NetSDK_JAVA Intelligent Building

Programming Manual

V2.0.1

Foreword

General

Welcome to use NetSDK_JAVA (hereinafter referred to be "SDK") programming manual (hereinafter referred to as "the Manual").

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

The Manual describes the SDK interfaces and processes of the general function modules for intelligent building products. 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
V2.0.1	Added the intelligent alarm event.	July 2025
V2.0.0	Updated Foreword.	February 2025

Version	Revision Content	Release Time
V1.0.0	First release.	December 2023

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.

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.dll	Library file
	avnetsdk.dll	Library file
Configuration library	dhconfigsdk.h	Header file
	dhconfigsdk.dll	Library file
Auxiliary library of playing (coding and decoding)	dhplay.dll	Playing library
Auxiliary library of "dhnetsdk.dll"	StreamConvertor.dll	Transcoding library

Table 1-2 The files included in 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	Header file
	libdhconfigsdk.so	Library file
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.
- Jdk version: jdk1.6; jdk1.8
- 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

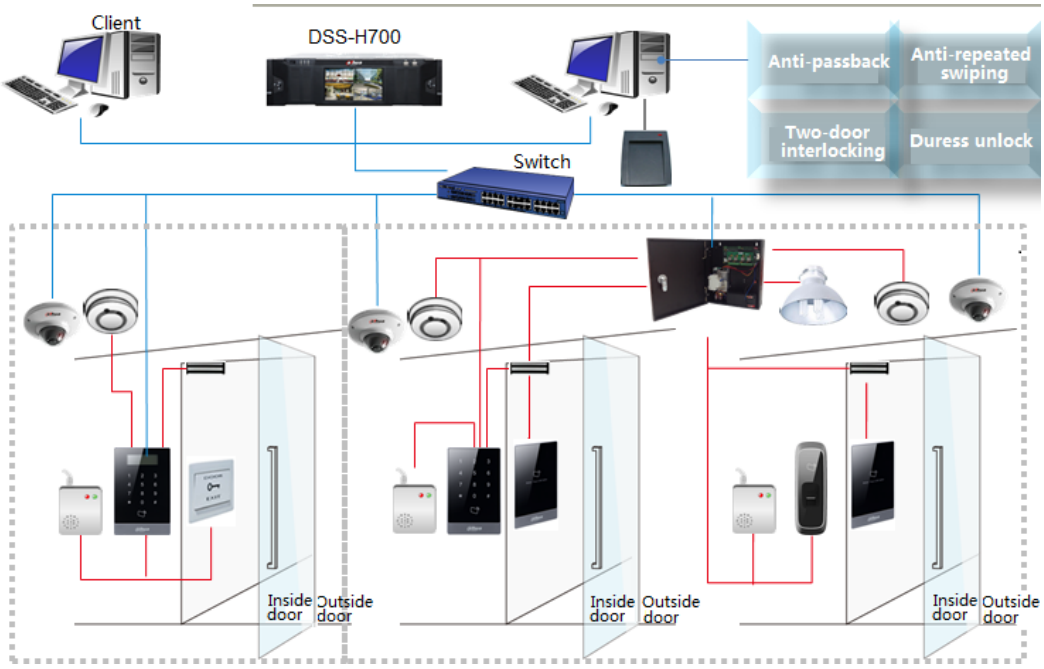
- ◇ DHI-ASI4213Y, DHI-ASI4214Y
- ◇ DHI-ASI7213X, DHI-ASI7213Y, DHI-ASI7213Y-D, DHI-ASI7213Y-V3
- ◇ DHI-ASI7214X, DHI-ASI7214Y, DHI-ASI7214Y-D, DHI-ASI7214Y-V3
- ◇ DHI-ASI7223X-A, DHI-ASI7223Y-A, DHI-ASI7223Y-A-V3
- ◇ DHI-ASI8213Y-V3
- ◇ DHI-ASI8214Y, DHI-ASI8214Y(V2), DHI-ASI8214Y-V3
- ◇ DHI-ASI8223Y, 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

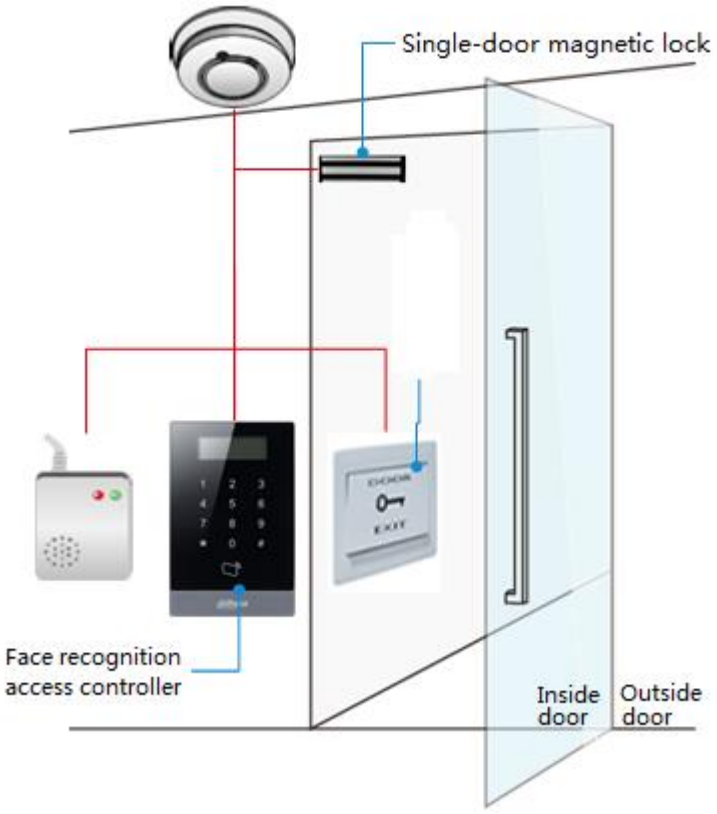
- Typical scenario.

Figure 1-1 Typical scenario



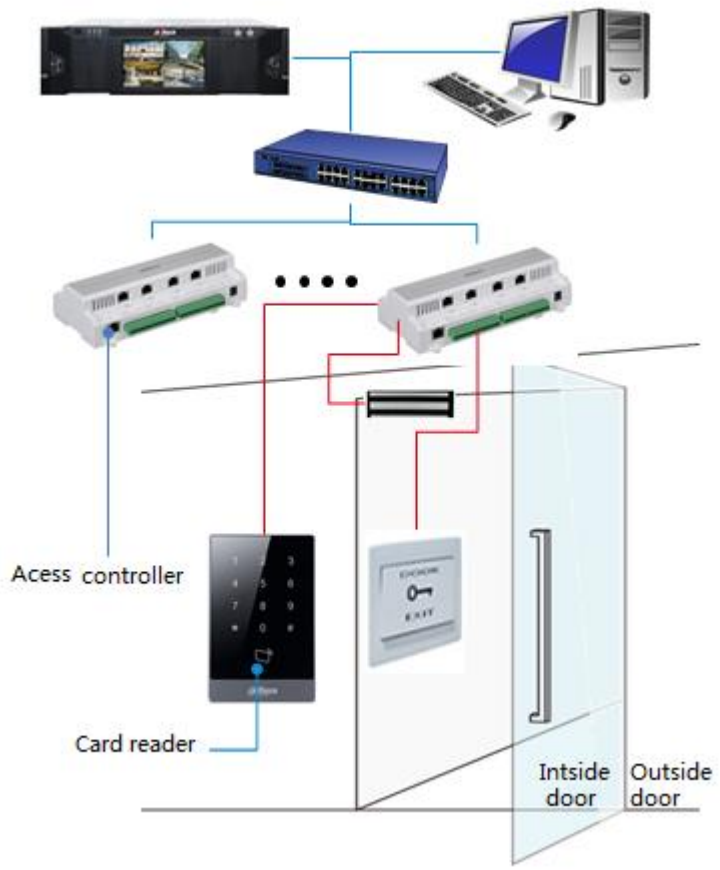
- Micro access control for small-sized office.

Figure 1-2 Micro access control



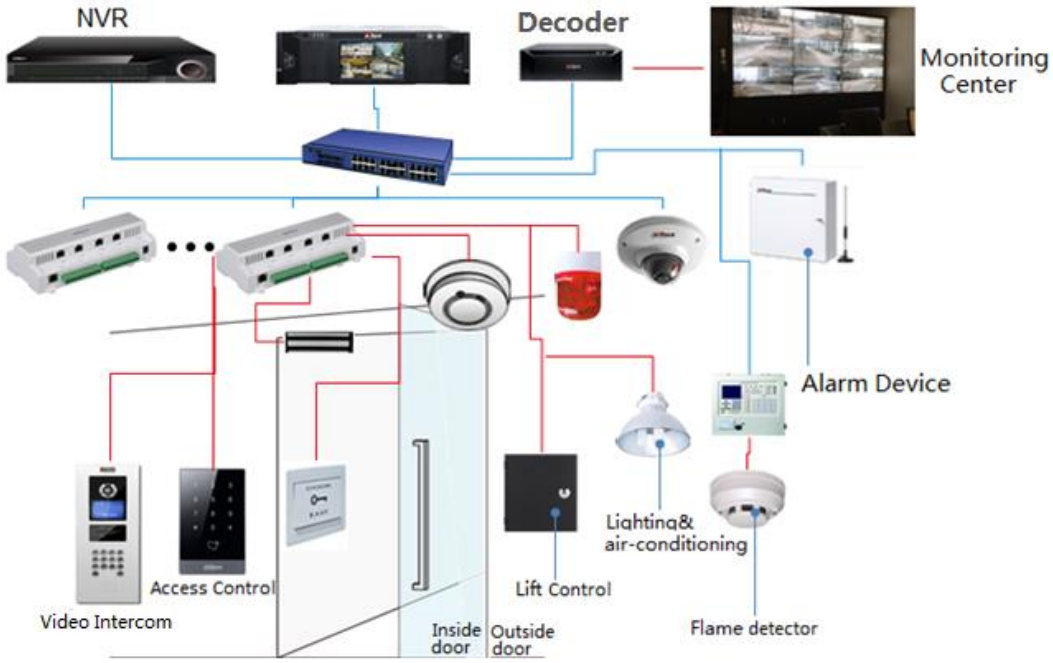
- Network access control for medium and small-sized intelligent building, treasury house and jail monitoring projects.

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.

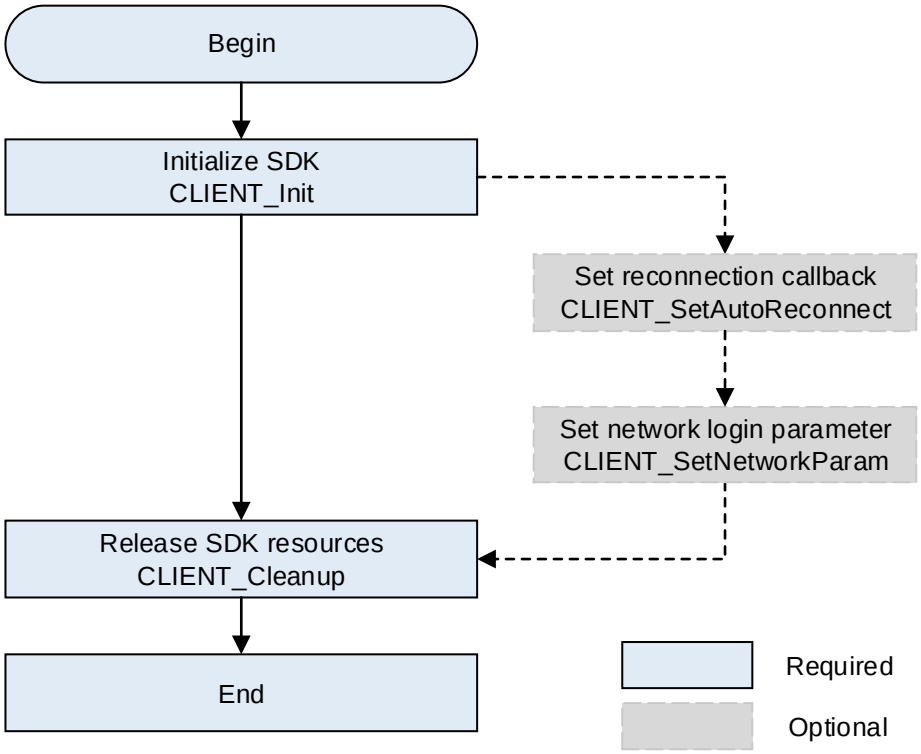
2.1.1.2 Interface Overview

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

```
/**
 * Realized by the login interface.
 * It mainly includes initialization, login and logout.
 */
public class LoginModule {

    public static NetSDKLib netsdk              = NetSDKLib.NETSDK_INSTANCE;
    public static NetSDKLib configsdk    = NetSDKLib.CONFIG_INSTANCE;

    // Login handle
    public static LLong m_hLoginHandle = new LLong(0);

    private static boolean bInit       = false;
    private static boolean bLogopen = false;

    //Initialization
    public static boolean init(NetSDKLib.fDisConnect disConnect,
NetSDKLib.fHaveReConnect haveReConnect) {
        bInit = netsdk.CLIENT_Init(disConnect, null);
        if(!bInit) {
            System.out.println("Initialize SDK failed");
            return false;
        }

        //Open the log, optional.
        NetSDKLib.LOG_SET_PRINT_INFO setLog = new
NetSDKLib.LOG_SET_PRINT_INFO();
        File path = new File("./sdklog/");
        if (!path.exists()) {
            path.mkdir();
        }
        String logPath = path.getAbsoluteFile().getParent() + "\\sdklog\\" + ToolKits.getDate()
+ ".log";
        setLog.nPrintStrategy = 0;
        setLog.bSetFilePath = 1;
        System.arraycopy(logPath.getBytes(), 0, setLog.szLogFilePath, 0,
logPath.getBytes().length);
        System.out.println(logPath);
```

```
        setLog.bSetPrintStrategy = 1;
        bLogopen = netsdk.CLIENT_LogOpen(setLog);
        if(!bLogopen ) {
            System.err.println("Failed to open NetSDK log");
        }

        // Set the callback interface for reconnection when it is disconnected. After the callback function is set for disconnection, the SDK automatically reconnects the device when disconnection occurs.
        // This operation is optional, but we recommend you set it.
        netsdk.CLIENT_SetAutoReconnect(haveReConnect, null);

        // Set the login timeout period and number of attempts, optional.
        int waitTime = 5000; // 5S Set the login request response timeout period to 5 seconds.
        int tryTimes = 1;       // Try to establish a link once during login.
        netsdk.CLIENT_SetConnectTime(waitTime, tryTimes);

        // Set more network parameters, nWaittime for NET PARAM, nConnectTryNum members and CLIENT SetConnectTime
        // The login timeout period set for the interface is the same as the login attempts. It is optional.
        NetSDKLib.NET_PARAM netParam = new NetSDKLib.NET_PARAM();
        netParam.nConnectTime = 10000;          // The timeout period for trying to establish a link during login.
        netParam.nGetConnInfoTime = 3000;       // Set the timeout period of the sub connection.          netsdk.CLIENT_SetNetworkParam(netParam);

        return true;
    }

    // Clear environment
    public static void cleanup() {
        if(bLogopen) {
            netsdk.CLIENT_LogClose();
        }

        if(bInit) {
            netsdk.CLIENT_Cleanup();
        }
    }
}
```

2.1.2 Device Login

2.1.2.1 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.

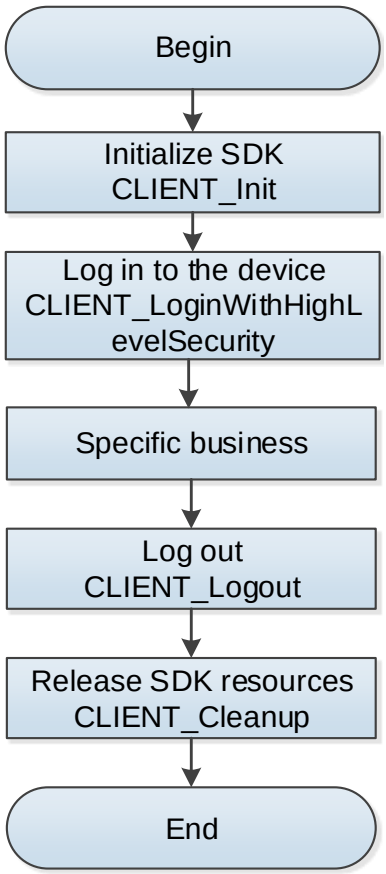
2.1.2.2 Interface Overview

Table 2-2 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.

2.1.2.3 Process Description

Figure 2-2 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 2-3.

Table 2-3 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.

2.1.2.4 Example Code

```
public class LoginModule {


    public static NetSDKLib netsdk             = NetSDKLib.NETSDK_INSTANCE;
```

```
    public static NetSDKLib configsdk    = NetSDKLib.CONFIG_INSTANCE;

    //SDK initialization. Skip SDK clearing.

    // Device information
    public static NetSDKLib.NET_DEVICEINFO_Ex m_stDeviceInfo = new
NetSDKLib.NET_DEVICEINFO_Ex();

    //Login handle
    public static LLong m_hLoginHandle = new LLong(0);

    //Log in to the device
    public static boolean login(String m_strIp, int m_nPort, String m_strUser, String
m_strPassword) {
    //Input parameters
NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY pstInParam=
new NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY();
pstInParam. szIP= m_strIp;
pstInParam.nport= m_nPort;
pstInParam.szUserName= m_strUser;
pstInParam.szPassword= m_strPassword;
//Output parameters
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY pstOutParam=
new NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY();
    m_hLoginHandle =
netsdk.CLIENT_LoginWithHighLevelSecurity(NET_IN_LOGIN_WITH_HIGHLEVEL_SECURIT
Y pstInParam, NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY pstOutParam);

if(m_hLoginHandle.longValue() == 0) {
            System.err.printf("Login  Device[%s]  Port[%d]Failed.  %s\n",  m_strIp,  m_nPort,
ToolKits.getErrorCodePrint());
        } else {
            System.out.println("Login Success ");
        }

        return m_hLoginHandle.longValue() == 0? false:true;
    }

    // Log out of the device
    public static boolean logout() {
```

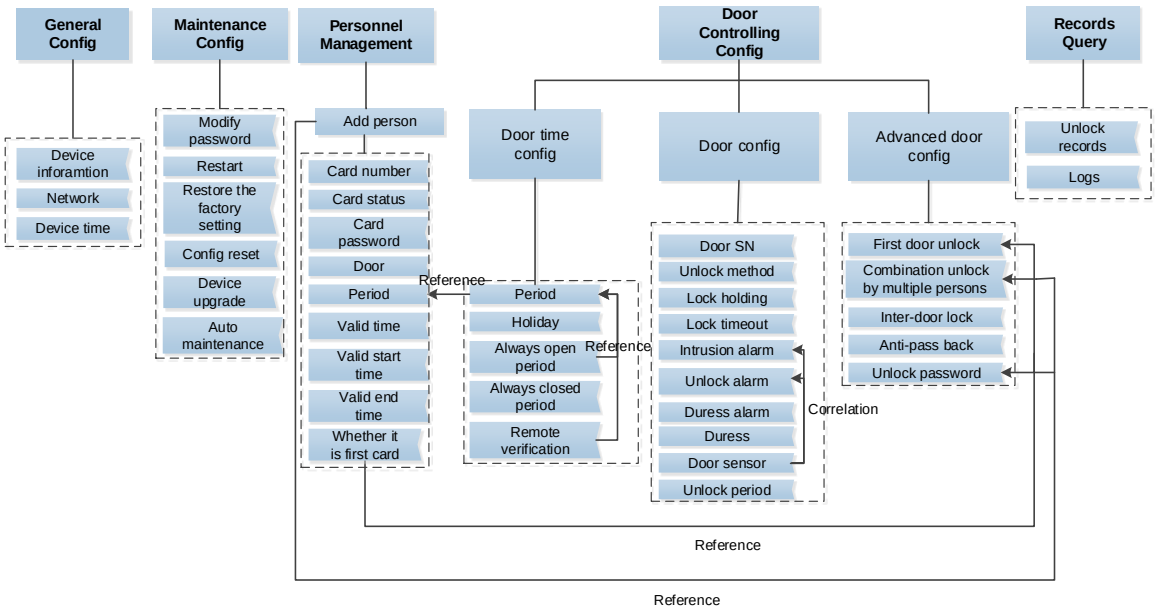
```
            if(m_hLoginHandle.longValue() == 0) {
                return false;
            }

            boolean bRet = netsdk.CLIENT_Logout(m_hLoginHandle);
            if(bRet) {
                m_hLoginHandle.setValue(0);
            }

            return bRet;
        }
}
```

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

Figure 2-3 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.

2.2.1 Access Control

2.2.1.1 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.

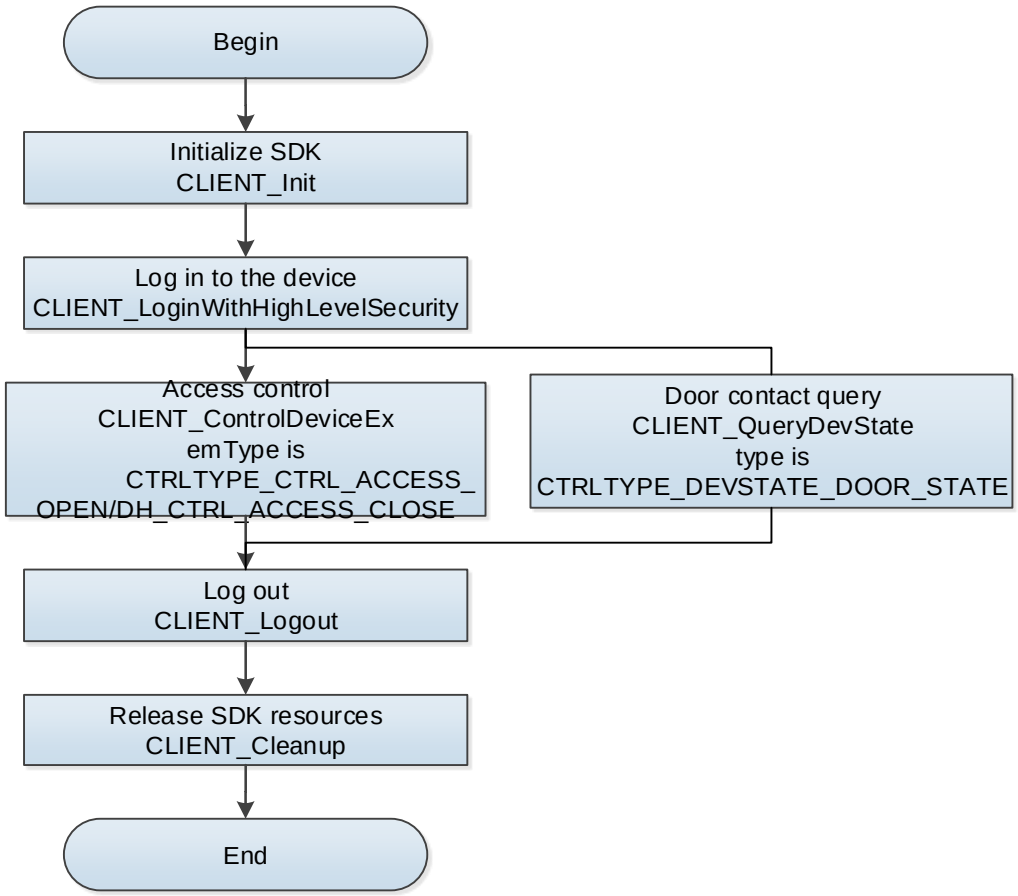
2.2.1.2 Interface Overview

Table 2-4 Description of access control interface

Interface	Description
CLIENT_ControlDeviceEx	Device control extension interface.
CLIENT_QueryDevState	Status query interface.

2.2.1.3 Process Description

Figure 2-4 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 **CTRLTYPE_CTRL_ACCESS_OPEN**.
- Close the access: The **emType** value is **CTRLTYPE_CTRL_ACCESS_CLOSE**.

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

Step 5 Type: CTRLTYPE_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.

2.2.1.4 Example Code

```
/**
 * Close the door
 */
public void closeDoor() {
    final NetSDKLib.NET_CTRL_ACCESS_CLOSE close = new
NetSDKLib.NET_CTRL_ACCESS_CLOSE();
    close.nChannelID = 0; // The corresponding door number. - How to open all the doors.
    close.write();
    boolean result = netsdkApi.CLIENT_ControlDeviceEx(loginHandle,
                                    NetSDKLib.CtrlType.CTRLTYPE_CTRL_ACCESS_CLOSE,
                                    close.getPointer(),
                                    null,
                                    5000);
    close.read();
    if (!result) {
        System.err.println("close error: 0x" + Long.toHexString(netsdkApi.CLIENT_GetLastError()));
    }
}

/**
 * Search for the door status (open or closed)
 */
public void queryDoorStatus() {
    int cmd = NetSDKLib.NET_DEVSTATE_DOOR_STATE;
    NetSDKLib.NET_DOOR_STATUS_INFO doorStatus = new
NetSDKLib.NET_DOOR_STATUS_INFO();
    IntByReference retLenByReference = new IntByReference(0);

    doorStatus.write();
    boolean bRet = netsdkApi.CLIENT_QueryDevState(loginHandle,
                                        cmd,
                                        doorStatus.getPointer(),
                                        doorStatus.size(),
                                        retLenByReference,
                                        3000);
    doorStatus.read();
    if (!bRet) {
```

```
            System.err.println("Failed to queryDoorStatus. Error Code 0x"
                    + Integer.toHexString(netsdkApi.CLIENT_GetLastError()));
            return;
        }

        String stateType[] = {"Unknown ", "Door Open", "Door Closed", "Door Abnormally Open"};
        System.out.println("doorStatus -> Channel: " + doorStatus.nChannel
                    + " type: " + stateType[doorStatus.emStateType]);
    }
```

2.2.2 Alarm Event

2.2.2.1 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.

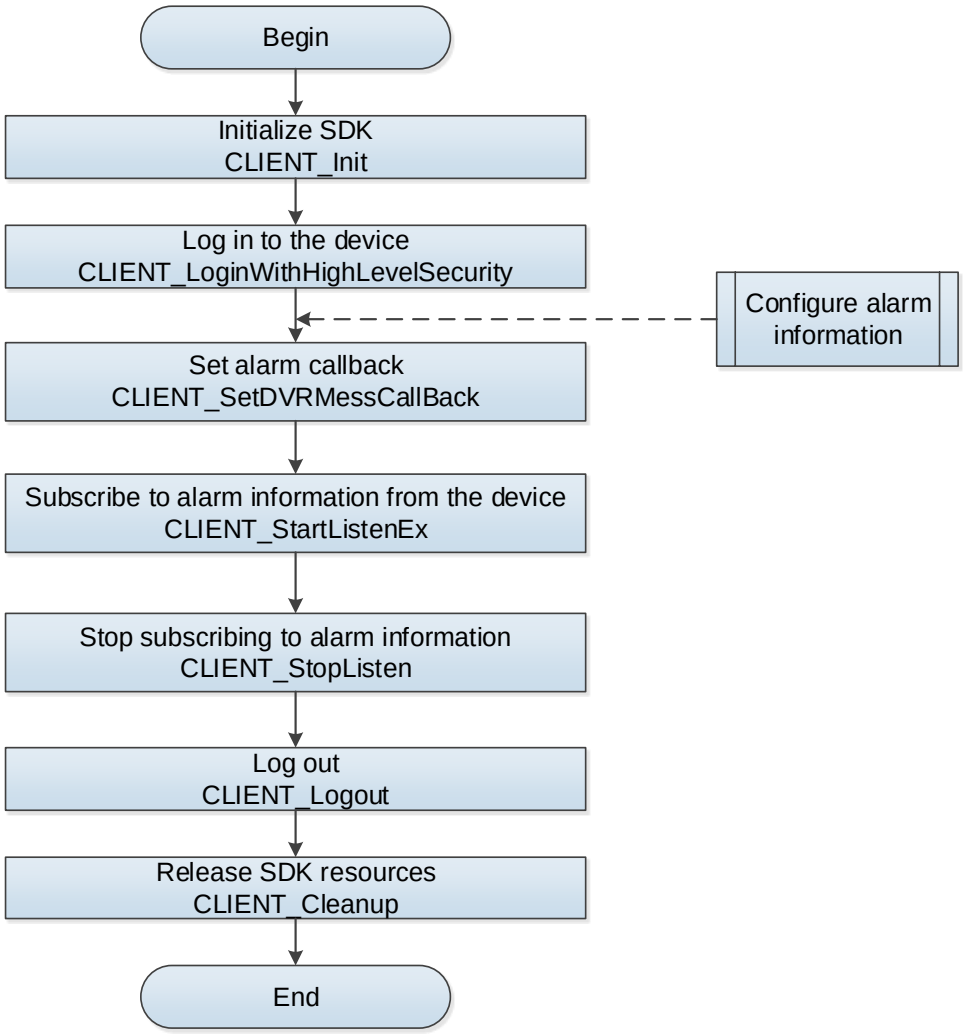
2.2.2.2 Interface Overview

Table 2-5 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.

2.2.2.3 Process Description

Figure 2-5 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.

