

NetSDK__C# (Intelligent Building)

Programming Manual



Foreword

General

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

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

The manual describes the SDK interfaces and processes of the general function modules for intelligent buildings. For more function modules and data structures, refer to *NetSDK Development Manual*.






The sample code 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

Safety Instructions

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

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

Revision History

Version	Revision Content	Release Time
V1.0.0	First release.	October 2020

About the Manual

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

Glossary

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

Term	Description
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

Foreword	I
Glossary	III
1 Overview	1
1.1 General.....	1
1.2 Applicability	2
1.2.1 Supported System	2
1.2.2 Supported Devices	2
1.3 Application Scenarios.....	3
2 Main Functions	5
2.1 General.....	5
2.1.1 SDK Initialization.....	5
2.1.2 Device Initialization	7
2.1.3 Device Login	10
2.1.4 Realtime Monitor.....	12
2.1.5 Voice Talk.....	18
2.1.6 Sending Alarm	22
2.2 Access Controller/All-in-one Fingerprint Machine (First-generation).....	26
2.2.1 Access Control	26
2.2.2 Alarm Event	30
2.2.3 Viewing Device Information	39
2.2.4 Network Setting	46
2.2.5 Device Time Getting and Setting.....	51
2.2.6 Maintenance Config.....	58
2.2.7 Personnel Management	68
2.2.8 Door Config.....	87
2.2.9 Door Time Config.....	90
2.2.10 Advanced Config of Door.....	104
2.2.11 Records Query	123
2.3 Access Controller/All-in-one Face Machine (Second-Generation).....	132
2.3.1 Access Control	132
2.3.2 Alarm Event	132
2.3.3 Viewing Device Information	133
2.3.4 Network Setting	134
2.3.5 Setting the Device Time.....	134
2.3.6 Maintenance Config.....	135
2.3.7 Personnel Management	135
2.3.8 Door Config.....	156
2.3.9 Door Time Config.....	156
2.3.10 Advanced Config of Door.....	162
2.3.11 Records Query	162
3 Interface Function	166
3.1 Common Interface	166
3.1.1 SDK Initialization.....	166
3.1.2 Device Initialization	167

3.1.3 Device Login	168
3.1.4 Realtime Monitor.....	169
3.1.5 Device Control	172
3.1.6 Alarm Listening	173
3.1.7 Getting Device Status	174
3.1.8 Voice Talk.....	174
3.2 Access Controller/ All-in-one Fingerprint Machine (First-generation).....	177
3.2.1 Access Control.....	177
3.2.2 Alarm Event	177
3.2.3 Viewing Device Information	178
3.2.4 Network Setting	180
3.2.5 Time Settings	181
3.2.6 Maintenance Config.....	182
3.2.7 Personnel Management	187
3.2.8 Door Config.....	187
3.2.9 Door Time Config.....	188
3.2.10 Advanced Config of Door.....	188
3.2.11 Records Query	191
3.3 Access Controller/All-in-one Face Machine (Second-Generation).....	193
3.3.1 Access Control.....	193
3.3.2 Alarm Event	193
3.3.3 Viewing Device Information	193
3.3.4 Network Setting	194
3.3.5 Time Settings	194
3.3.6 Maintenance Config.....	194
3.3.7 Personnel Management	194
3.3.8 Door Config.....	203
3.3.9 Door Time Config.....	204
3.3.10 Advanced Config of Door.....	206
3.3.11 Records Query	206
4 Callback Function	207
4.1 Device Querying Callback fSearchDevicesCB	207
4.2 Device Querying Callback fSearchDevicesCBEx.....	207
4.3 Disconnection Callback ffDisConnectCallBack	207
4.4 Reconnection Callback fHaveReConnectCallBack	208
4.5 Callback for Real-time Monitoring Data fRealDataCallBackEx2	208
4.6 Audio Data Callback pfAudioDataCallBack	209
4.7 Alarm Callback fMessCallBackEx.....	209
4.8 Upgrade Progress Callback fUpgradeCallBackEx	213
Appendix 1 Cybersecurity Recommendations	215

1 Overview

1.1 General

This manual is about reference information of NetSDKCS, the packaging project of C# NetSDK library. The main content is main functions, interface functions, and callback functions.

Main functions include: NetSDK initialization, device login, real-time monitoring, audio talk, alarms reporting, and other functions.

- For files included by C# NetSDK library, see Table 1-1 and Table 1-2.

Table 1-1 Files included in Windows development kit

Library Type	Windows Library File Name	Linux Library File Name	Library Description
Function library	dhnetsdk.dll	libdhnetsdk.so	Lib file
	avnetsdk.dll	libavnetsdk.so	Lib file
Configuration library	dhconfigsdk.dll	libdhconfigsdk.so	Lib file
Play (encoding/decoding) auxiliary library	dhplay.dll	libdhplay.so	Play library
	fisheye.dll	Not included	Fisheye correction library
Dependent library of avnetsdk.dll	Infra.dll	libInfra.so	Infrastructure library
	json.dll	Not included	Json library
	NetFramework.dll	libNetFramework.so	Network infrastructure library
	Stream.dll	libStream.so	Media transmission structure package library
	StreamSvr.dll	libStreamSvr.so	Streaming service
dhnetsdk auxiliary library	lvsDrawer.dll	Not included	Image display library
	ImageAlg.dll	libImageAlg.so	Thermal image auxiliary library
	libcurl.dll	libcurl.so	Cloud update auxiliary library

- For files included by C# packaging project, see Table 1-2.

Table 1-2 Files included in NetSDKCS project

File Name	Description
NetSDK.cs	Packages C++ interfaces, provides callback C# interfaces.
NetSDKStruct.cs	Stores structural body enumerations.
OriginalSDK.cs	Introduces C interfaces in NetSDK library to C# projects.



- The function library and configuration library are required libraries.
- The function library is the main body of NetSDK, 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.

- NetSDK is the base of NetSDKCS project. The OriginalSDK.cs file defined the citing path of NetSDK library. In actual use, put the NetSDK library in the citing path, which can be customized.
- Customers can cite this packaging project in their own projects directly, or they can import files in the packaging project to their own projects, or they can package programs by referring to this packaging project.

1.2 Applicability

1.2.1 Supported System

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

1.2.2 Supported Devices



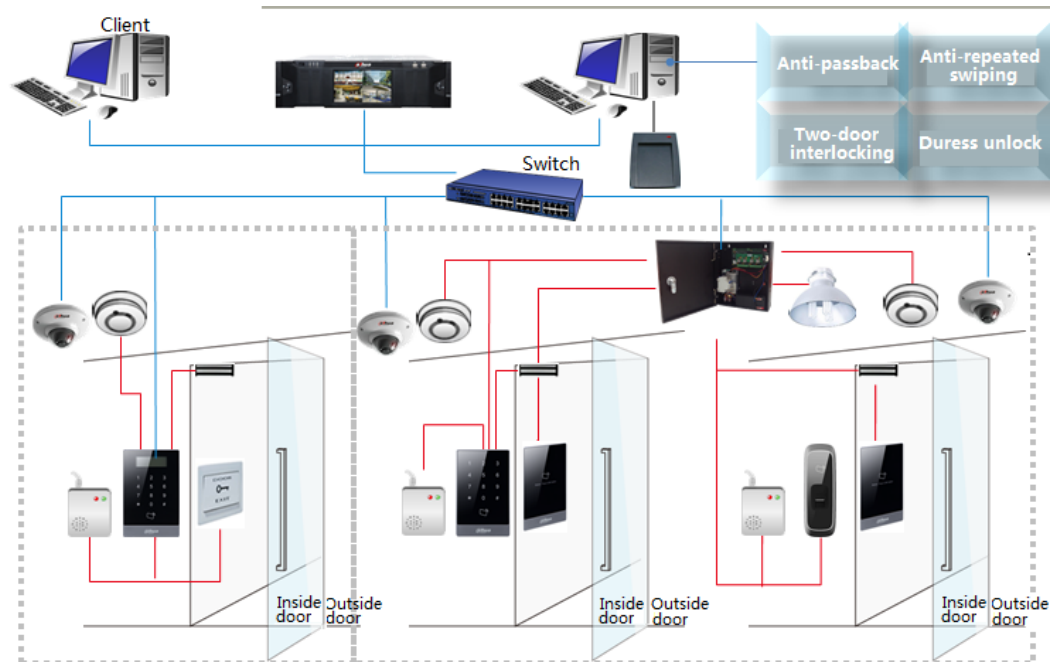
Not all models are listed here.

- Access Control (First-generation Devices)
 - ◇ ASC1201B-D, ASC1201C-D
 - ◇ ASC1202B-S, ASC1202B-D, ASC1202C-S, ASC1202C-D
 - ◇ ASC1204B-S, SC1204C-S, ASC1204C-D
 - ◇ ASC1208C-S
 - ◇ ASI1201A, ASI1201A-D, ASI1201E, ASI1201E-D
 - ◇ ASI1212A(V2), ASI1212A-D(V2), ASI1212D, ASI1212D-D
- Access Control (Second-generation Devices)
 - ◇ 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

1.3 Application Scenarios

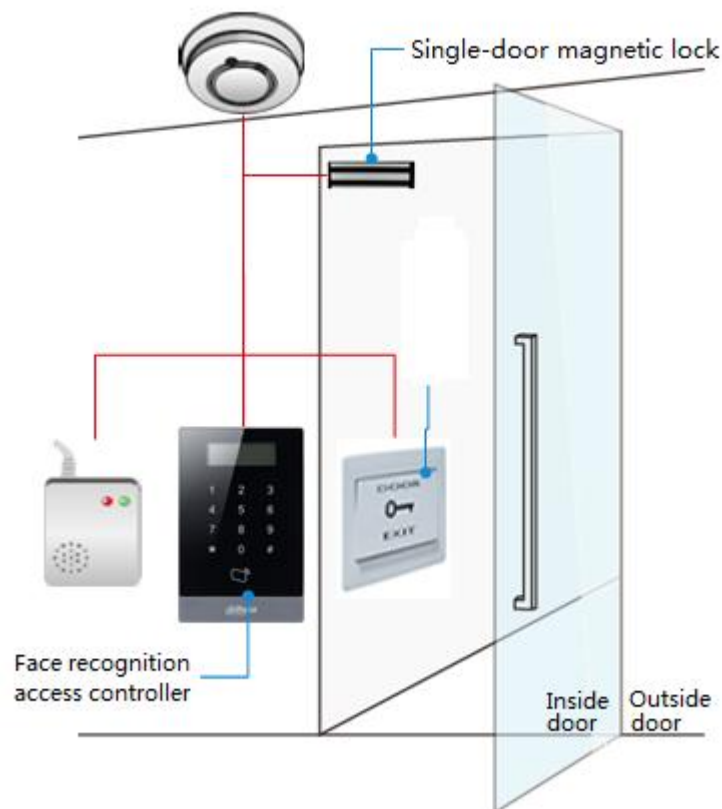
- Typical scenario.

Figure 1-1 Typical scenario



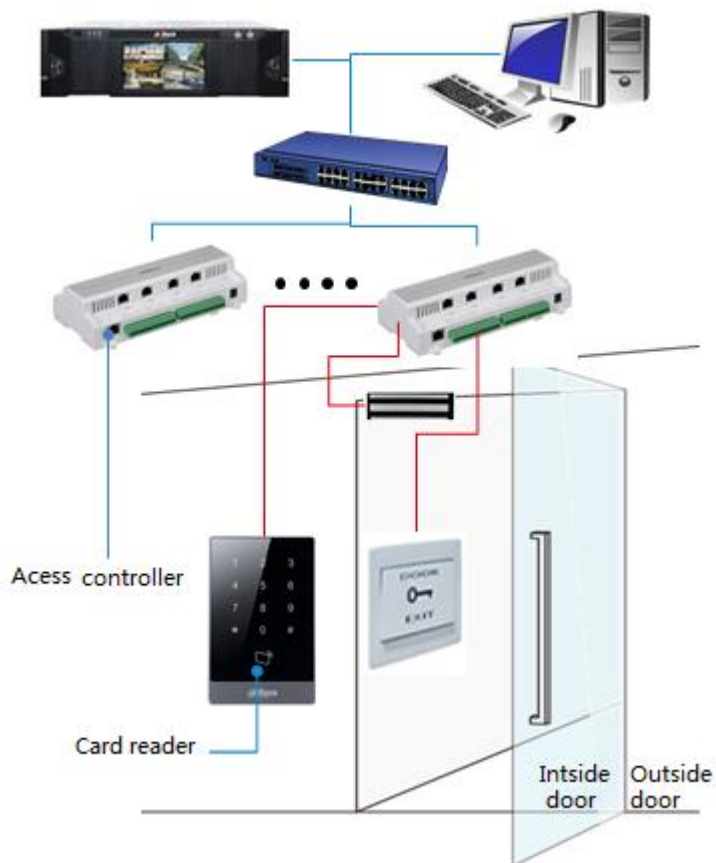
- Micro access control for small-sized office.

Figure 1-2 Micro access control



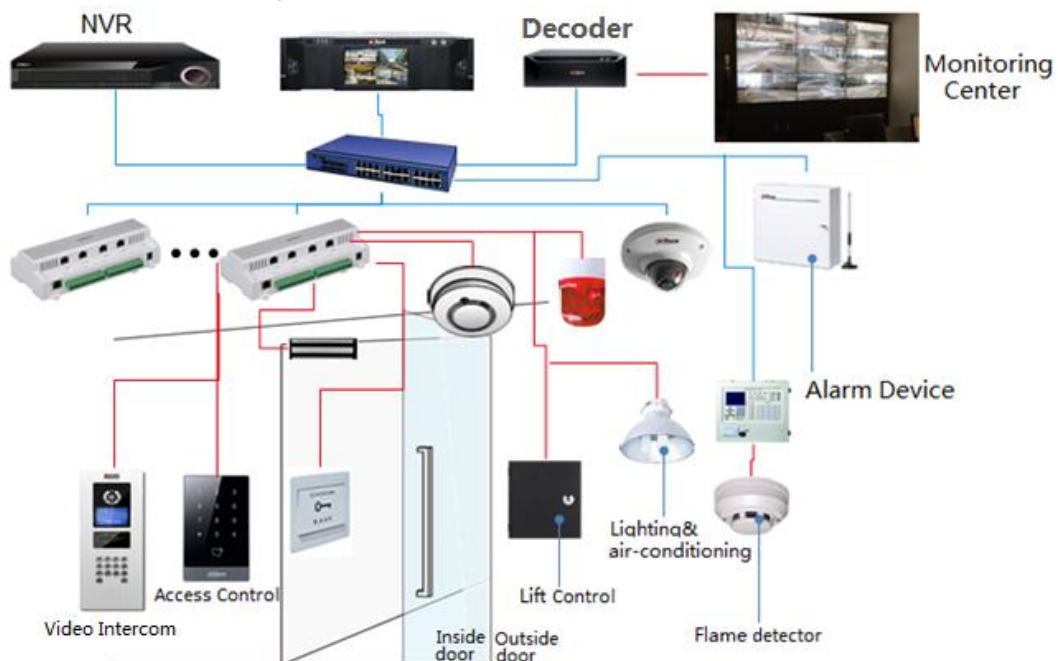
- Network access control for medium and small-sized intelligent building, treasury house and jail monitoring projects.