2.2.2.4 Example Code

```
/**
 * Subscribe to alarm information
```

```
 */
public void startListen() {
	// Set the alarm callback function
	netsdk.CLIENT_SetDVRMessCallBack(fAlarmDataCB.getCallBack(), null);

	// Subscribe to alarms
	boolean bRet = netsdk.CLIENT_StartListenEx(m_hLoginHandle);
	if (!bRet) {
		System.err.println("Failed to subscribe to alarms. LastError = 0x%x\n" +
netsdk.CLIENT_GetLastError());
	}
	else {
		System.out.println("Successfully subscribed to alarm.");
	}
}

/**
 * Unsubscribe from alarm information
 */
public void stopListen() {
	// Stop subscribing to alarms
	boolean bRet = netsdk.CLIENT_StopListen(m_hLoginHandle);
	if (bRet) {
		System.out.println("Unsubscribe from alarm information.");
	}
}
/**
 * Alarm information callback function prototype. We recommend writing it as singleton pattern.
 */
private static class fAlarmDataCB implements NetSDKLib.fMessCallBack{
	private fAlarmDataCB(){}

	private static class fAlarmDataCBHolder {
		private static fAlarmDataCB callback = new fAlarmDataCB();
	}

	public static fAlarmDataCB getCallBack() {
		return fAlarmDataCB.fAlarmDataCBHolder.callback;
	}

	public boolean invoke(int lCommand, NetSDKLib.LLong lLoginID, Pointer pStuEvent, int dwBufLen,
String strDeviceIP, NativeLong nDevicePort, Pointer dwUser){
		switch (lCommand)
		{
			case NetSDKLib.NET_ALARM_TALKING_INVITE:	{	// The device requests the other
party to initiate an intercom event (corresponding to the structure ALARM_TALKING_INVITE_INFO)
				System.out.println("The device requests the other party to initiate an intercom event");
```

```
                    NetSDKLib.ALARM_TALKING_INVITE_INFO msg = new
NetSDKLib.ALARM_TALKING_INVITE_INFO();
                    ToolKits.GetPointerData(pStuEvent, msg);
                    System.out.println("emCaller :" + msg.emCaller);
                    System.out.println("szCallID :" + new String(msg.szCallID).trim());
                    System.out.println("nLevel :" + msg.nLevel);

                    /**
                     Whether RealUTC is valid. When bRealUTC is TRUE, use RealUTC. Otherwise use
the stuTime field.
                     */
                    int bRealUTC
                            = msg.bRealUTC;
                    System.out.println("bRealUTC :" + bRealUTC);
                    if(bRealUTC==1){
                        System.out.println("RealUTC :" + msg.RealUTC.toStringTime());
                    }else {
                        System.out.println("stuTime :" + msg.stuTime.toStringTime());

                    }

                    NetSDKLib.TALKINGINVITE_REMOTEDEVICEINFO stuRemoteDeviceInfo
                            = msg.stuRemoteDeviceInfo;
                    int emProtocol = stuRemoteDeviceInfo.emProtocol;

                    System.out.println("emProtocol:" + emProtocol);


                    try {
                        System.out.println("szIP:" +     new String(stuRemoteDeviceInfo.szIP,encode));

                        System.out.println("nPort:" +    stuRemoteDeviceInfo.nPort);

                        System.out.println("szUser:" +     new
String(stuRemoteDeviceInfo.szUser,encode));

                        System.out.println("szPassword:" +     new
String(stuRemoteDeviceInfo.szPassword,encode));

                    } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                    }


                    break;
                }
                case NetSDKLib.NET_ALARM_TALKING_HANGUP    :{ // The device hangs up the
```

```
intercom event (corresponding to structure ALARM_TALKING_HANGUP_INFO)
                    System.out.println("The device hangs up the intercom event");
                    ALARM_TALKING_HANGUP_INFO msg=new ALARM_TALKING_HANGUP_INFO();
                    ToolKits.GetPointerData(pStuEvent, msg);

                    try {
                        System.out.println("szRoomNo :" + new String(msg.szRoomNo,encode));
                    } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                    }
                    /**
                     Whether RealUTC is valid. When bRealUTC is TRUE, use RealUTC. Otherwise use
the stuTime field.
                     */
                    int bRealUTC
                            = msg.bRealUTC;
                    System.out.println("bRealUTC :" + bRealUTC);
                    if(bRealUTC==1){
                        System.out.println("RealUTC :" + msg.RealUTC.toStringTime());

                    }else {
                        System.out.println("stuTime :" + msg.stuTime.toStringTime());

                    }


                    byte[] szCaller = msg.szCaller;
                    try {
                        System.out.println("szCaller :" + new String(szCaller,encode));
                    } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                    }
                    break;
                }

                default:
                    System.out.println("Event:"+lCommand);

                    break;
            }
            return true;
        }
}
```

2.2.3 Intelligent Alarm Event with Image

2.2.3.1 Introduction

Intelligent subscription means that intelligent devices perform analysis on real-time streams. If specific events are detected, they will be sent to the user. Intelligent events include traffic violation, available parking spaces, and more.

Intelligent subscription implementation: SDK automatically connects to the device and subscribes to the intelligent event function from the device. When the device detects an intelligent event, it will send the event to SDK immediately.

For supported intelligent subscription events, see the constants starting with EVENT_IVS_ in NetSDKLib.java, including general events such as occupied lane and vehicle violations.

2.2.3.2 Interface Overview

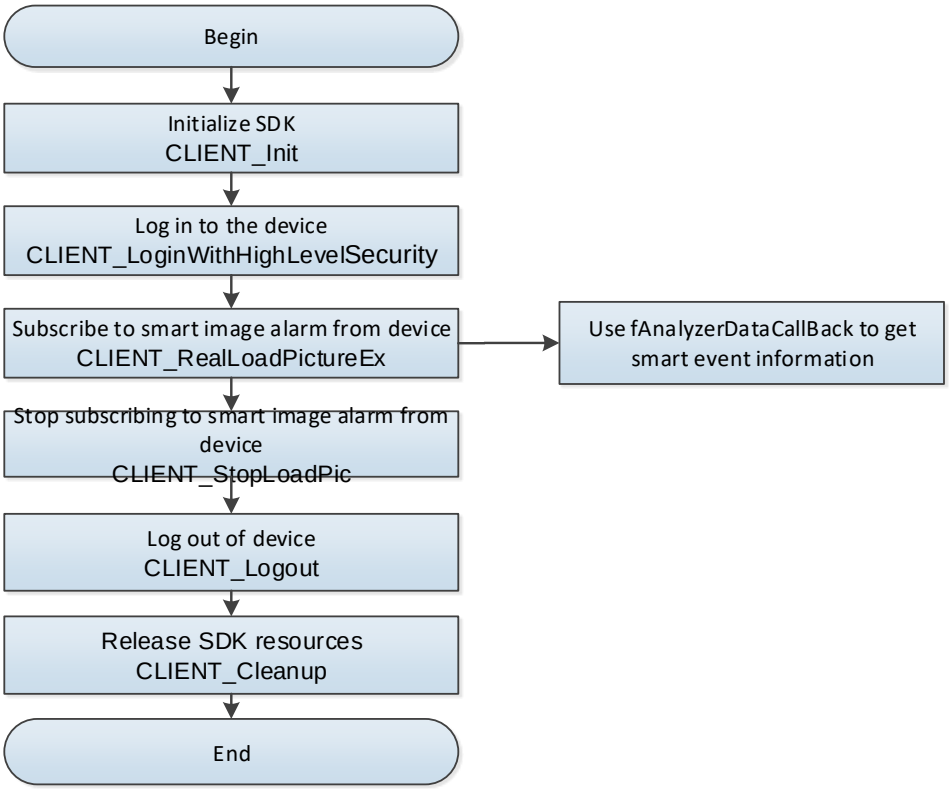
Table 2-6 Description of reporting intelligent traffic time interfaces

Interface	**Implication**
CLIENT_RealLoadPictureEx	Subscribe for intelligent events.
CLIENT_StopLoadPic	Cancel subscription for intelligent events.
fAnalyzerDataCallBack	Callback to get the intelligent event information.

2.2.3.3 Process Description

For the process of reporting intelligent subscription events, see Figure 2-6.

Figure 2-6 Service of reporting intelligent subscription events



Process

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

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

Step 3 Call **CLIENT_RealLoadPictureEx** to subscribe to intelligent events from the device.

Step 4 After successful subscription, use the callback set by **fAnalyzerDataCallBack** to inform the user of intelligent events reported by the device.

Step 5 To stop subscribing to intelligent events, call **CLIENT_StopLoadPic**.

Step 6 After using the function, call **CLIENT_Logout** to log out of the device.

Step 7 After using SDK, call **CLIENT_Cleanup** function to release SDK resources.

Note

- Subscription event type: If you need to report different intelligent events at the same time, you can subscribe to all intelligent events (EVENT_IVS_ALL) or a single intelligent event.
- Set whether to receive images: Because some devices are in 3G or 4G network environment, when SDK connects to the device, if you do not need to receive images, you can set the parameter bNeedPicFile in CLIENT_ RealLoadPictureEx interface to False, for only receiving intelligent traffic event information without images.
- Subscribe to all channels by sending -1 as the channel number. Some intelligent traffic products do not support full channel subscriptions. If subscribing with -1 does not work, try subscribing to a single channel instead.

2.2.3.4 Example Code

```
//This is an example of access control event
```

```
//Omit SDK initialization and access control device login
    // Login handle
    private NetSDKLib.LLong loginHandle = new NetSDKLib.LLong(0);
    // Intelligent event subscription handle
    private NetSDKLib.LLong m_attachHandle = new NetSDKLib.LLong(0);
    // Listen
    /**
     * Subscribe to intelligent tasks
     */
    public void AttachEventRealLoadPic() {
        // Unsubscribe first. The device does not check for duplicate subscriptions, so if you subscribe again without unsubscribing, you might receive duplicate events.
        this.DetachEventRealLoadPic();
        // Needs images
        int bNeedPicture = 1;
        //EVENT_IVS_ALL means subscribing to all intelligent events
        m_attachHandle = netsdk.CLIENT_RealLoadPictureEx(loginHandle, channelId, EVENT_IVS_ALL, bNeedPicture, AnalyzerDataCB.getInstance(), null, null);
        /**
         * // EVENT_IVS_WORKCLOTHES_DETECT Helmet detection event
         * // EVENT_IVS_SMOKING_DETECT Smoking detection event
         **/
        if (m_attachHandle.longValue() != 0) {
            System.out.printf("Chn[%d] CLIENT_RealLoadPictureEx Success\n", channelId);
        } else {
            System.out.printf("Ch[%d] CLIENT_RealLoadPictureEx Failed!LastError = %s\n", channelId,
                    ToolKits.getErrorCode());
        }
    }


    /**
     * Alarm event (intelligent) callback
     */
    private static class AnalyzerDataCB implements NetSDKLib.fAnalyzerDataCallBack {
        private final File picturePath;
        private static AnalyzerDataCB instance;

        private AnalyzerDataCB() {
```

```
            picturePath = new File("./AnalyzerPicture/");
            if (!picturePath.exists()) {
                picturePath.mkdirs();
            }
        }

        public static AnalyzerDataCB getInstance() {
            if (instance == null) {
                synchronized (AnalyzerDataCB.class) {
                    if (instance == null) {
                        instance = new AnalyzerDataCB();
                    }
                }
            }
            return instance;
        }

        @Override
        public int invoke(NetSDKLib.LLong lAnalyzerHandle, int dwAlarmType, Pointer pAlarmInfo, Pointer pBuffer, int dwBufSize,
                          Pointer dwUser, int nSequence, Pointer reserved) {
            if (lAnalyzerHandle == null || lAnalyzerHandle.longValue() == 0) {
                return -1;
            }

            switch (dwAlarmType) {
                case EVENT_IVS_WORKCLOTHES_DETECT: // Helmet detection event
                {
                    NetSDKLib.DEV_EVENT_WORKCLOTHES_DETECT_INFO msg = new NetSDKLib.DEV_EVENT_WORKCLOTHES_DETECT_INFO();
                    ToolKits.GetPointerData(pAlarmInfo, msg);
                    if (msg.stuSceneImage != null && msg.stuSceneImage.nLength > 0) {
                        String bigPicture = picturePath + "\\" + System.currentTimeMillis() + ".jpg";
                        ToolKits.savePicture(pBuffer, msg.stuSceneImage.nOffSet, msg.stuSceneImage.nLength, bigPicture);
                        if (msg.stuHumanImage != null && msg.stuHumanImage.nLength > 0) {
                            String smallPicture = picturePath + "\\" + System.currentTimeMillis() + "small.jpg";
```

```
                    ToolKits.savePicture(pBuffer, msg.stuHumanImage.nOffSet,
msg.stuHumanImage.nLength, smallPicture);
                }
            }
            System.out.println(" Helmet detection event(UTC)：" + msg.UTC + "
Channel number:" + msg.nChannelID);
            break;
        }
        case NetSDKLib.EVENT_IVS_SMOKING_DETECT : { // Smoking
detection event (Corresponding to DEV_EVENT_SMOKING_DETECT_INFO)
            System.out.printf("Smoking detection event");
            DEV_EVENT_SMOKING_DETECT_INFO msg = new
DEV_EVENT_SMOKING_DETECT_INFO();
            ToolKits.GetPointerData(pAlarmInfo, msg);
            String Picture = picturePath + "\\" + "smoking_" +
System.currentTimeMillis() + ".jpg";

            if (dwBufSize > 0) {
                ToolKits.savePicture(pBuffer, msg.stuImageInfo[0].nOffset,
msg.stuImageInfo[0].nLength, Picture);
            }
            System.out.println("Smoking detection event time(UTC)：" + msg.UTC +
" Start time:" + msg.stuObject.stuStartTime + " End time:"
                    + msg.stuObject.stuEndTime);
            break;
        }
        case EVENT_IVS_GRANARY_TRANS_ACTION_DETECTION:{ //Report
changes in grain levels detection event (Corresponding to
NET_DEV_EVENT_GRANARY_TRANS_ACTION_DETECTION_INFO)
            System.out.println("Report changes in grain levels detection event");
            NET_DEV_EVENT_GRANARY_TRANS_ACTION_DETECTION_INFO
msg = new NET_DEV_EVENT_GRANARY_TRANS_ACTION_DETECTION_INFO();
            ToolKits.GetPointerData(pAlarmInfo, msg);
            System.out.println("Report changes in grain levels detection event
time(stuUTC):" + msg.stuUTC);
            System.out.println("Total number of messages in the object list
(nObjectsCount):" + msg.nObjectsCount);
            break;
        }
        case EVENT_IVS_REGION_PROPORTION_DETECTION:{ // Area
proportion detection event (Corresponding to
NET_DEV_EVENT_REGION_PROPORTION_DETECTION_INFO)
```

```
                        System.out.println("Area proportion detection event");
                        NET_DEV_EVENT_REGION_PROPORTION_DETECTION_INFO msg
= new NET_DEV_EVENT_REGION_PROPORTION_DETECTION_INFO();
                        ToolKits.GetPointerData(pAlarmInfo, msg);
                        System.out.println("Area proportion detection event reporting
time(stuUTC):" + msg.stuUTC);
                        System.out.println("Total number of messages in the object list
(nObjectsCount): " + msg.nObjectsCount);
                        break;
                    }
                    default:
                        System.out.println("Other events----------"+ dwAlarmType);
                        break;
                }
                return 0;
            }
        }
        /**
         * Stop listening to intelligent events
         */
        public void DetachEventRealLoadPic() {
            if (m_attachHandle.longValue() != 0) {
                netsdk.CLIENT_StopLoadPic(m_attachHandle);
                System.out.println("CLIENT_StopLoadPic Success");
            }
        }
```

2.2.4 Viewing Device Information

2.2.4.1 Capability Set Query

2.2.4.1.1 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.

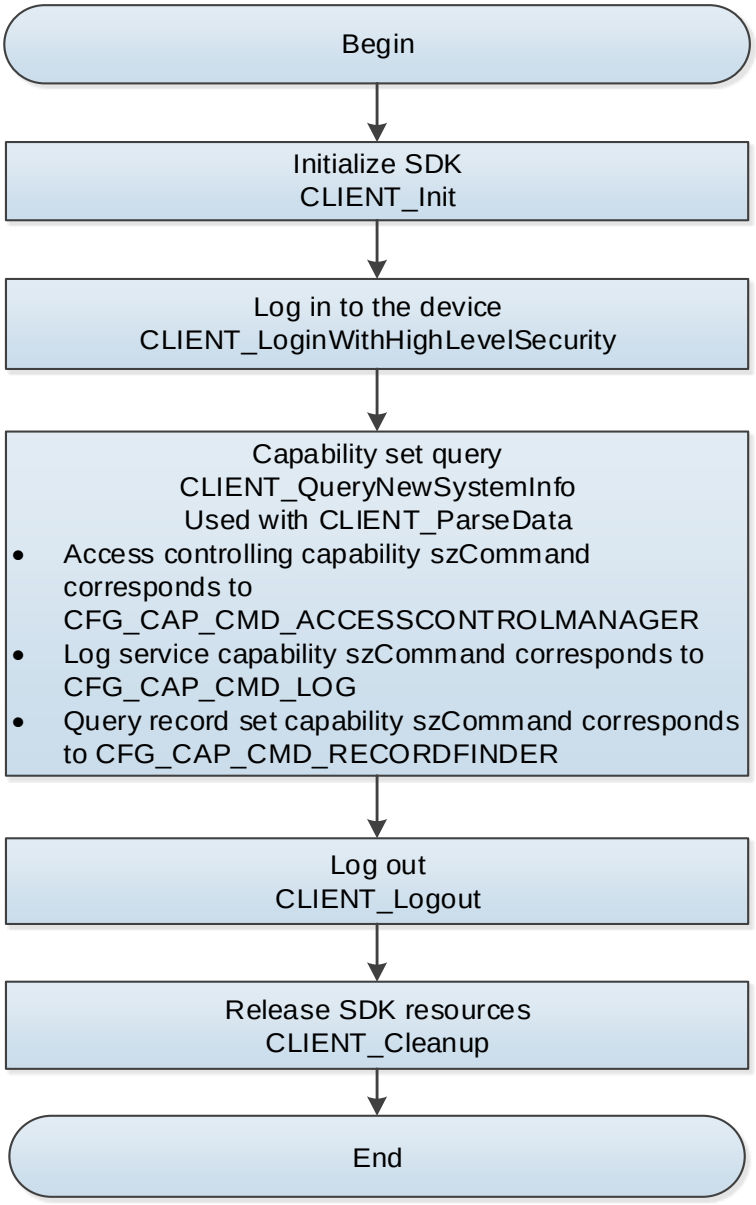
2.2.4.1.2 Interface Overview

Table 2-7 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.

2.2.4.1.3 Process Description

Figure 2-7 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 2-8 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.

2.2.4.1.4 Example Code

```
/**
 * Optimized the CLIENT QueryNewSystemInfo interface to obtain the video analysis capability set.
 * 1. The structure CFG CAP ANALYSE INFO has a byte length of 54874632. It takes a lot of time to construct the object and analyse it. It calculates bytes and offsets to obtain the corresponding field information.
 */
public void queryNewSystemInfoOpt() {
    // The structure CFG CAP ANALYSE INFO has a byte length of 54874632. It takes a lot of time to construct the object and analyse it. It calculates bytes and offsets to obtain the corresponding field information.
    //long len = new CFG_CAP_ANALYSE_INFO().size(); len = 54874632
    byte[] data = new byte[54874632];
    IntByReference error = new IntByReference(0);
    boolean result = netsdk.CLIENT_QueryNewSystemInfo(m_hLoginHandle, EM_NEW_QUERY_SYSTEM_INFO.CFG_CAP_CMD_VIDEOANALYSE.getValue(), 0, data, data.length, error, 5000);
    if (!result) {
        System.out.println("query system info failed.error is" + ENUMERROR.getErrorMessage());
    }
    //Parse the capability information
    Pointer pointer = new Memory(data.length);
    result = configsdk.CLIENT_ParseData(EM_NEW_QUERY_SYSTEM_INFO.CFG_CAP_CMD_VIDEOANALYSE.getValue(), data, pointer, data.length, null);
    if (!result) {
        System.out.println("parse system info failed.error is " + ENUMERROR.getErrorMessage());
    }
    CFG_CAP_ANALYSE_INFO_OPT info = new CFG_CAP_ANALYSE_INFO_OPT();
    // The byte shift is 0 before the nSupportedData field of Structure CFG CAP ANALYSE. The byte shift is 4100 before the szSceneName field.
    info.getPointer().write(0, pointer.getByteArray(0, 4100), 0, info.size());
    info.read();
    System.out.println("Supported number of scenes:"+info.nSupportedSceneNum);
    System.out.println("List of supported scenes:");// Enumeration. Refer to @{ @link EM_SCENE_TYPE}
    MaxNameByteArrInfo[] szSceneName = info.szSceneName;// Determines whether there are enumerated values FaceAnalysis.
```

```
    String value = EM_SCENE_TYPE.getNoteByValue(27);//value = FaceAnalysis Face Analysis
    for (int i = 0; i < info.nSupportedSceneNum; i++) {
        //System.out.println(new String(szSceneName[i].name).trim());
        if(value.trim().equals(new String(szSceneName[i].name).trim())) {
            System.out.println("IPC has target recognition capability ");
        }
    }
}
```

2.2.4.2 Viewing Device Version and MAC

2.2.4.2.1 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.

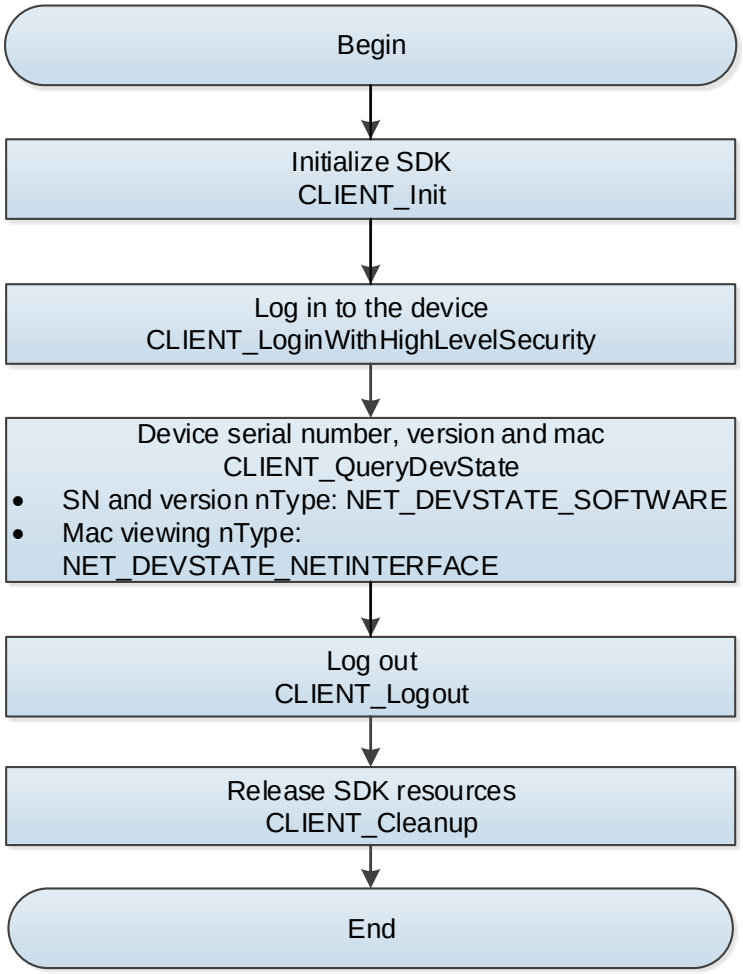
2.2.4.2.2 Interface Overview

Table 2-9 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).

2.2.4.2.3 Process Description

Figure 2-8 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.

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.

2.2.4.2.4 Example Code

```
/**
 * Obtain the device software version.
 *
 */
public void QueryDevDeviceVersionStateTest() {
    NetSDKLib.NETDEV_VERSION_INFO info = new NetSDKLib.NETDEV_VERSION_INFO();
    info.write();
    boolean bRet = netSdk.CLIENT_QueryDevState(loginHandle, NET_DEVSTATE_SOFTWARE,
info.getPointer(), info.size(), new IntByReference(0), 3000);
    if (!bRet) {
```

```
            System.err.println("QueryDevState DEV STATE of SOFTWARE failed: " +
ToolKits.getErrorCode());
            return;
        }
        info.read();
        System.out.println("QueryDevState DEV STATE of SOFTWARE succeed");

        System.out.println("szSoftWareVersion Software version: " + new
String(info.szSoftWareVersion).trim());
        System.out.println("szDevSerialNo Serial number: " + new String(info.szDevSerialNo).trim());
        int buildData = info.dwSoftwareBuildDate;
        int day = buildData & 0xff;
        buildData >>= 8;
        int month = buildData & 0xff;
        int year = buildData >> 8;
        System.out.println("BuildData Date Compiled: " + year + "-" + month + "-" + day);
}

/**
 * Get all mobile network interfaces
 */
public void getNetAppMobileInterface() {
        // Input parameters
        NET_IN_NETAPP_GET_MOBILE_INTERFACE pstuIn = new
NET_IN_NETAPP_GET_MOBILE_INTERFACE();
        // Output parameters
        NET_OUT_NETAPP_GET_MOBILE_INTERFACE pstuOut = new
NET_OUT_NETAPP_GET_MOBILE_INTERFACE();

        pstuIn.write();
        pstuOut.write();
        boolean bRet = netsdk.CLIENT_RPC_NetApp(m_hLoginHandle,
EM_RPC_NETAPP_TYPE.EM_PRC_NETAPP_TYPE_GET_MOBILE_INTERFACE.getId(),
pstuIn.getPointer(), pstuOut.getPointer(), 3000);
        if(bRet) {
            pstuOut.read();
            System.out.println("Number of valid network interfaces:"+pstuOut.nInterfaceNum);
            System.out.println("Mobile network interface information ");
            NETDEV_NETINTERFACE_INFO[] stuInterface = pstuOut.stuInterface;
            for (int i = 0; i < pstuOut.nInterfaceNum; i++) {
                int a = i+1;
                System.out.println("-----No. "+a+" ----");
                System.out.println("Valid or not:"+stuInterface[i].bValid);
                try {
                    System.out.println("Network port name:"+new String(stuInterface[i].szName,encode));
                    System.out.println("Actual number of network modes supported by
3G:"+stuInterface[i].nSupportedModeNum);
```

```
                    SupportedModeByteArr[] szSupportedModes= stuInterface[i].szSupportedModes;
                    String prt = "";
                    for (int j = 0; j < stuInterface[i].nSupportedModeNum; j++) {
                        if(j == (stuInterface[i].nSupportedModeNum-1) ) {
                            prt += new String
(szSupportedModes[j].supportedModeByteArr,encode).trim();
                        }else {
                            prt += new String
(szSupportedModes[j].supportedModeByteArr,encode).trim() +",";
                        }
                    }
                    System.out.println("Network mode supported by 3G:"+prt);

                    System.out.println("International mobile subscriber identity:"+new
String(stuInterface[i].szIMEI,encode));
                    System.out.println("Integrated Circuit Card Identification number or SIM card
number:"+new String(stuInterface[i].szICCID,encode));
                } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
                }

            }

        } else {
            System.err.println("getNetAppMobileInterface Failed!" + ToolKits.getErrorCode());
        }
}
```

2.2.5 Network Setting

2.2.5.1 IP Settings

2.2.5.1.1 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.

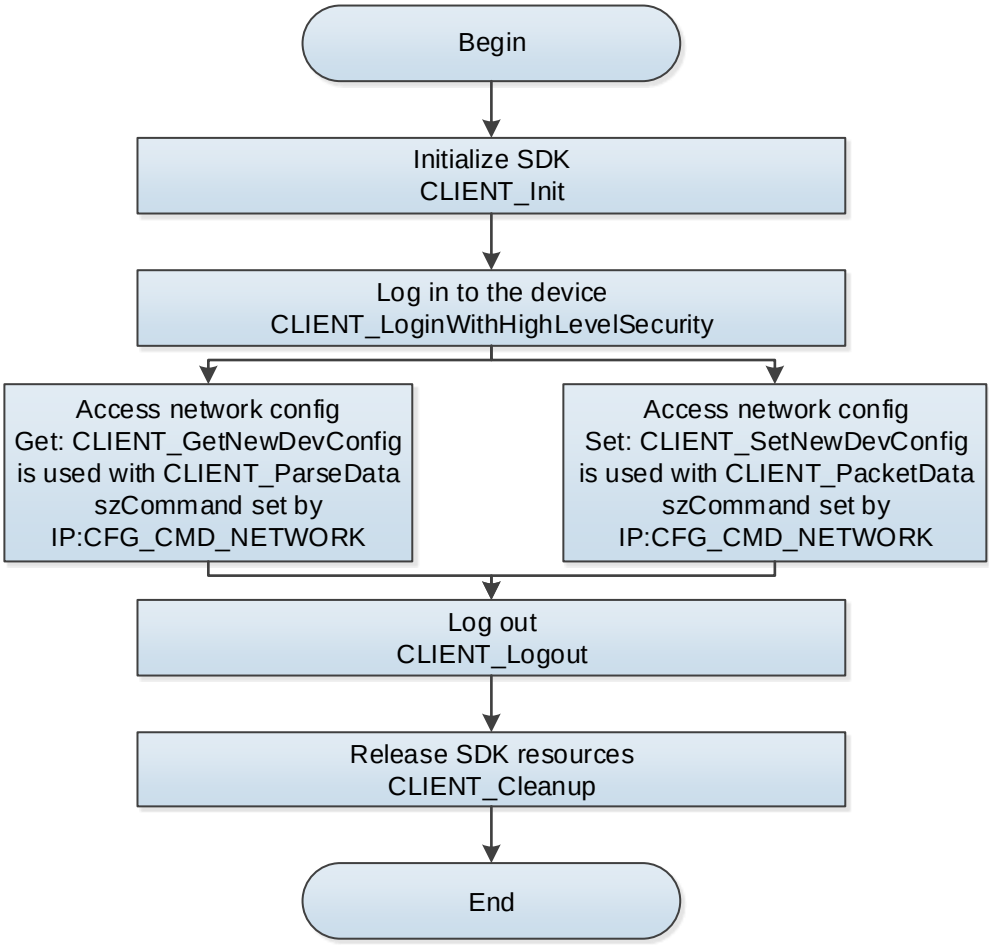
2.2.5.1.2 Interface Overview

Table 2-10 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

2.2.5.1.3 Process Description

Figure 2-9 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.

2.2.5.1.4 Example Code

```
// Configure the network
public void getSetNetWorkInfo() {
    String command = NetSDKLib.CFG_CMD_NETWORK;
    int channel = 0;
    CFG_NETWORK_INFO network = new CFG_NETWORK_INFO();
```

```
        // gET
        if (ToolKits.GetDevConfig(m_hLoginHandle, channel, command, network)) {
            System.out.println("Host name : " + new String(network.szHostName).trim());
            for (int i = 0; i < network.nInterfaceNum; i++) {
                System.out.println("Network interface name : " + new
String(network.stuInterfaces[i].szName).trim());
                System.out.println("IP Address : " + new String(network.stuInterfaces[i].szIP).trim());
                System.out.println("Subnet mask : " + new
String(network.stuInterfaces[i].szSubnetMask).trim());
                System.out.println("Defult gateway : " + new
String(network.stuInterfaces[i].szDefGateway).trim());
                System.out.println(
                        "DNS server address : " + new
String(network.stuInterfaces[i].szDnsServersArr[0].szDnsServers).trim()
                                + "\n" + new
String(network.stuInterfaces[i].szDnsServersArr[1].szDnsServers).trim());
                System.out.println("MAC address : " + new
String(network.stuInterfaces[i].szMacAddress).trim());
            }

            if (ToolKits.SetDevConfig(m_hLoginHandle, channel, command, network)) {
                System.out.println("Set NETWORK Succeed!");
            } else {
                System.err.println("Set NETWORK Failed!" + netsdk.CLIENT_GetLastError());
            }

        } else {
            System.err.println("Get NETWORK Failed!" + netsdk.CLIENT_GetLastError());
        }
}
```

2.2.5.2 Auto Register Config

2.2.5.2.1 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.

2.2.5.2.2 Interface Overview

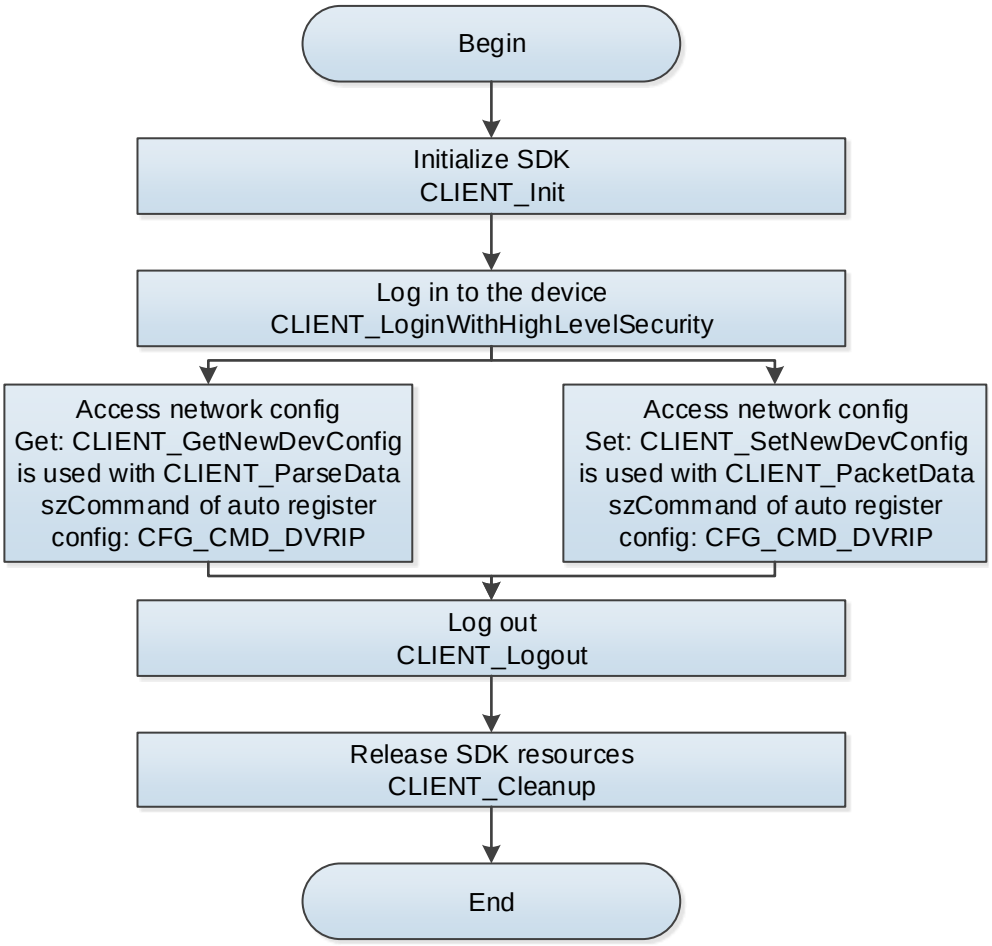
Table 2-11 Description of interfaces for setting auto register

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.

2.2.5.2.3 Process Description

Figure 2-10 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.

2.2.5.2.4 Example Code

```
// Configure the network protocol
public void getSetDVRIPInfo() {
    String command = NetSDKLib.CFG_CMD_DVRIP;
    int channel = 0;
    CFG_DVRIP_INFO dvrIp = new CFG_DVRIP_INFO();

    // Get
    if (ToolKits.GetDevConfig(m_hLoginHandle, channel, command, dvrIp)) {
        System.out.println("TCP service port : " + dvrIp.nTcpPort);
        System.out.println("SSL service port :" + dvrIp.nSSLPort);
        System.out.println("UDP service port :" + dvrIp.nUDPPort);
        System.out.println("Multicast port :" + dvrIp.nMCASTPort);

        // Configure the settings based on the obtained information
        if (ToolKits.SetDevConfig(m_hLoginHandle, channel, command, dvrIp)) {
            System.out.println("Set DVRIP Succeed!");
        } else {
            System.err.println("Set DVRIP Failed!" + netsdk.CLIENT_GetLastError());
        }
    } else {
        System.err.println("Get DVRIP Failed!" + netsdk.CLIENT_GetLastError());
    }
}
```

2.2.6 Device Time Setting

2.2.6.1 DeviceTime Setting

2.2.6.1.1 Introduction

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

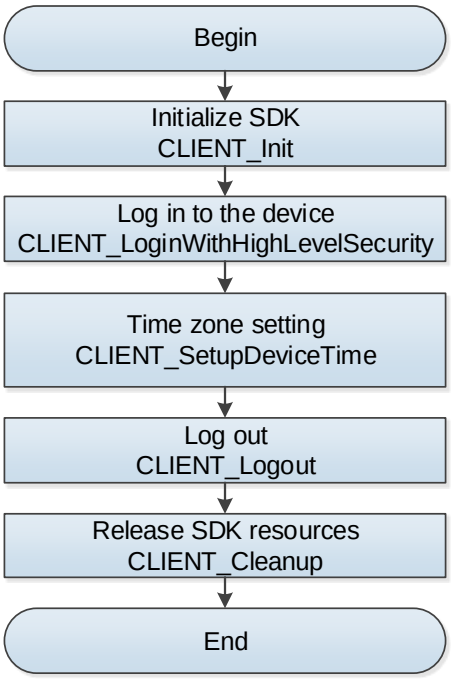
2.2.6.1.2 Interface Overview

Table 2-12 Description of time setting interfaces

Interface	Description
CLIENT_SetupDeviceTime	Set the current time of the device.

2.2.6.1.3 Process Description

Figure 2-11 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.

2.2.6.1.4 Example Code

```
/**
 * Synchronize the time
 */
public void setupDeviceTime() {
    SimpleDateFormat simpleDate = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    String date = simpleDate.format(new java.util.Date());

    String[] dateTime = date.split(" ");
    String[] mDate1 = dateTime[0].split("-");
    String[] mDate2 = dateTime[1].split(":");

    NetSDKLib.NET_TIME pDeviceTime = new NetSDKLib.NET_TIME();
    pDeviceTime.dwYear = Integer.parseInt(mDate1[0]);
    pDeviceTime.dwMonth = Integer.parseInt(mDate1[1]);
    pDeviceTime.dwDay = Integer.parseInt(mDate1[2]);
    pDeviceTime.dwHour = Integer.parseInt(mDate2[0]);
    pDeviceTime.dwMinute = Integer.parseInt(mDate2[1]);
    pDeviceTime.dwSecond = Integer.parseInt(mDate2[2]);
```

```
        if(netsdkApi.CLIENT_SetupDeviceTime(loginHandle, pDeviceTime)) {
            System.out.println("Successfully synchronized the time. ");
        } else {
            System.out.println("Failed to synchronize the time." + ToolKits.getErrorCode());
        }
    }
```

2.2.6.2 NTP Server and Time Zone Setting

2.2.6.2.1 Introduction

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

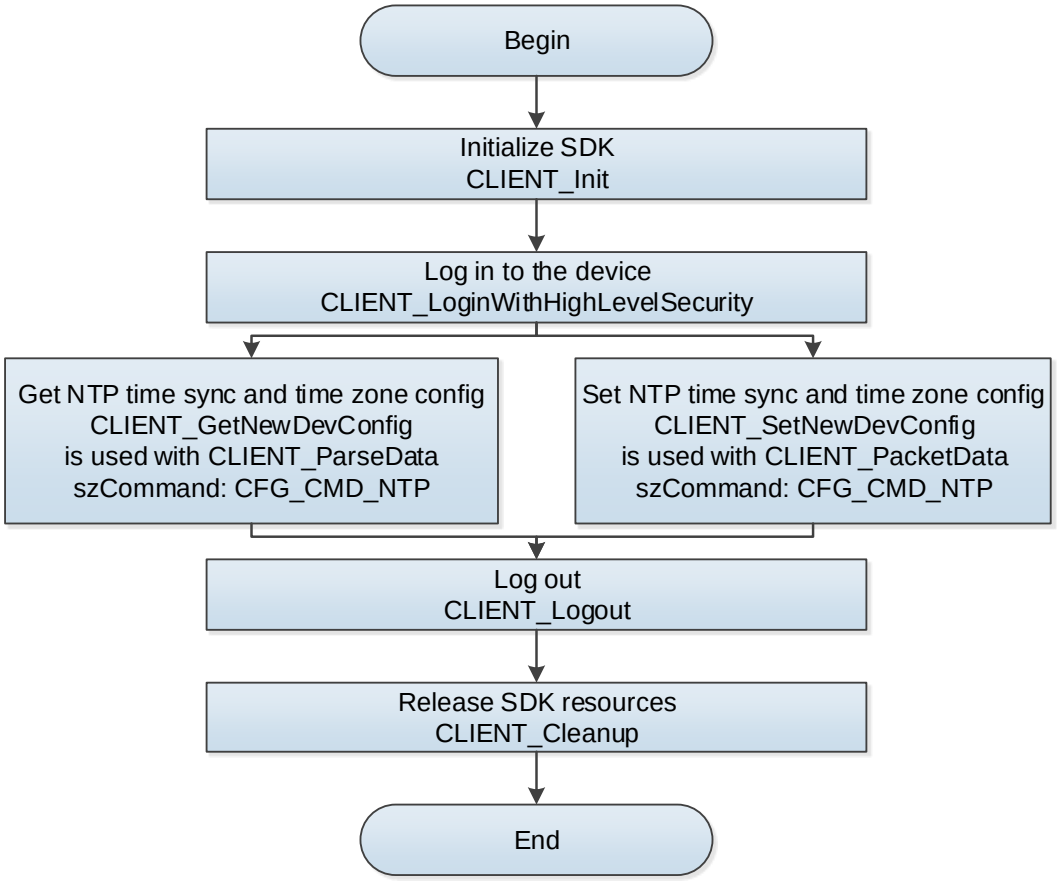
2.2.6.2.2 Interface Overview

Table 2-13 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.

2.2.6.2.3 Process Description

Figure 2-12 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.

2.2.6.2.4 Example Code

```
// Synchronize the time zone
public void SetNTP() {
    NetSDKLib.CFG_NTP_INFO ntpInfo = new NetSDKLib.CFG_NTP_INFO();
    if (ToolKits.GetDevConfig(loginHandle, -1, NetSDKLib.CFG_CMD_NTP, ntpInfo)) {
        System.out.println("bEnable : " + ntpInfo.bEnable);
        System.out.println("emTimeZoneType : " + ntpInfo.emTimeZoneType);
```

```
            System.out.println("szTimeZoneDesc : " + new String(ntpInfo.szTimeZoneDesc).trim());
        }

        // Configure
        ntpInfo.bEnable = 1;
        ntpInfo.emTimeZoneType = NetSDKLib.EM_CFG_TIME_ZONE_TYPE.EM_CFG_TIME_ZONE_0;
        if (ToolKits.SetDevConfig(loginHandle, -1, NetSDKLib.CFG_CMD_NTP, ntpInfo)) {
            System.out.println("SetNTP Succeed!");
        }
    }
```

2.2.7 Personnel Management

2.2.7.1 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).

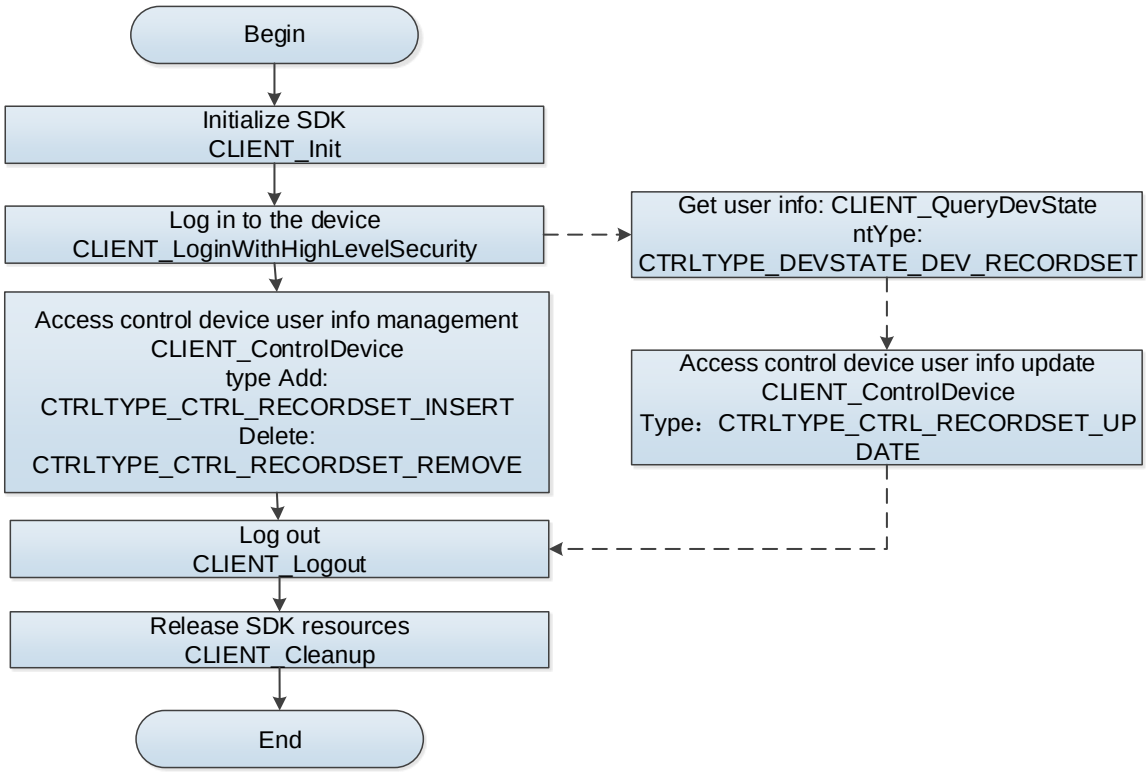
2.2.7.2 Interface Overview

Table 2-14 Description of personnel information interfaces

Interface	Description
CLIENT_ControlDevice	Control device.
CLIENT_QueryDevState	Query device status.

2.2.7.3 Process Description

Figure 2-13 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 2-15 Description and structure of type

Type	Description	emType	Param
● CTRLTYPE_CTRL_RECORDSET_INSERT ● CTRLTYPE_CTRL_RECORDSET_INSERTEX	Add user info	NET_RECORD_ACCESSCTLCARD	● NET_CTRL_RECORDSET_INSERT_PARAM ● NET_RECORDSET_ACCESS_CTL_CARD
CTRLTYPE_CTRL_RECORDSET_REMOVE	Delete user info	NET_RECORD_ACCESSCTLCARD	● NET_CTRL_RECORDSET_PARAM ● NET_RECORDSET_ACCESS_CTL_CARD
CTRLTYPE_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 2-16 Description and structure of type

Type	Description	emType	Param
NET_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 2-17 Description and structure of type

Type	Description	emType	Param
● CTRLTYPE_CTRL_RECORDSET_UPDATE ● CTRLTYPE_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 "2.2.9.1 Period Config."
- Unlock password: After setting this password, you can directly enter the password to open the door without swiping card.
- 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 "2.2.10.1 Unlock at Designated Intervals and First Card Unlock."

2.2.7.4 Example Code

```
/**
 * Insert the password
 */
public void insertPassword() {

    // A maximum of 500 unique password numbers are supported.
    final String userId = "1";

    // Unlock password
    final String openDoorPassword = "888887";

```

```
        NetSDKLib.NET_RECORDSET_ACCESS_CTL_PWD accessInsert = new
NetSDKLib.NET_RECORDSET_ACCESS_CTL_PWD();

        System.arraycopy(userId.getBytes(), 0, accessInsert.szUserID,
                0, userId.getBytes().length);
        System.arraycopy(openDoorPassword.getBytes(), 0, accessInsert.szDoorOpenPwd,
                0, openDoorPassword.getBytes().length);

        /// The following fields can be fixed. The values must be provided because of the current restrictions.
        accessInsert.nDoorNum = 2; // Number of doors. It indicates double door controller.
        accessInsert.sznDoors[0] = 0; // It indicates having the permission for the first door.
        accessInsert.sznDoors[1] = 1; // It indicates having the permission for the second door.
        accessInsert.nTimeSectionNum = 2; // It corresponds to the number of doors.
        accessInsert.nTimeSectionIndex[0] = 255; // It indicates that the setting of the first door is valid for
the whole day.
        accessInsert.nTimeSectionIndex[1] = 255; // It indicates that the setting of the second door is valid
for the whole day.

        NetSDKLib.NET_CTRL_RECORDSET_INSERT_PARAM insert = new
NetSDKLib.NET_CTRL_RECORDSET_INSERT_PARAM();
        insert.stuCtrlRecordSetInfo.emType =
NetSDKLib.EM_NET_RECORD_TYPE.NET_RECORD_ACCESSCTLPWD;       // Type of the recordset
information.
        insert.stuCtrlRecordSetInfo.pBuf = accessInsert.getPointer();

        accessInsert.write();
        insert.write();
        boolean success = netSdk.CLIENT_ControlDevice(loginHandle,
                NetSDKLib.CtrlType.CTRLTYPE_CTRL_RECORDSET_INSERT, insert.getPointer(),
5000);
        insert.read();
        accessInsert.read();

        if(!success) {
            System.err.println("insert password failed. 0x" +
Long.toHexString(netSdk.CLIENT_GetLastError()));
            return;
        }

        System.out.println("Password nRecNo : " + insert.stuCtrlRecordSetResult.nRecNo);
        passwordRecordNo = insert.stuCtrlRecordSetResult.nRecNo;
}

/**
 * Update the password
 */
public void updatePassword() {
```

```
        NetSDKLib.NET_RECORDSET_ACCESS_CTL_PWD accessUpdate = new
NetSDKLib.NET_RECORDSET_ACCESS_CTL_PWD();
        accessUpdate.nRecNo = passwordRecordNo; // The recordset number to be modified, obtained by
insertion.

        /// Password number. This parameter is required. Otherwise, password update does not take effect.
        final String userId = String.valueOf(accessUpdate.nRecNo);
        System.arraycopy(userId.getBytes(), 0, accessUpdate.szUserID,
                0, userId.getBytes().length);

        // New unlock password
        final String newPassord = "333333";
        System.arraycopy(newPassord.getBytes(), 0,
                accessUpdate.szDoorOpenPwd, 0, newPassord.getBytes().length);

        /// The following fields can be fixed. The values must be provided because of the current restrictions.
        accessUpdate.nDoorNum = 2; // Number of doors. It indicates double door controller.
        accessUpdate.sznDoors[0] = 0; // It indicates having the permission for the first door.
        accessUpdate.sznDoors[1] = 1; // It indicates having the permission for the second door.
        accessUpdate.nTimeSectionNum = 2; // It corresponds to the number of doors.
        accessUpdate.nTimeSectionIndex[0] = 255; // It indicates that the setting of the first door is valid for
the whole day.
        accessUpdate.nTimeSectionIndex[1] = 255; // It indicates that the setting of the second door is valid
for the whole day.

        NetSDKLib.NET_CTRL_RECORDSET_PARAM update = new
NetSDKLib.NET_CTRL_RECORDSET_PARAM();
        update.emType = NetSDKLib.EM_NET_RECORD_TYPE.NET_RECORD_ACCESSCTLPWD;
// Type of the recordset information.
        update.pBuf = accessUpdate.getPointer();

        accessUpdate.write();
        update.write();
        boolean result = netSdk.CLIENT_ControlDevice(loginHandle,
                NetSDKLib.CtrlType.CTRLTYPE_CTRL_RECORDSET_UPDATE, update.getPointer(),
5000);
        update.read();
        accessUpdate.read();
        if (!result) {
            System.err.println("update password failed. 0x" +
Long.toHexString(netSdk.CLIENT_GetLastError()));
        }else {
            System.out.println("update password success");
        }
}
```

```
/**
 * Delete all
 */
public void alldeleteOperate() {
	int type = NetSDKLib.CtrlType.CTRLTYPE_CTRL_RECORDSET_CLEAR;
	NetSDKLib.NET_CTRL_RECORDSET_PARAM param = new
NetSDKLib.NET_CTRL_RECORDSET_PARAM();
	if (flg) {
		param.emType = NetSDKLib.EM_NET_RECORD_TYPE.NET_RECORD_TRAFFICREDLIST;
	} else {
		param.emType =
NetSDKLib.EM_NET_RECORD_TYPE.NET_RECORD_TRAFFICBLACKLIST;
	}
	param.write();
	boolean zRet = netsdk.CLIENT_ControlDevice(m_hLoginHandle, type, param.getPointer(), 5000);
	if (zRet) {
		System.out.println("Successfully deleted all. ");
	} else {
		System.err.println("Failed to delete all " + String.format("0x%x",
netsdk.CLIENT_GetLastError()));
	}
}
	public static boolean queryRecordState(SdkStructure condition) {

		int emType = getRecordType(condition);
		if (emType == EM_NET_RECORD_TYPE.NET_RECORD_UNKNOWN) {
			System.err.println("the input query condition SdkStructure [" + condition.getClass() + "]
invalid!");
			return false;
		}

		NetSDKLib.NET_CTRL_RECORDSET_PARAM record = new
NetSDKLib.NET_CTRL_RECORDSET_PARAM();
		record.emType = emType;
		record.pBuf = condition.getPointer();

		IntByReference intRetLen = new IntByReference();

		condition.write();
		record.write();
		if (!netsdkApi.CLIENT_QueryDevState(loginHandle,
NetSDKLib.NET_DEVSTATE_DEV_RECORDSET,
				record.getPointer(), record.size(), intRetLen, 3000)) {
			return false;
		}
```

```
        record.read();
        condition.read();

        output(condition);

        return true;
    }

    private static void output(SdkStructure record) { // The specific output is defined at the upper level.

        if (record instanceof NET_RECORDSET_ACCESS_CTL_CARDREC) {

            printRecord((NET_RECORDSET_ACCESS_CTL_CARDREC)record);

        }else if(record instanceof NET_RECORDSET_ACCESS_CTL_CARD) {

            printRecord((NET_RECORDSET_ACCESS_CTL_CARD)record);

        }
    }

    public static int getRecordNo() {
        return nRecordNo;
    }

    private static int getRecordType(SdkStructure object) {
        int type = EM_NET_RECORD_TYPE.NET_RECORD_UNKNOWN;

        if (object instanceof NET_RECORDSET_ACCESS_CTL_CARD
                || object instanceof FIND_RECORD_ACCESSCTLCARD_CONDITION) {

            type = EM_NET_RECORD_TYPE.NET_RECORD_ACCESSCTLCARD;

        }else if (object instanceof NET_RECORD_ACCESSQRCODE_INFO) {

            type = EM_NET_RECORD_TYPE.NET_RECORD_ACCESSQRCODE;

        }else if (object instanceof NET_RECORDSET_ACCESS_CTL_PWD) {

            type = EM_NET_RECORD_TYPE.NET_RECORD_ACCESSCTLPWD;

        }else if (object instanceof NET_RECORDSET_ACCESS_CTL_CARDREC
                || object instanceof FIND_RECORD_ACCESSCTLCARDREC_CONDITION_EX) {

            type = EM_NET_RECORD_TYPE.NET_RECORD_ACCESSCTLCARDREC_EX;

        }
```

```
            return type;
        }
}
```

2.2.8 Door Config

2.2.8.1 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.

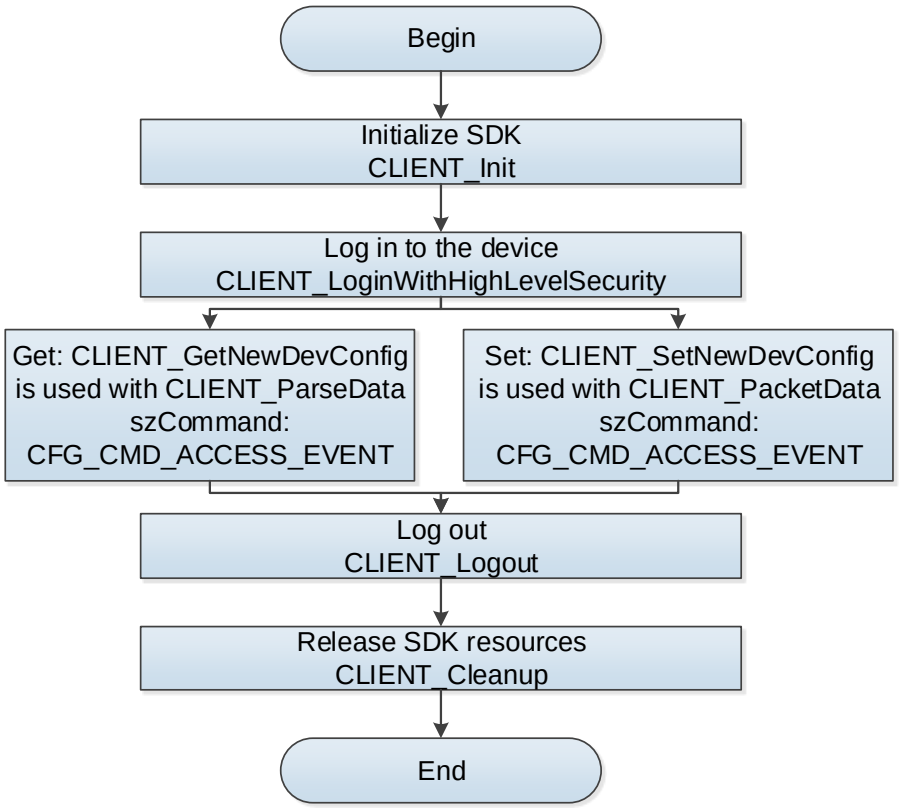
2.2.8.2 Interunlockface Overview

Table 2-18 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.

2.2.8.3 Process Description

Figure 2-14 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 2-19 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 2-20 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 "2.2.9.1 Period Config."

2.2.8.4 Example Code

```
// Configure the access control events
public void AccessConfig() {
    // Get
    String szCommand = NetSDKLib.CFG_CMD_ACCESS_EVENT;
    int nChn = 0;  // Channel
    CFG_ACCESS_EVENT_INFO access = new CFG_ACCESS_EVENT_INFO();  //
m_stDeviceInfo.byChanNum It is the number of device channels.

    if (ToolKits.GetDevConfig(loginHandle, nChn, szCommand, access)) {
        System.out.println("Access control channel name:" + new
String(access.szChannelName).trim());
        System.out.println("Enable the first card:" + access.stuFirstEnterInfo.bEnable);    // 0-false;
1-true
        System.out.println("Access control status after the first card permission verification passes:" +
access.stuFirstEnterInfo.emStatus); // Status reference enumeration
CFG_ACCESS_FIRSTENTER_STATUS
        System.out.println("The time period for which the first card verification is required. The value is
the channel number:" + access.stuFirstEnterInfo.nTimeIndex);
    }

    // Configure
    boolean bRet = ToolKits.SetDevConfig(loginHandle, nChn, szCommand, access);
    if (bRet) {
        System.out.println("Set Succeed!");
```

```
    }
}
```

2.2.9 Door Time Config

2.2.9.1 Period Config

2.2.9.1.1 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.

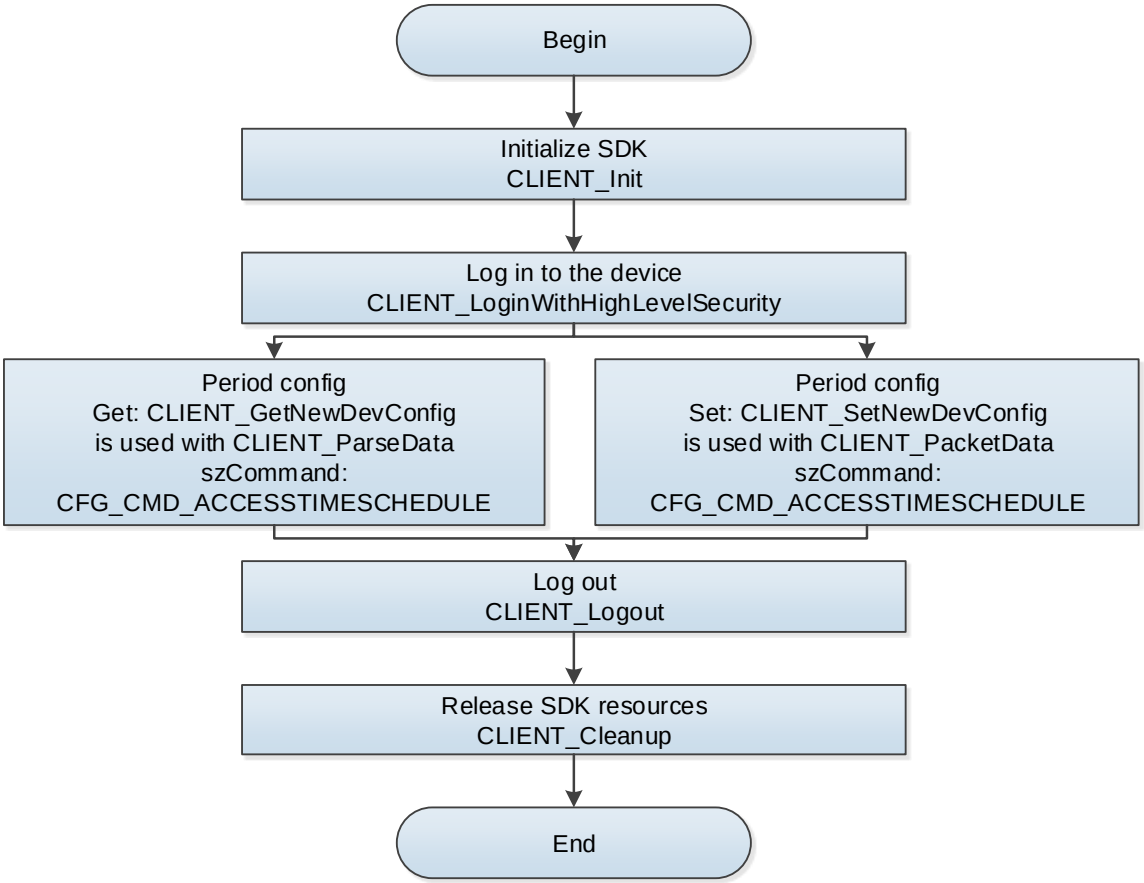
2.2.9.1.2 Interface Overview

Table 2-21 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.