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 NETSDK cleaning interfaces to release resources.

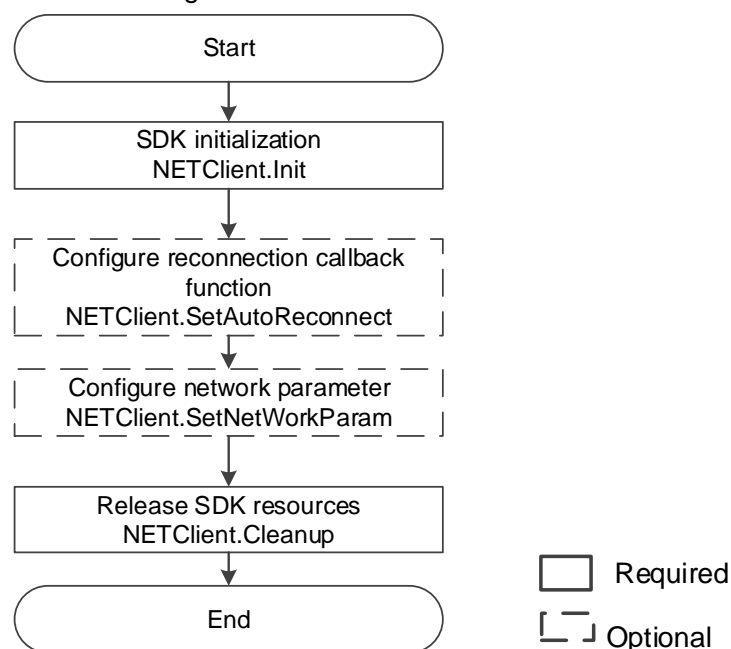
2.1.1.2 Interface Overview

Table 2-1 Description of SDK initialization interface

Interface	Description
NETClient.Init	NetSDK initialization interface.
NETClient.Cleanup	NetSDK cleaning interface.
NETClient.SetAutoReconnect	Configure reconnection callback interface.
NETClient.SetNetworkParam	Configure login network environment interface.

2.1.1.3 Process Description

Figure 2-1 SDK initialization



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 (Optional) Call **NETClient.SetAutoReconnect** to set reconnection callback to allow the auto reconnecting after disconnection within SDK.
- Step 3 (Optional) Call **NETClient.SetNetworkParam** to network login parameter that includes the timeout period for device login and the number of attempts.
- Step 4 After using all NetSDK functions, call **NETClient.Cleanup** to release NetSDK resources.

Note

- You need to call the interfaces **NETClient.Init** and **NETClient.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 **NETClient.Init** multiple times is only for internal count without repeating applying resources.
- Cleaning up: The interface **NETClient.Cleanup** clears all the opened processes, such as login, real-time monitoring, and alarm subscription.
- Reconnection: NetSDK can set the reconnection function for the situations such as network disconnection and power disconnection. NetSDK will keep logging in to the device until login succeeded. Only the real-time monitoring, playback, intelligent event subscription, and alarm function will be resumed after the reconnection.

2.1.1.4 Sample Code

```
// statement static call back entrusting (for common entrusting, releasing before callback might occur)
private static fDisconnectCallBack m_DisConnectCallBack;    //disconnection callback
private static fHaveReConnectCallBack m_ReConnectCallBack;  /reconnection callback

// realize entrusting
m_DisConnectCallBack = new fDisconnectCallBack(DisconnectCallBack);
m_ReConnectCallBack = new fHaveReConnectCallBack(ReConnectCallBack);

// initialize NetSDK, realize disconnection callback during initialization
bool result = NETClient.Init(m_DisConnectCallBack, IntPtr.Zero, null);
if (!result)
{
    MessageBox.Show(NETClient.GetLastError()); //display error information
    return;
}

//configure reconnection callback
NETClient.SetAutoReconnect(m_ReConnectCallBack, IntPtr.Zero);
```

```
//configure network parameter
NET_PARAM param = new NET_PARAM()
{
    nWaittime = 10000,// waiting timeout duration (ms)
    nConnectTime = 5000,// connection timeout (ms)
};
NETClient.SetNetworkParam(param);

// cleaning up initialization resource
NETClient.Cleanup();
```

2.1.2 Device Initialization

2.1.2.1 Introduction

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

- Uninitialized devices cannot be logged in.
- A password will be set up for the default admin account during initialization.
- You can reset the password if you forgot it.

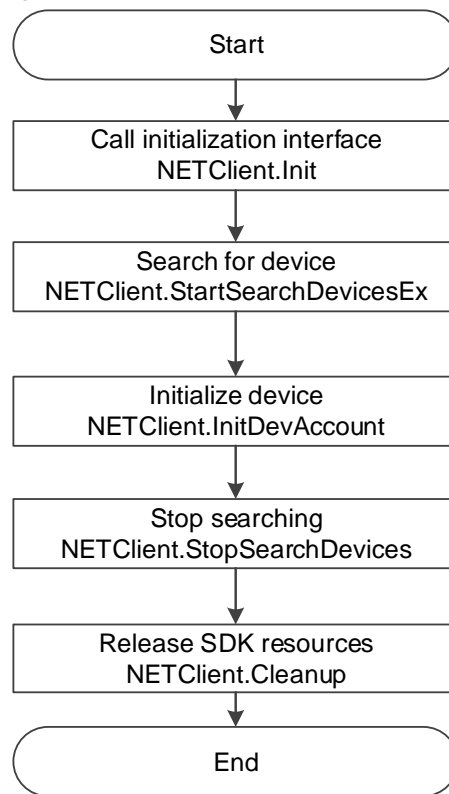
2.1.2.2 Interface Overview

Table 2-2 Description of device initialization interfaces

Interface	Description
NETClient.StartSearchDevicesEx	Search for devices in the LAN, and find the uninitialized devices.
NETClient.InitDevAccount	Device initialization interface.
NETClient.StopSearchDevices	Used to stop searching for devices.

2.1.2.3 Process Description

Figure 2-2 Device initialization



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.StartSearchDevicesEx** to search for the devices within the LAN and get the device information (multi-thread calling is not supported).
- Step 3 Call **NETClient.InitDevAccount** to initialize device.
- Step 4 Call **NETClient.StopSearchDevices** to stop searching.
- Step 5 After using all SDK functions, call **NETClient.Cleanup** to release SDK resources.

Note

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

2.1.2.4 Sample Code

```
Task task = new Task(() =>
{
    NET_IN_INIT_DEVICE_ACCOUNT inParam = new
NET_IN_INIT_DEVICE_ACCOUNT();
    inParam.dwSize =
(uint)Marshal.SizeOf(typeof(NET_IN_INIT_DEVICE_ACCOUNT));
```

```

        if (initDeviceDialog.IsEmail)
        {
            inParam.szMail = initDeviceDialog.RestStr;
        }
        else
        {
            inParam.szCellPhone = initDeviceDialog.RestStr;
        }
        inParam.szMac = selectInfo.stuDevInfo.szMac;
        inParam.szUserName = initDeviceDialog.UserName;
        if (initDeviceDialog.Passwrod.Length > 127)
        {
            string password = initDeviceDialog.Passwrod.Substring(0, 127);
            inParam.szPwd = password;
        }
        else
        {
            inParam.szPwd = initDeviceDialog.Passwrod;
        }
        inParam.byPwdResetWay = selectInfo.stuDevInfo.byPwdResetWay;
        NET_OUT_INIT_DEVICE_ACCOUNT outParam = new
NET_OUT_INIT_DEVICE_ACCOUNT();
        outParam.dwSize =
(uint)Marshal.SizeOf(typeof(NET_OUT_INIT_DEVICE_ACCOUNT));
        bool res = NETClient.InitDevAccount(inParam, ref outParam, 5000,
selectInfo.szLocallp);
        if (!res)
        {
            string errormsg = NETClient.GetLastError();
            this.BeginInvoke(new Action(() =>
            {
                MessageBox.Show(this, errormsg);
            }));
        }
        else
        {
            this.BeginInvoke(new Action(() =>
            {
                listView_device.SelectedItems[0].BackColor = Color.White;
                listView_device.SelectedItems[0].SubItems[1].Text = "Initialized";
            }));
        }
    }
}

```

```

        ));

        selectInfo.stuDevInfo.byInitStatus = 2;
    }
});

task.Start();

```

2.1.3 Device Login

2.1.3.1 Introduction

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

You can obtain a unique login ID upon logging in to the device and should use the login ID when using other NetSDK interfaces. The login ID becomes invalid once logged out of the device.

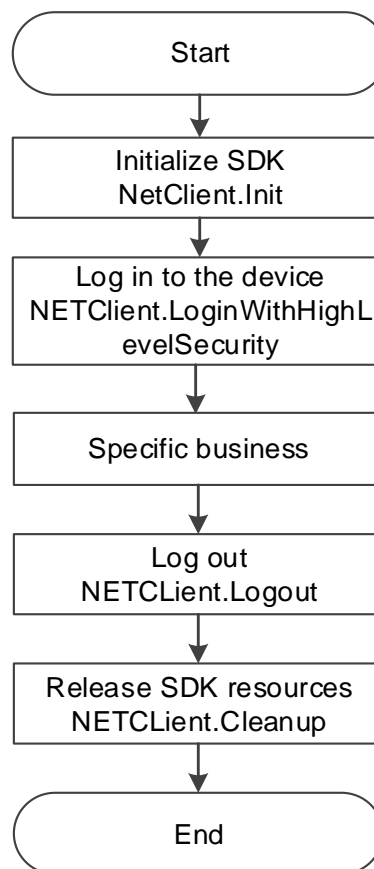
2.1.3.2 Interface Overview

Table 2-3 Description of device login interfaces

Interface	Description
NETClient.LoginWithHighLevelSecurity	Login interface.
NETClient.Logout	Logout interface.

2.1.3.3 Process Description

Figure 2-3 Login



Process

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.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 **NETClient.Logout** to log out of the device.
- Step 5 After using all SDK functions, call **NETClient.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.
- Duplicate handle: The login handle might be the same as the existing handle, and it is normal. For example, log in to device A, you will get loginIDA; log out loginIDA, and then log in to the device again, you will get LoginIDA again. However, in the life cycle of the handle, no duplicate handle will appear.
- 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.
- After the device is offline, the login ID of the device will be invalid, and the login ID will be valid again if the device is logged in again.

2.1.3.4 Sample Code

```
//log in to the device
NET_DEVICEINFO_Ex m_DeviceInfo = new NET_DEVICEINFO_Ex();
IntPtr m_LoginID = NETClient.LoginWithHighLevelSecurity(ip, port, name, password,
EM_LOGIN_SPAC_CAP_TYPE.TCP, IntPtr.Zero, ref m_DeviceInfo);
if (IntPtr.Zero == m_LoginID)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

// log out of the device
if (IntPtr.Zero != m_LoginID)
{
    bool result = NETClient.Logout(m_LoginID);
    if (!result)
    {
        MessageBox.Show(this, NETClient.GetLastError());
        return;
    }
}
m_LoginID = IntPtr.Zero;
}
```

2.1.4 Realtime Monitor

2.1.4.1 Introduction

Real-time monitoring obtains the real-time stream from the storage device or front-end device, which is an important part of the surveillance system.

SDK can get the main stream and sub stream from the device once logged in.

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

2.1.4.2 Interface Overview

Table 2-4 Description of real-time monitoring interfaces

Interface	Description
NETClient.RealPlay	Extension interface for starting the real-time monitoring.
NETClient.StopRealPlay	Extension interface for stopping the real-time monitoring.
NETClient.SaveRealData	Start saving the real-time monitoring data to the local path.
NETClient.StopSaveRealData	Stop saving the real-time monitoring data to the local path.
NETClient.SetRealDataCallBack	Extension interface for setting the real-time monitoring data callback.