2.2.9.1.3 Process Description

Figure 2-15 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.

2.2.9.1.4 Example Code

```
/**
 * Configure the card swiping period of the access control device
 */
public void setAccessTimeSchedule() {
    CFG_ACCESS_TIMESCHEDULE_INFO msg = new CFG_ACCESS_TIMESCHEDULE_INFO();

```

```
        String strCmd = NetSDKLib.CFG_CMD_ACCESSTIMESCHEDULE;
        int nChannel = 120; // Channel number

        // Get
        if(ToolKits.GetDevConfig(loginHandle, nChannel, strCmd, msg)) {
            System.out.println("Enable:" + msg.bEnable);
            try {
                System.out.println("Custom name:" + new String(msg.szName, "GBK").trim());
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }

            for(int i = 0; i < 7; i++) {
                for(int j = 0; j < 4; j++) {
                    System.out.println("dwRecordMask:" +
msg.stuTimeWeekDay[i].stuTimeSection[j].dwRecordMask);
                    System.out.println(msg.stuTimeWeekDay[i].stuTimeSection[j].startTime() + "-" +
                                        msg.stuTimeWeekDay[i].stuTimeSection[j].endTime() + "\n");
                }
            }

            // Configure
            if(ToolKits.SetDevConfig(loginHandle, nChannel, strCmd, msg)) {
                System.out.println("Set AccessTimeSchedule Succeed!");
            } else {
                System.err.println("Set AccessTimeSchedule Failed!" + ToolKits.getErrorCode());
            }
        } else {
            System.err.println("Get AccessTimeSchedule Failed!" + ToolKits.getErrorCode());
        }
}
```

2.2.9.2 Always Open and Always Closed Period Config

2.2.9.2.1 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.

2.2.9.2.2 Interface Overview

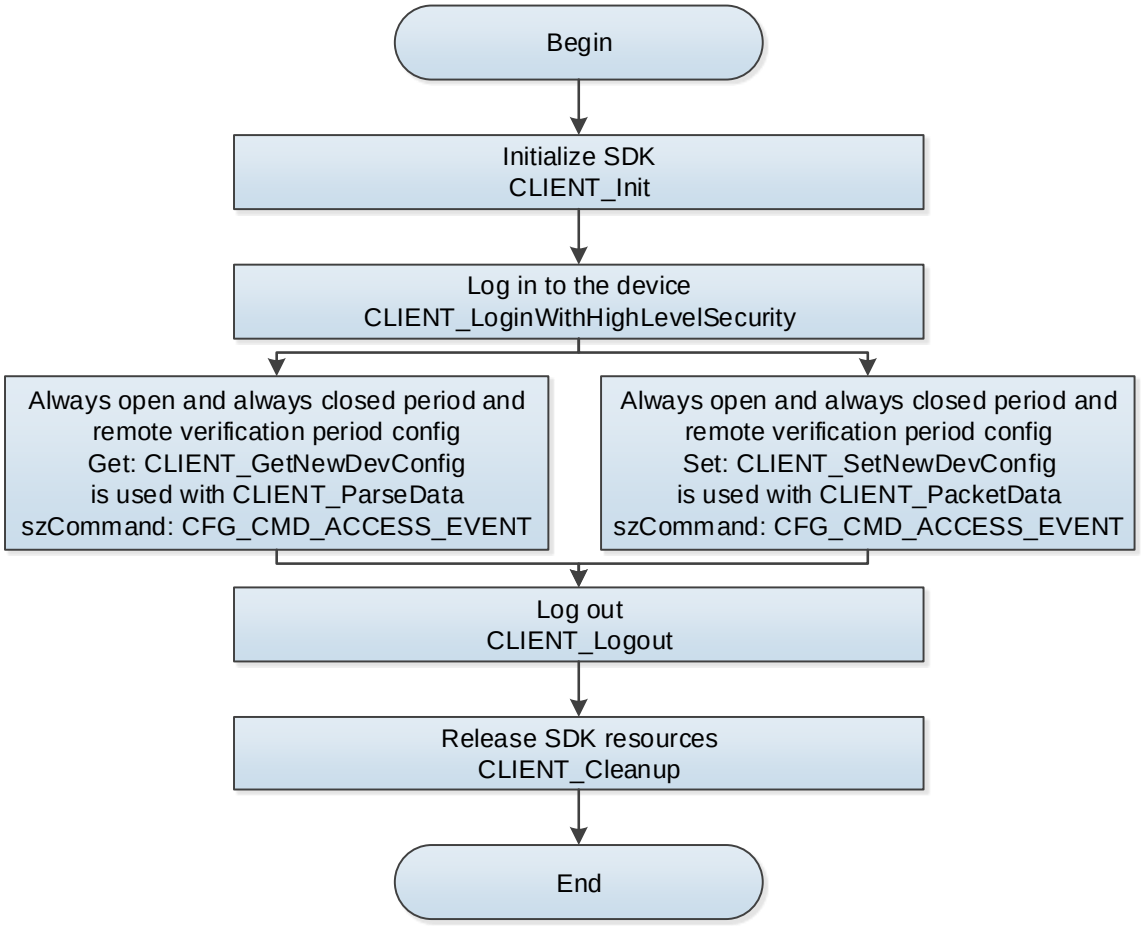
Table 2-22 Description of always open and always closed period config interfaces

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.

2.2.9.2.3 Process Description

Figure 2-16 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 2-23 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 2-24 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 "2.2.9.1 Period Config."

2.2.9.2.4 Example Code

```
/**
 * Configure the card swiping period of the access control device
 */
public void setAccessTimeSchedule() {
    CFG_ACCESS_TIMESCHEDULE_INFO msg = new CFG_ACCESS_TIMESCHEDULE_INFO();

    String strCmd = NetSDKLib.CFG_CMD_ACCESSTIMESCHEDULE;
    int nChannel = 120; // Channel number

    // Get
    if(ToolKits.GetDevConfig(loginHandle, nChannel, strCmd, msg)) {
        System.out.println("Enable:" + msg.bEnable);
        try {
            System.out.println("Custom name:" + new String(msg.szName, "GBK").trim());
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }

        for(int i = 0; i < 7; i++) {
            for(int j = 0; j < 4; j++) {
                System.out.println("dwRecordMask:" +
msg.stuTimeWeekDay[i].stuTimeSection[j].dwRecordMask);
                System.out.println(msg.stuTimeWeekDay[i].stuTimeSection[j].startTime() + "-" +
                                   msg.stuTimeWeekDay[i].stuTimeSection[j].endTime() + "\n");
            }
        }

        // Configure
        if(ToolKits.SetDevConfig(loginHandle, nChannel, strCmd, msg)) {
            System.out.println("Set AccessTimeSchedule Succeed!");
        } else {
            System.err.println("Set AccessTimeSchedule Failed!" + ToolKits.getErrorCode());
        }
```

```
    } else {
        System.err.println("Get AccessTimeSchedule Failed!" + ToolKits.getErrorCode());
    }
}
```

2.2.10 Advanced Config of Door

2.2.10.1 Unlock at Designated Intervals and First Card Unlock

2.2.10.1.1 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.

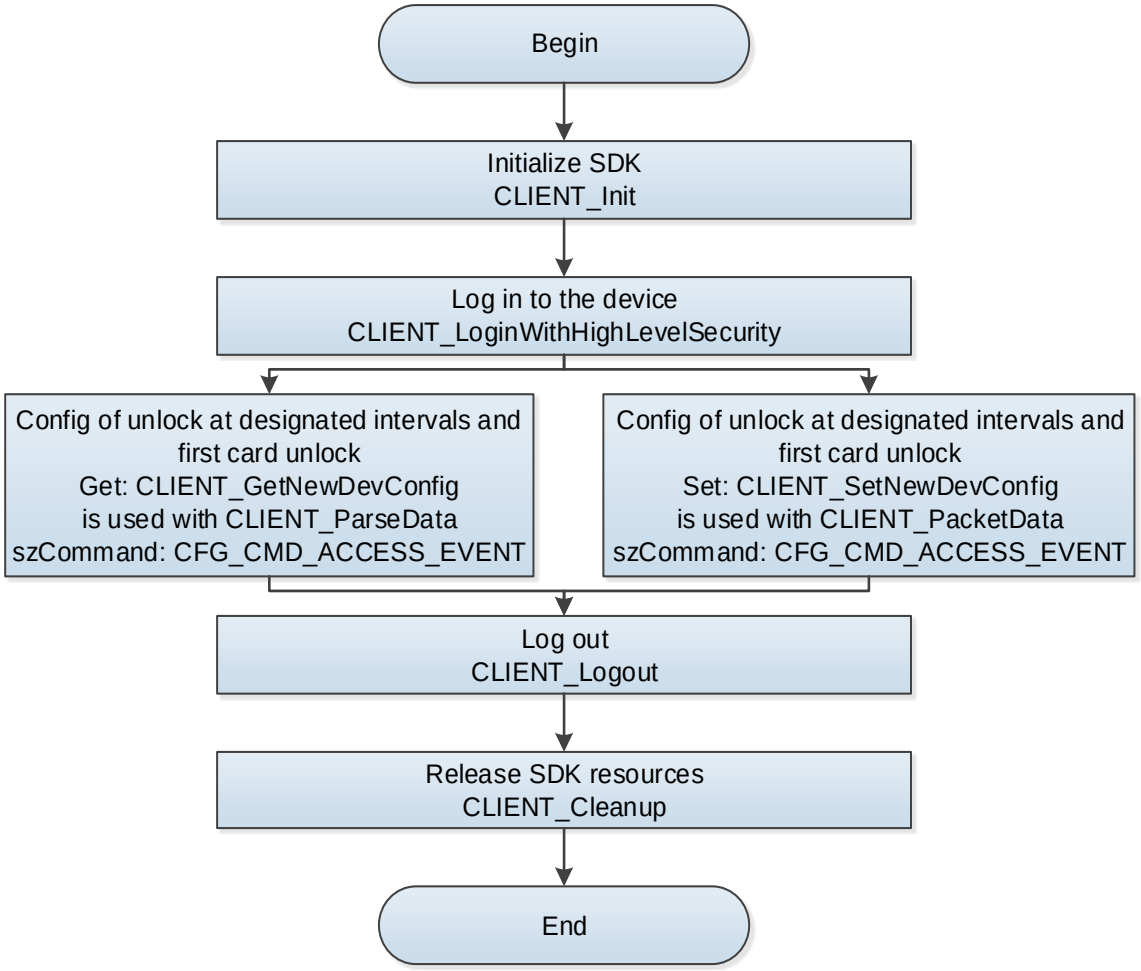
2.2.10.1.2 Interface Overview

Table 2-25 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.
CLIENT_SetNewDevConfig	Set config information.
CLIENT_PacketData	Pack the config information to be set into the string format.

2.2.10.1.3 Process Description

Figure 2-17 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 2-26 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 2-27 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.

2.2.10.1.4 Example Code

```
// Configure the access control events
public void AccessConfig() {
	// Get
	String szCommand = NetSDKLib.CFG_CMD_ACCESS_EVENT;
	int nChn = 0; // Channel
	CFG_ACCESS_EVENT_INFO access = new CFG_ACCESS_EVENT_INFO(); //
m_stDeviceInfo.byChanNum It is the number of device channels.

	if (ToolKits.GetDevConfig(loginHandle, nChn, szCommand, access)) {
		System.out.println("Access control channel name:"
				+ new String(access.szChannelName).trim());
		System.out.println("Enable the first card:" + access.stuFirstEnterInfo.bEnable); // 0-false;
                                                                                    // 1-true
		System.out.println("Access control status after the first card permission verification passes:"
				+ access.stuFirstEnterInfo.emStatus); // Status reference enumeration
                                                      //
CFG_ACCESS_FIRSTENTER_STATUS
		System.out.println("The time period for which the first card verification is required. The value is
the channel number:"
				+ access.stuFirstEnterInfo.nTimeIndex);

		System.out.println(" The acquisition status of the current door:" + access.emReadCardState);
	}

	// Configure
//		access.emReadCardState = EM_CFG_CARD_STATE.EM_CFG_CARD_STATE_SWIPE;	//
Swipe card on the access control device
	access.emReadCardState = EM_CFG_CARD_STATE.EM_CFG_CARD_STATE_COLLECTION;
	 // Collect the card information by the access control device
//		access.emReadCardState = EM_CFG_CARD_STATE.EM_CFG_CARD_STATE_UNKNOWN;
	// Exit card reading

	boolean bRet = ToolKits.SetDevConfig(loginHandle, nChn, szCommand,
			access);
	if (bRet) {
		System.out.println("Set Succeed!");
```

```
    }
}
```

2.2.10.2 Combination Unlock by Multiple Persons

2.2.10.2.1 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.

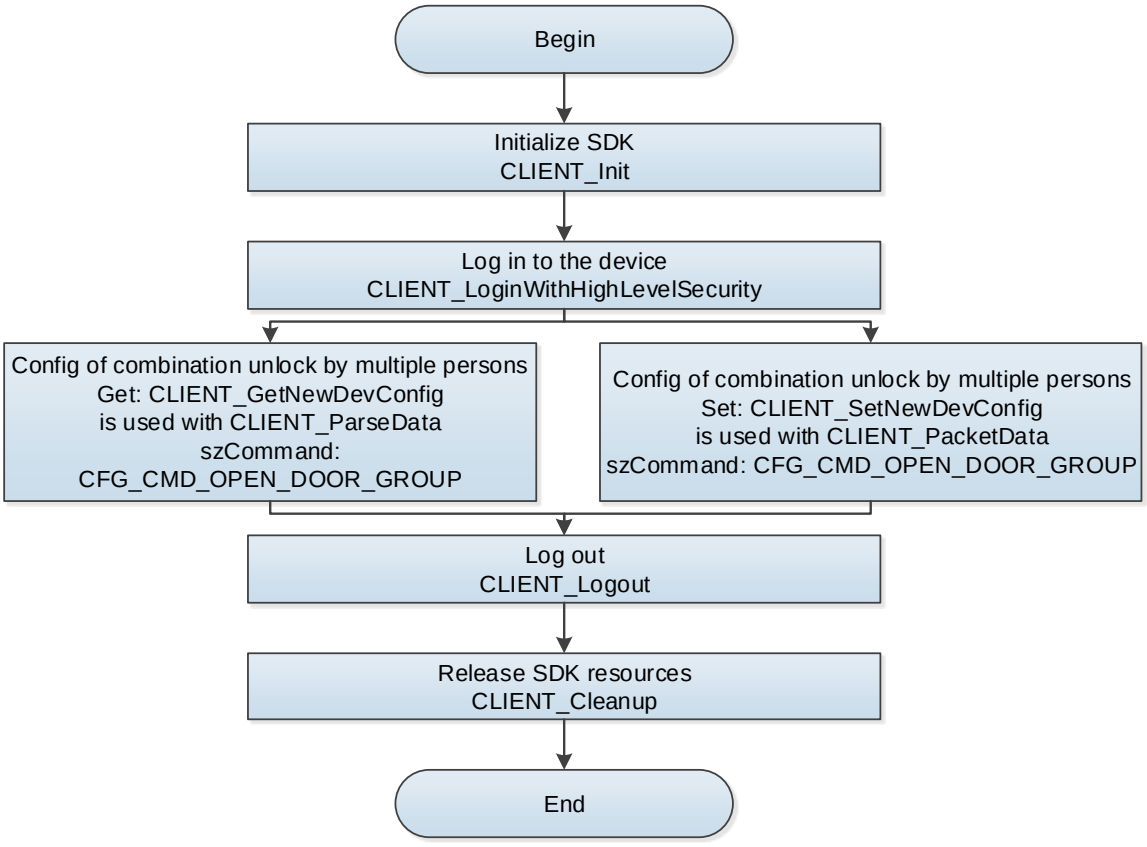
2.2.10.2.2 Interface Overview

Table 2-28 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.

2.2.10.2.3 Process Description

Figure 2-18 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.

2.2.10.2.4 Example Code

```
/**
 * Configure combination unlock by multiple persons
 * @param nChannel      Channel number
 * @param msg           Configure combination unlock by multiple persons
 */
public boolean MoreOpenDoor(int nChannel,CFG_OPEN_DOOR_GROUP_INFO msg){
    // Get
    String szCommand = NetSDKLib.CFG_CMD_OPEN_DOOR_GROUP;
    if (!ToolKits.GetDevConfig(loginHandle, nChannel, szCommand, msg)) {
            System.err.println("Get more open door Failed.");
            return false;
    }
    System.out.println("Number of valid combinations" + msg.nGroup + " Number of users " +
msg.stuGroupInfo.length );

    //Configure
    msg.stuGroupInfo[0].stuGroupDetail[0].emMethod=1;
    msg.stuGroupInfo[0].stuGroupDetail[1].emMethod=1;
    System.arraycopy("123".getBytes(), 0, msg.stuGroupInfo[0].stuGroupDetail[0].szUserID, 0,
```

```
"123".getBytes().length);
    System.arraycopy("234".getBytes(), 0, msg.stuGroupInfo[0].stuGroupDetail[1].szUserID, 0,
"234".getBytes().length);
    for(int i=0;i<msg.stuGroupInfo.length;i++){
        msg.stuGroupInfo[i].nUserCount=2;
        msg.stuGroupInfo[i].nGroupNum=2;
        msg.stuGroupInfo[i].bGroupDetailEx=true;
    }
    if(!ToolKits.SetDevConfig(loginHandle, nChannel, szCommand, msg)){
        System.err.println("Set more open door Failed!");
        return false;
    }
    System.out.println("Set more open door Succeed!");
    return true;
}
```

2.2.10.3 Inter-door Lock

2.2.10.3.1 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.

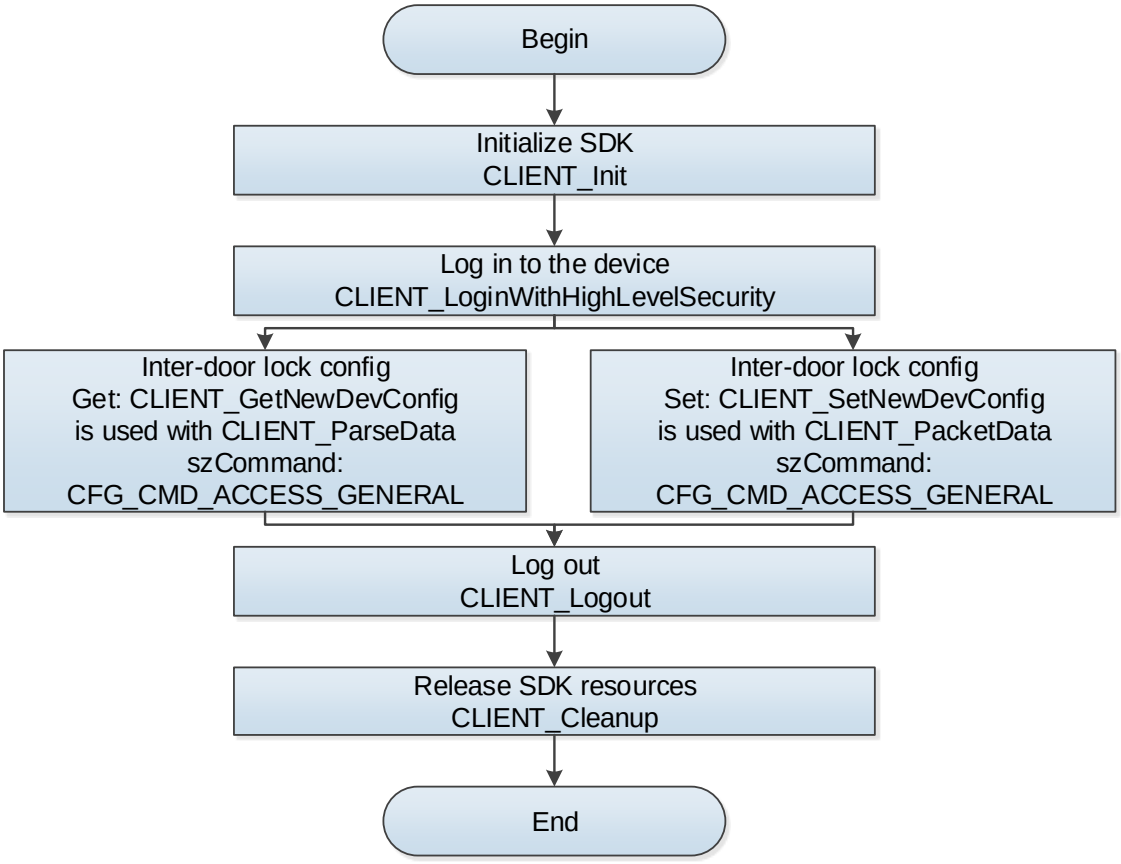
2.2.10.3.2 Interface Overview

Table 2-29 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.

2.2.10.3.3 Process Description

Figure 2-19 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.

2.2.10.3.4 Example Code

```
/** Get and send the inter-lock configuration */
  public void ABLockInfo() {
    // To send the configuration, obtain and modify the configuration items to avoid missing or
```

```
mistakenly modifying other irrelevant configuration items.
    CFG_ACCESS_GENERAL_INFO inParam = new CFG_ACCESS_GENERAL_INFO();
    inParam =
        (CFG_ACCESS_GENERAL_INFO)
            configModule.getNewConfig(loginHandler, CFG_CMD_ACCESS_GENERAL, inParam, 0,
3000);
    System.out.println("Before configuration, enable inter-lock:" + inParam.stuABLockInfo.bEnable + ",
configure to:");
    for (int i = 0; i < inParam.stuABLockInfo.nDoors; i++) {
      for (int j = 0; j < inParam.stuABLockInfo.stuDoors[i].nDoor; j++) {
        // Inter-lock information
        System.out.println(i + "," + j + ",door:" + inParam.stuABLockInfo.stuDoors[i].anDoor[j]);
      }
    }
    // Modify inter-lock configuration
    inParam.abABLockInfo = 1;
    inParam.stuABLockInfo.bEnable = !inParam.stuABLockInfo.bEnable;
    inParam.stuABLockInfo.nDoors = 1;
    int doors = 2;
    inParam.stuABLockInfo.stuDoors[0].nDoor = doors;
    inParam.stuABLockInfo.stuDoors[0].anDoor[0] = 1;
    inParam.stuABLockInfo.stuDoors[0].anDoor[1] = 2;

    inParam.stuABLockInfo.stuDoors[1].nDoor = doors;
    inParam.stuABLockInfo.stuDoors[1].anDoor[0] = 0;
    inParam.stuABLockInfo.stuDoors[1].anDoor[1] = 3;
    boolean result =
        configModule.setNewConfig(loginHandler, CFG_CMD_ACCESS_GENERAL, inParam, 0,
3000);
    // If the configuration was sent successfully, get the configuration again and print
    if (result) {
      inParam =
          (CFG_ACCESS_GENERAL_INFO)
              configModule.getNewConfig(loginHandler, CFG_CMD_ACCESS_GENERAL, inParam,
3, 3000);
      System.out.println("After the modification, enable inter-lock:" + inParam.stuABLockInfo.bEnable +
",configure to:");
    }
  }
```

2.2.10.4 Unlock Password

2.2.10.4.1 Introduction

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

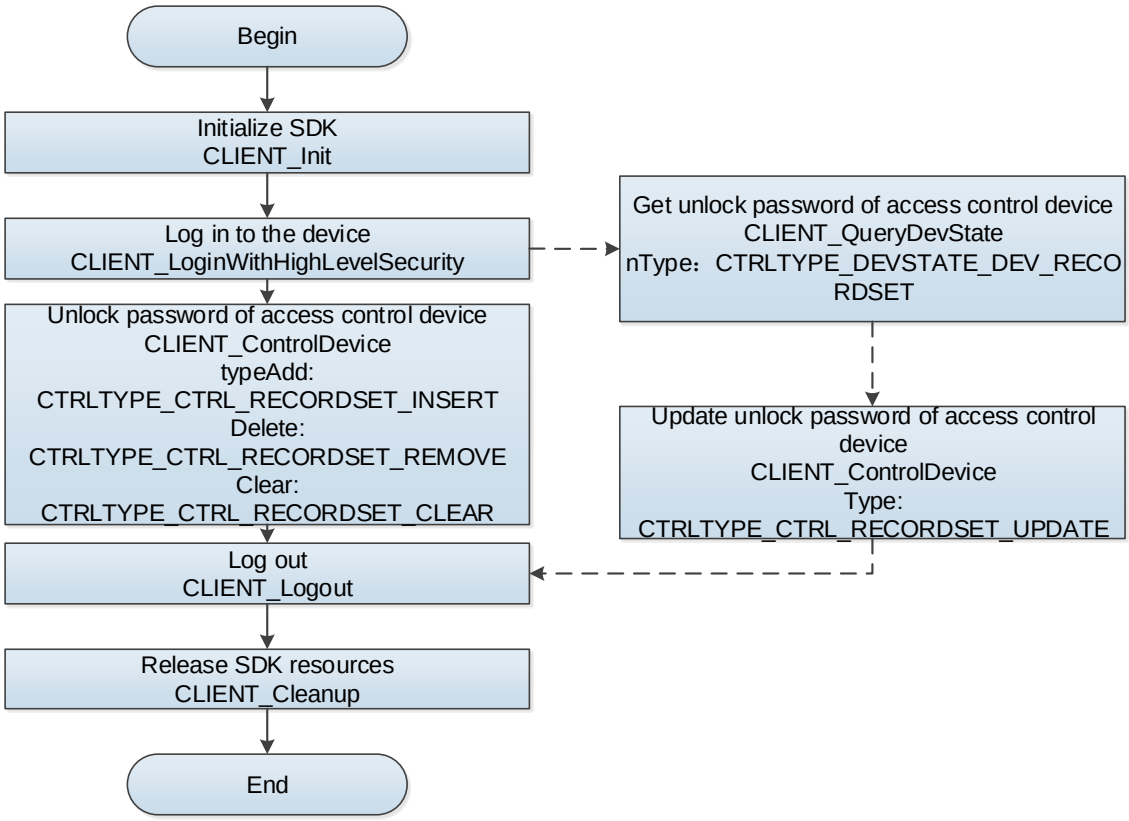
2.2.10.4.2 Interface Overview

Table 2-30 Description of unlock password interface

Interface	Description
CLIENT_ControlDevice	Device control.

2.2.10.4.3 Process Description

Figure 2-20 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 2-31 Description and structure of type

Type	Description	emType	Param
CTRLTYPE_CTRL_RECORDSET_INSERT	Add unlock password	NET_RECORD_ACCESSCTLPWD	NET_CTRL_RECORDSET_INSERT_PARA NET_RECORDSET_ACCESS_CTL_PWD
CTRLTYPE_CTRL_RECORDSET_REMOVE	Delete unlock password	NET_RECORD_ACCESSCTLPWD	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_ACCESS_CTL_PWD
CTRLTYPE_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 2-32 Description and structure of type

Type	Description	emType	Param
NET_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 2-33 Description and structure of type

Type	Description	emType	Param
CTRLTYPE_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.