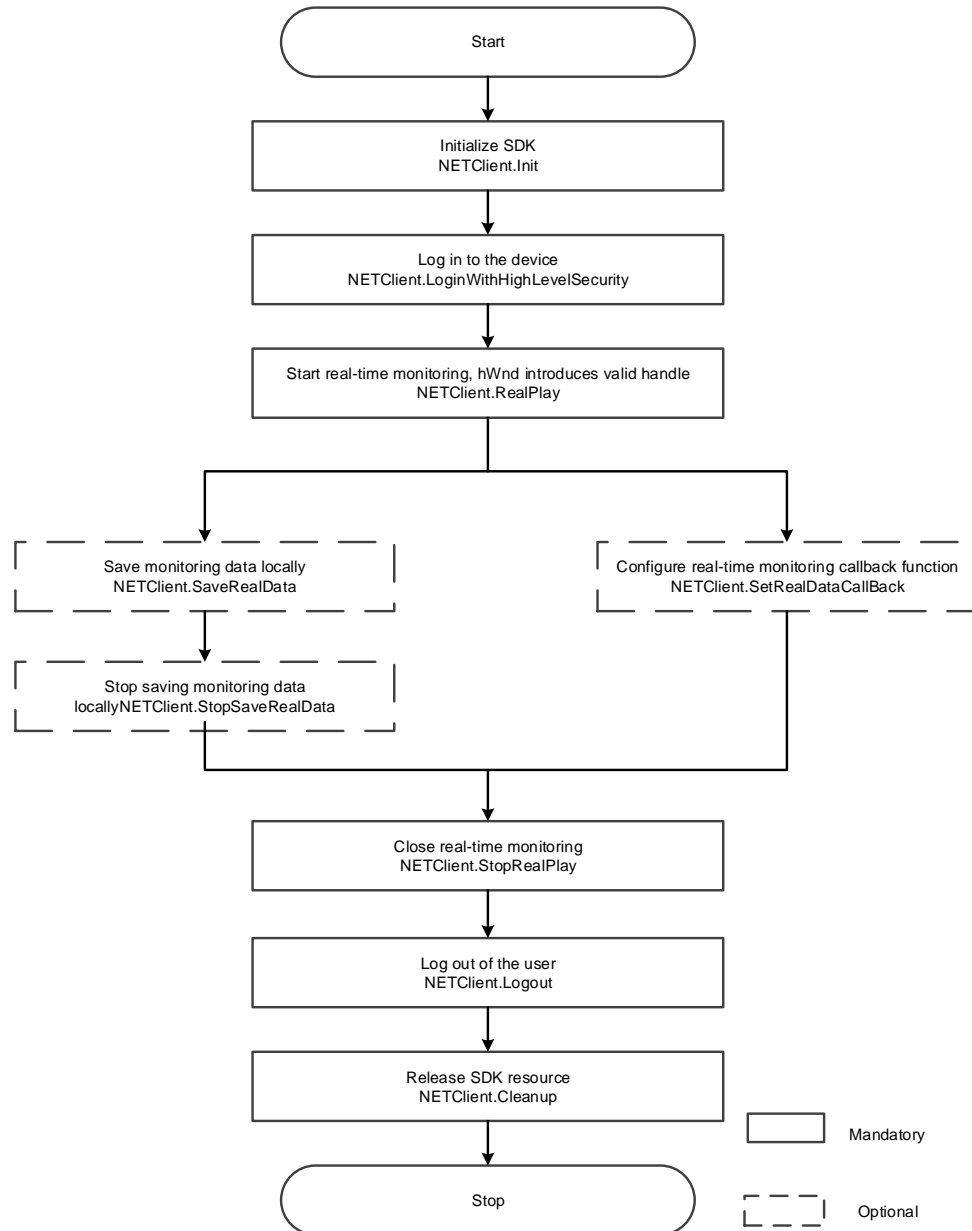
2.1.4.3 Process Description

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

2.1.4.3.1 SDK Decoding Play

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

Figure 2-4 SDK decoding play



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.RealPlay** to start the real-time monitoring. The parameter **hWnd** is a valid window handle.
- Step 4 (Optional) Call **NETClient.SaveRealData** to start saving the monitoring data.
- Step 5 (Optional) Call **NETClient.StopSaveRealData** to end the saving process and generate a local video file.
- Step 6 (Optional) If you call **NETClient.SetRealDataCallBack**, you can choose to save or forward the video data. If the video data is saved as a file, see the step 4 and step 5.
- Step 7 After using the real-time monitoring, call **NETClient.StopRealPlay** to stop it.
- Step 8 After using the function module, call **NETClient.Logout** to log out of the device.
- Step 9 After using all NetSDK functions, call **NETClient.Cleanup** to release SDK resources.

Note

- NetSDK decoding play only supports Windows system. You need to call the decoding after getting the stream for display in other systems.
- Multi-thread calling: Multi-thread calling is not supported for the functions within the same login session; however, multi-thread calling can deal with the functions of different login sessions although such calling is not recommended.
- Timeout: The application for monitoring resources in the interface should make some agreements with the device before requesting the monitoring data. There are some timeout settings (see "NET_PARAM structure"), and the field related to monitoring is **nGetConnInfoTime**. If there is timeout due to the reasons such as bad network connection, you can modify the value of **nGetConnInfoTime** bigger. The sample code is as follows. Call it for only one time after having called the **NETClient.Init**.

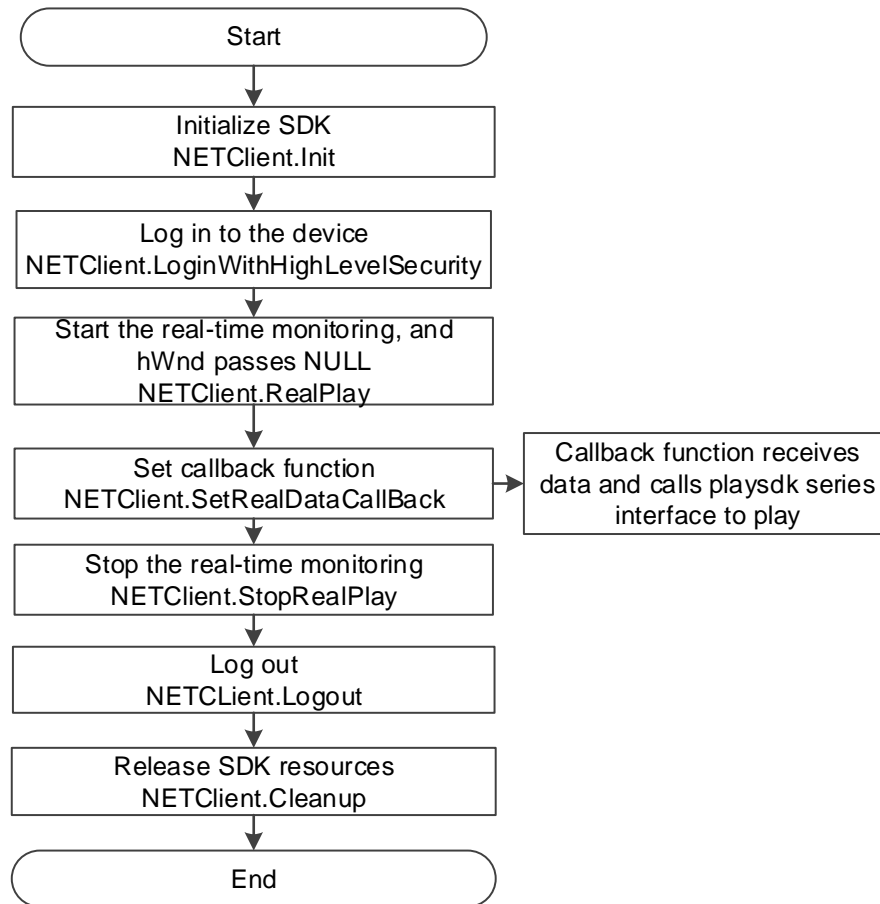
```
NET_PARAM param = new NET_PARAM()
{
    nGetConnInfoTime = 5000, // get sub connection info timeout duration (ms)
};
NETClient.SetNetworkParam(param);
```

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

2.1.4.3.2 Calling the Third-party Decoding Play Library

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

Figure 2-5 Third-party decoding play



Process

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 After successful login, call **NETClient.RealPlay** to start real-time monitoring. The parameter **hWnd** is **NULL**.
- Step 4 Call **NETClient.SetRealDataCallBack** to set the real-time data callback.
- Step 5 In the callback, pass the data to PlaySDK to finish decoding.
- Step 6 After using the real-time monitoring, call **NETClient.StopRealPlay** to stop it.
- Step 7 After using the function module, call **NETClient.Logout** to log out of the device.
- Step 8 After using all SDK functions, call **NETClient.Cleanup** to release SDK resources.

Note

- Stream format: It is recommended to use PlaySDK for decoding.
- Image lag:
 - ◇ When using PlaySDK for decoding, there is a default channel cache size (the `PLAY_OpenStream` interface in PlaySDK) for decoding. If the stream resolution value is big, it is recommended to change the parameter value (for example, changed to `3*1024*1024`).
 - ◇ NetSDK callbacks can only move into the next process after returning from you. Do not do time consuming operations; otherwise the performance could be affected.

2.1.4.4 Sample Code

2.1.4.4.1 SDK Decoding Play

```
//take enabling main stream of the first channel as an example, hWnd is window handle of the
interface
IntPtr m_RealPlayID = NETClient.RealPlay(m_LoginID, 0, hWnd, EM_RealPlayType.Realplay);
if (IntPtr.Zero == m_RealPlayID)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

// Disable monitoring
bool ret = NETClient.StopRealPlay(m_RealPlayID);
if (!ret)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}
m_RealPlayID = IntPtr.Zero;
```

2.1.4.4.2 Calling Play Library

```
// take enabling main stream of the first channel as an example
IntPtr m_RealPlayID = NETClient.RealPlay(m_LoginID, 0, null, EM_RealPlayType.Realplay);
if (IntPtr.Zero == m_RealPlayID)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

//configure real-time monitoring callback function
private static fRealDataCallBackEx2 m_RealDataCallBackEx2;
m_RealDataCallBackEx2 = new fRealDataCallBackEx2(RealDataCallBackEx);
NETClient.SetRealDataCallBack(m_RealPlayID, m_RealDataCallBackEx2, IntPtr.Zero,
EM_REALDATA_FLAG.DATA_WITH_FRAME_INFO |
EM_REALDATA_FLAG.PCM_AUDIO_DATA | EM_REALDATA_FLAG.RAW_DATA |
EM_REALDATA_FLAG.YUV_DATA);
private void RealDataCallBackEx(IntPtr IRealHandle, uint dwDataType, IntPtr pBuffer, uint
dwBufSize, IntPtr param, IntPtr dwUser)
{
    // To get stream data from the device, you need to call PlaySDK interfaces. For details, see
    NetSDK monitoring demo source code
    // For example, save data, send data, transform to YUV and more.
    EM_REALDATA_FLAG type = (EM_REALDATA_FLAG)dwDataType;
    switch (type)
    {

```

```

        case EM_REALDATA_FLAG.RAW_DATA:
            //processing operations
            break;
    }
}

// close monitoring
bool ret = NETClient.StopRealPlay(m_RealPlayID);
if (!ret)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}
m_RealPlayID = IntPtr.Zero;

```

2.1.5 Voice Talk

2.1.5.1 Introduction

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

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

2.1.5.2 Interface Overview

Table 2-5 Description of voice talk interfaces

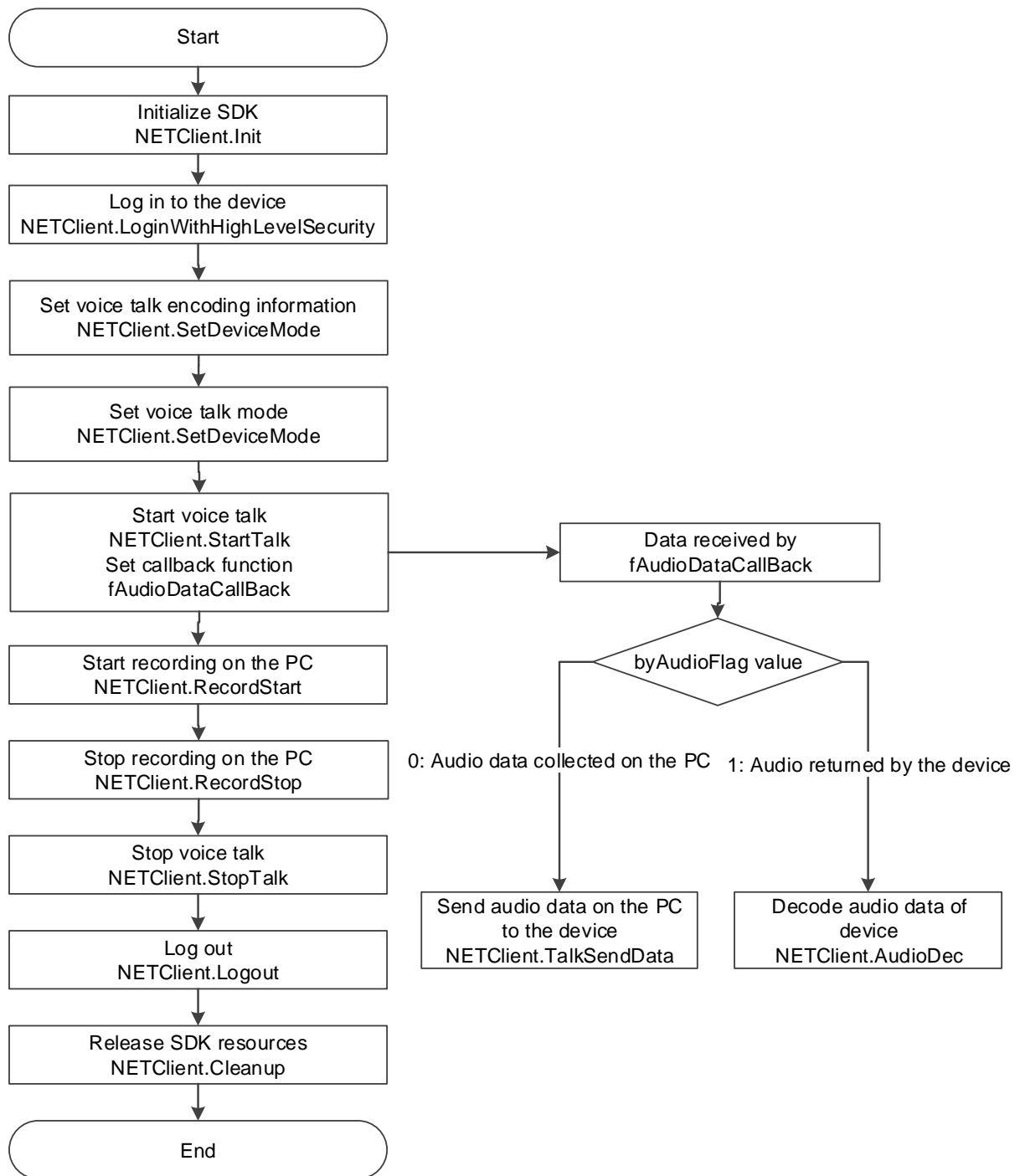
Interface	Description
NETClient.StartTalk	Extension interface for starting the voice talk.
NETClient.StopTalk	Extension interface for stopping the voice talk.
NETClient.RecordStart	Extension interface for starting the client record (valid only in Windows system).
NETClient.RecordStop	Extension interface for stopping the client record (valid only in Windows system).
NETClient.TalkSendData	Send voice data to the device.
NETClient.AudioDec	Extension interface for decoding audio data (valid only in Windows system).

2.1.5.3 Process Description

When NetSDK collects the audio data from the local audio card or receives the audio data from the front-end devices, it will call the audio data callback. You can call the NetSDK interface in the callback to send the local audio data collected to the front-end devices, or call the NetSDK interface to decode and play back the audio data received from the front-end devices.

The process is valid only in Windows system.

Figure 2-6 Second-generation voice talk



Procedure

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 After successful initialization, call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.SetDeviceMode** to get support for the second-generation or third-generation voice talk. The value of **emType** should be **EM_USEDEV_MODE.TALK_ENCODE_TYPE**.
- Step 4 Call **NETClient.SetDeviceMode** to set voice talk parameters. The value of **emType** should be **EM_USEDEV_MODE.TALK_SPEAK_PARAM**.