2.2.10.4.4 Example Code

```
/**
 * The number returned after the password was successfully inserted is used for subsequent operation s
such as update and deletion.
 */
private int passwordRecordNo = 0;

/**
 * Insert the password
 */
public void insertPassword() {

    // A maximum of 500 unique password numbers are supported.
    final String userId = "1";

    // Unlock password
    final String openDoorPassword = "888887";

    NetSDKLib.NET_RECORDSET_ACCESS_CTL_PWD accessInsert = new
NetSDKLib.NET_RECORDSET_ACCESS_CTL_PWD();

    System.arraycopy(userId.getBytes(), 0, accessInsert.szUserID,
            0, userId.getBytes().length);
    System.arraycopy(openDoorPassword.getBytes(), 0, accessInsert.szDoorOpenPwd,
            0, openDoorPassword.getBytes().length);
```

```
        /// The following fields can be fixed. The values must be provided because of the current restrictions.
        accessInsert.nDoorNum = 2; // Number of doors. It indicates double door controller.
        accessInsert.sznDoors[0] = 0; // It indicates having the permission for the first door.
        accessInsert.sznDoors[1] = 1; // It indicates having the permission for the second door.
        accessInsert.nTimeSectionNum = 2; // It corresponds to the number of doors.
        accessInsert.nTimeSectionIndex[0] = 255; // It indicates that the setting of the first door is valid for
the whole day.
        accessInsert.nTimeSectionIndex[1] = 255; // It indicates that the setting of the second door is valid
for the whole day.

        NetSDKLib.NET_CTRL_RECORDSET_INSERT_PARAM insert = new
NetSDKLib.NET_CTRL_RECORDSET_INSERT_PARAM();
        insert.stuCtrlRecordSetInfo.emType =
NetSDKLib.EM_NET_RECORD_TYPE.NET_RECORD_ACCESSCTLPWD;       // Type of the recordset
information.
        insert.stuCtrlRecordSetInfo.pBuf = accessInsert.getPointer();

        accessInsert.write();
        insert.write();
        boolean success = netSdk.CLIENT_ControlDevice(loginHandle,
                    NetSDKLib.CtrlType.CTRLTYPE_CTRL_RECORDSET_INSERT, insert.getPointer(),
5000);
        insert.read();
        accessInsert.read();

        if(!success) {
            System.err.println("insert password failed. 0x" +
Long.toHexString(netSdk.CLIENT_GetLastError()));
            return;
        }

        System.out.println("Password nRecNo : " + insert.stuCtrlRecordSetResult.nRecNo);
        passwordRecordNo = insert.stuCtrlRecordSetResult.nRecNo;
}

/**
 * Update the password
 */
public void updatePassword() {

        NetSDKLib.NET_RECORDSET_ACCESS_CTL_PWD accessUpdate = new
NetSDKLib.NET_RECORDSET_ACCESS_CTL_PWD();
        accessUpdate.nRecNo = passwordRecordNo; // The recordset number to be modified, obtained by
insertion

        /// Password number. This parameter is required. Otherwise, password update does not take effect.
```

```
        final String userId = String.valueOf(accessUpdate.nRecNo);
        System.arraycopy(userId.getBytes(), 0, accessUpdate.szUserID,
                0, userId.getBytes().length);

        // New unlock password
        final String newPassord = "333333";
        System.arraycopy(newPassord.getBytes(), 0,
                accessUpdate.szDoorOpenPwd, 0, newPassord.getBytes().length);

        /// The following fields can be fixed. The values must be provided because of the current restrictions.
        accessUpdate.nDoorNum = 2; // Number of doors. It indicates double door controller.
        accessUpdate.sznDoors[0] = 0; // It indicates having the permission for the first door.
        accessUpdate.sznDoors[1] = 1; // It indicates having the permission for the second door.
        accessUpdate.nTimeSectionNum = 2; // It corresponds to the number of doors.
        accessUpdate.nTimeSectionIndex[0] = 255; // It indicates that the setting of the first door is valid for
the whole day.
        accessUpdate.nTimeSectionIndex[1] = 255; // It indicates that the setting of the second door is valid
for the whole day.

        NetSDKLib.NET_CTRL_RECORDSET_PARAM update = new
NetSDKLib.NET_CTRL_RECORDSET_PARAM();
        update.emType = NetSDKLib.EM_NET_RECORD_TYPE.NET_RECORD_ACCESSCTLPWD;
// Type of the recordset information.
        update.pBuf = accessUpdate.getPointer();

        accessUpdate.write();
        update.write();
        boolean result = netSdk.CLIENT_ControlDevice(loginHandle,
                NetSDKLib.CtrlType.CTRLTYPE_CTRL_RECORDSET_UPDATE, update.getPointer(),
5000);
        update.read();
        accessUpdate.read();
        if (!result) {
            System.err.println("update password failed. 0x" +
Long.toHexString(netSdk.CLIENT_GetLastError()));
        }else {
            System.out.println("update password success");
        }
}


/**
 * Delete the password
 */
public void deletePassword() {
        NetSDKLib.NET_CTRL_RECORDSET_PARAM remove = new
NetSDKLib.NET_CTRL_RECORDSET_PARAM();
```

```
        remove.emType = NetSDKLib.EM_NET_RECORD_TYPE.NET_RECORD_ACCESSCTLPWD;
        remove.pBuf = new IntByReference(passwordRecordNo).getPointer();

        remove.write();
        boolean result = netSdk.CLIENT_ControlDevice(loginHandle,
                NetSDKLib.CtrlType.CTRLTYPE_CTRL_RECORDSET_REMOVE, remove.getPointer(),
5000);
        remove.read();

        if(!result){
            System.err.println(" remove pawssword failed. 0x" +
Long.toHexString(netSdk.CLIENT_GetLastError()));
        }else {
            System.out.println("remove password success");
        }
}
```

2.2.11 Records Query

2.2.11.1 Unlock Records

2.2.11.1.1 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.

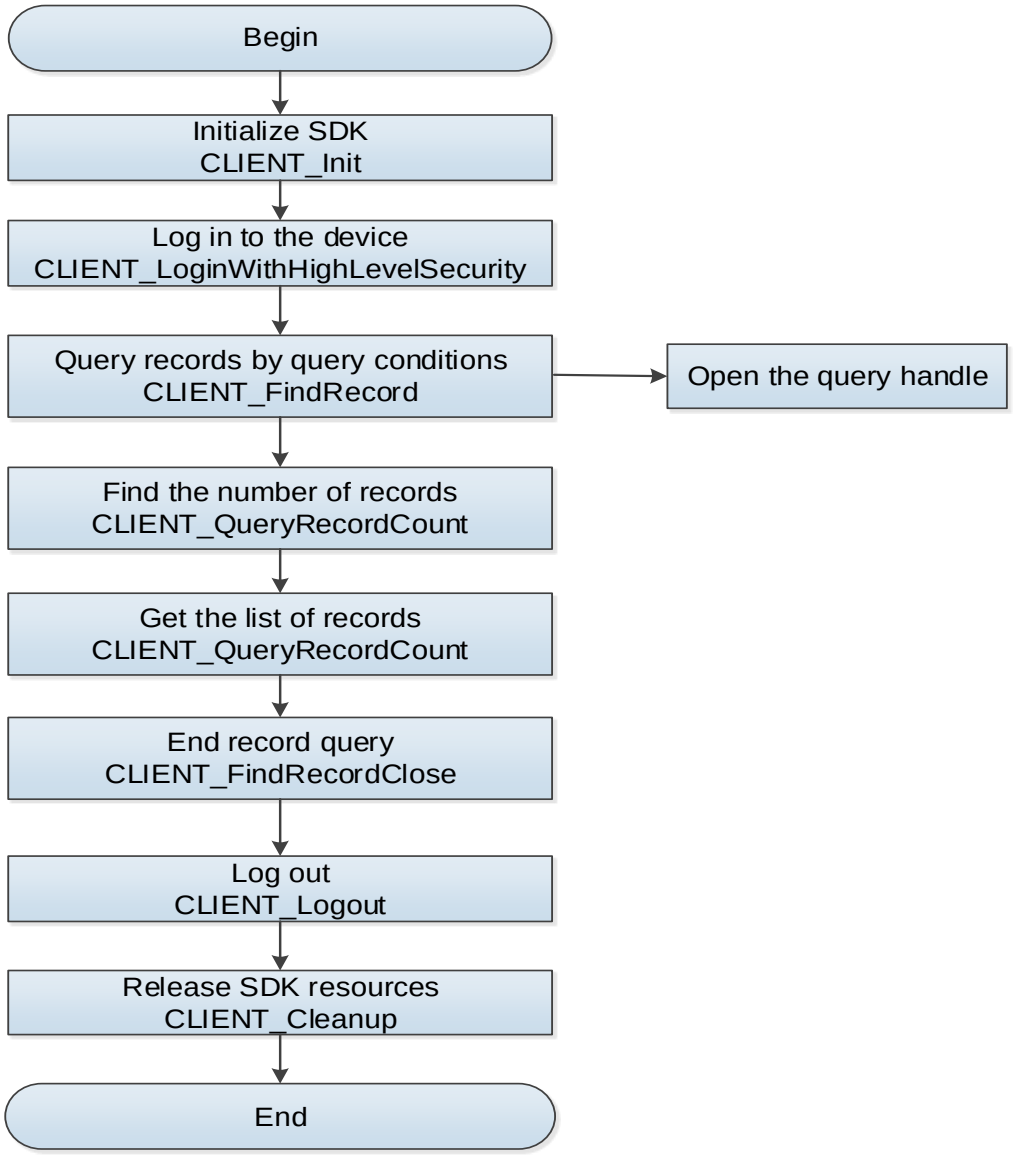
2.2.11.1.2 Interface Overview

Table 2-34 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.

2.2.11.1.3 Process Description

Figure 2-21 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.

2.2.11.1.4 Example Code

```
///Search for the access control card swiping records by card number
public void findAccessRecordByCardNo() {
    /// Search conditions
```

```
    NetSDKLib.FIND_RECORD_ACCESSCTLCARDREC_CONDITION_EX recordCondition = new
NetSDKLib.FIND_RECORD_ACCESSCTLCARDREC_CONDITION_EX();
    recordCondition.bCardNoEnable = 1; //Enable searching by card number
    String cardNo = "FB0DCF65";
    System.arraycopy(cardNo.getBytes(), 0, recordCondition.szCardNo, 0, cardNo.getBytes().length);

    ///CLIENT_FindRecord Input parameters
    NetSDKLib.NET_IN_FIND_RECORD_PARAM stuFindInParam = new
NetSDKLib.NET_IN_FIND_RECORD_PARAM();
    stuFindInParam.emType =
NetSDKLib.EM_NET_RECORD_TYPE.NET_RECORD_ACCESSCTLCARDREC_EX;
    stuFindInParam.pQueryCondition = recordCondition.getPointer();

    ///CLIENT_FindRecord Output parameters
    NetSDKLib.NET_OUT_FIND_RECORD_PARAM stuFindOutParam = new
NetSDKLib.NET_OUT_FIND_RECORD_PARAM();

    recordCondition.write();
    if (netsdkApi.CLIENT_FindRecord(loginHandle, stuFindInParam, stuFindOutParam, 5000)) {
        recordCondition.read();
        System.out.println("FindRecord Succeed" + "\n" + "FindHandle :" +
stuFindOutParam.lFindeHandle);

        int count = 0;    //Times of cycles
        int nFindCount = 0;
        int nRecordCount = 10;    // Number of records for each search
        ///Recordset information of access control card swiping records
        NetSDKLib.NET_RECORDSET_ACCESS_CTL_CARDREC[] pstRecord = new
NetSDKLib.NET_RECORDSET_ACCESS_CTL_CARDREC[nRecordCount];
        for (int i = 0; i < nRecordCount; i++) {
            pstRecord[i] = new NetSDKLib.NET_RECORDSET_ACCESS_CTL_CARDREC();
        }

        ///CLIENT_FindNextRecord Input parameters
        NetSDKLib.NET_IN_FIND_NEXT_RECORD_PARAM stuFindNextInParam = new
NetSDKLib.NET_IN_FIND_NEXT_RECORD_PARAM();
        stuFindNextInParam.lFindeHandle = stuFindOutParam.lFindeHandle;
        stuFindNextInParam.nFileCount = nRecordCount;    //Number of records for the search

        ///CLIENT_FindNextRecord Output parameters
        NetSDKLib.NET_OUT_FIND_NEXT_RECORD_PARAM stuFindNextOutParam = new
NetSDKLib.NET_OUT_FIND_NEXT_RECORD_PARAM();
        stuFindNextOutParam.nMaxRecordNum = nRecordCount;
        stuFindNextOutParam.pRecordList = new Memory(pstRecord[0].dwSize * nRecordCount);
        stuFindNextOutParam.pRecordList.clear(pstRecord[0].dwSize * nRecordCount);

        ToolKits.SetStructArrToPointerData(pstRecord, stuFindNextOutParam.pRecordList);       //
```

```
Copy the array memory to the Pointer

            while (true) {    // Cycle search
                if (netsdkApi.CLIENT_FindNextRecord(stuFindNextInParam, stuFindNextOutParam,
5000)) {
                    ToolKits.GetPointerDataToStructArr(stuFindNextOutParam.pRecordList, pstRecord);

                    for (int i = 0; i < stuFindNextOutParam.nRetRecordNum; i++) {
                        nFindCount = i + count * nRecordCount;

                        System.out.println("[" + nFindCount + "]Card swiping time:" +
pstRecord[i].stuTime.toStringTime());
                        System.out.println("[" + nFindCount + "]User ID:" + new
String(pstRecord[i].szUserID).trim());
                        System.out.println("[" + nFindCount + "]Card number:" + new
String(pstRecord[i].szCardNo).trim());
                        System.out.println("[" + nFindCount + "]Door number:" + pstRecord[i].nDoor);
                        if (pstRecord[i].emDirection == 1) {
                            System.out.println("[" + nFindCount + "]Unlock direction: Enter");
                        } else if (pstRecord[i].emDirection == 2) {
                            System.out.println("[" + nFindCount + "]Unlock direction: Exit");
                        }
                    }

                    if (stuFindNextOutParam.nRetRecordNum < nRecordCount) {
                        break;
                    } else {
                        count++;
                    }
                } else {
                    System.err.println("FindNextRecord Failed" + netsdkApi.CLIENT_GetLastError());
                    break;
                }
            }
            netsdkApi.CLIENT_FindRecordClose(stuFindOutParam.lFindeHandle);
        } else {
            System.err.println("Can Not Find This Record" + String.format("0x%x",
netsdkApi.CLIENT_GetLastError()));
        }
}
```

2.2.11.2 Device log

2.2.11.2.1 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.

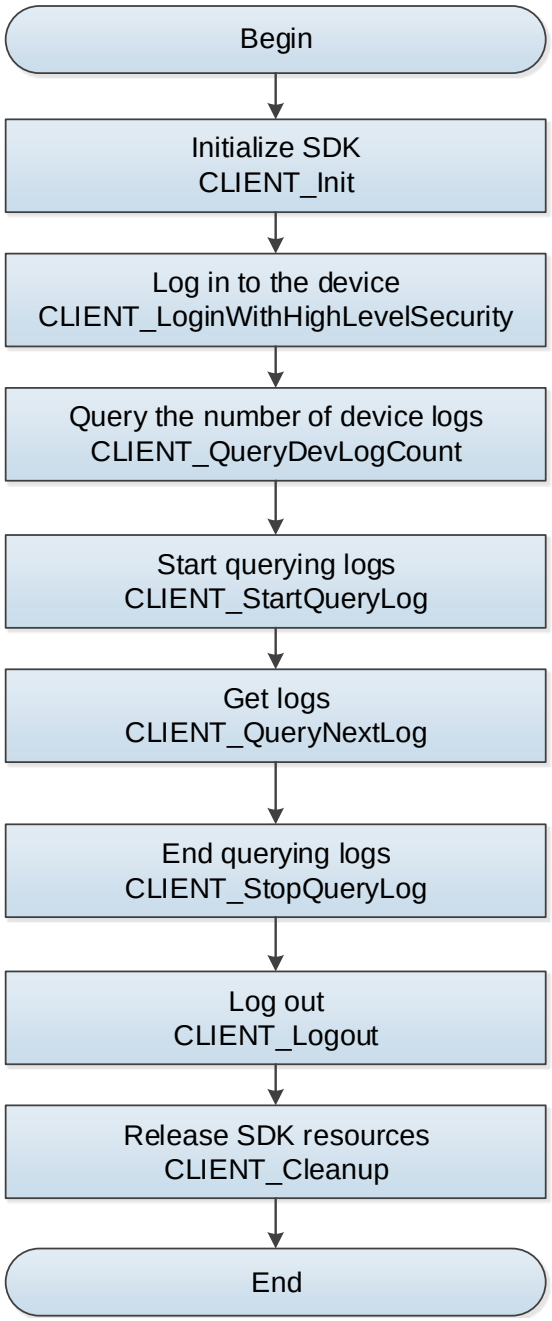
2.2.11.2.2 Interface Overview

Table 2-35 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.

2.2.11.2.3 Process Description

Figure 2-22 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.

2.2.11.2.4 Example Code

```
/**
 * Search for device logs
 */
public void queryDeviceLog() {
    NET_IN_START_QUERYLOG stIn = new NET_IN_START_QUERYLOG();
    // Type of logs to be searched for
    stIn.emLogType = 0;
    // Search time period.
    stIn.stuStartTime = new NET_TIME("2023/1/6/0/0/0");
    stIn.stuEndTime = new NET_TIME("2023/6/6/20/0/0");
    // In ascending chronological order
    stIn.emResultOrder = 1;
    stIn.write();
    NET_OUT_START_QUERYLOG stOut = new NET_OUT_START_QUERYLOG();
    stOut.write();

    LLong lLogID = netsdk.CLIENT_StartQueryLog(m_hLoginHandle, stIn.getPointer(),
stOut.getPointer(), 5000);
    stIn.read();
    stOut.read();
    if (lLogID.longValue() == 0) {
        System.err.println("CLIENT_StartQueryLog Failed!" + ToolKits.getErrorCode());
        return;
    }

    int num = 1024;// If the number of logs in a certain period is too large, search for logs in different
batches. Do not select a large value to avoid memory issues.
    // Initialize
    NET_LOG_INFO[] info = new NET_LOG_INFO[num];
    for (int i = 0; i < info.length; i++) {
        info[i] = new NET_LOG_INFO();
    }

    NET_IN_QUERYNEXTLOG stIn1 = new NET_IN_QUERYNEXTLOG();
```

```
        // Number of logs to be searched for.
        stIn1.nGetCount = num;
        stIn1.write();

        NET_OUT_QUERYNEXTLOG stOut1 = new NET_OUT_QUERYNEXTLOG();
        stOut1.nMaxCount = num;
        stOut1.pstuLogInfo = new Memory(info[0].dwSize * num);
        stOut1.pstuLogInfo.clear(info[0].dwSize * num);
        // Initialize native data
        ToolKits.SetStructArrToPointerData(info, stOut1.pstuLogInfo);
        stOut1.write();
       boolean flg1 = netsdk.CLIENT_QueryNextLog(lLogID, stIn1.getPointer(), stOut1.getPointer(), 5000);
        stIn1.read();
        stOut1.read();
        if (!flg1) {
            System.err.println("CLIENT_QueryNextLog Failed!" + ToolKits.getErrorCode());
            return;
        }
        // Number of logs actually returned
        int nRetCount = stOut1.nRetCount;
        System.out.println("Number of logs actually returned: " + nRetCount);
        // Convert native data to Java data
        ToolKits.GetPointerDataToStructArr(stOut1.pstuLogInfo, info);
        try {
            for (int i = 0; i < nRetCount; i++) {
                NET_LOG_INFO log = info[i];
                System.out.println("Time: " + log.stuTime.toStringTime()); // If the number of logs in a
certain period is too large, when you search for logs in batches, the system obtains the latest log time as
the start time of the next search.
                System.out.println("Operated by: " + new String(log.szUserName, encode));
                System.out.println("Type: " + new String(log.szLogType, encode));
                System.out.println("Log information: " + new String(log.stuLogMsg.szLogMessage,
encode));
            }
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        System.out.println("Number of logs actually returned: " + nRetCount);
        // Manually release the memory
        long peer = Pointer.nativeValue(stOut1.pstuLogInfo);
        Native.free(peer);
        Pointer.nativeValue(stOut1.pstuLogInfo, 0);

        boolean flg2 = netsdk.CLIENT_StopQueryLog(lLogID);
        if (!flg2) {
            System.err.println("CLIENT_StopQueryLog Failed!" + ToolKits.getErrorCode());
            return;
```

```
    }
}
```

2.2.12 Access Control Event

2.2.12.1 Introduction

When the access control device opens the door, it reports related unlock events, including the type of event, unlock method, and information about the person who unlocks it.

2.2.12.2 Process Description

This chapter introduces how to handle callbacks for specific events. For the process of subscribing for and receiving events, see "2.2.3 Intelligent Alarm Event with Image".

2.2.12.3 Enumeration and Structure

- Enumerated value corresponding to the event: EVENT_IVS_ACCESS_CTL
- Structure corresponding to the event: DEV_EVENT_ACCESS_CTL_INFO

2.2.12.4 Example Code

```
private class AnalyzerDataCB implements NetSDKLib.fAnalyzerDataCallBack {
        private BufferedImage gateBufferedImage = null;

        public int invoke(LLong lAnalyzerHandle, int dwAlarmType,
                          Pointer pAlarmInfo, Pointer pBuffer, int dwBufSize,
                          Pointer dwUser, int nSequence, Pointer reserved)
        {
            if (lAnalyzerHandle.longValue() == 0 || pAlarmInfo == null) {
                return -1;
            }

            File path = new File("./GateSnapPicture/");
            if (!path.exists()) {
                path.mkdir();
            }

            ///< Access control event
            if(dwAlarmType == NetSDKLib.EVENT_IVS_ACCESS_CTL) {
                DEV_EVENT_ACCESS_CTL_INFO msg = new
                DEV_EVENT_ACCESS_CTL_INFO();
```

```
                ToolKits.GetPointerData(pAlarmInfo, msg);

                // Save the image and get the image cache
                String snapPicPath = path + "\\" + System.currentTimeMillis() + "GateSnapPicture.jpg";    // Save image address
                byte[] buffer = pBuffer.getByteArray(0, dwBufSize);
            ByteArrayInputStream byteArrInputGlobal = new ByteArrayInputStream(buffer);

                try {
                    gateBufferedImage = ImageIO.read(byteArrInputGlobal);
                    if(gateBufferedImage != null) {
                        ImageIO.write(gateBufferedImage, "jpg", new File(snapPicPath));
                    }
                } catch (IOException e2) {
                    e2.printStackTrace();
                }

                // Display of images and access control information page

                EventQueue eventQueue = Toolkit.getDefaultToolkit().getSystemEventQueue();
                if (eventQueue != null) {
                eventQueue.postEvent( new AccessEvent(target,gateBufferedImage,msg));
                }
            }

            return 0;
        }
```

2.2.13 Face-ID Comparison Event

2.2.13.1 Introduction

Compare the detected person information with the ID information to see if they match.

2.2.13.2 Process Description

This chapter introduces how to handle callbacks for specific events. For the process of subscribing for and receiving events, see "2.2.3 Intelligent Alarm Event with Image".

2.2.13.3 Enumeration and Structure

- Enumerated value corresponding to the event: EVENT_IVS_CITIZEN_PICTURE_COMPARE
- Structure corresponding to the event: DEV_EVENT_CITIZEN_PICTURE_COMPARE_INFO

2.2.13.4 Example Code

```
/* Intelligent alarm event callback */
    public static class fAnalyzerDataCB implements NetSDKLib.fAnalyzerDataCallBack {
        private BufferedImage snapBufferedImage = null;
        private BufferedImage idBufferedImage = null;

        private fAnalyzerDataCB() {}

        private static class fAnalyzerDataCBHolder {
            private static final fAnalyzerDataCB instance = new fAnalyzerDataCB();
        }
        public static fAnalyzerDataCB getInstance() {
            return fAnalyzerDataCBHolder.instance;
        }

        @Override
        public int invoke(LLong lAnalyzerHandle, int dwAlarmType,
                Pointer pAlarmInfo, Pointer pBuffer, int dwBufSize,
                Pointer dwUser, int nSequence, Pointer reserved) {
            if(pAlarmInfo == null) {
                return 0;
            }

            File path = new File("./CitizenCompare/");
            if (!path.exists()) {
                path.mkdir();
            }

            switch(dwAlarmType)
            {
                case NetSDKLib.EVENT_IVS_CITIZEN_PICTURE_COMPARE:
                //Face-ID comparison event
                {
```

```
                    DEV_EVENT_CITIZEN_PICTURE_COMPARE_INFO msg = new
                    DEV_EVENT_CITIZEN_PICTURE_COMPARE_INFO();
                    ToolKits.GetPointerData(pAlarmInfo, msg);

                    try {
                        System.out.println("Time of event:" + msg.stuUTC.toString());
                        System.out.println("Event name :" + new String(msg.szName,
                        "GBK").trim());

                        // Face-ID comparison result,If the similarity is greater than or equal
                        to the threshold, it is a successful match, 1-Success, 0-Failure
                        System.out.println("Comparison result:" + msg.bCompareResult);

                        System.out.println("Similarity between 2 images:" +
msg.nSimilarity);
                        System.out.println("Detection threshold:" + msg.nThreshold);

                        if (msg.emSex == 1) {
                            System.out.println("Gender:Male");
                        }else if (msg.emSex == 2){
                            System.out.println("Gender:Female");
                        }else {
                            System.out.println("Gender:Unknown");
                        }



                        System.out.println("Resident name:" + new String(msg.szCitizen,
                        "GBK").trim());
                        System.out.println("Home address:" + new String(msg.szAddress,
                        "GBK").trim());
                        System.out.println("Credential number:" + new
String(msg.szNumber).trim());
                        System.out.println("Issued by:" + new String(msg.szAuthority,
                        "GBK").trim());

                        System.out.println("Birth date:" + msg.stuBirth.toStringTimeEx());
                        System.out.println("Effective start date:" +
                        msg.stuValidityStart.toStringTimeEx());
                        if (msg.bLongTimeValidFlag == 1) {
                            System.out.println("Valid to: no expiration date");
```

```
            }else{
                System.out.println("Valid to:"+
            msg.stuValidityEnd.toStringTimeEx());
            }
            System.out.println("IC card number:" + new String(msg.szCardNo,
            "GBK").trim());
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Photo
            String strFileName = path + "\\" + System.currentTimeMillis() +
        "citizen_snap.jpg";
                byte[] snapBuffer =
         pBuffer.getByteArray(msg.stuImageInfo[0].dwOffSet,
         msg.stuImageInfo[0].dwFileLenth);
                ByteArrayInputStream snapArrayInputStream = new
         ByteArrayInputStream(snapBuffer);
        try {
            snapBufferedImage = ImageIO.read(snapArrayInputStream);
            if(snapBufferedImage == null) {
                return 0;
            }
            ImageIO.write(snapBufferedImage, "jpg", new File(strFileName));

        } catch (IOException e) {
            e.printStackTrace();
        }

        // Credential image
        strFileName = path + "\\" + System.currentTimeMillis() + "citizen_id.jpg";
                byte[] idBuffer =
         pBuffer.getByteArray(msg.stuImageInfo[1].dwOffSet,
         msg.stuImageInfo[1].dwFileLenth);
                ByteArrayInputStream idArrayInputStream = new
         ByteArrayInputStream(idBuffer);
        try {
            idBufferedImage = ImageIO.read(idArrayInputStream);
            if(idBufferedImage == null) {
                return 0;
            }
```

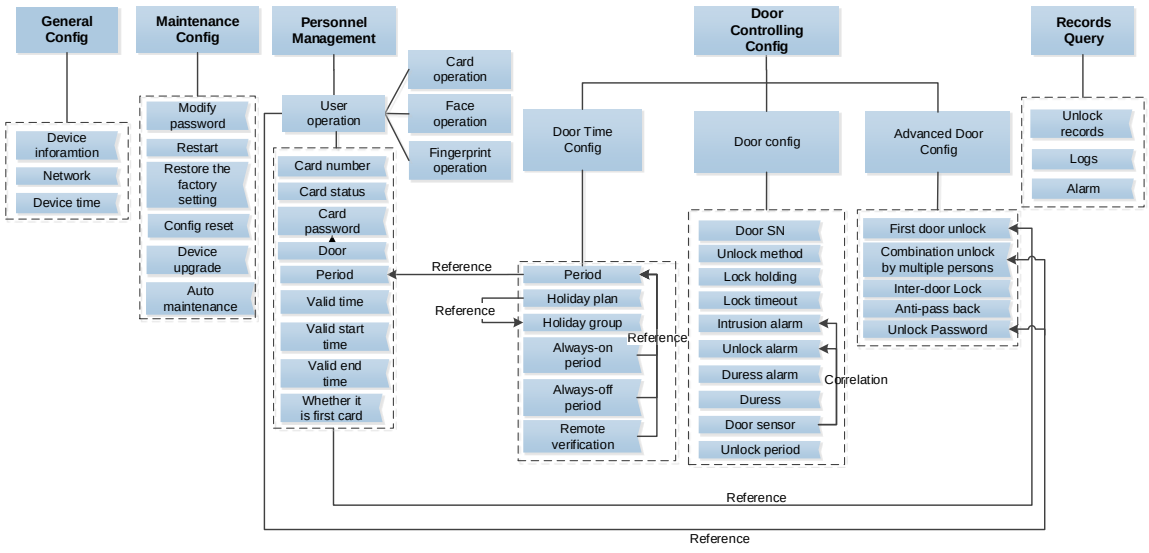
```
                    ImageIO.write(idBufferedImage, "jpg", new File(strFileName));
                } catch (IOException e) {
                    e.printStackTrace();
                }

                break;
            }
            default:
                break;
        }

        return 0;
    }
}
```

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

Figure 2-23 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

See "2.2.1 Access Control."

2.3.2 Alarm Event

See "2.2.2 Alarm Event"

2.3.3 Intelligent Alarm Event with Image

- See "2.2.3 Intelligent Alarm Event with Image".

2.3.4 Viewing Device Information

2.3.4.1 Capability Set Query

2.3.4.1.1 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.

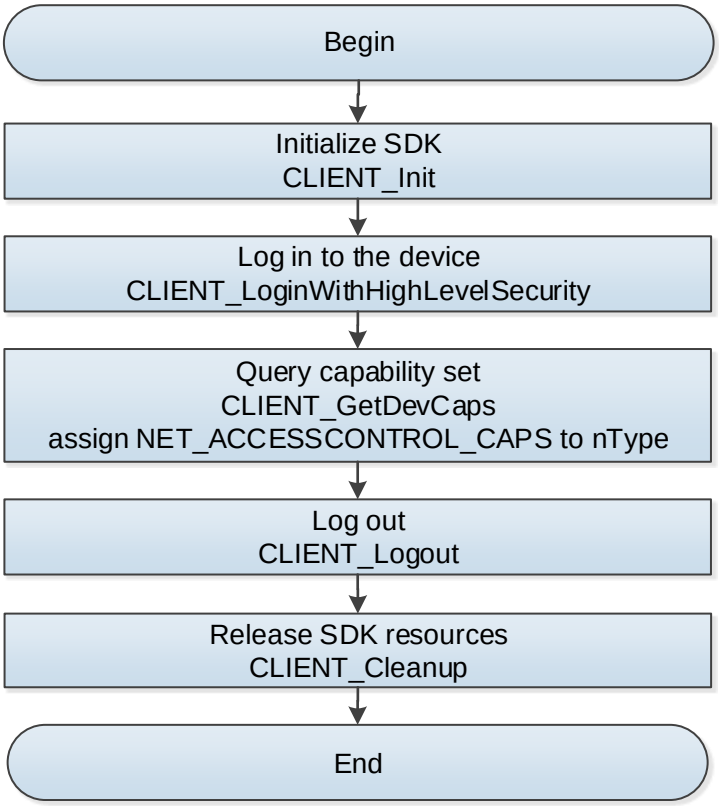
2.3.4.1.2 Interface Overview

Table 2-36 Description of capability set query interface

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

2.3.4.1.3 Process Description

Figure 2-24 Device information viewing



Process

Step 9 Call the **CLIENT_Init** to initialize SDK.

Step 10 Call the **CLIENT_LoginWithHighLevelSecurity** to log in to the device.

Step 11 Call **CLIENT_GetDevCaps** and assign **NET_ACCESSCONTROL_CAPS** to nType, to get the access control.

Step 12 After completing this process, call the **CLIENT_Logout** to log out of the device.

Step 13 After using all SDK functions, call the **CLIENT_Cleanup** to release SDK resources.

2.3.4.1.4 Example Code

```
    public void    accesscontrolCaps(){
//      Get the access control capabilities, pInBuf = NET_IN_AC_CAPS*, pOutBuf =
NET_OUT_AC_CAPS*
        int nType = NetSDKLib.NET_ACCESSCONTROL_CAPS;

        // Input parameters
        NET_IN_AC_CAPS stIn = new NET_IN_AC_CAPS();

        // Output parameters
        NET_OUT_AC_CAPS stOut = new NET_OUT_AC_CAPS();
```

```
        stIn.write();
        stOut.write();

        boolean bRet = netsdk.CLIENT_GetDevCaps(m_hLoginHandle, nType, stIn.getPointer(),
stOut.getPointer(), 3000);
        if(bRet) {
            stOut.read();
            /**
             ACCaps capability set
             */
            NET_AC_CAPS stuACCaps
                    = stOut.stuACCaps;
            System.out.println("ACCaps capability set");

            System.out.println("nChannels:"+stuACCaps.nChannels);
            /**
             Whether the access control alarm logs can be recorded in the recordset
             */
            int bSupAccessControlAlarmRecord
                    = stuACCaps.bSupAccessControlAlarmRecord;
            System.out.println("bSupAccessControlAlarmRecord:"+bSupAccessControlAlarmRecord);

            /**
             AccessControlCustomPassword Storage method of record passwords, 0: Plaintext. 0 by
default; 1: MD5
             */
            int nCustomPasswordEncryption
                    = stuACCaps.nCustomPasswordEncryption;
            System.out.println("nCustomPasswordEncryption:"+nCustomPasswordEncryption);
            /**
             Supports information function or not. 0 : Unknown. It is the default value. 1: Do not
support. 2: Support
             */
            int nSupportFingerPrint
                    = stuACCaps.nSupportFingerPrint;
            System.out.println("nSupportFingerPrint:"+nSupportFingerPrint);

            /**
             User operation capability set
             */
```

```
System.out.println("User operation capability set ");

NET_ACCESS_USER_CAPS stuUserCaps=    stOut.stuUserCaps;
/**
 Maximum number of items that can be sent at a time
 */
int nMaxInsertRate = stuUserCaps.nMaxInsertRate;
System.out.println("nMaxInsertRate:"+nMaxInsertRate);

/**
 Maximum number of users
 */
int nMaxUsers
        = stuUserCaps.nMaxUsers;
System.out.println("nMaxUsers:"+nMaxUsers);
/**
 The maximum amount of information that can be recorded per user
 */
int nMaxFingerPrintsPerUser
        = stuUserCaps.nMaxFingerPrintsPerUser;
System.out.println("nMaxFingerPrintsPerUser:"+nMaxFingerPrintsPerUser);
/**
 The maximum number of cards that can be recorded per user
 */
int nMaxCardsPerUser
        = stuUserCaps.nMaxCardsPerUser;
System.out.println("nMaxCardsPerUser:"+nMaxCardsPerUser);

/**
 Card operation capability set
 */
NET_ACCESS_CARD_CAPS stuCardCaps
        = stOut.stuCardCaps;
System.out.println("Card operation capability set ");

/**
 Maximum number of items that can be sent at a time
 */
int nMaxInsertRate1
        = stuCardCaps.nMaxInsertRate;
```

```
System.out.println("nMaxInsertRate1:"+nMaxInsertRate1);

/**
 Maximum number of cards
 */
int nMaxCards
        = stuCardCaps.nMaxCards;
System.out.println("nMaxCards:"+nMaxCards);


/**
 Information operation capability set
 */
NET_ACCESS_FINGERPRINT_CAPS stuFingerprintCaps
        = stOut.stuFingerprintCaps;

System.out.println("Information operation capability set ");

/**
 Maximum number of items that can be sent at a time
 */
int nMaxInsertRate2
        = stuFingerprintCaps.nMaxInsertRate;
System.out.println("nMaxInsertRate2:"+nMaxInsertRate2);

/**
 The maximum number of bytes of a single message
 */
int nMaxFingerprintSize = stuFingerprintCaps.nMaxFingerprintSize;
System.out.println("nMaxFingerprintSize:"+nMaxFingerprintSize);

/**
 Maximum amount of information
 */
int nMaxFingerprint = stuFingerprintCaps.nMaxFingerprint;

System.out.println("nMaxFingerprint:"+nMaxFingerprint);


/**
```

```
 Target operation capability set
 */
NET_ACCESS_FACE_CAPS stuFaceCaps
        = stOut.stuFaceCaps;

System.out.println("Target operation capability set ");
/**
 Maximum number of items that can be sent at a time
 */
int nMaxInsertRate3
        = stuFaceCaps.nMaxInsertRate;
System.out.println("nMaxInsertRate3:"+nMaxInsertRate3);

/**
 Target storage limit
 */
int nMaxFace
        = stuFaceCaps.nMaxFace;
System.out.println("nMaxFace:"+nMaxFace);

/**
 Target recognition type. 0: White light. 1: IR
 */
int nRecognitionType
        = stuFaceCaps.nRecognitionType;
System.out.println("nRecognitionType:"+nRecognitionType);
/**
 Target recognition algorithm.
 */
int nRecognitionAlgorithm
        = stuFaceCaps.nRecognitionAlgorithm;
System.out.println("nRecognitionAlgorithm:"+nRecognitionAlgorithm);

/**
 Eyes related capability set
 */
NET_ACCESS_IRIS_CAPS stuIrisCaps
        = stOut.stuIrisCaps;

System.out.println("Eyes related capability set ");
```

```
/**
 Maximum number of items that can be inserted at a time.
 */
int nMaxInsertRate4
        = stuIrisCaps.nMaxInsertRate;
System.out.println("nMaxInsertRate4:"+nMaxInsertRate4);

/**
 Minimum size of eye information image in KB
 */
int nMinIrisPhotoSize
        = stuIrisCaps.nMinIrisPhotoSize;
System.out.println("nMinIrisPhotoSize:"+nMinIrisPhotoSize);

/**
 Minimum size of eye information image in KB
 */
int nMaxIrisPhotoSize
        = stuIrisCaps.nMaxIrisPhotoSize;
System.out.println("nMaxIrisPhotoSize:"+nMaxIrisPhotoSize);

/**
 The supported maximum number of groups for each user
 */
int nMaxIrisGroup
        = stuIrisCaps.nMaxIrisGroup;
System.out.println("nMaxIrisGroup:"+nMaxIrisGroup);

/**
 Eye recognition algorithm provider identifier.
 */
int nRecognitionAlgorithmVender
        = stuIrisCaps.nRecognitionAlgorithmVender;
System.out.println("nRecognitionAlgorithmVender:"+nRecognitionAlgorithmVender);

/**
 Version number of the algorithm (model). If the version number has more than one digit, it indicates a version every 8 bits in ascending order from Major to Minor. For example, 1.5.2 indicates 0x00010502
```

```
             */
            int nRecognitionVersion
                        = stuIrisCaps.nRecognitionVersion;
            System.out.println("nRecognitionVersion:"+nRecognitionVersion);

            /**
             Eye information storage limit
             */
            int nMaxIrisesCount
                        = stuIrisCaps.nMaxIrisesCount;
            System.out.println("nMaxIrisesCount:"+nMaxIrisesCount);


        } else {
            System.err.println("GetDevCaps Failed!" + ToolKits.getErrorCode());
        }
    }
```

2.3.4.2 Viewing Device Version and MAC

See "2.3.4.2 Viewing Device Version and MAC"

2.3.5 Network Setting

See "2.2.5 Network Setting"

2.3.6 Setting the Device Time

See "2.2.6 Device Time Setting"

2.3.7 Personnel Management

2.3.7.1 User Management

2.3.7.1.1 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).

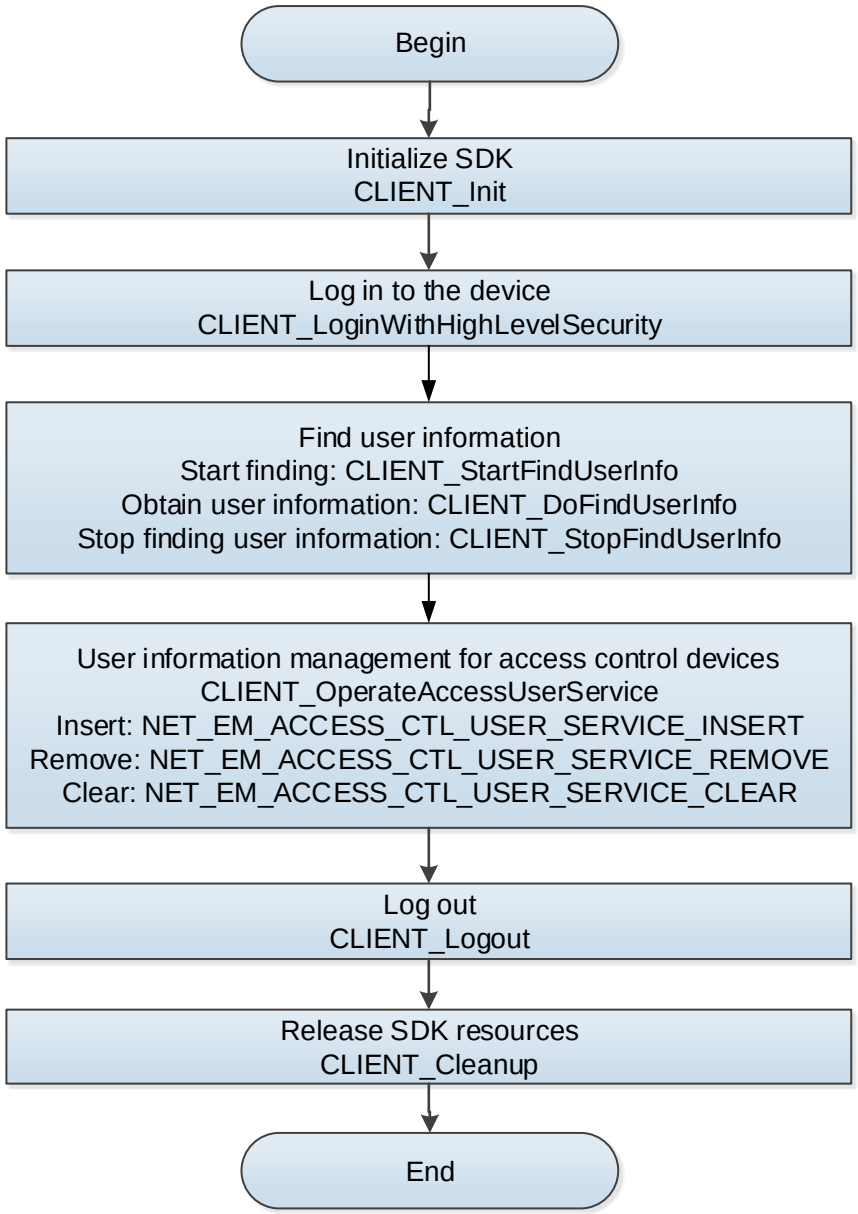
2.3.7.1.2 Interface Overview

Table 2-37 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.

2.3.7.1.3 Process Description

Figure 2-25 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.

2.3.7.1.4 Example Code

```
/**
 * Get user information by user ID
 */
public void operateAccessUserService(){

    String[] userIDs = {"3"};

    // Number of users that were obtained
    int nMaxNum = userIDs.length;

    // ////////////////////////// The following are fixed writing methods
    // //////////////////////////////////
    // User operation type
    // Get users
    int emtype =
NetSDKLib.NET_EM_ACCESS_CTL_USER_SERVICE.NET_EM_ACCESS_CTL_USER_SERVICE_G
ET;

    /**
     * User information array
     */
    // Initialize the user information array first
    NetSDKLib.NET_ACCESS_USER_INFO[] users = new
NetSDKLib.NET_ACCESS_USER_INFO[nMaxNum];


    // Array of failure information returned by initialization
    NetSDKLib.FAIL_CODE[] failCodes = new NetSDKLib.FAIL_CODE[nMaxNum];

    for (int i = 0; i < nMaxNum; i++) {
        NetSDKLib.NET_ACCESS_USER_INFO info      = new
NetSDKLib.NET_ACCESS_USER_INFO();
        int size
                = new NET_FLOORS_INFO().size();

        Pointer floors =new Memory(size);

        floors.clear(size);
```

```
            info.pstuFloorsEx2=floors;

            NET_ACCESS_USER_INFO_EX pstuUserInfoEx=
                    new NET_ACCESS_USER_INFO_EX();

            Pointer pstuUserInfo
                    = new Memory(pstuUserInfoEx.size());

            pstuUserInfo.clear(pstuUserInfoEx.size());

            info.pstuUserInfoEx=pstuUserInfo;

            users[i]=info;

            failCodes[i] = new NetSDKLib.FAIL_CODE();
        }

        /**
         * Input parameters NET_IN_ACCESS_USER_SERVICE_GET
         */
        NetSDKLib.NET_IN_ACCESS_USER_SERVICE_GET stIn = new
NetSDKLib.NET_IN_ACCESS_USER_SERVICE_GET();
        // Number of user IDs
        stIn.nUserNum = userIDs.length;

        // User ID
        for (int i = 0; i < userIDs.length; i++) {
            System.arraycopy(userIDs[i].getBytes(), 0,
                    stIn.szUserIDs[i].szUserID, 0, userIDs[i].getBytes().length);
        }

        /**
         * Output parameters    NET_OUT_ACCESS_USER_SERVICE_GET
         */
        NetSDKLib.NET_OUT_ACCESS_USER_SERVICE_GET stOut = new
NetSDKLib.NET_OUT_ACCESS_USER_SERVICE_GET();

        stOut.nMaxRetNum = nMaxNum;

        stOut.pUserInfo = new Memory(users[0].size() * nMaxNum); // Request memory
        stOut.pUserInfo.clear(users[0].size() * nMaxNum);

        stOut.pFailCode = new Memory(failCodes[0].size() * nMaxNum); // Request memory
        stOut.pFailCode.clear(failCodes[0].size() * nMaxNum);

        ToolKits.SetStructArrToPointerData(users, stOut.pUserInfo);
```

```
        ToolKits.SetStructArrToPointerData(failCodes, stOut.pFailCode);


        stIn.write();
        stOut.write();


        if (netSdk.CLIENT_OperateAccessUserService(loginHandle, emtype,
                stIn.getPointer(), stOut.getPointer(), 3000)) {
            // Convert pointers into specific information
            ToolKits.GetPointerDataToStructArr(stOut.pUserInfo, users);
            ToolKits.GetPointerDataToStructArr(stOut.pFailCode, failCodes);

            /**
             * Print the specific information
             */
            for (int i = 0; i < nMaxNum; i++) {
                try {
                    System.out.println("[" + i + "]Username:"
                            + new String(users[i].szName, "GBK").trim());
                } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
                }
                System.out.println("[" + i + "]Password:"
                        + new String(users[i].szPsw).trim());
                System.out.println("[" + i + "]User search result: "
                        + failCodes[i].nFailCode);
            }
        } else {
            System.err.println("Failed to search for the users, " + ToolKits.getErrorCode());
        }
    }
```

2.3.7.2 Card Management

2.3.7.2.1 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).

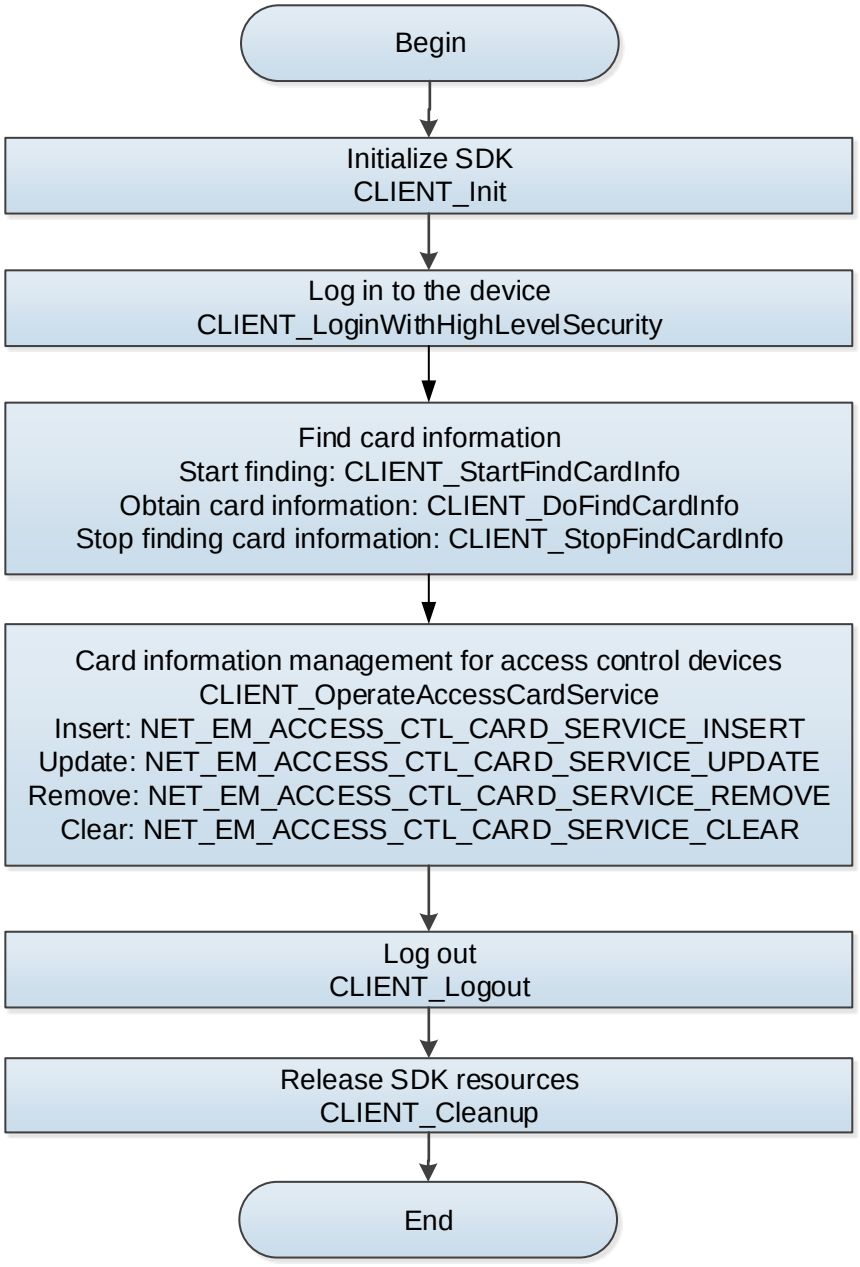
2.3.7.2.2 Interface Overview

Table 2-38 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

2.3.7.2.3 Process Description

Figure 2-26 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.

2.3.7.2.4 Example Code

```
/**
 * Search for all card information
 */
public void queryAllCard() {
//          String userId = "1011";
    // ////////////////////////////////////////////////////////////////////////////////////////
    /**
     * Input parameters
     */
    NET_IN_CARDINFO_START_FIND stInFind = new NET_IN_CARDINFO_START_FIND();
    // User ID. When it is left blank or not specified, all cards of all users will be searched for.
//          System.arraycopy(userId.getBytes(), 0, stInFind.szUserID, 0,
//                  userId.getBytes().length);

    /**
     * Output parameters
     */
    NET_OUT_CARDINFO_START_FIND stOutFind = new NET_OUT_CARDINFO_START_FIND();

    LLong lFindHandle = netsdkApi.CLIENT_StartFindCardInfo(loginHandle,
            stInFind, stOutFind, TIME_OUT);

    if (lFindHandle.longValue() == 0) {
        return;
    }

    System.out.println("Total number of searched results:" + stOutFind.nTotalCount);

    if (stOutFind.nTotalCount <= 0) {
        return;
    }
    // ////////////////////////////////////////////////////////////////////////////////////////
    // Start number
    int startNo = 0;

    // Number of records for each search
    int nFindCount = stOutFind.nCapNum == 0 ? 5 : stOutFind.nCapNum;

    while (true) {
```

```
NET_ACCESS_CARD_INFO[] cardInfos = new NET_ACCESS_CARD_INFO[nFindCount];
for (int i = 0; i < nFindCount; i++) {
    cardInfos[i] = new NET_ACCESS_CARD_INFO();
    cardInfos[i].bUserIDEx = 1;
}

/**
 * Input parameter
 */
NET_IN_CARDINFO_DO_FIND stInDoFind = new NET_IN_CARDINFO_DO_FIND();
// Start number
stInDoFind.nStartNo = startNo;

// Number of records for this search
stInDoFind.nCount = nFindCount;

/**
 * Output parameters
 */
NET_OUT_CARDINFO_DO_FIND stOutDoFind = new NET_OUT_CARDINFO_DO_FIND();
stOutDoFind.nMaxNum = nFindCount;

stOutDoFind.pstuInfo = new Memory(cardInfos[0].size() * nFindCount);
stOutDoFind.pstuInfo.clear(cardInfos[0].size() * nFindCount);

ToolKits.SetStructArrToPointerData(cardInfos, stOutDoFind.pstuInfo);

if (netsdkApi.CLIENT_DoFindCardInfo(lFindHandle, stInDoFind,
        stOutDoFind, TIME_OUT)) {
    if (stOutDoFind.nRetNum <= 0) {
        return;
    }

    ToolKits.GetPointerDataToStructArr(stOutDoFind.pstuInfo,
            cardInfos);

    for (int i = 0; i < stOutDoFind.nRetNum; i++) {
        System.out.println("[" + (startNo + i) + "]User ID:"
                + new String(cardInfos[i].szUserID).trim());
        System.out.println("[" + (startNo + i) + "]Card number:"
                + new String(cardInfos[i].szCardNo).trim());
        System.out.println("[" + (startNo + i) + "]Card type:"
                + cardInfos[i].emType + "\n");
    }
}

if (stOutDoFind.nRetNum < nFindCount) {
```

```
                break;
            } else {
                startNo += nFindCount;
            }
        }

        // ////////////////////////////////////////////////////////////////////////////////////////
        // Stop the search
        if (lFindHandle.longValue() != 0) {
            netsdkApi.CLIENT_StopFindCardInfo(lFindHandle);
            lFindHandle.setValue(0);
        }
    }
```

2.3.7.3 Face Management

2.3.7.3.1 Introduction

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

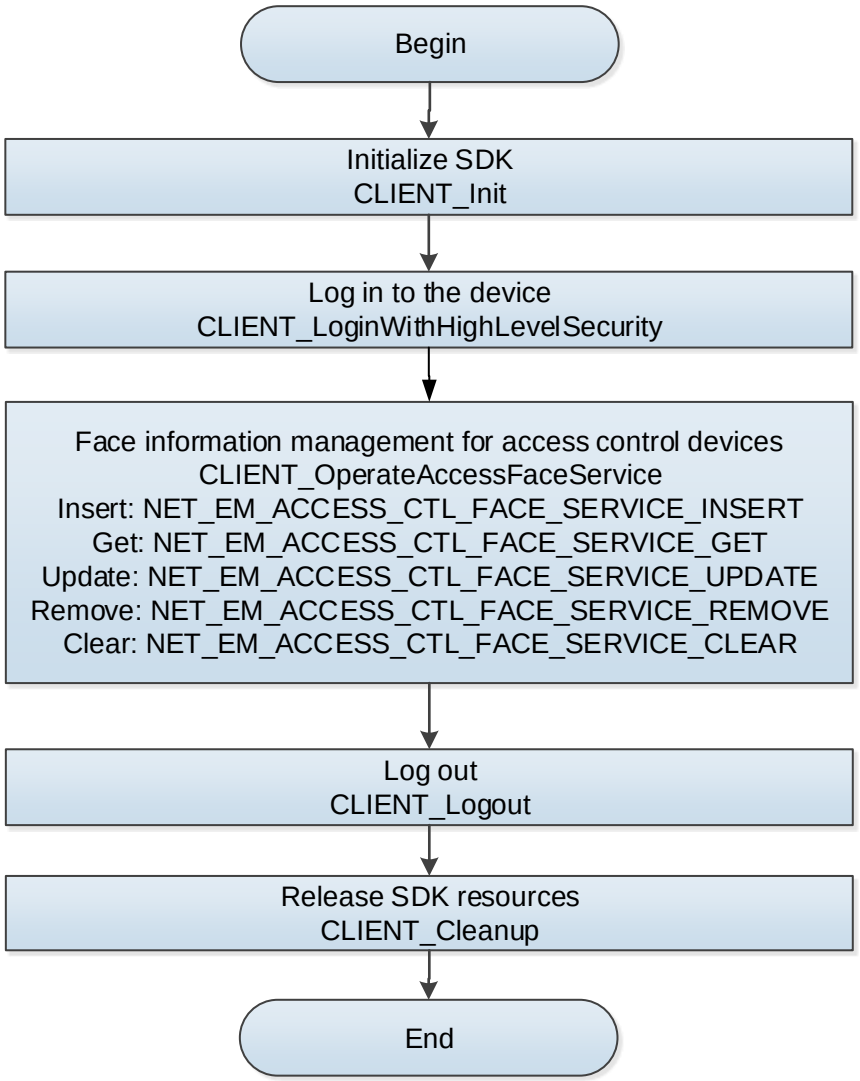
2.3.7.3.2 Interface Overview

Table 2-39 Description of face information interface

Interface	Description
CLIENT_OperateAccessFaceService	Face information management interface for access control devices

2.3.7.3.3 Process Description

Figure 2-27 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.

2.3.7.3.4 Example Code

```
/**
 * Get the face information
 */
public void getFace() {
        String[] userIDs = { "1","2","3" };
 //     String[] userIDs = { "3423" };
        // The maximum number of users whose faces that you can obtain
        int nMaxCount = userIDs.length;
```

```
        // ////////////////////// Initialize the face information of each user ////////////////////////
        NetSDKLib.NET_ACCESS_FACE_INFO[] faces = new
NetSDKLib.NET_ACCESS_FACE_INFO[nMaxCount];
        for (int i = 0; i < faces.length; i++) {
            faces[i] = new NetSDKLib.NET_ACCESS_FACE_INFO();

            // Request the memory according to the actual number of face images of each user. Up to 5
images are supported for each user.

            faces[i].nFacePhoto = 1; // Number of images for each user

            // Request the memory for each image
            faces[i].nInFacePhotoLen[0] = 200 * 1024;
            faces[i].pFacePhotos[0].pFacePhoto = new Memory(200 * 1024); // Face image data. The size
cannot exceed 200 KB
            faces[i].pFacePhotos[0].pFacePhoto.clear(200 * 1024);
        }

        // Initialize
        NetSDKLib.FAIL_CODE[] failCodes = new NetSDKLib.FAIL_CODE[nMaxCount];
        for (int i = 0; i < failCodes.length; i++) {
            failCodes[i] = new NetSDKLib.FAIL_CODE();
        }

        // Face image operation type
        // Get face information
        int emtype =
NetSDKLib.NET_EM_ACCESS_CTL_FACE_SERVICE.NET_EM_ACCESS_CTL_FACE_SERVICE_G
ET;

        /**
         * Input parameters
         */
        NetSDKLib.NET_IN_ACCESS_FACE_SERVICE_GET stIn = new
NetSDKLib.NET_IN_ACCESS_FACE_SERVICE_GET();
        stIn.nUserNum = nMaxCount;
        for (int i = 0; i < nMaxCount; i++) {
            System.arraycopy(userIDs[i].getBytes(), 0,
                    stIn.szUserIDs[i].szUserID, 0, userIDs[i].getBytes().length);
        }

        /**
         * Output parameters NET_OUT_ACCESS_FACE_SERVICE_GET
         */
```

```
    NetSDKLib.NET_OUT_ACCESS_FACE_SERVICE_GET stOut = new
NetSDKLib.NET_OUT_ACCESS_FACE_SERVICE_GET();
    stOut.nMaxRetNum = nMaxCount;

    stOut.pFaceInfo = new Memory(faces[0].size() * nMaxCount);
    stOut.pFaceInfo.clear(faces[0].size() * nMaxCount);

    stOut.pFailCode = new Memory(failCodes[0].size() * nMaxCount);
    stOut.pFailCode.clear(failCodes[0].size() * nMaxCount);

    ToolKits.SetStructArrToPointerData(faces, stOut.pFaceInfo);
    ToolKits.SetStructArrToPointerData(failCodes, stOut.pFailCode);

    stIn.write();
    stOut.write();
    if (netSdk.CLIENT_OperateAccessFaceService(loginHandle, emtype,
            stIn.getPointer(), stOut.getPointer(), 3000)) {
        // Convert the obtained result information to a concrete structure
        ToolKits.GetPointerDataToStructArr(stOut.pFaceInfo, faces);
        ToolKits.GetPointerDataToStructArr(stOut.pFailCode, failCodes);

        // Print the specific information
        // nMaxCount Number of users
        for (int i = 0; i < nMaxCount; i++) {
            System.out.println("[" + i + "]User ID: "
                    + new String(faces[i].szUserID).trim());

            int nFacePhoto = faces[i].nFacePhoto;
            System.out.println("nFacePhoto:"+nFacePhoto);


            int nFaceData = faces[i].nFaceData;
            System.out.println("nFaceData:"+nFaceData);
            NetSDKLib.FACEDATA[] szFaceDatas = faces[i].szFaceDatas;

            for(int n=0;n<nFaceData;n++){
                NetSDKLib.FACEDATA szFaceData = szFaceDatas[i];

                byte[] szFaceData1 = szFaceData.szFaceData;
                new DefaultAnalyseTaskResultCallBack().fileOut(szFaceData1);
            }


            System.out.println("[" + i + "]The result of obtaining the face information: "
                    + failCodes[i].nFailCode);
        }
    } else {
```

```
            System.err.println("Failed to get the face., " + ToolKits.getErrorCode());
        }

        stIn.read();
        stOut.read();
    }
```

2.3.7.4 Fingerprint Management

2.3.7.4.1 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).