- Step 5 Call **NETClient.StartTalk** to set callback and start voice talk. In the callback, call **NETClient.AudioDec** to decode the audio data sent from the decoding device, and call **NETClient.TalkSendData** to send the audio data from the PC to the device.
- Step 6 Call **NETClient.RecordStart** to start recording on the PC. After this interface is called, the voice talk callback set by **NETClient.StartTalk** will receive the local audio data.
- Step 7 After using the voice talk function, call **NETClient.RecordStop** to stop recording.
- Step 8 Call **NETClient.StopTalk** to stop voice talk.
- Step 9 Call **NETClient.Logout** to log out of the device.
- Step 10 After using NetSDK, call **NETClient.Cleanup** to release NetSDK resources.

Note

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

2.1.5.4 Sample Code

```

        if (IntPtr.Zero == m_TalkID)
        {
            IntPtr talkEncodePointer = IntPtr.Zero;
            IntPtr talkSpeakPointer = IntPtr.Zero;
            IntPtr talkTransferPointer = IntPtr.Zero;
            IntPtr channelPointer = IntPtr.Zero;

            NET_DEV_TALKDECODE_INFO talkCodeInfo = new
NET_DEV_TALKDECODE_INFO();
            talkCodeInfo.encodeType = EM_TALK_CODING_TYPE.PCM;
            talkCodeInfo.dwSampleRate = SampleRate;
            talkCodeInfo.nAudioBit = AudioBit;
            talkCodeInfo.nPacketPeriod = PacketPeriod;
            talkCodeInfo.reserved = new byte[60];

            talkEncodePointer =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_DEV_TALKDECODE_INFO)));
            Marshal.StructureToPtr(talkCodeInfo, talkEncodePointer, true);
            // set talk encode type
            NETClient.SetDeviceMode(m_LoginID,
EM_USEDEV_MODE.TALK_ENCODE_TYPE, talkEncodePointer);

            NET_SPEAK_PARAM speak = new NET_SPEAK_PARAM();
            speak.dwSize = (uint)Marshal.SizeOf(typeof(NET_SPEAK_PARAM));
            speak.nMode = 0;

```

```

        speak.bEnableWait = false;
        speak.nSpeakerChannel = 0;
        talkSpeakPointer
    Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_SPEAK_PARAM)));
        Marshal.StructureToPtr(speak, talkSpeakPointer, true);
        //set talk speak mode
        NETClient.SetDeviceMode(m_LoginID,
EM_USEDEV_MODE.TALK_SPEAK_PARAM, talkSpeakPointer);

        NET_TALK_TRANSFER_PARAM        transfer        =        new
NET_TALK_TRANSFER_PARAM();
        transfer.dwSize
    (uint)Marshal.SizeOf(typeof(NET_TALK_TRANSFER_PARAM));
        if (local_radioButton.Checked)
        {
            transfer.bTransfer = false;
        }
        else
        {
            transfer.bTransfer = true;
            channelPointer = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(int)));
            Marshal.WriteInt32(channelPointer, channel_comboBox.SelectedIndex);
            NETClient.SetDeviceMode(m_LoginID,
EM_USEDEV_MODE.TALK_TALK_CHANNEL, channelPointer);
        }
        talkTransferPointer
    Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_TALK_TRANSFER_PARAM)));
        Marshal.StructureToPtr(transfer, talkTransferPointer, true);
        //set talk transfer mode
        NETClient.SetDeviceMode(m_LoginID,
EM_USEDEV_MODE.TALK_TRANSFER_MODE, talkTransferPointer);

        m_TalkID    =    NETClient.StartTalk(m_LoginID,    m_AudioDataCallBack,
IntPtr.Zero);

        Marshal.FreeHGlobal(talkEncodePointer);
        Marshal.FreeHGlobal(talkSpeakPointer);
        Marshal.FreeHGlobal(talkTransferPointer);
        Marshal.FreeHGlobal(channelPointer);
        if(IntPtr.Zero == m_TalkID)
        {
            MessageBox.Show(this, NETClient.GetLastError());
            return;
        }
        bool ret = NETClient.RecordStart(m_LoginID);
        if(!ret)
        {
            NETClient.StopTalk(m_TalkID);

```

```

        m_TalkID = IntPtr.Zero;
        MessageBox.Show(this, NETClient.GetLastError());
        return;
    }
    talk_button.Text = "StopTalk";
}
else
{
    NETClient.RecordStop(m_LoginID);
    NETClient.StopTalk(m_TalkID);
    m_TalkID = IntPtr.Zero;
    talk_button.Text = "StartTalk";
}

private void AudioDataCallBack(IntPtr ITalkHandle, IntPtr pDataBuf, uint dwBufSize,
byte byAudioFlag, IntPtr dwUser)
{
    if (ITalkHandle == m_TalkID)
    {
        if (SendAudio == byAudioFlag)
        {
            //send talk data
            NETClient.TalkSendData(ITalkHandle, pDataBuf, dwBufSize);
        }
        else if (ReviceAudio == byAudioFlag)
        {
            //here call netsdk decode audio,or can send data to other user.
            try
            {
                NETClient.AudioDec(pDataBuf, dwBufSize);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
}
}

```

2.1.6 Sending Alarm

2.1.6.1 Introduction

Here is how to realize alarm sending: Log in to the device through NetSDK, and then subscribe alarm from devices. The device sends the information to NetSDK when it detects alarm events, and the alarm information can be received through alarm callback function.

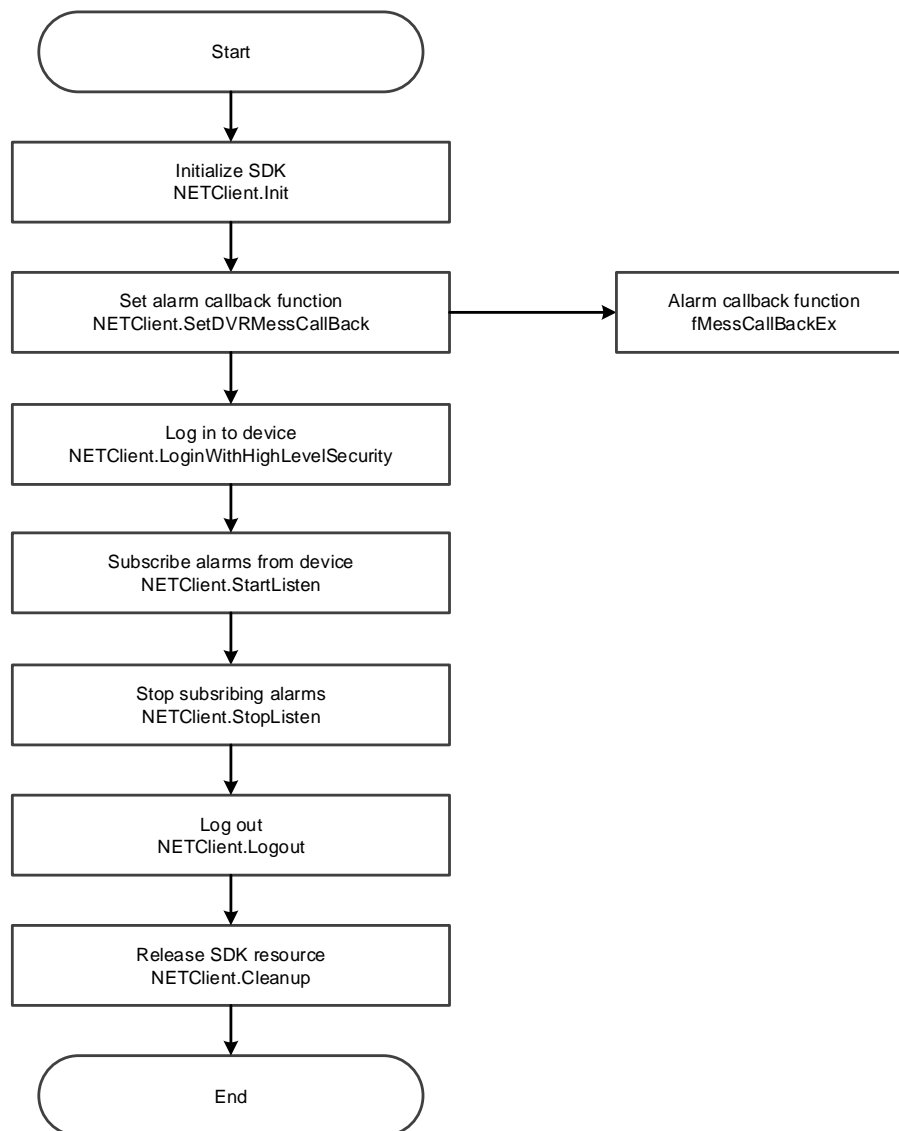
2.1.6.2 Interface Overview

Table 2-6 Description of alarm sending interfaces

Interface	Description
NETClient.SetDVRMessCallBack	Set alarm callback function.
NETClient.StartListen	Extension interface for alarm subscription
NETClient.StopListen	Stop alarm subscription

2.1.6.3 Process Description

Figure 2-7 Arming and disarming



Procedure

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.SetDVRMessCallBack** to set alarm callback function. This interface needs to be called before alarm subscription.
- Step 3 Call **NETClient.LoginWithHighLevelSecurity** to log in to device.

Step 4 Call **NETClient.StartListen** to subscribe alarms from device. After the subscription, alarm events sent by device will be sent to users through the callback function set through **NETClient.SetDVRMessCallBack**.

Step 5 After alarm sending is completed, call **NETClient.StopListen** to stop subscribing alarms from device.

Step 6 Call **NETClient.Logout** to log out of device.

Step 7 After using NetSDK, call **NETClient.Cleanup** to release NetSDK resources.

Note

- If alarms that can be sent before cannot be sent now, you need to check whether the device is disconnected from network. If yes, the alarm subscription will be resumed automatically after the device connects to network again. You do not need to cancel subscription manually, and then subscribe again.
- It is recommended that you asynchronously process alarm information in the alarm callback function fMessCallBack. Do not perform too many operations in the callback; otherwise the callback might be blocked.

2.1.6.4 Sample Code

```
// State static callback entrusting
private static fMessCallBackEx m_AlarmCallBack;
m_AlarmCallBack = new fMessCallBackEx(AlarmCallBackEx);
// Set alarm callback
NETClient.SetDVRMessCallBack(m_AlarmCallBack, IntPtr.Zero);

// Process alarm callback
private bool AlarmCallBackEx(int ICommand, IntPtr ILoginID, IntPtr pBuf, uint dwBufLen, IntPtr
pchDVRIP, int nDVRPort, bool bAlarmAckFlag, int nEventID, IntPtr dwUser)
{
    EM_ALARM_TYPE type = (EM_ALARM_TYPE)ICommand;
    switch (type)
    {
        case EM_ALARM_TYPE.ALARM_ALARM_EX:
            data = new byte[dwBufLen];
            Marshal.Copy(pBuf, data, 0, (int)dwBufLen);
            for (int i = 0; i < dwBufLen; i++)
            {
                if (data[i] == ALARM_START) // alarm start
                {
                    // Custom processing
                }
                else //alarm stop
                {
                }
            }
            break;
        case EM_ALARM_TYPE.ALARM_RECORD_SCHEDULE_CHANGE:
```

```

{
    NET_ALARM_RECORD_SCHEDULE_CHANGE_INFO info =
    (NET_ALARM_RECORD_SCHEDULE_CHANGE_INFO)Marshal.PtrToStructure(pBuf,
    typeof(NET_ALARM_RECORD_SCHEDULE_CHANGE_INFO));
    // Custom processing
}
break;

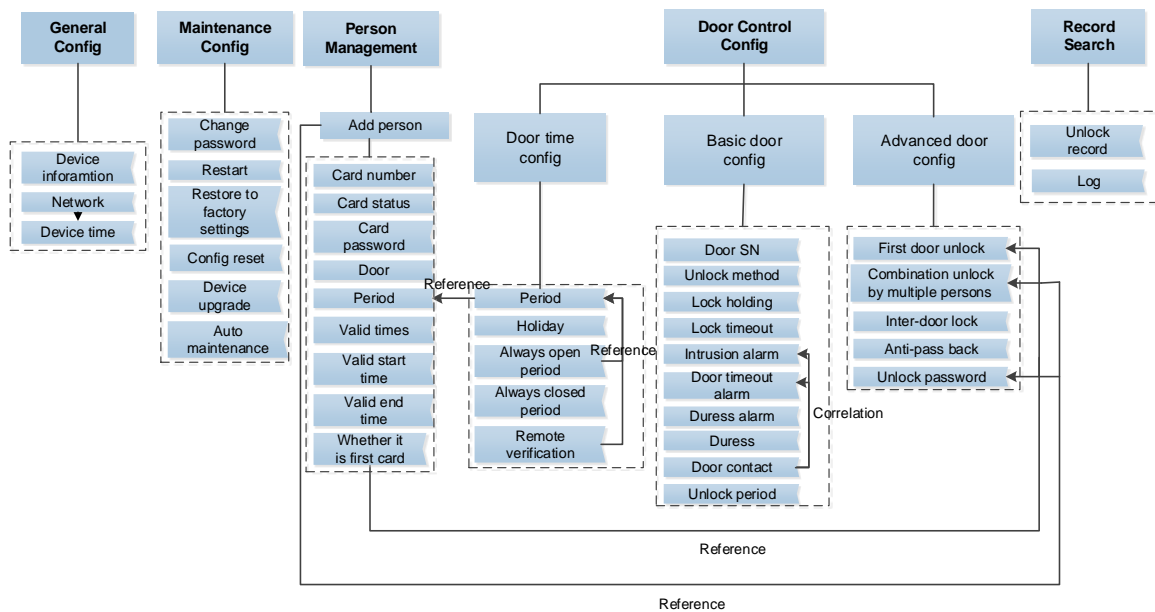
    default:
        Console.WriteLine(!Command.ToString("X"));
        break;
    }
    return true;
}

// Subscribe alarms
bool ret = NETClient.StartListen(m_LoginID);
if (!ret)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}
// Stop subscribing alarms
bool ret = NETClient.StopListen(m_LoginID);
if (!ret)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}
}

```

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

Figure 2-8 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 pointed by the start point of the arrow can be used normally depends on the function configuration pointed by the end point of the arrow.

2.2.1 Access Control

2.2.1.1 Introduction

You can control the opening and closing of the access, and get door contact status. You can remotely open and close the door directly without person information.

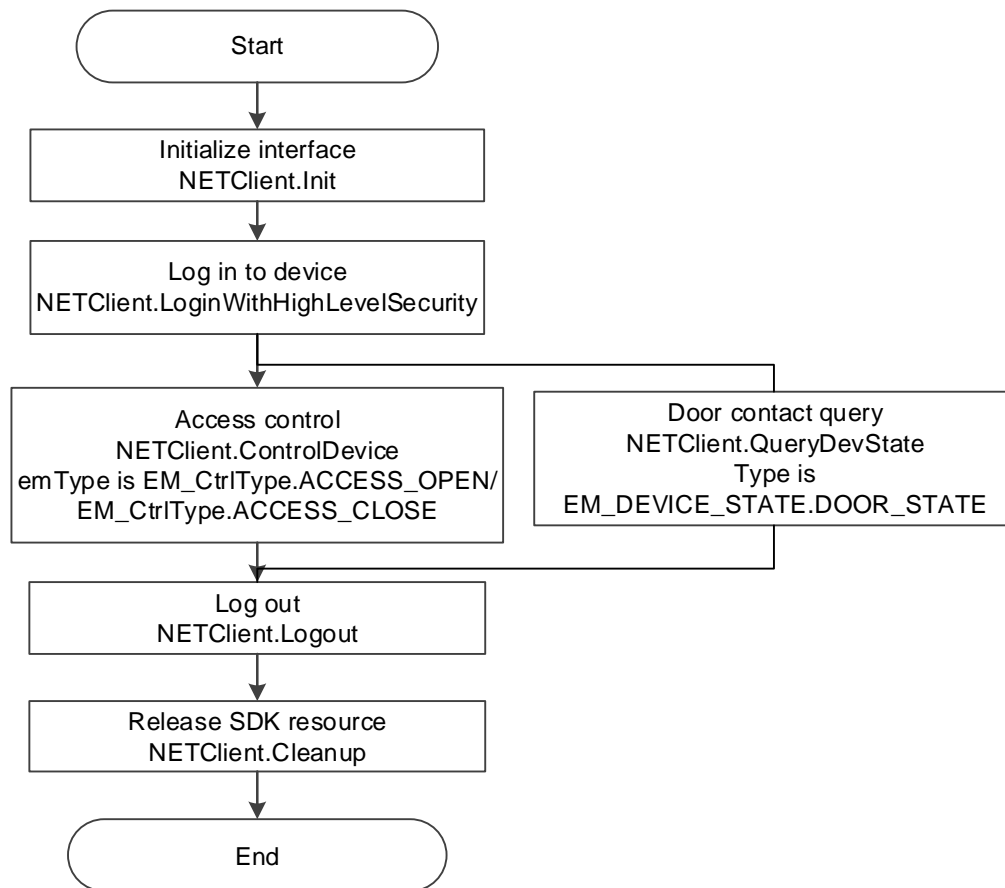
2.2.1.2 Interface Overview

Table 2-7 Description of access control interface

Interface	Description
NETClient.ControlDevice	Device control interface.
NETClient.QueryDevState	Status search interface.

2.2.1.3 Process Description

Figure 2-9 Access control business flow



Procedure

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to device.
- Step 3 Call **NETClient.ControlDevice** to control access.
- Open access: **emType** value is **EM_CtrlType.ACCESS_OPEN**.
 - Close access: **emType** value is **EM_CtrlType.ACCESS_CLOSE**.
- Step 4 Call **NETClient.QueryDevState** to query door contact.
- Type: EM_DEVICE_STATE_DOOR_STATE
pBuf: NET_DOOR_STATUS_INFO.
- Step 5 After completing this process, call **NETClient.Logout** to log out of device.
- Step 6 After using SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.2.1.4 Sample Code

```
#region Open door Access control-open
IntPtr m_LoginHandle = IntPtr.Zero;
GetConfig();
if (cfg.emState != EM_CFG_ACCESS_STATE.NORMAL)
```

```

    {
        cfg.emState = EM_CFG_ACCESS_STATE.NORMAL;
        SetConfig(cfg);
    }
    NET_CTRL_ACCESS_OPEN openInfo = new NET_CTRL_ACCESS_OPEN();
    openInfo.dwSize = (uint)Marshal.SizeOf(typeof(NET_CTRL_ACCESS_OPEN));
    openInfo.nChannelID = 0;
    openInfo.szTargetID = IntPtr.Zero;
    openInfo.emOpenDoorType = EM_OPEN_DOOR_TYPE.REMOTE;
    IntPtr inPtr = IntPtr.Zero;
    try
    {
        inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_ACCESS_OPEN)));
        Marshal.StructureToPtr(openInfo, inPtr, true);
        bool ret = NETClient.ControlDevice(m_LoginID,
EM_CtrlType.ACCESS_OPEN, inPtr, 10000);
        if (!ret)
        {
            MessageBox.Show(NETClient.GetLastError());
            return;
        }
    }
    finally
    {
        Marshal.FreeHGlobal(inPtr);
    }
    MessageBox.Show("Opened door successfully");
#endregion

#region Close door Access control-close
IntPtr m_LoginHandle = IntPtr.Zero;
GetConfig();
if (cfg.emState != EM_CFG_ACCESS_STATE.NORMAL)
{
    cfg.emState = EM_CFG_ACCESS_STATE.NORMAL;
    SetConfig(cfg);
}
    NET_CTRL_ACCESS_CLOSE closeInfo = new
NET_CTRL_ACCESS_CLOSE();

```

```

        closeInfo.dwSize = (uint)Marshal.SizeOf(typeof(NET_CTRL_ACCESS_CLOSE));
        closeInfo.nChannelID = 0;
        IntPtr inPtr = IntPtr.Zero;
        try
        {
            inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_ACCESS_CLOSE)));
            Marshal.StructureToPtr(closeInfo, inPtr, true);
            bool ret = NETClient.ControlDevice(m_LoginID,
EM_CtrlType.ACCESS_CLOSE, inPtr, 10000);
            if (!ret)
            {
                MessageBox.Show(NETClient.GetLastError());
                return;
            }
        }
        finally
        {
            Marshal.FreeHGlobal(inPtr);
        }
        MessageBox.Show("Closed door successfully");
    #endregion
    #region Search for door contact status
    NET_DOOR_STATUS_INFO info = new NET_DOOR_STATUS_INFO();
    info.dwSize = (uint)Marshal.SizeOf(typeof(NET_DOOR_STATUS_INFO));
    info.nChannel = 0;
    object objInfo = info;
    bool ret = NETClient.QueryDevState(m_LoginID,
EM_DEVICE_STATE.DOOR_STATE, ref objInfo, typeof(NET_DOOR_STATUS_INFO),
10000);
    if (!ret)
    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }
    info = (NET_DOOR_STATUS_INFO)objInfo;

    switch (info.emStateType)
    {
        case EM_NET_DOOR_STATUS_TYPE.BREAK:

```

```

        MessageBox.Show("Door opened abnormally");
        break;
    case EM_NET_DOOR_STATUS_TYPE.CLOSE:
        MessageBox.Show("Door closed");
        break;
    case EM_NET_DOOR_STATUS_TYPE.OPEN:
        MessageBox.Show("Door opened");
        break;
    case EM_NET_DOOR_STATUS_TYPE.UNKNOWN:
        MessageBox.Show("Unknown");
        break;
    default:
        break;
}
#endregion

```

2.2.2 Alarm Event

2.2.2.1 Introduction

Here is the process to get event. You call the SDK interface. SDK actively connects to the device, and subscribes events from the device, including door opening event and alarm event. Device sends information to SDK immediately when events occur. You can stop susbscription if you do not want to receive events from device any more.

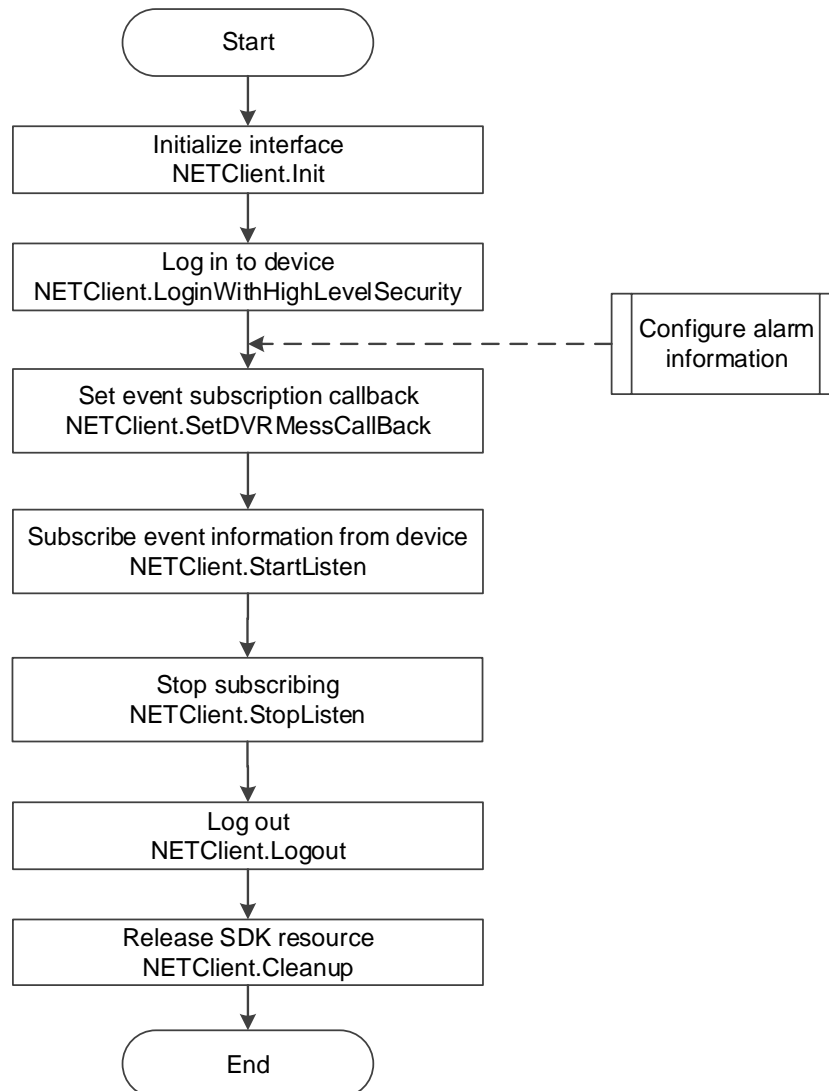
2.2.2.2 Interface Overview

Table 2-8 Description of alarm event interface

Interface	Description
NETClient.StartListen	Subscribe alarms from device.
NETClient.SetDVRMessCallBa ck	Set device message callback to get the current device status; this function is independent of the calling sequence, and SDK is not called back by default. The callback must call the alarm message subscription interface NETClient.StartListen first before it takes effect.
NETClient.StopListen	Stop subscription.

2.2.2.3 Process Description

Figure 2-10 Alarm event business flow



Procedure

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to device.
- Step 3 Configure alarm arming (skip this step if alarm arming has been configured).
- Step 4 Set alarm callback **NETClient.SetDVRMessCallBack**.
- Step 5 Call **NETClient.StartListen** to subscribe alarm information from device.
- Step 6 After the alarm sending process is completed, you need to call **NETClient.StopListen** to stop subscribing alarm events.
- Step 7 After completing this process, call **NETClient.Logout** to log out of device.
- Step 8 After using SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.2.2.4 Sample Code

```
// State static callback entrusting
```

```

private static fMessCallBackEx m_AlarmCallBack;
m_AlarmCallBack = new fMessCallBackEx(AlarmCallBackEx);
// Set alarm callback
NETClient.SetDVRMessCallBack(m_AlarmCallBack, IntPtr.Zero);
// Process alarm callback
private bool AlarmCallBack(int ICommand, IntPtr ILoginID, IntPtr pBuf, uint dwBufLen, IntPtr
pchDVRIP, int nDVRPort, bool bAlarmAckFlag, int nEventID, IntPtr dwUser)
{
    EM_ALARM_TYPE type = (EM_ALARM_TYPE)ICommand;
    var item = new ListViewItem();
    switch (type)
    {
        case EM_ALARM_TYPE.ALARM_ACCESS_CTL_EVENT:
            NET_ALARM_ACCESS_CTL_EVENT_INFO access_info =
(NET_ALARM_ACCESS_CTL_EVENT_INFO)Marshal.PtrToStructure(pBuf,
typeof(NET_ALARM_ACCESS_CTL_EVENT_INFO));
            item.Text = Alarm_Index.ToString();
            item.SubItems.Add(access_info.stuTime.ToString());
            item.SubItems.Add("Entry");
            item.SubItems.Add(access_info.szUserID.ToString());
            item.SubItems.Add(access_info.szCardNo.ToString());
            item.SubItems.Add(access_info.nDoor.ToString());
            switch (access_info.emOpenMethod)
            {
                case EM_ACCESS_DOOROPEN_METHOD.CARD:
                    item.SubItems.Add("Card");
                    break;
                case
EM_ACCESS_DOOROPEN_METHOD.FACE_RECOGNITION:
                    item.SubItems.Add("Face recognition");
                    break;
                default:
                    item.SubItems.Add("Unknown");
                    break;
            }
            if (access_info.bStatus)
            {
                item.SubItems.Add("Success");
            }
            else

```

```

        {
            item.SubItems.Add("Failure");
        }

this.BeginInvoke(new Action(() =>
{
    listView_event.BeginUpdate();
    listView_event.Items.Insert(0, item);
    if (listView_event.Items.Count > ListViewCount)
    {
        listView_event.Items.RemoveAt(ListViewCount);
    }
    listView_event.EndUpdate();
}));
Alarm_Index++;
break;
case EM_ALARM_TYPE.ALARM_ACCESS_CTL_NOT_CLOSE:
    NET_ALARM_ACCESS_CTL_NOT_CLOSE_INFO notclose_info =
(NET_ALARM_ACCESS_CTL_NOT_CLOSE_INFO)Marshal.PtrToStructure(pBuf,
typeof(NET_ALARM_ACCESS_CTL_NOT_CLOSE_INFO));
    item.Text = Alarm_Index.ToString();
    item.SubItems.Add(notclose_info.stuTime.ToString());
    item.SubItems.Add("Not closed");
    item.SubItems.Add("");
    item.SubItems.Add("");
    item.SubItems.Add(notclose_info.nDoor.ToString());
    item.SubItems.Add("");
    if (notclose_info.nAction == ALARM_START)
    {
        item.SubItems.Add("Start");
    }
    else if (notclose_info.nAction == ALARM_STOP)
    {
        item.SubItems.Add("Stop");
    }
    else
    {
        item.SubItems.Add("");
    }
}