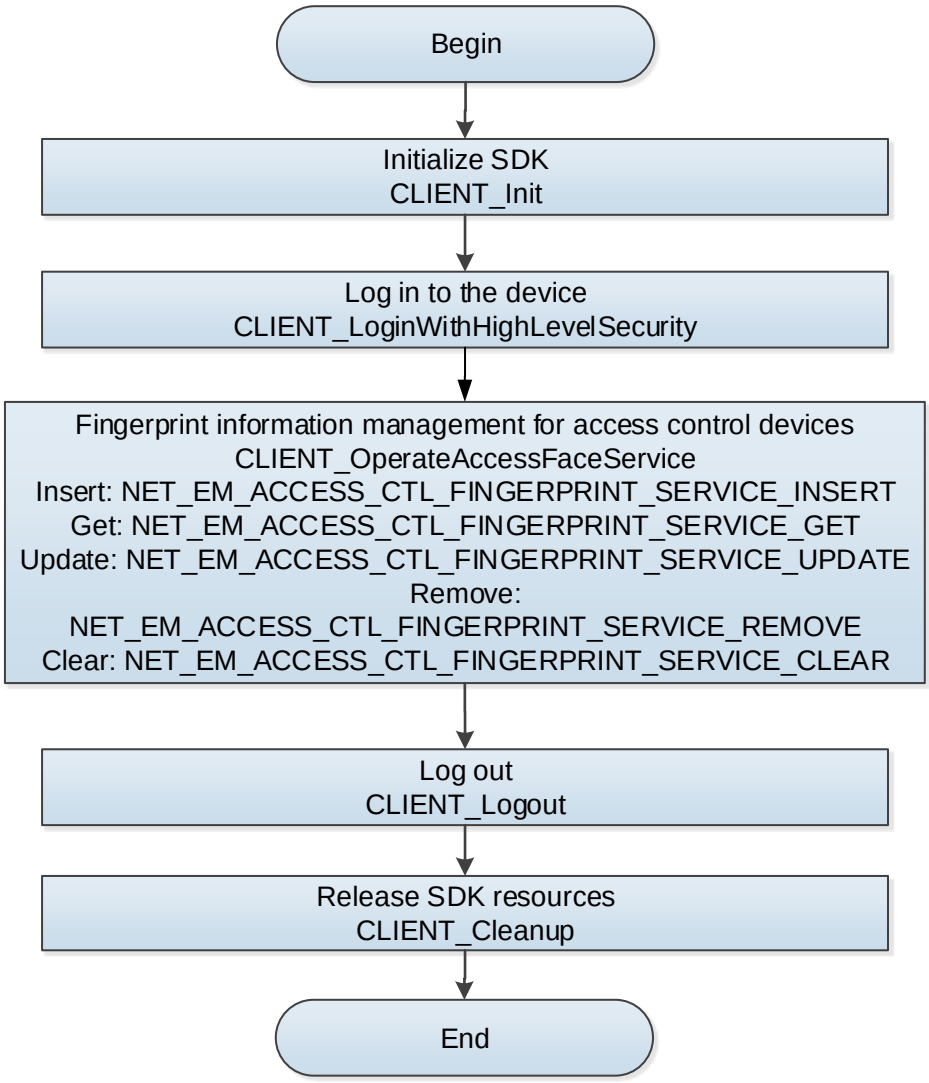
2.3.7.4.2 Interface Overview

Table 2-40 Description of fingerprint information interface

Interface	Description
CLIENT_OperateAccessFingerprintService	Fingerprint information management interface

2.3.7.4.3 Process Description

Figure 2-28 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.

2.3.7.4.4 Example Code

```
/**********************************************************************************************
 * Operations on information: Add, modify, delete, obtain or clear one piece of information for only one user at a time. Up to 3 pieces of information can be added for each user.
 *
 * If you need to perform an operation on the information of a user, repeat the operation. For example, call the interface: User ID: 1011 + one piece of information data. Call the interface user ID: 1011 + one piece of information data again.
```

```
 ***********************************************************************************************/
/**
 * Add the information by user ID (One user ID corresponds to one piece of information)
 */
public void addFingerprint() {
	String userId = "1011";

	String finger = "xTxpAASFsk3+8hoDh4ky0ghH2oR7hjp658Wp"
			+ "Q4eJQyEQBdoFgIerDuhFuAQGhsr6BvQpguuF804HMjpFaIUjOQ"
			+ "AJyUSGiAwlCEupSGSFnA3gS7nF2IUUVe62SYVXhcSBoJG5iGSGvJqI"
			+ "lZsRkYhksjBNnUmjh4VOEFEpizyGVUnol4oQQohVft6sOQdWh93lzarniBeJK"
			+
"4EANhtCKoqEmuA0C4M+im2+/zPJAzCKrcX5OipCtIpF9g6DqQOth32CAA8bUiiGdbog0wqKWofGYtxwecgQhsZi"
			+
"ONMO3VqH5p7b8kmGiooaSQfF/MEWilKZ+Dgeg4iJuwYQBgyDJ4uEMfg3/kOsi53VB/++RTaLbf74N/uFY4NzPegFyQNfg3Ot6EXawda"
			+
"CO63/fhpAWIS0YehJyoRSg9SR+IfYRFCCbQXnxcrBR4RtTeCJykPKgrU5/rYagr6EvbIOeDhCOoWt5dB/yI"
			+
"dDgd3FB8f4BEGDpf3oR/oDQYImAQfJ+kM+iy5x+HXMRUaJznnQtCrGNoteifk4CkUohZaWyIepBTGEdsLX/9qBHoZO0pD"
			+
"JGog6iybq+Tn3x1yH7vrbco2HN4R+8s//+EIhhbcW18f3RZqGnx4LfhvXO4qvOvlB+0qQhpc9/L5MVlJH878zUyMvjxF2YSIlVRJUamE"
			+
"lNH8nRUGD8niPUp/xI6f28/czRPMPMfH2QvIvQSNEcyNB/zIyRSM/8zXzUvMzYfNFQ/IzUh9CNSf1X//xL0U6MohGebIWlDNWhRZi/xIVgkVP+Hhn"
			+
"9f9RAUWhRi8eafk+QXSytDwtFjGCklxSMTK8pNZYFILYkrkkYmS30qhUEcKqtNswRwKFkq8QQSHHkRBtESNRkfhdEbK9ovslIVKMoYYdMtGdgSM0"
			+ "ERE1cnIOQQTYoLJkIQJ0gJsNFFTugetWEhME5zESEnLokpsbFFS+kYVfVLGbs"
			+ "E4qQTaYgcEOIRPEkfQRJGDhkWsOEmXWoLdpRKKNoA5tRENBoM"
			+
"5GEFEkosISZBLUkKlXIHbdkGRRAQHRoAwLDw4NFwgJChglKxEoKiYaGQcSBDIbIy0pNBQWFVVs";

	// Initialize the information data
	NET_ACCESS_FINGERPRINT_INFO fingerprint = new NET_ACCESS_FINGERPRINT_INFO();

	/**
	 * Add the information data
```

```
     */
    // User ID
    System.arraycopy(userId.getBytes(), 0, fingerprint.szUserID, 0,
            userId.getBytes().length);

    // Convert the strings to information data
    byte[] fingerPrintBuffer = Base64Util.getDecoder().decode(finger);

    // Length of a single piece of information
    fingerprint.nPacketLen = fingerPrintBuffer.length;

    // Number of information packages
    fingerprint.nPacketNum = 1;

    // Information data
    // Request the memory first
    fingerprint.szFingerPrintInfo = new Memory(fingerPrintBuffer.length);
    fingerprint.szFingerPrintInfo.clear(fingerPrintBuffer.length);

    fingerprint.szFingerPrintInfo.write(0, fingerPrintBuffer, 0,
            fingerPrintBuffer.length);

    // ////////////////////////////////////////////////////////////////////////////////

    // Initialize
    FAIL_CODE failCode = new FAIL_CODE();

    // Type of information operations
    // Add information type
    int emtype =
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE.NET_EM_ACCESS_CTL_FINGERPRINT_SERVI
CE_INSERT;

    /**
     * Input parameters
     */
    NET_IN_ACCESS_FINGERPRINT_SERVICE_INSERT stIn = new
NET_IN_ACCESS_FINGERPRINT_SERVICE_INSERT();
    stIn.nFpNum = 1;
```

```
        stIn.pFingerPrintInfo = fingerprint.getPointer();

        /**
         * Output parameters
         */
        NET_OUT_ACCESS_FINGERPRINT_SERVICE_INSERT stOut = new
NET_OUT_ACCESS_FINGERPRINT_SERVICE_INSERT();
        stOut.nMaxRetNum = 1;

        stOut.pFailCode = failCode.getPointer();

        stIn.write();
        stOut.write();
        fingerprint.write();
        failCode.write();
        if (netsdkApi.CLIENT_OperateAccessFingerprintService(loginHandle,
                        emtype, stIn.getPointer(), stOut.getPointer(), TIME_OUT)) {

                // Print the specific information
                System.out.println("Result of adding the information: " + failCode.nFailCode);
        } else {
                System.err.println("Failed to add the information, " + getErrorCode());
        }

        fingerprint.read();
        failCode.read();
        stIn.read();
        stOut.read();
}

/**
 * Modify the information by user ID (one user ID corresponds to one piece of information)
 */
public void modifyFingerprint() {
        String userId = "1011";

        String finger = "xTxpAASFsk3+8hoDh4ky0ghH2oR7hjp658Wp"
                        + "Q4eJQyEQBdoFgIerDuhFuAQGhsr6BvQpguuF804HMjpFaIUjOQ"
                        + "AJyUSGiAwlCEupSGSFnA3gS7nF2IUUVe62SYVXhcSBoJG5iGSGvJqI"
                        + "lZsRkYhksjBNnUmjh4VOEFEpizyGVUnol4oQQohVft6sOQdWh93lzarniBeJK"
```

```
                +
"4EANhtCKoqEmuA0C4M+im2+/zPJAzCKrcX5OipCtIpF9g6DqQOth32CAA8bUiiGdbog0wqKWofGYtxwecgQhsZi"
                +
"ONMO3VqH5p7b8kmGiooaSQfF/MEWilKZ+Dgeg4iJuwYQBgyDJ4uEMfg3/kOsi53VB/++RTaLbf74N/uFY4NzPegFyQNfg3Ot6EXawda"
                +
"CO63/fhpAWIS0YehJyoRSg9SR+IfYRFCCbQXnxcrBR4RtTeCJykPKgrU5/rYagr6EvbIOeDhCOoWt5dB/yI"
                +
"dDgd3FB8f4BEGDpf3oR/oDQYImAQfJ+kM+iy5x+HXMRUaJznnQtCrGNoteifk4CkUohZaWyIepBTGEdsLX/9qBHoZO0pD"
                +
"JGog6iybq+Tn3x1yH7vrbco2HN4R+8s//+EIhhbcW18f3RZqGnx4LfhvXO4qvOvlB+0qQhpc9/L5MVlJH878zUyMvjxF2YSIlVRJUamE"
                +
"lNH8nRUGD8niPUp/xI6f28/czRPMPMfH2QvIvQSNEcyNB/zIyRSM/8zXzUvMzYfNFQ/IzUh9CNSf1X//xL0U6MohGebIWlDNWhRZi/xIVgkVP+Hhn"
                +
"9f9RAUWhRi8eafk+QXSytDwtFjGCklxSMTK8pNZYFILYkrkkYmS30qhUEcKqtNswRwKFkq8QQSHHkRBtESNRkfhdEbK9ovslIVKMoYYdMtGdgSM0"
                + "ERE1cnIOQQTYoLJkIQJ0gJsNFFTugetWEhME5zESEnLokpsbFFS+kYVfVLGbs"
                + "E4qQTaYgcEOIRPEkfQRJGDhkWsOEmXWoLdpRKKNoA5tRENBoM"
                +
"5GEFEkosISZBLUkKlXIHbdkGRRAQHRoAwLDw4NFwgJChglKxEoKiYaGQcSBDIbIy0pNBQWFVVs";

    // Intialize the information data
    NET_ACCESS_FINGERPRINT_INFO fingerprint = new NET_ACCESS_FINGERPRINT_INFO();

    /**
     * Add the information data
     */
    // User ID
    System.arraycopy(userId.getBytes(), 0, fingerprint.szUserID, 0,
                userId.getBytes().length);

    // Convert the strings to information data
    byte[] fingerPrintBuffer = Base64Util.getDecoder().decode(finger);

    // Length of a single piece of information
    fingerprint.nPacketLen = fingerPrintBuffer.length;
```

```
    // Number of information packages
    fingerprint.nPacketNum = 1;

    // Information data
    // Request the memory first
    fingerprint.szFingerPrintInfo = new Memory(fingerPrintBuffer.length);
    fingerprint.szFingerPrintInfo.clear(fingerPrintBuffer.length);

    fingerprint.szFingerPrintInfo.write(0, fingerPrintBuffer, 0,
            fingerPrintBuffer.length);

    // //////////////////////////////////////////////////////////////////////////////
    // //////////////////////////////////////////////////////////////////////////////

    // Initialize
    FAIL_CODE failCode = new FAIL_CODE();

    // Type of information operations
    // Modify the information data
    int emtype =
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE.NET_EM_ACCESS_CTL_FINGERPRINT_SERVI
CE_UPDATE;

    /**
     * Input parameters
     */
    NET_IN_ACCESS_FINGERPRINT_SERVICE_UPDATE stIn = new
NET_IN_ACCESS_FINGERPRINT_SERVICE_UPDATE();
    stIn.nFpNum = 1;

    stIn.pFingerPrintInfo = fingerprint.getPointer();

    /**
     * Output parameters
     */
    NET_OUT_ACCESS_FINGERPRINT_SERVICE_UPDATE stOut = new
NET_OUT_ACCESS_FINGERPRINT_SERVICE_UPDATE();
    stOut.nMaxRetNum = 1;

    stOut.pFailCode = failCode.getPointer();
```

```
        stIn.write();
        stOut.write();
        fingerprint.write();
        failCode.write();
        if (netsdkApi.CLIENT_OperateAccessFingerprintService(loginHandle,
                emtype, stIn.getPointer(), stOut.getPointer(), TIME_OUT)) {

            // Print the specific information
            System.out.println("Result of modifying the information : " + failCode.nFailCode);
        } else {
            System.err.println("Failed to modify the information, " + getErrorCode());
        }

        fingerprint.read();
        failCode.read();
        stIn.read();
        stOut.read();
}

/**
 * Get a single piece of information by user ID (one user ID corresponds to one piece of information)
 */
public void getFingerprint() {
        String userId = "1011";

        // Type of information operations
        // Get the information data
        int emtype =
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE.NET_EM_ACCESS_CTL_FINGERPRINT_SERVI
CE_GET;

        /**
         * Input parameters
         */
        NET_IN_ACCESS_FINGERPRINT_SERVICE_GET stIn = new
NET_IN_ACCESS_FINGERPRINT_SERVICE_GET();
        // User ID
        System.arraycopy(userId.getBytes(), 0, stIn.szUserID, 0,
                userId.getBytes().length);
```

```
        /**
         * Output parameters
         */
        NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET stOut = new
NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET();
        // The maximum length of the cache that receives information data
        stOut.nMaxFingerDataLength = 1024;

        stOut.pbyFingerData = new Memory(1024);
        stOut.pbyFingerData.clear(1024);

        stIn.write();
        stOut.write();
        if (netsdkApi.CLIENT_OperateAccessFingerprintService(loginHandle,
                    emtype, stIn.getPointer(), stOut.getPointer(), TIME_OUT)) {
            // Get the specific information at this point.
            stIn.read();
            stOut.read();

            byte[] buffer = stOut.pbyFingerData.getByteArray(0,
                        stOut.nRetFingerDataLength);

            // Convert the obtained information into strings without garbled characters
            String figerStr = Base64Util.getEncoder().encodeToString(buffer);

            System.out.println("Obtained information data:" + figerStr);
        } else {
            System.err.println("Failed to get the information, " + getErrorCode());
        }
}

/**
 * Delete the information by user ID
 */
public void deleteFingerprint() {
        String userID = "1011";

        // Initialize
        FAIL_CODE failCode = new FAIL_CODE();
```

```
    // Type of information operations
    // Delete the information data
    int emtype =
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE.NET_EM_ACCESS_CTL_FINGERPRINT_SERVI
CE_REMOVE;

    /**
     * Input parameterts
     */
    NET_IN_ACCESS_FINGERPRINT_SERVICE_REMOVE stIn = new
NET_IN_ACCESS_FINGERPRINT_SERVICE_REMOVE();
    stIn.nUserNum = 1;
    System.arraycopy(userID.getBytes(), 0, stIn.szUserIDs[0].szUserID, 0,
            userID.getBytes().length);

    /**
     * Output parameters
     */
    NET_OUT_ACCESS_FINGERPRINT_SERVICE_REMOVE stOut = new
NET_OUT_ACCESS_FINGERPRINT_SERVICE_REMOVE();
    stOut.nMaxRetNum = 1;

    stOut.pFailCode = failCode.getPointer();

    stIn.write();
    stOut.write();
    if (netsdkApi.CLIENT_OperateAccessFingerprintService(loginHandle,
            emtype, stIn.getPointer(), stOut.getPointer(), TIME_OUT)) {

        // Print the specific information
        System.out.println("Result of deleting the information: " + failCode.nFailCode);

    } else {
        System.err.println("Failed to delete the information, " + getErrorCode());
    }

    stIn.read();
    stOut.read();
}
```

```
/**
 * Clear all information
 */
public void clearFingerprint() {
	// Type of information operations
	// Clear the information data
	int emtype =
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE.NET_EM_ACCESS_CTL_FINGERPRINT_SERVI
CE_CLEAR;

	/**
	 * Input parameters
	 */
	NET_IN_ACCESS_FINGERPRINT_SERVICE_CLEAR stIn = new
NET_IN_ACCESS_FINGERPRINT_SERVICE_CLEAR();

	/**
	 * Output parameters
	 */
	NET_OUT_ACCESS_FINGERPRINT_SERVICE_CLEAR stOut = new
NET_OUT_ACCESS_FINGERPRINT_SERVICE_CLEAR();

	stIn.write();
	stOut.write();
	if (netsdkApi.CLIENT_OperateAccessFingerprintService(loginHandle,
			emtype, stIn.getPointer(), stOut.getPointer(), TIME_OUT)) {
		System.out.println("Successfully cleared the information.");
	} else {
		System.err.println("Failed to clear the information, " + getErrorCode());
	}

	stIn.read();
	stOut.read();
}
```

2.3.8 Door Config

See "2.3.8 Door Config"

2.3.9 Door Time Config

2.3.9.1 Period Config

See "2.2.9.1 Period Config."

2.3.9.2 Always Open and Always Closed Period Config

See "2.2.9.2 Always Open and Always Closed Period Config"

2.3.9.3 Holiday Group

2.3.9.3.1 Introduction

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

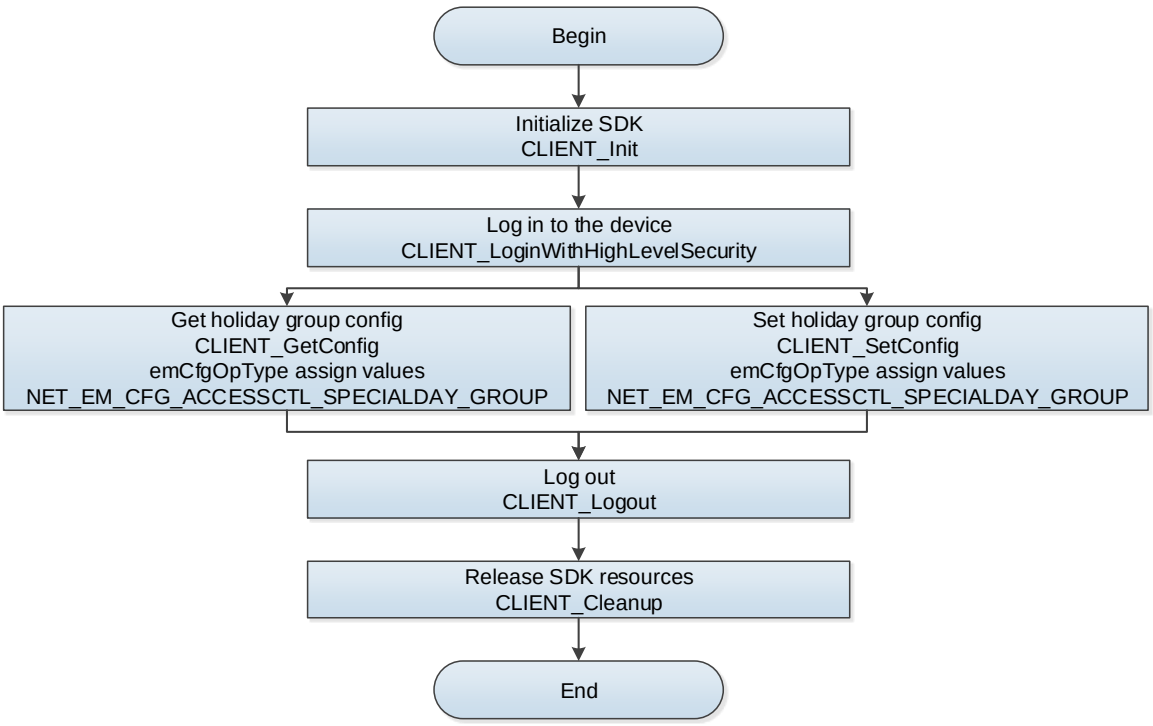
2.3.9.3.2 Interface Overview

Table 2-41 Description of holiday group interface

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

2.3.9.3.3 Process Description

Figure 2-29 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 2-42 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 2-43 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.

2.3.9.3.4 Example Code

```
/**
 * Send holiday group configuration
 *
 * @return Whether the configuration was sent successfully
 */
public boolean setSpecialDayGroup() {
    NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO config = new
NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO();
    config.bGroupEnable = true;
    config.nSpeciaday = 2;
    // byte[]Use System.arraycopy() to assign the values
    System.arraycopy("test".getBytes(), 0, config.szGroupName, 0, "test".getBytes().length);
    System.arraycopy(
            "test1".getBytes(), 0, config.stuSpeciaday[0].szDayName, 0, "test1".getBytes().length);
    config.stuSpeciaday[0].stuStartTime = new NET_TIME(2020, 10, 19, 0, 0, 0);
    config.stuSpeciaday[0].stuEndTime = new NET_TIME(2020, 10, 21, 23, 59, 59);
    config.stuSpeciaday[1].szDayName = "test2".getBytes();
    System.arraycopy(
            "test2".getBytes(), 0, config.stuSpeciaday[1].szDayName, 0, "test2".getBytes().length);
    config.stuSpeciaday[1].stuStartTime = new NET_TIME(2020, 10, 22, 10, 10, 10);
    config.stuSpeciaday[1].stuEndTime = new NET_TIME(2020, 10, 23, 12, 0, 0);
```

```
    return configModule.setConfig(
            loginHandler,
NET_EM_CFG_OPERATE_TYPE.NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP, config, 0);
}

/** Get the holiday group configuration */
public void getSpecialDayGroup() {
    NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO config = new
NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO();
    config =
            (NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO)
                configModule.getConfig(
                    loginHandler,

NET_EM_CFG_OPERATE_TYPE.NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP,
                    config,
                    0);
    if (config != null) {
        System.out.println(config.toString());
    }
}
```

2.3.9.4 Holiday Plan

2.3.9.4.1 Introduction

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

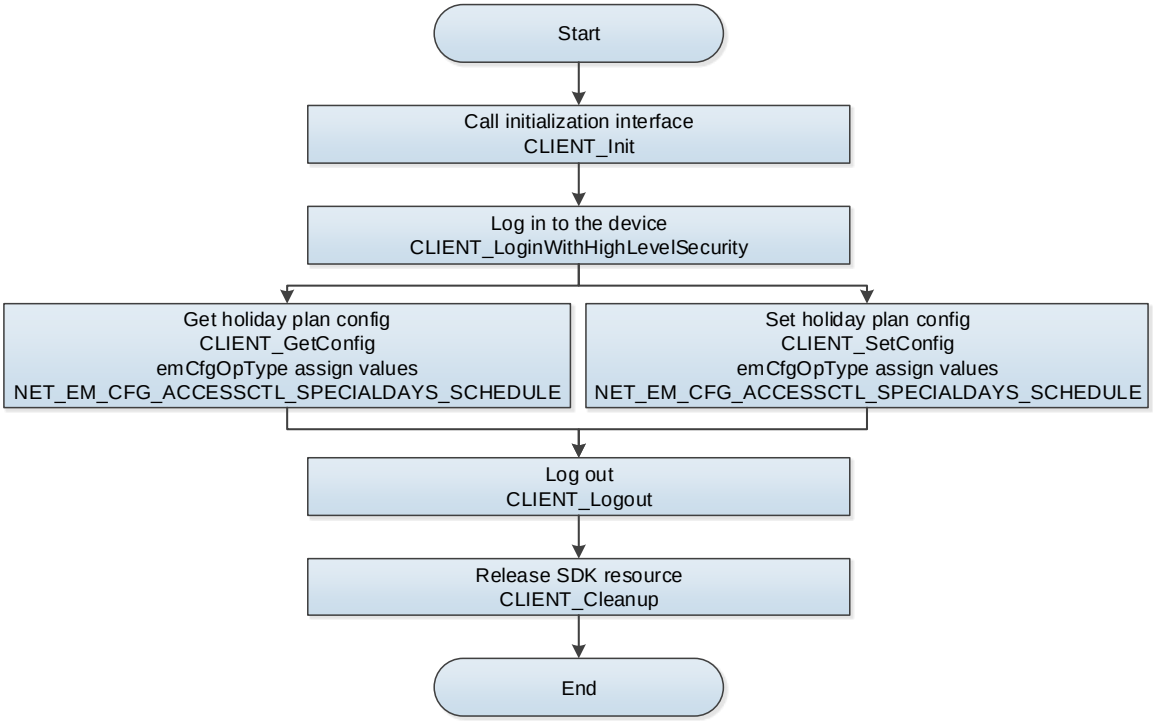
2.3.9.4.2 Interface Overview

Table 2-44 Description of holiday plan interface

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

2.3.9.4.3 Process Description

Figure 2-30 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 2-45 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 2-46 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.

2.3.9.4.4 Example Code

```
/** Send the holiday plan */
public void setSpecialDaysSchedule() {
   NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO config =
         new NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO();
   System.arraycopy(
         "scheduleName".getBytes(), 0, config.szSchduleName, 0, "scheduleName".getBytes().length);
   // Enable the plan
   config.bSchdule = true;
   // Holiday group subscript.    The subscript of stuSpeciaday in
NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO
   config.nGroupNo = 1;
   // Time period
   config.nTimeSection = 2;
   config.stuTimeSection[0].setTime(1, 5, 0, 0, 14, 0, 0);
   config.stuTimeSection[1].setTime(0, 15, 0, 0, 21, 0, 0);
   // Number of doors that take effect
   config.nDoorNum = 1;
   // Doors that take effect
   config.nDoors[0] = 1;
   configModule.setConfig(
         loginHandler,
NET_EM_CFG_OPERATE_TYPE.NET_EM_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE, config,
0);
}

/** Get the holiday plan */
public void getSpecialDaysSchedule() {
   NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO config =
         new NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO();
   config.nGroupNo = 1;
   config =
         (NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO)
               configModule.getConfig(
                     loginHandler,

NET_EM_CFG_OPERATE_TYPE.NET_EM_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE,
                     config,
                     0);
   if (config != null) {
      System.out.println(config.toString());
   }
}
```

2.3.10 Advanced Config of Door

See "2.3.10 Advanced Config of Door"

2.3.11 Records Query

2.3.11.1 Unlock Records

See "2.2.11.1 Unlock Records."

2.3.11.2 Device Log

See "2.2.11.2 Device log"

2.3.12 Access Control Event

See "2.2.12 Access Control Event".

2.3.13 Face-ID Comparison Event

See "2.2.13 Face-ID Comparison Event".

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	public boolean CLIENT_Init(Callback cbDisConnect, Pointer 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	public 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.	
Function	public void CLIENT_SetAutoReconnect(Callback cbAutoConnect, Pointer dwUser);	
Parameter	[in]cbAutoConnect	Reconnection callback.
	[in]dwUser	User parameters for reconnection callback.
Return Value	None.	

Item	Description
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	public 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 Login

3.1.2.1 Logging in to the Device CLIENT_LoginWithHighLevelSecurity

Table 3-5 Description of user logging in to the device

Item	Description	
Description	Log in to the device.	
Function	Public 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 3-6 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.

Error code	Corresponding meanings
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.
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.

3.1.2.2 User Logging Out of the Device CLIENT_Logout

Table 3-7 Description of user logging out of the device

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

3.1.3 Device Control

3.1.3.1 Device Controlling CLIENT_ControlDeviceEx

Table 3-8 Device control description

Item	Description	
Description	Device control.	
Function	public boolean CLIENT_ControlDeviceEx(LLong lLoginID, int emType, Pointer pInBuf, Pointer pOutBuf, int nWaitTime);	
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 3-9 Comparison of emType, pInBuf and pOutBuf

emType	Description	pInBuf	pOutBuf
CTRLTYPE_CTRL_ARMED_EX	Arming and Disarming	CTRL_ARM_DISARM_PARAM	NULL
CTRLTYPE_CTRL_SET_BYPASS	Bypass setting function	NET_CTRL_SET_BYPASS	NULL
CTRLTYPE_CTRL_ACCESS_OPEN	Access control--open	NET_CTRL_ACCESS_OPEN	NULL
CTRLTYPE_CTRL_ACCESS_CLOSE	Access control--close	NET_CTRL_ACCESS_CLOSE	NULL

3.2 Intelligent Subscription

3.2.1 Starting Subscription for Intelligent Events CLIENT_RealLoadPictureEx

Table 3-10 Description of starting subscription for intelligent events

Appendix 1 Item	Appendix 2 Description	
Name	Start subscription for intelligent events	
Function	public LLong CLIENT_RealLoadPictureEx(LLong lLoginID, int nChannelID, int dwAlarmType, int bNeedPicFile, StdCallCallback cbAnalyzerData, Pointer dwUser, Pointer Reserved);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in]nChannelID	Device channel number, starting from 0.
	[in]dwAlarmType	Type of subscribed alarm event.
	[in]bNeedPicFile	Subscribe to image files or not.
	[in]cbAnalyzerData	Intelligent event callback function.
	[in]dwUser	User-defined data type.
	[in]Reserved	Reserved field.
Return Value	Returns a LLONG type subscription handle on success; returns 0 on failure.	
Note	If the interface does not respond, use CLIENT_GetLastError to obtain the error code.	

- For the intelligent alarm event type, see the table below.

Table 3-11 Description of intelligent event type

Appendix 3 dwAlarmType Macro Definition	Appendix 4 Value of Macro Definition	Appendix 5 Description	Appendix 6 Structure Corresponding to Callback pAlarmInfo
EVENT_IVS_ALL	0x00000001	All events.	None.
EVENT_IVS_CROSSFENCEDETECTION	0x0000011F	Crossing fence.	DEV_EVENT_CROSSFENCEDETECTION_INFO
EVENT_IVS_CROSSLINEDETECTION	0x00000002	Tripwire.	DEV_EVENT_CROSSLINE_INFO
EVENT_IVS_CROSSREGIONDETECTION	0x00000003	Intrusion.	DEV_EVENT_CROSSREGION_INFO
EVENT_IVS_LEFTDETECTION	0x00000005	Abandoned object.	DEV_EVENT_LEFT_INFO
EVENT_IVS_PRESERVATION	0x00000008	Object preservation.	DEV_EVENT_PRESERVATION_INFO
EVENT_IVS_TAKENAWAYDETECTION	0x00000115	Missing object.	DEV_EVENT_TAKENAWAYDETECTION_INFO
EVENT_IVS_WANDERDETECTION	0x00000007	Loitering event.	DEV_EVENT_WANDER_INFO
EVENT_IVS_VIDEOABNORMALDETECTION	0x00000013	Video exception.	DEV_EVENT_VIDEOABNORMALDETECTION_INFO
EVENT_IVS_AUDIO_ABNORMALDETECTION	0x00000126	Abnormal audio.	DEV_EVENT_IVS_AUDIO_ABNORMALDETECTION_INFO
EVENT_IVS_CLIMBDETECTION	0x00000128	Climbing detection.	DEV_EVENT_IVS_CLIMB_INFO
EVENT_IVS_FIGHTDETECTION	0x0000000E	Fight detection.	DEV_EVENT_FLOWSTAT_INFO
EVENT_IVS_LEAVEDETECTION	0x00000129	Off-duty detection.	DEV_EVENT_IVS_LEAVE_INFO
EVENT_IVS_PSRISEDETECTION	0x0000011E	Getting up detection	DEV_EVENT_PSRISEDETECTION_INFO
EVENT_IVS_PASTEDETECTION	0x00000004	Detection of illegal stickers.	DEV_EVENT_PASTE_INFO

3.2.2 Stopping Subscription for Intelligent Events CLIENT_StopLoadPic

Table 3-12 Description of stopping subscription for intelligent events

Appendix 7 Item	Appendix 8 Description	
Name	Stop subscribing to intelligent event	
Function	public boolean CLIENT_StopLoadPic(LLong lAnalyzerHandle);	
Parameter	[in]lAnalyzerHandle	Intelligent event subscription handle.
Return Value	BOOL Type ● Success: TRUE ● Failure: FALSE	
Note	If the interface does not respond, use CLIENT_GetLastError to obtain the error code.	

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

3.3.1 Access Control

For details of the door control interface, see "3.1.3.1 Device Controlling CLIENT_ControlDeviceEx."

For details of the door sensor status interface, see "3.3.2.4 Querying Device Status CLIENT_QueryDevState."

3.3.2 Viewing Device Information

3.3.2.1 Querying System Capability Information CLIENT_QueryNewSystemInfo

Table 3-13 Description of querying system capability information

Item	Description
Description	Query system capability information in string format.

Item	Description	
Function	public boolean CLIENT_QueryNewSystemInfo(LLong lLoginID, String szCommand, int nChannelID, byte[] szOutBuffer, int dwOutBufferSize, IntByReference error, int waittime);	
Parameter	[in]lLoginID	Return value of CLIENT_Login or CLIENT_LoginEx.
	[in] szCommand	Command parameter. See "3.3.2.2 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 3-14 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

3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData

Table 3-15 Description of parsing the queried config information

Item	Description
Description	Parse the queried config information.
Function	public boolean CLIENT_ParseData(String szCommand, byte[] szInBuffer, Pointer lpOutBuffer, int dwOutBufferSize, Pointer pReserved);

Item	Description	
Parameter	[in] szCommand	Command parameter. See Table 3-16 for details.
	[in] szInBuffer	Input buffer, character config buffer.
	[out] lpOutBuffer	Output buffer. For structure types, see Table 3-16.
	[in] dwOutBufferSize	Output buffer size.
	[in] pReserved	Reserved parameter.
Return Value	● Success: TRUE, ● Failure: FALSE	
Note	None.	

Table 3-16 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

3.3.2.3 Getting Device Capabilities CLIENT_ GetDevCaps

Table 3-17 Description of getting device capabilities

Item	Description
Description	Get device capabilities.
Function	public boolean CLIENT_GetDevCaps(LLong lLoginID, int nType, Pointer pInBuf, Pointer pOutBuf, int nWaitTime);

Item	Description	
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 3-18 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

3.3.2.4 Querying Device Status CLIENT_QueryDevState

Table 3-19 Description of querying device status

Item	Description	
Description	Get the current working status of the front-end device.	
Function	public boolean CLIENT_QueryDevState(LLong lLoginID, int nType, Pointer pBuf, int nBufLen, IntByReference pRetLen, int waittime);	
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 3-20 Correspondence between nType, query type and structure

nType	Description	pBuf
NET_DEVSTATE_SOFTWARE	Query device software version information	NETDEV_VERSION_INFO
NET_DEVSTATE_NETINTERFACE	Query network port information	NETDEV_NETINTERFACE_INFO
NET_DEVSTATE_DEV_RECORDSET	Query device record set information	NET_CTRL_RECORDSET_PARAM
NET_DEVSTATE_DOOR_STATE	Query access control status (door sensor)	NET_DOOR_STATUS_INFO

3.3.3 Network Setting

3.3.3.1 IP Settings

3.3.3.1.1 Parsing Config Information CLIENT_GetNewDevConfig

For details about CLIENT_ParseData, see "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.3.1.2 Querying Config Information CLIENT_GetNewDevConfig

Table 3-21 Description of querying config information

Item	Description	
Description	Get config in string format.	
Function	public boolean CLIENT_GetNewDevConfig(LLong lLoginID, String szCommand, int nChannelID, byte[] szOutBuffer, int dwOutBufferSize, IntByReference error, int waiitime, Pointer pReserved);	
Parameter	[in] lLoginID	Login handle, return value of CLIENT_LoginWithHighLevelSecurity.
	[in] szCommand	Command parameter. See "3.3.2.2 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	

Item	Description
Note	Get config in string format, and information contained in each string is parsed by CLIENT_ParseData.

Table 3-22 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

3.3.3.1.3 Setting Config Information CLIENT_SetNewDevConfig

Table 3-23 Description of setting config information

Item	Description	
Description	Get config in string format.	
Function	public boolean CLIENT_SetNewDevConfig(LLong lLoginID, String szCommand, int nChannelID, byte[] szInBuffer, int dwInBufferSize, IntByReference error, IntByReference restart, int waittime);	
Parameter	[in] lLoginID	Login handle, return value of CLIENT_LoginWithHighLevelSecurity.
	[in] szCommand	Command parameter information. See "3.3.2.2 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 3-24 Description of error codes and meanings of the parameter error

Error code	Corresponding meanings
0	Successful
1	Failed

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

3.3.3.1.4 Packing into String Format CLIENT_PacketData

Table 3-25 Description of packing into string format

Item	Description	
Description	Pack the config information to be set into the string format.	
Function	public boolean CLIENT_PacketData(String szCommand, Pointer lpInBuffer, int dwInBufferSize, byte[] szOutBuffer, int dwOutBufferSize);	
Parameter	[in] szCommand	Command parameter. See "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData" for details.
	[in] lpInBuffer	Input buffer. For structure types, see "3.3.2.2 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.	

3.3.3.2 Auto Register Config

3.3.3.2.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.3.2.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.3.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.3.2.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.3.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.3.2.4 Packing into String Format CLIENT_PacketData

See "3.3.3.1.4 Packing into String Format CLIENT_PacketData."

3.3.4 Time Settings

3.3.4.1 Time Settings

Table 3-26 Description of time settings

Item	Description	
Description	Set the current time of the device.	
Function	public boolean CLIENT_SetupDeviceTime(LLong lLoginID, NET_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.	

3.3.4.2 NTP Time Sync, Time Zone Config

3.3.4.2.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.4.2.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.3.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.4.2.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.3.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.4.2.4 Packing into String Format CLIENT_PacketData

See "3.3.3.1.4 Packing into String Format CLIENT_PacketData."

3.3.4.3 DST Setting

3.3.4.3.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.4.3.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.3.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.4.3.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.3.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.4.3.4 Packing into String Format CLIENT_PacketData

See "3.3.3.1.4 Packing into String Format CLIENT_PacketData."

3.3.5 Personnel Management

3.3.5.1 Collection of Personnel Information Fields

See "3.3.2.4 Querying Device Status CLIENT_QueryDevState."

3.3.6 Door Config

3.3.6.1 Door Config Information

3.3.6.1.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.6.1.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.3.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.6.1.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.3.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.6.1.4 Packing into String Format CLIENT_PacketData

See "3.3.3.1.4 Packing into String Format CLIENT_PacketData."

3.3.7 Door Time Config

3.3.7.1 Period Config

3.3.7.1.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.7.1.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.3.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.7.1.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.3.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.7.1.4 Packing into String Format CLIENT_PacketData

See "3.3.3.1.4 Packing into String Format CLIENT_PacketData."

3.3.7.2 Always Open and Always Closed Period Config

3.3.7.2.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.7.2.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.3.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.7.2.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.3.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.7.2.4 Packing into String Format CLIENT_PacketData

See "3.3.3.1.4 Packing into String Format CLIENT_PacketData."

3.3.8 Advanced Config of Door

3.3.8.1 Unlock at Designated Intervals and First Card Unlock

3.3.8.1.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.8.1.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.3.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.8.1.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.3.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.8.1.4 Packing into String Format CLIENT_PacketData

See "3.3.3.1.4 Packing into String Format CLIENT_PacketData."

3.3.8.2 Combination Unlock by Multiple Persons

3.3.8.2.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.8.2.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.3.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.8.2.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.3.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.8.2.4 Packing into String Format CLIENT_PacketData

See "3.3.3.1.4 Packing into String Format CLIENT_PacketData."

3.3.8.3 Inter-door Lock

3.3.8.3.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.8.3.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.3.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.8.3.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.3.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.8.3.4 Packing into String Format CLIENT_PacketData

See "3.3.3.1.4 Packing into String Format CLIENT_PacketData."

3.3.8.4 Device Log

3.3.8.4.1 Querying the Count of Device Logs CLIENT_QueryDevLogCount

Table 3-27 Description of querying the count of device logs

Item	Description
Description	Query the count of device logs.

Item	Description	
Function	public boolean CLIENT_QueryDevLogCount(LLong lLoginID, Pointer pInParam, Pointer 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.
	[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.	

3.3.8.4.2 Starting Querying Logs CLIENT_StartQueryLog

Table 3-28 Description of starting querying logs

Item	Description	
Description	Start querying device logs.	
Function	public LLong CLIENT_StartQueryLog(LLong lLoginID, Pointer pInParam, Pointer 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.	

3.3.8.4.3 Getting Logs CLIENT_QueryNextLog

Table 3-29 Description of getting logs

Item	Description
Description	Get logs.

Item	Description	
Function	public boolean CLIENT_QueryNextLog(LLong lLogID, Pointer pInParam, Pointer 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.
Return Value	● Success: TRUE, ● Failure: FALSE	
Note	None.	

3.3.8.4.4 Ending Querying Logs CLIENT_StopQueryLog

Table 3-30 Description of ending querying logs

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

3.3.9 Records Query

3.3.9.1 Unlock Records

3.3.9.1.1 Querying Record Count CLIENT_QueryRecordCount

Table 3-31 Description of querying record count

Item	Description	
Description	Query the count of records.	
Function	public boolean 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.

Item	Description	
	[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.	

3.3.9.1.2 Querying Records by Query Conditions CLIENT_FindRecord

Table 3-32 Description of querying records by query conditions

Item	Description	
Description	Query records by query conditions.	
Function	public boolean CLIENT_FindRecord(LLong lLoginID, NET_IN_FIND_RECORD_PARAM pInParam, NET_OUT_FIND_RECORD_PARAM pOutParam, int waittime);	
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 3-33 Description of pInParam

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

3.3.9.1.3 Querying Records CLIENT_FindNextRecord

Table 3-34 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.

Item	Description	
Function	public boolean 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 3-35 Description of pOutParam

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

3.3.9.1.4 Ending Record Query CLIENT_FindRecordClose

Table 3-36 Description of ending record query

Item	Description	
Description	Stop record query.	
Function	public boolean 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 "3.1.3.1 Device Controlling CLIENT_ControlDeviceEx."

For details of the door contact status interface, see 3.3.2.4 Querying Device Status CLIENT_QueryDevState."

3.4.2 Viewing Device Information

3.4.2.1 Getting Device Capabilities CLIENT_QueryDevState

Table 3-37 Description of getting device capabilities

Item	Description	
Description	Get device capabilities.	
Function	public boolean CLIENT_GetDevCaps(LLong lLoginID, int nType, Pointer pInBuf, Pointer 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 3-38 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

3.4.2.2 Querying Device Status CLIENT_QueryDevState

For details about CLIENT_QueryDevState, see "3.3.2.4 Querying Device Status CLIENT_QueryDevState."

3.4.3 Network Setting

See "3.3.3 Network Setting."

3.4.4 Time Settings

See "3.3.4 Time Settings."

3.4.5 Personnel Management

3.4.5.1 User Management

3.4.5.1.1 User Information Management Interface for Access Control Devices CLIENT_OperateAccessUserService

Table 3-39 Description of user information management interface for access control devices

Item	Description	
Description	Personnel information management interface for access control devices.	
Function	public boolean CLIENT_OperateAccessUserService(LLong lLoginID, int emtype, Pointer pstInParam, Pointer 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 3-40 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

3.4.5.1.2 Starting to Find the Personnel Information CLIENT_StartFindUserInfo

Table 3-41 Description of starting to find the personnel information interface

Item	Description
Description	Starting to find the personnel information interface.

Item	Description	
Function	public 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	

3.4.5.1.3 Finding the Personnel Information Interface CLIENT_DoFindUserInfo

Table 3-42 Description of finding the personnel information interface

Item	Description	
Description	Finding the personnel information interface.	
Function	public boolean 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.	

3.4.5.1.4 Stopping Finding the Personnel Information Interface CLIENT_StartFindUserInfo

Table 3-43 Stopping finding the personnel information interface

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

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

3.4.5.2 Card Management

3.4.5.2.1 Card Information Management Interface for Access Control Devices CLIENT_OperateAccessCardService

Table 3-44 Description of card information management interface for access control devices

Item	Description	
Description	Card information management interface for access control devices.	
Function	public boolean CLIENT_OperateAccessCardService(LLong lLoginID, int emtype, Pointer pstInParam, Pointer 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 3-45 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

3.4.5.2.2 Starting to Find the Card Information Interface CLIENT_StartFindCardInfo

Table 3-46 Description of starting to find the card information interface

Item	Description
Description	Starting to find the card information interface.

Item	Description	
Function	public 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.	

3.4.5.2.3 Finding the Card Information Interface CLIENT_DoFindCardInf

Table 3-47 Description of finding the card information interface

Item	Description	
Description	Finding the card information interface.	
Function	public boolean 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.	

3.4.5.2.4 Stopping Finding the Card Information Interface CLIENT_StopFindUserInfo

Table 3-48 Description of stopping finding the card information interface

Item	Description	
Description	Stopping finding the card information interface.	
Function	public boolean CLIENT_StopFindCardInfo(LLong lFindHandle);	
Parameter	[in] lFindHandle	Return value of CLIENT_StartFindCardInf.
Return value	● Success: TRUE ● Failure: FALSE	

Item	Description
Description	None.

3.4.5.3 Face Management

3.4.5.3.1 Face Information Management Interface for Access Control Devices CLIENT_OperateAccessFaceService

Table 3-49 Description of face information management interface for access control devices

Item	Description	
Description	Face information management interface for access control devices.	
Function	public boolean CLIENT_OperateAccessFaceService(LLong lLoginID, int emtype, Pointer pstInParam, Pointer 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 3-50 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

3.4.5.4 Fingerprint Management

3.4.5.4.1 Fingerprint Information Management Interface for Access Control Devices CLIENT_OperateAccessFingerprintService

Table 3-51 Description of fingerprint information management interface for access control devices

Item	Description	
Description	Fingerprint information management interface for access control devices.	
Function	public boolean CLIENT_OperateAccessFingerprintService(LLong lLoginID, int emtype, Pointer pstInParam, Pointer 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 3-52 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

3.4.6 Door Config

3.4.6.1 Door Config Information

3.4.6.1.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData."

3.4.6.1.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.3.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.4.6.1.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.3.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.4.6.1.4 Packing into String Format CLIENT_PacketData

See "3.3.3.1.4 Packing into String Format CLIENT_PacketData."

3.4.7 Door Time Config

3.4.7.1 Period Config

3.4.7.1.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData."

3.4.7.1.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.3.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.4.7.1.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.3.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.4.7.1.4 Packing into String Format CLIENT_PacketData

See "3.3.3.1.4 Packing into String Format CLIENT_PacketData."

3.4.7.2 Always open and always closed period config

3.4.7.2.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.2.2 Parsing the Queried Config Information CLIENT_ParseData."

3.4.7.2.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.3.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.4.7.2.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.3.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.4.7.2.4 Packing into String Format CLIENT_PacketData

See "3.3.3.1.4 Packing into String Format CLIENT_PacketData."

3.4.7.3 Holiday group

3.4.7.3.1 Getting the Holiday Group Interface CLIENT_GetConfig

Table 3-53 Description of getting the holiday group interface

Item	Description	
Description	Getting the holiday group interface.	
Function	public boolean CLIENT_GetConfig(LLong lLoginID, int emCfgOpType, int nChannelID, Pointer szOutBuffer, int dwOutBufferSize, int waittime, Pointer reserve ●);	
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 3-54 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

3.4.7.3.2 Setting the Holiday Group Interface CLIENT_SetConfig

Table 3-55 Description of setting the holiday group interface

Item	Description	
Description	Setting the holiday group interface.	
Function	public boolean CLIENT_SetConfig(LLong lLoginID, int emCfgOpType, int nChannelID, Pointer szInBuffer, int dwInBufferSize, int waittime, IntByReference restart, Pointer reserve);	
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 3-56 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

3.4.7.4 Holiday plan

For details, see "3.4.7.3 Holiday group."

Table 3-57 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 3-58 Description of emCfgOpType

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

3.4.8 Advanced Config of Door

See "3.3.8 Advanced Config of Door."

3.4.9 Records Query

3.4.9.1 Unlock Records

See "3.3.9.1 Unlock Records."

3.4.9.2 Alarm Records

3.4.9.2.1 Querying Record Count CLIENT_QueryRecordCount

Table 3-59 Description of querying record count

Item	Description	
Description	Query the count of records.	
Function	public boolean 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.	

3.4.9.2.2 Querying Records by Query Conditions CLIENT_FindRecord

Table 3-60 Description of querying records by query conditions

Item	Description
Description	Query records by query conditions.

Item	Description	
Function	public boolean CLIENT_FindRecord(LLong lLoginID, NET_IN_FIND_RECORD_PARAM pInParam, NET_OUT_FIND_RECORD_PARAM pOutParam, int waittime);	
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 3-61 Description of pInParam

pInParam Structure	Value Assignment	Description
emType	NET_RECORD_ACCESS _ALARMRECORD	Query alarm records.

3.4.9.2.3 Querying Records CLIENT_FindNextRecord

Table 3-62 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	public boolean 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.	

Item	Description
Note	None.

Table 3-63 Description of pOutParam

pOutParam Structure	Value Assignment	Description
pRecordList	NET_RECORD_ACCESS_ALARMRECORD_INFO	Query alarm records.

3.4.9.2.4 Ending Record Query CLIENT_FindRecordClose

Table 3-64 Description of ending record query

Item	Description	
Description	Stop record query.	
Function	public boolean 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 Disconnection Callback fDisConnect

Table 4-1 Description of disconnecting callback function

Item	Description	
Description	Disconnection callback.	

Function	public interface fDisConnect extends StdCallCallback {
	public void invoke(
	LLong lLoginID,
	String pchDVRIP, int nDVRPort,
	Pointer 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.	

4.2 Reconnection Callback fHaveReConnect

Table 4-2 Description of reconnecting callback function

Item	Description	
Description	Reconnection callback.	

Function	public interface fHaveReConnect extends StdCallCallback {
	public void invoke(
	LLong lLoginID,
	String pchDVRIP, int nDVRPort,
	Pointer 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.	

4.3 Callback for Real-time Monitoring Data fRealDataCallBackEx2

Table 4-3 Description of callback function for real-time monitoring data

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

Function	public interface fRealDataCallBackEx2 extends SDKCallback{
	void invoke(
	LLong lRealHandle,
	int dwDataType,
	Pointer pBuffer,
	int dwBufSize,
	LLong param,
	Pointer 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.	

4.4 Audio Data Callback pfAudioDataCallBack

Table 4-4 Description of audio data callback function

Item	Description
Description	Audio data callback for voice talk.

Item	Description	

Function	public interface pfAudioDataCallBack extends StdCallCallback {
	public void invoke(
	LLong lTalkHandle,
	Pointer pDataBuf, int dwBufSize,
	byte byAudioFlag, Pointer dwUser);
	}	

Parameter	[out]lTalkHandle	Return value of CLIENT_StartTalkEx.
	[out]pDataBuf	Audio data block address.
	[out]dwBufSize	Length of audio data block, in bytes.
	[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.	

4.5 Alarm Callback fMessCallBack

Table 4-5 Description of alarm callback function

Item	Description	
Description	Alarm callback function.	

Function	public interface fMessCallBack extends SDKCallback{
	public boolean invoke(
	int lCommand,
	LLong lLoginID,
	Pointer pStuEvent,
	int dwBufLen,
	String strDeviceIP,
	NativeLong nDevicePort,
	Pointer dwUser);
	}	

Parameter	[out]lCommand	Alarm type. See Table 4-6 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	

Item	Description
	command values.

Table 4-6 Correspondence between alarm type and structure

Alarm business	Alarm type name	lCommand	pBuf
Alarm host	Local alarm event	NET_ALARM_ALARM_EX2	ALARM_ALARM_INFO_EX2
	Power failure event	NET_ALARM_POWERFAULT	ALARM_POWERFAULT_INFO
	Dismantlement prevention event	NET_ALARM_CHASSISINTRUDED	ALARM_CHASSISINTRUDED_INFO
	Extended alarm input channel event	NET_ALARM_ALARMEXTENDED	ALARM_ALARMEXTENDED_INFO
	Emergency event	NET_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	NET_ALARM_BATTERYLOWPOWER	ALARM_BATTERYLOWPOWER_INFO
	Device inviting platform to talk event	NET_ALARM_TALKING_INVITE	ALARM_TALKING_INVITE_INFO
	Device arming mode change event	NET_ALARM_ARMMODE_CHANGE_EVENT	ALARM_ARMMODE_CHANGE_INFO
	Protection zone bypass status change event	NET_ALARM_BYPASSMODE_CHANGE_EVENT	ALARM_BYPASSMODE_CHANGE_INFO
	Alarm input source signal event	NET_ALARM_INPUT_SOURCE_SIGNAL	ALARM_INPUT_SOURCE_SIGNAL_INFO
	Alarm clearing event	NET_ALARM_ALARMCLEAR	ALARM_ALARMCLEAR_INFO
	Sub-system status change event	NET_ALARM_SUBSYSTEM_STATE_CHANGE	ALARM_SUBSYSTEM_STATE_CHANGE_INFO
	Extension module offline event	NET_ALARM_MODULE_LOST	ALARM_MODULE_LOST_INFO

Alarm business	Alarm type name	lCommand	pBuf
	PSTN offline event	NET_ALARM_PSTN_BREAK_LINE	ALARM_PSTN_BREAK_LINE_INFO
	Analog quantity alarm event	NET_ALARM_ANALOG_PULSE	ALARM_ANALOGPULSE_INFO
	Alarm transmission event	NET_ALARM_PROFILE_ALARM_TRANSMIT	ALARM_PROFILE_ALARM_TRANSMIT_INFO
	Wireless device low battery alarm event	NET_ALARM_WIRELESSDEV_LOWPOWER	ALARM_WIRELESSDEV_LOWPOWER_INFO
	Protection zone arming and disarming status change event	NET_ALARM_DEFENCE_ARMMODE_CHANGE	ALARM_DEFENCE_ARMMODECHANGE_INFO
	Sub-system arming and disarming status change event	NET_ALARM_SUBSYSTEM_ARMMODE_CHANGE	ALARM_SUBSYSTEM_ARMMODECHANGE_INFO
	Detector abnormality alarm	NET_ALARM_SENSOR_ABNORMAL	ALARM_SENSOR_ABNORMAL_INFO
	Patient activity status alarm event	NET_ALARM_PATIENTDETECTION	ALARM_PATIENTDETECTION_INFO
Access Control	Access control event	NET_ALARM_ACCESS_CTL_EVENT	ALARM_ACCESS_CTL_EVENT_INFO
	Details of access control unlocking event	NET_ALARM_ACCESS_CTL_NOT_CLOSE	ALARM_ACCESS_CTL_NOT_CLOSE_INFO
	Details of intrusion event	NET_ALARM_ACCESS_CTL_BREAK_IN	ALARM_ACCESS_CTL_BREAK_IN_INFO
	Details of repeated entry event	NET_ALARM_ACCESS_CTL_REPEAT_ENTER	ALARM_ACCESS_CTL_REPEAT_ENTER_INFO
	Malicious unlocking event	NET_ALARM_ACCESS_CTL_MALICIOUS	ALARM_ACCESS_CTL_MALICIOUS

Alarm business	Alarm type name	lCommand	pBuf
	Details of forced card swiping event	NET_ALARM_ACCESS_CTL_DURESS	ALARM_ACCESS_CTL_DURESS_INFO
	Combination unlocking by multiple persons event	NET_ALARM_OPENDOORGROUP	ALARM_OPEN_DOOR_GROUP_INFO
	Dismantlement prevention event	NET_ALARM_CHASSISINTRUDED	ALARM_CHASSISINTRUDED_INFO
	Local alarm event	NET_ALARM_ALARM_EX2	ALARM_ALARM_INFO_EX2
	Access control status event	NET_ALARM_ACCESS_CTL_STATUS	ALARM_ACCESS_CTL_STATUS_INFO
	Bolt alarm	NET_ALARM_ACCESS_CTL_STATUS	ALARM_ACCESS_CTL_STATUS_INFO
	Fingerprint acquisition event	NET_ALARM_FINGER_PRINT	ALARM_CAPTURE_FINGER_PRINT_INFO
Video Intercom	No response to the call in direct connection event	NET_ALARM_CALL_NO_ANSWERED	NET_ALARM_CALL_NO_ANSWERED_INFO
	Mobile phone number report event	NET_ALARM_TELEPHONE_CHECK	ALARM_TELEPHONE_CHECK_INFO
	VTS status report	NET_ALARM_VTSTATE_UPDATE	ALARM_VTSTATE_UPDATE_INFO
	VTO object recognition	NET_ALARM_ACCESSIDENTIFY	NET_ALARM_ACCESSIDENTIFY
	Device inviting another device to start talk event	NET_ALARM_TALKING_INVITE	ALARM_TALKING_INVITE_INFO
	Device canceling talk request event	NET_ALARM_TALKING_IGNORE_INVITE	ALARM_TALKING_IGNORE_INVITE_INFO

Alarm business	Alarm type name	lCommand	pBuf
	Device actively hanging up talk event	NET_ALARM_TALKING_HANGUP	ALARM_TALKING_HANGUP_INFO
	Radar monitoring overspeed alarm event	NET_ALARM_RADAR_HIGH_SPEED	ALARM_RADAR_HIGH_SPEED_INFO

4.6 Upgrade Progress Callback fUpgradeCallBackEx

Table 4-7 Description of upgrade progress callback function

Item	Description	
Description	Upgrade progress callback function.	

Function	public interface fUpgradeCallBackEx extends SDKCallback {
	public void invoke(
	LLong lLoginID,
	LLong lUpgradechannel,
	int nTotalSize,
	int nSendSize,
	Pointer dwUserData);
	}	

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.	

4.7 Intelligent Event Callback fAnalyzerDataCallBack

Table 4-8 Intelligent event callback fAnalyzerDataCallBack

Appendix 9 Item	Appendix 10 Description	
Name	Remote device status callback	

Function	public interface fAnalyzerDataCallBack extends Callback {
	public int invoke(LLong lAnalyzerHandle, int dwAlarmType, Pointer pAlarmInfo, Pointer pBuffer, int dwBufSize, Pointer dwUser, int nSequence, Pointer reserved);
	}	

Parameter	[out]lAnalyzerHandle	Return value of CLIENT_RealLoadPictureEx
	[out]dwEventType	Intelligent event type
	[out] pAlarmInfo	Event information cache
	[out]pBuffer	Image cache
	[out]dwBufSize	Image cache size
	[out]dwUser	User data
	[out]nSequence	Serial number
	[out]reserved	Reserved
Return Value	None.	
Note	After subscribing to the intelligent event of remote device, if an intelligent event is triggered, the camera will report relevant information of the event.	

Appendix 11 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.