```

```

        this.BeginInvoke(new Action(() =>
        {
            listView_event.BeginUpdate();
            listView_event.Items.Insert(0, item);
            if (listView_event.Items.Count > ListViewCount)
            {
                listView_event.Items.RemoveAt(ListViewCount);
            }
            listView_event.EndUpdate();
        }));
        Alarm_Index++;
        break;

    case EM_ALARM_TYPE.ALARM_ACCESS_CTL_BREAK_IN:
        NET_ALARM_ACCESS_CTL_BREAK_IN_INFO breakin_info =
        (NET_ALARM_ACCESS_CTL_BREAK_IN_INFO)Marshal.PtrToStructure(pBuf,
        typeof(NET_ALARM_ACCESS_CTL_BREAK_IN_INFO));
        item.Text = Alarm_Index.ToString();
        item.SubItems.Add(breakin_info.stuTime.ToString());
        item.SubItems.Add("Break in");
        item.SubItems.Add("");
        item.SubItems.Add("");
        item.SubItems.Add(breakin_info.nDoor.ToString());
        item.SubItems.Add("");
        item.SubItems.Add("");

        this.BeginInvoke(new Action(() =>
        {
            listView_event.BeginUpdate();
            listView_event.Items.Insert(0, item);
            if (listView_event.Items.Count > ListViewCount)
            {
                listView_event.Items.RemoveAt(ListViewCount);
            }
            listView_event.EndUpdate();
        }));
        Alarm_Index++;
        break;

    case EM_ALARM_TYPE.ALARM_ACCESS_CTL_REPEAT_ENTER:

```



```

        NET_ALARM_ACCESS_CTL_REPEAT_ENTER_INFO repeat_info =
(NET_ALARM_ACCESS_CTL_REPEAT_ENTER_INFO)Marshal.PtrToStructure(pBuf,
typeof(NET_ALARM_ACCESS_CTL_REPEAT_ENTER_INFO));

        item.Text = Alarm_Index.ToString();
        item.SubItems.Add(repeat_info.stuTime.ToString());
        item.SubItems.Add("Passback");
        item.SubItems.Add("");
        item.SubItems.Add(repeat_info.szCardNo.ToString());
        item.SubItems.Add(repeat_info.nDoor.ToString());
        item.SubItems.Add("");
        item.SubItems.Add("");

        this.BeginInvoke(new Action(() =>
        {
            listView_event.BeginUpdate();
            listView_event.Items.Insert(0, item);
            if (listView_event.Items.Count > ListViewCount)
            {
                listView_event.Items.RemoveAt(ListViewCount);
            }
            listView_event.EndUpdate();
        }));
        Alarm_Index++;
        break;

    case EM_ALARM_TYPE.ALARM_ACCESS_CTL_DURESS:
        NET_ALARM_ACCESS_CTL_DURESS_INFO duress_info =
(NET_ALARM_ACCESS_CTL_DURESS_INFO)Marshal.PtrToStructure(pBuf,
typeof(NET_ALARM_ACCESS_CTL_DURESS_INFO));

        item.Text = Alarm_Index.ToString();
        item.SubItems.Add(duress_info.stuTime.ToString());
        item.SubItems.Add("Duress");
        item.SubItems.Add(duress_info.szUserID.ToString());
        item.SubItems.Add(duress_info.szCardNo.ToString());
        item.SubItems.Add(duress_info.nDoor.ToString());
        item.SubItems.Add("");
        item.SubItems.Add("");

        this.BeginInvoke(new Action(() =>
        {
            listView_event.BeginUpdate();

```

```

        listView_event.Items.Insert(0, item);
        if (listView_event.Items.Count > ListViewCount)
        {
            listView_event.Items.RemoveAt(ListViewCount);
        }
        listView_event.EndUpdate();
    }));
    Alarm_Index++;
    break;
case EM_ALARM_TYPE.ALARM_CHASSISINTRUDED:
    NET_ALARM_CHASSISINTRUDED_INFO chassisintruded_info =
(NET_ALARM_CHASSISINTRUDED_INFO)Marshal.PtrToStructure(pBuf,
typeof(NET_ALARM_CHASSISINTRUDED_INFO));
    item.Text = Alarm_Index.ToString();
    item.SubItems.Add(chassisintruded_info.stuTime.ToString());
    if (chassisintruded_info.szReaderID.Length > 0)
    {
        item.SubItems.Add("Card reader anti-demolition");
    }
    else
    {
        item.SubItems.Add("Chassis intruded");
    }
    item.SubItems.Add("");
    item.SubItems.Add("");
    item.SubItems.Add(chassisintruded_info.nChannelID.ToString());
    item.SubItems.Add("");
    if (chassisintruded_info.nAction == ALARM_START)
    {
        item.SubItems.Add("Start");
    }
    else if (chassisintruded_info.nAction == ALARM_STOP)
    {
        item.SubItems.Add("Stop");
    }
    else
    {
        item.SubItems.Add("");
    }
}

```

```

        this.BeginInvoke(new Action(() =>
        {
            listView_event.BeginUpdate();
            listView_event.Items.Insert(0, item);
            if (listView_event.Items.Count > ListViewCount)
            {
                listView_event.Items.RemoveAt(ListViewCount);
            }
            listView_event.EndUpdate();
        }));
        Alarm_Index++;
        break;

    case EM_ALARM_TYPE.ALARM_ALARM_EX2:
        NET_ALARM_ALARM_INFO_EX2 alarm_info =
(NET_ALARM_ALARM_INFO_EX2)Marshal.PtrToStructure(pBuf,
typeof(NET_ALARM_ALARM_INFO_EX2));
        item.Text = Alarm_Index.ToString();
        item.SubItems.Add(alarm_info.stuTime.ToString());
        item.SubItems.Add("External alarm");
        item.SubItems.Add("");
        item.SubItems.Add("");
        item.SubItems.Add(alarm_info.nChannelID.ToString());
        item.SubItems.Add("");
        if (alarm_info.nAction == ALARM_START)
        {
            item.SubItems.Add("Start");
        }
        else if (alarm_info.nAction == ALARM_STOP)
        {
            item.SubItems.Add("Stop");
        }
        else
        {
            item.SubItems.Add("");
        }

        this.BeginInvoke(new Action(() =>
        {
            listView_event.BeginUpdate();
            listView_event.Items.Insert(0, item);

```

```

        if (listView_event.Items.Count > ListViewCount)
        {
            listView_event.Items.RemoveAt(ListViewCount);
        }
        listView_event.EndUpdate();
    }));
    Alarm_Index++;
    break;
default:
    break;
}

return true;
}
#endregion
// Alarm subscription
if (!m_IsListen)
{
    bool ret = NETClient.StartListen(m_LoginID);
    if (!ret)
    {
        MessageBox.Show(this, NETClient.GetLastError());
        return;
    }
    m_IsListen = true;
    btn_StartListen.Text = "Stop subscribing";
}
else
{
    bool ret = NETClient.StopListen(m_LoginID);
    if (!ret)
    {
        MessageBox.Show(this, NETClient.GetLastError());
        return;
    }
    m_IsListen = false;
    listView_Event.Items.Clear();
    btn_StartListen.Text = "Start subscribing";
}

```

2.2.3 Viewing Device Information

2.2.3.1 Capability Set Query

2.2.3.1.1 Introduction

Here is the process to view device information: You issue a command through SDK to access control device to get device capability set.

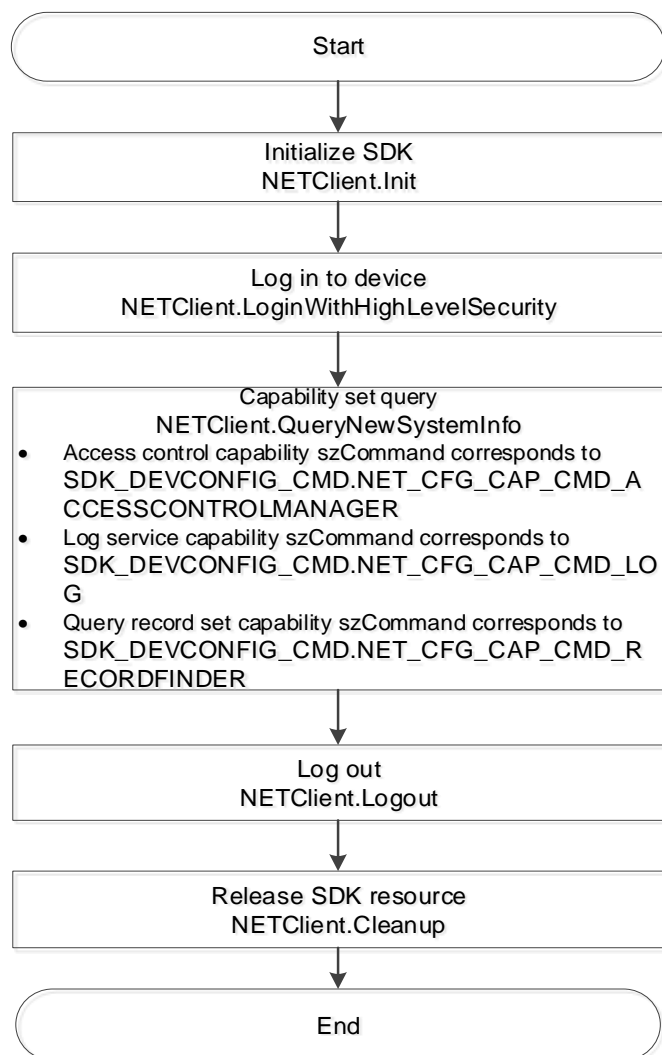
2.2.3.1.2 Interface Overview

Table 2-9 Description of capability set query interface

Interface	Description
NETClient.QueryNewSystemInfo	Query information on system capabilities (such as logs, record sets, and door control capabilities).

2.2.3.1.3 Process Description

Figure 2-11 Device information viewing



Procedure

Step 1 Call **NETClient.Init** to initialize SDK.

Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to device.

Step 3 Call **NETClient.QueryNewSystemInfo** to query access control capability set.

Table 2-10 Description and structure of szCommand

szCommand	Description	szOutBuffer
SDK_DEVCONFIG_CMD.NET_CFG_CAP_CMD_ACCESSCONTROLMANAGER	Access control capability	CFG_CAP_ACCESSCONTROL
SDK_DEVCONFIG_CMD.NET_CFG_CAP_CMD_LOG	Log service capability	NET_CFG_CAP_LOG
SDK_DEVCONFIG_CMD.NET_CFG_CAP_CMD_RECORDFINDER	Query record set capability	NET_CFG_CAP_RECORDFINDER_INFO

Step 4 After completing this process, call **NETClient.Logout** to log out of device.

Step 5 After using SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.2.3.1.4 Sample Code

```
#region Query access control capability
this.button_Query.Enabled = false;
textBox_Caps.Text = "";
textBox_Version.Text = "";

int nCount = 0;
CFG_CAP_ACCESSCONTROL info = new CFG_CAP_ACCESSCONTROL();
object obj = info;
string strCommand =
SDK_DEVCONFIG_CMD.NET_CFG_CAP_CMD_ACCESSCONTROLMANAGER;
try
{
    bool bQuery = NETClient.QueryNewSystemInfo(loginID, -1, strCommand, ref
obj, typeof(CFG_CAP_ACCESSCONTROL), 3000);
    if (bQuery)
    {
        nCount =
((CFG_CAP_ACCESSCONTROL)obj).nAccessControlGroups;
        textBox_Caps.Text = "Access control capability:" +
System.Environment.NewLine + "Number of access control devices=" + nCount.ToString() +
System.Environment.NewLine + System.Environment.NewLine;
        this.button_Query.Enabled = true;
    }
}
```

```

    }
    catch (NETClientExcetion ex)
    {
        Console.WriteLine("GetAccessCount error:" + ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine("GetAccessCount error:" + ex.Message);
    }
    #endregion

    #region Query record capability
    NET_CFG_CAP_RECORDFINDER_INFO RecordInfo = new
NET_CFG_CAP_RECORDFINDER_INFO();
    obj = RecordInfo;
    strCommand =
SDK_DEVCONFIG_CMD.NET_CFG_CAP_CMD_RECORDFINDER;
    try
    {
        bool bQuery = NETClient.QueryNewSystemInfo(loginID, 0, strCommand, ref
obj, typeof(NET_CFG_CAP_RECORDFINDER_INFO), 3000);
        if (bQuery)
        {
            int nMaxPageSize =
((NET_CFG_CAP_RECORDFINDER_INFO)obj).nMaxPageSize;
            textBox_Caps.Text += "Record set query capability:" +
System.Environment.NewLine + "Max number of records per page=" +
nMaxPageSize.ToString() + System.Environment.NewLine + System.Environment.NewLine;
            this.button_Query.Enabled = true;
        }
    }
    catch (NETClientExcetion ex)
    {
        Console.WriteLine("GetAccessCount error:" + ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine("GetAccessCount error:" + ex.Message);
    }
    #endregion

```

```

#region Query log service capability
NET_CFG_CAP_LOG LogInfo = new NET_CFG_CAP_LOG();
obj = LogInfo;
strCommand = SDK_DEVCONFIG_CMD.NET_CFG_CAP_CMD_LOG;
try
{
    bool bQuery = NETClient.QueryNewSystemInfo(loginID, 0, strCommand, ref
obj, typeof(NET_CFG_CAP_LOG), 3000);

    if (bQuery)
    {
        int dwMaxLogItems = (int)((NET_CFG_CAP_LOG)obj).dwMaxLogItems;
        int dwMaxPageItems =
(int)((NET_CFG_CAP_LOG)obj).dwMaxPageItems;
        string strSupportStartNo = "";
        if (((NET_CFG_CAP_LOG)obj).bSupportStartNo)
        {
            strSupportStartNo = "Yes";
        }
        else
        {
            strSupportStartNo = "No";
        }

        string strSupportTypeFilter = "";
        if (((NET_CFG_CAP_LOG)obj).bSupportTypeFilter)
        {
            strSupportTypeFilter = "Yes";
        }
        else
        {
            strSupportTypeFilter = "No";
        }

        string strSupportTimeFilter = "";
        if (((NET_CFG_CAP_LOG)obj).bSupportTimeFilter)
        {
            strSupportTimeFilter = "Yes";
        }
        else

```



```

        {
            strSupportTimeFilter = "No";
        }
        textBox_Caps.Text += "Log service capability:" +
System.Environment.NewLine + "Max number of Logs =" + dwMaxLogItems.ToString() +
System.Environment.NewLine;
        textBox_Caps.Text += "Max number of logs per page=" +
dwMaxPagelItems.ToString() + System.Environment.NewLine;
        textBox_Caps.Text += "Support start No. or not=" + strSupportStartNo +
System.Environment.NewLine;
        textBox_Caps.Text += "Support type filter or not=" +
strSupportTypeFilter + System.Environment.NewLine;
        textBox_Caps.Text += "Support time filter or not=" +
strSupportTimeFilter + System.Environment.NewLine;
        this.button_Query.Enabled = true;
    }
}
catch (NETClientExcetion ex)
{
    Console.WriteLine("GetAccessCount error:" + ex.Message);
}
catch (Exception ex)
{
    Console.WriteLine("GetAccessCount error:" + ex.Message);
}
}
#endregion

```

2.2.3.2 Viewing Device Version and MAC Address

2.2.3.2.1 Introduction

To view device version and MAC address, you need to issue a command through SDK to access control device, so as to get device information such as serial number, version and MAC address.

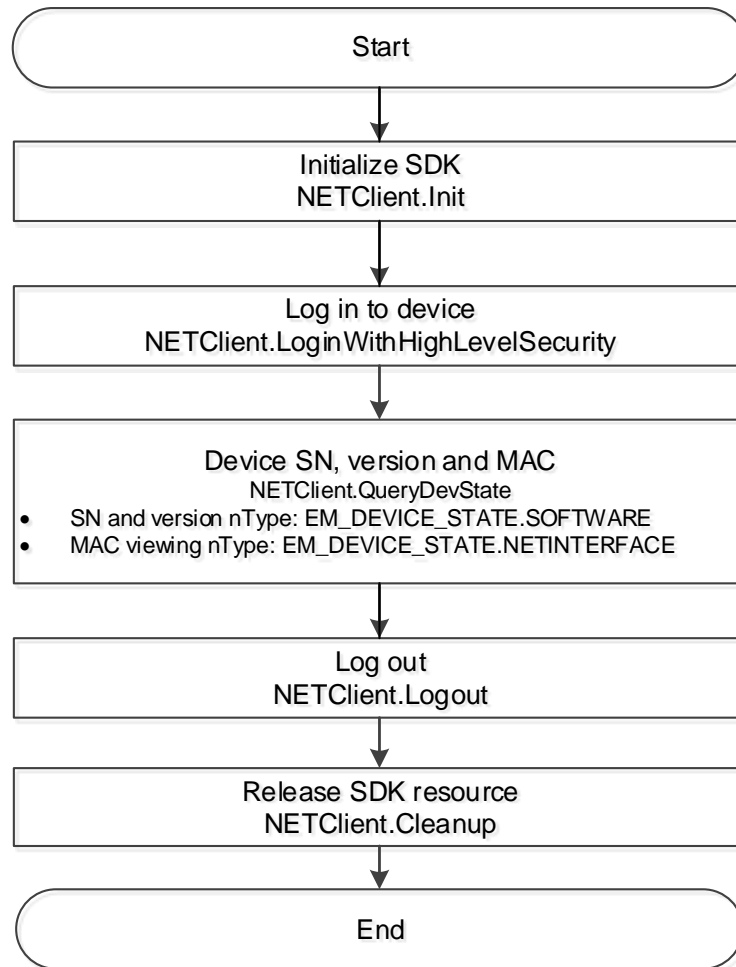
2.2.3.2.2 Interface Overview

Table 2-11 Description of interfaces for viewing device version and MAC

Interface	Description
NETClient.QueryDevState	Query device status (serial number, software version, compiling time, MAC address).

2.2.3.2.3 Process Description

Figure 2-12 Device information viewing



Procedure

Step 1 Call **NETClient.Init** to initialize SDK.

Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to device.

Step 3 Call **NETClient.QueryDevState** to query access control device information such as serial number, version and MAC.

Table 2-12 Description and structure of nType

nType	Description	pBuf
EM_DEVICE_STATE.SOFTWARE	Serial number and version	NET_DEV_VERSION_INFO
EM_DEVICE_STATE.NETINTERFACE	MAC address	NET_DEV_NETINTERFACE_INFO

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

Step 5 After using SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.2.3.2.4 Sample Code

```
#region Query device version info
    NET_DEV_VERSION_INFO VersionInfo = new NET_DEV_VERSION_INFO();
```

```

        object objInfo = VersionInfo;

        bool ret = NETClient.QueryDevState(loginID, EM_DEVICE_STATE.SOFTWARE,
ref objInfo, typeof(NET_DEV_VERSION_INFO), 10000);
        if (!ret)
        {
            MessageBox.Show(NETClient.GetLastError());
            return;
        }
        this.button_Query.Enabled = true;
        VersionInfo = (NET_DEV_VERSION_INFO)objInfo;

        textBox_Version.Text += "Serial number:" + VersionInfo.szDevSerialNo +
System.Environment.NewLine;
        textBox_Version.Text += "Software version:" + VersionInfo.szSoftWareVersion +
System.Environment.NewLine;
        textBox_Version.Text += "Compiling time:" +
((VersionInfo.dwSoftwareBuildDate >> 16) & 0xffff) + "-" +
((VersionInfo.dwSoftwareBuildDate >> 8) & 0xff) + "-" + (VersionInfo.dwSoftwareBuildDate &
0xff) + System.Environment.NewLine;

        // Query MAC address
        NET_DEV_NETINTERFACE_INFO[] stuNetInfo = new
NET_DEV_NETINTERFACE_INFO[64];

        for (int i = 0; i < 64; i++)
        {
            stuNetInfo[i].dwSize = (int)Marshal.SizeOf(stuNetInfo[i].GetType());
        }
        object[] objInfo2 = new object[64];
        for (int i = 0; i < 64; i++)
        {
            objInfo2[i] = stuNetInfo[i];
        }
        bool Macret = NETClient.QueryDevState(loginID,
(int)EM_DEVICE_STATE.NETINTERFACE, ref objInfo2,
typeof(NET_DEV_NETINTERFACE_INFO), 5000);
        if (!Macret)
        {
            MessageBox.Show(NETClient.GetLastError());
            return;

```

```

    }
    for (int i = 0; i < objInfo2.Length; i++)
    {
        stuNetInfo[i] = (NET_DEV_NETINTERFACE_INFO)objInfo2[i];
    }
    textBox_Version.Text += "MAC" + stuNetInfo[0].szMAC +
System.Environment.NewLine;
#endregion

```

2.2.4 Network Setting

2.2.4.1 IP Settings

2.2.4.1.1 Introduction

To configure IP address, you need to call SDK interface to get and configure device information, including IP address, subnet mask, and default gateway.

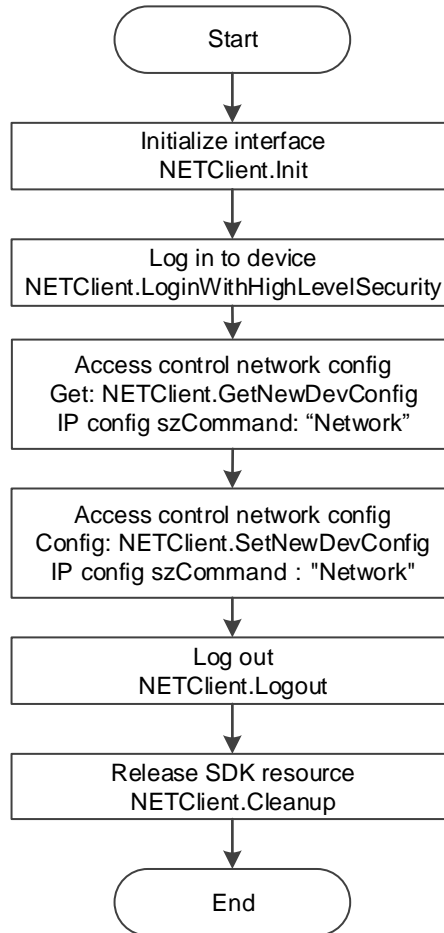
2.2.4.1.2 Interface Overview

Table 2-13 Description of IP setting interface

Interface	Description
NETClient.GetNewDevConfig	Query config information
NETClient.SetNewDevConfig	Set config information

2.2.4.1.3 Process Description

Figure 2-13 IP setting



Procedure

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to device.
- Step 3 Call **NETClient.GetNewDevConfig** to query IP settings.
- szCommand: "Network".
 - pBuf: **CFG_NETWORK_INFO**.
- Step 4 Call **NETClient.SetNewDevConfig** to configure IP settings.
- szCommand: "Network".
 - pBuf: **CFG_NETWORK_INFO**.
- Step 5 After completing this process, call **NETClient.Logout** to log out of device.
- Step 6 After using SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.2.4.1.4 Sample Code

```
// Get IP settings
CFG_NETWORK_INFO cfg = new CFG_NETWORK_INFO();

public CFG_NETWORK_INFO GetConfig_Network()
{
```

```

        try
        {
            object objTemp = new object();
            bool bRet = NETClient.GetNewDevConfig(loginID, -1, "Network", ref
objTemp, typeof(CFG_NETWORK_INFO), 5000);
            if (!bRet)
            {
                MessageBox.Show(NETClient.GetLastError());
                return cfg;
            }
            cfg = (CFG_NETWORK_INFO)objTemp;
        }
        catch (NETClientExcetion nex)
        {
            MessageBox.Show(nex.Message);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        return cfg;
    }

```

```

public bool SetConfig_Network(CFG_NETWORK_INFO cfg)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID, -1, "Network", (object)cfg,
typeof(CFG_NETWORK_INFO), 5000);
    }
    catch (NETClientExcetion nex)
    {
        Console.WriteLine(nex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

```

```

return bRet;
}

```

2.2.4.2 Auto Register Config

2.2.4.2.1 Introduction

To configure auto register, you need to call SDK interface to configure auto register information of the device, including auto register enabling, device ID, and server.

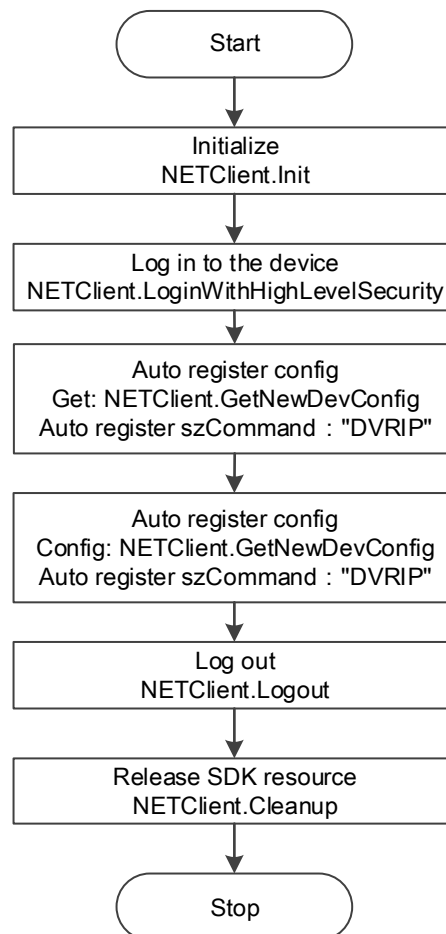
2.2.4.2.2 Interface Overview

Table 2-14 Description of interfaces for setting auto register

Interface	Description
NETClient.GetNewDevConfig	Query config information.
NETClient.SetNewDevConfig	Set config information.

2.2.4.2.3 Process Description

Figure 2-14 Auto register setting



Process

Step 1 Call **NETClient.Init** to initialize SDK.

Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.

Step 3 Auto register config.

- Call **NETClient.GetNewDevConfig** to query the auto register config.
 - ◇ szCommand: DVRIP.
 - ◇ pBuf: NET_CFG_DVRIP_INFO.
- Call **NETClient.SetNewDevConfig** to set the auto register config.
 - ◇ szCommand: DVRIP.
 - ◇ pBuf: NET_CFG_DVRIP_INFO.

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

Step 5 After using all SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.2.4.2.4 Sample Code

```
// get auto register network config
NET_CFG_DVRIP_INFO cfg_Dvrip = new NET_CFG_DVRIP_INFO();
public NET_CFG_DVRIP_INFO GetConfig_Dvrip()
{
    try
    {
        object objTemp = new object();
        bool bRet = NETClient.GetNewDevConfig(loginID, -1, "DVRIP", ref objTemp,
        typeof(NET_CFG_DVRIP_INFO), 5000);
        if (bRet)
        {
            cfg_Dvrip = (NET_CFG_DVRIP_INFO)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return cfg_Dvrip;
}
```



```

public bool SetConfig_Dvrip(NET_CFG_DVRIP_INFO cfg_Dvrip)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID, -1, "DVRIP", (object)cfg_Dvrip,
typeof(NET_CFG_DVRIP_INFO), 5000);
    }
    catch (NETClientExcetion nex)
    {
        Console.WriteLine(nex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return bRet;
}

```

2.2.5 Device Time Getting and Setting

2.2.5.1 DeviceTime Getting and Setting

2.2.5.1.1 Introduction

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

2.2.5.1.2 Interface Overview

Table 2-15 Description of time setting interfaces

Interface	Description
NETClient.QueryDeviceTime	Get the current time of the device.
NETClient.SetupDeviceTime	Configure the current time of the device.

2.2.5.1.3 Process Description

Figure 2-15 Time getting

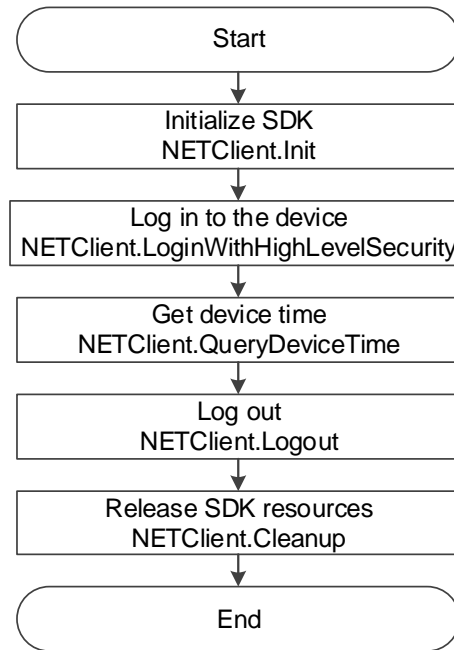
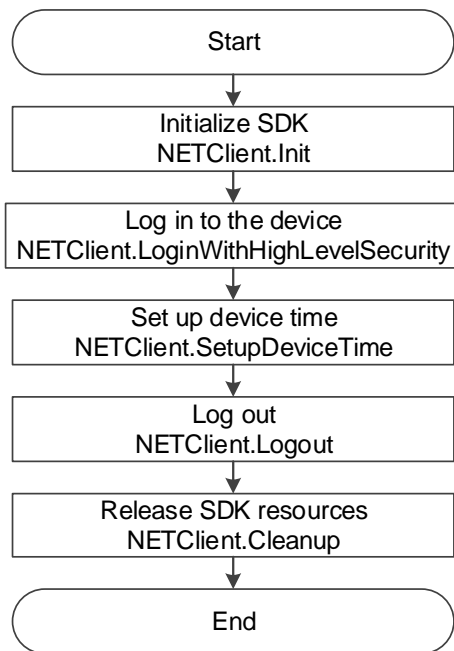


Figure 2-16 Time setting



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.SetupDeviceTime** to set the access control time.
- Step 4 After completing this process, call **NETClient.Logout** to log out of the device.
- Step 5 After using all SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.2.5.1.4 Sample Code

```
#region Get Deice Time get device time
NET_TIME stuInfo = new NET_TIME();

bool ret = NETClient.QueryDeviceTime(loginID, ref stuInfo, 5000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}
dateTimePicker_DevTime.Value = stuInfo.ToDateTime();
MessageBox.Show("Get Success(get successfully)");
#endregion

#region Set Device Timeconfigure device time
NET_TIME stuSet = new NET_TIME();
stuSet.dwYear = (uint)dateTimePicker_DevTime.Value.Year;
stuSet.dwMonth = (uint)dateTimePicker_DevTime.Value.Month;
stuSet.dwDay = (uint)dateTimePicker_DevTime.Value.Day;
stuSet.dwHour = (uint)dateTimePicker_DevTime.Value.Hour;
stuSet.dwMinute = (uint)dateTimePicker_DevTime.Value.Minute;
stuSet.dwSecond = (uint)dateTimePicker_DevTime.Value.Second;

bool ret = NETClient.SetupDeviceTime(loginID, stuSet);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}
MessageBox.Show("Set Success(configured successfully)");
#endregion
```

2.2.5.2 NTP Server and Time Zone Setting

2.2.5.2.1 Introduction

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

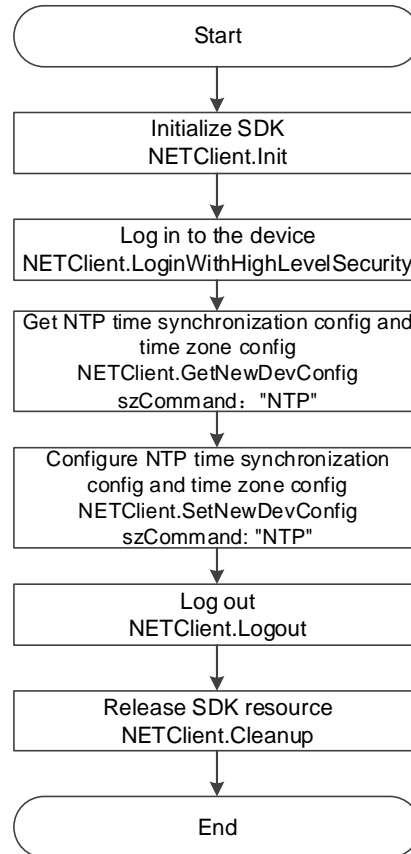
2.2.5.2.2 Interface Overview

Table 2-16 Description of NTP server and time zone setting interfaces

Interface	Description
NETClient.GetNewDevConfig	Query config information.
NETClient.SetNewDevConfig	Configure config information.

2.2.5.2.3 Process Description

Figure 2-17 NTP server and time zone setting



Process

- Step 1** Call **NETClient.Init** to initialize SDK.
- Step 2** Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3** Call **NETClient.GetNewDevConfig** to query the access NTP time sync and time zone config.
- szCommand: NTP.
 - pBuf: NET_CFG_NTP_INFO.
- Step 4** Call **NETClient.SetNewDevConfig** to set the access NTP time sync and time zone config.
- szCommand: NTP.
 - pBuf: NET_CFG_NTP_INFO.
- Step 5** After completing this process, call **NETClient.Logout** to log out of the device.
- Step 6** After using all SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.2.5.2.4 Sample Code

```
// NTP config
NET_CFG_NTP_INFO cfg = new NET_CFG_NTP_INFO();
```

```

public NET_CFG_NTP_INFO GetConfig_NTP()
{
    try
    {
        object objTemp = new object();
        bool bRet = NETClient.GetNewDevConfig(loginID, -1, "NTP", ref objTemp,
typeof(NET_CFG_NTP_INFO), 5000);
        if (bRet)
        {
            cfg = (NET_CFG_NTP_INFO)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return cfg;
}

public bool SetConfig_NTP(NET_CFG_NTP_INFO cfg)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID, -1, "NTP", (object)cfg,
typeof(NET_CFG_NTP_INFO), 5000);
    }
    catch (NETClientExcetion nex)
    {
        Console.WriteLine(nex.Message);
    }
}

```

```

        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
        return bRet;
    }

```

DST Setting

2.2.5.3 DST Settings

2.2.5.3.1 Introduction

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

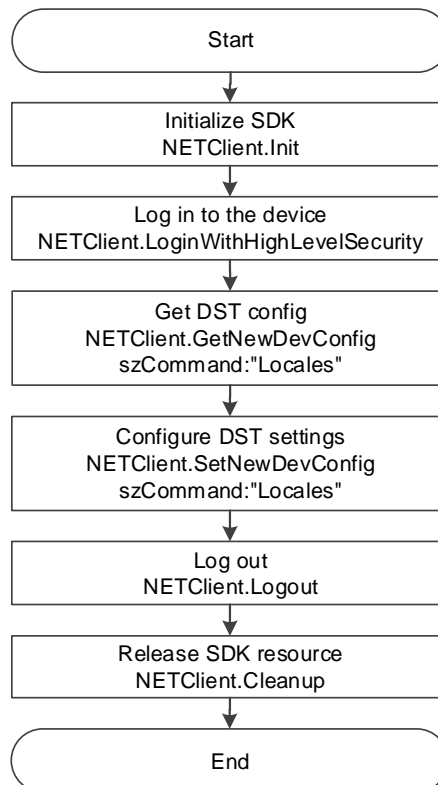
2.2.5.3.2 Interface Overview

Table 2-17 Description of DST setting interfaces

Interface	Description
NETClient. GetNewDevConfig	Query config information.
NETClient. SetNewDevConfig	Set config information.

2.2.5.3.3 Process Description

Figure 2-18 DST setting



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.GetNewDevConfig** to query the access control DST config.
- szCommand: Locales.
 - pBuf: NET_AV_CFG_Locales.
- Step 4 Call **NETClient.SetNewDevConfig** to set the access control DST config.
- szCommand: Locales.
 - pBuf: NET_AV_CFG_Locales.
- Step 5 After completing this process, call **NETClient.Logout** to log out of the device.
- Step 6 After using all SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.2.5.3.4 Sample Code

```
// Locales config

public bool GetConfig_Locales()
{
    bool bRet = false;
    try
    {
        cfg_Locales.stuDstStart.nStructSize =
Marshal.SizeOf(typeof(AV_CFG_DSTTime));
        cfg_Locales.stuDstEnd.nStructSize =
Marshal.SizeOf(typeof(AV_CFG_DSTTime));
        object objTemp = (object)cfg_Locales;
        bRet = NETClient.GetNewDevConfig(loginID, -1, "Locales", ref objTemp,
typeof(NET_AV_CFG_Locales), 5000);
        if (bRet)
        {
            cfg_Locales = (NET_AV_CFG_Locales)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
}
```

```

        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        return bRet;
    }

    public bool SetConfig_Locales(NET_AV_CFG_Locales cfg_Locales)
    {
        bool bRet = false;
        try
        {
            {
                bRet = NETClient.SetNewDevConfig(loginID, -1, "Locales",
(object)cfg_Locales, typeof(NET_AV_CFG_Locales), 5000);
            }
            catch (NETClientExcetion nex)
            {
                Console.WriteLine(nex.Message);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            return bRet;
        }
    }

```

2.2.6 Maintenance Config

2.2.6.1 Changing Login Password

2.2.6.1.1 Introduction

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

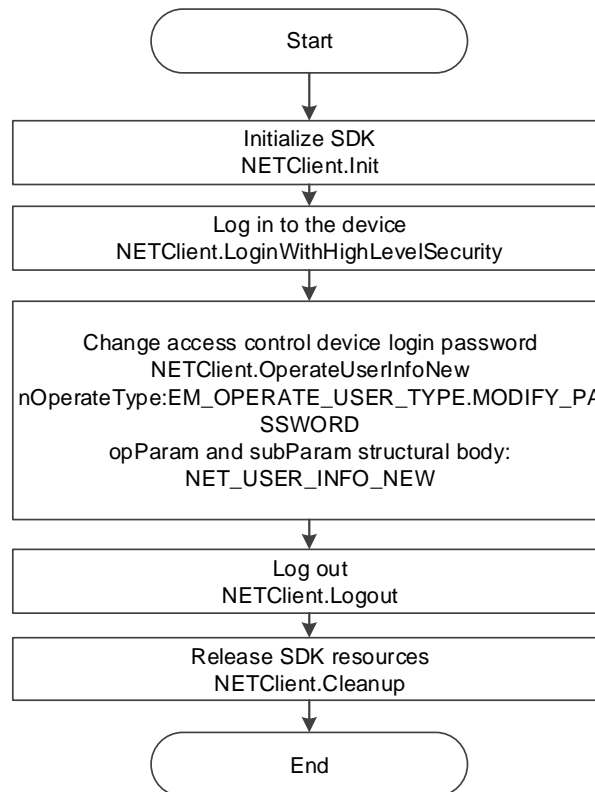
2.2.6.1.2 Interface Overview

Table 2-18 Description of interfaces for changing login password

Interface	Description
NETClient.OperateUserInfoNew	Operate device user info.

2.2.6.1.3 Process Description

Figure 2-19 Maintenance config



Process

- Step 1 Call **NETClient. Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.OperateUserInfoNew** to operate user info to change the device login password. **nOperateType** is **EM_OPERATE_USER_TYPE.MODIFY_PASSWORD**, opParam and subParam structural body is **NET_USER_INFO_NEW**.
- Step 4 After completing this process, call **NETClient. Logout** to log out of the device.
- Step 5 After using all SDK functions, call **NETClient. Cleanup** to release SDK resources.

2.2.6.1.4 Sample Code

```
// change device login password
NET_USER_INFO_NEW userInfo = new NET_USER_INFO_NEW();
userInfo.dwSize = (uint)Marshal.SizeOf(typeof(NET_USER_INFO_NEW));
userInfo.name = textBox_User.Text.Trim();
userInfo.passWord = textBox_OldPasswd.Text.Trim();
IntPtr inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_USER_INFO_NEW)));
NET_USER_INFO_NEW stuModifyInfo = new NET_USER_INFO_NEW();
stuModifyInfo.dwSize = (uint)Marshal.SizeOf(typeof(NET_USER_INFO_NEW));
stuModifyInfo.passWord = textBox_NewPasswd.Text.Trim();
```

```

        IntPtr      insubPtr      =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_USER_INFO_NEW)));

        try
        {

            Marshal.StructureToPtr(userInfo, IntPtr, true);

            Marshal.StructureToPtr(stuModifyInfo, insubPtr, true);

            bool      ret      =      NETClient.OperateUserInfoNew(loginID,
EM_OPERATE_USER_TYPE.MODIFY_PASSWORD, insubPtr, IntPtr, 10000);
            if (!ret)
            {
                MessageBox.Show( NETClient.GetLastError());
                return;
            }
        }
        finally
        {
            Marshal.FreeHGlobal(IntPtr);
            Marshal.FreeHGlobal(insubPtr);
        }
        MessageBox.Show("Modified password successfully.");

```

2.2.6.2 Restart

2.2.6.2.1 Introduction

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

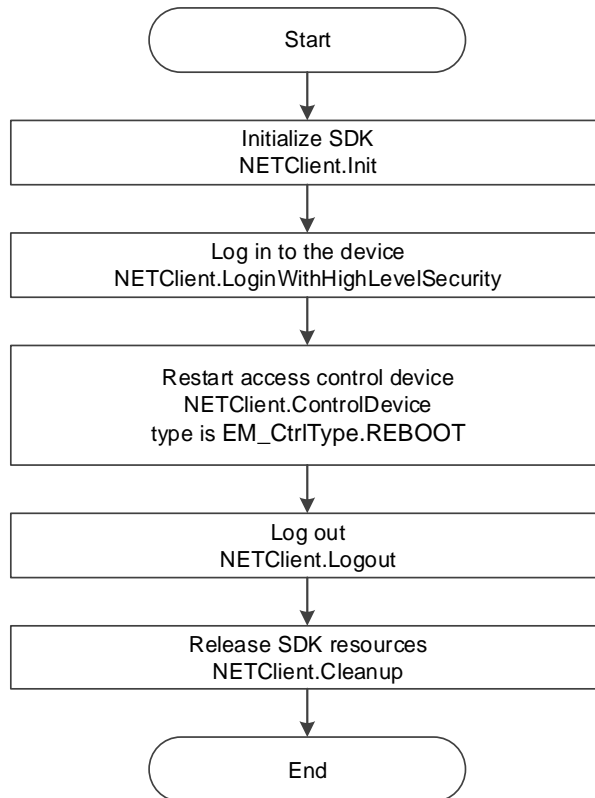
2.2.6.2.2 Interface Overview

Table 2-19 Description of device restart interface

Interface	Description
NETClient.ControlDevice	Device control.

2.2.6.2.3 Process Description

Figure 2-20 Device restart



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.ControlDevice** to restart the device.
Type: EM_CtrlType.REBOOT.
- Step 4 After completing this process, call **NETClient. Logout** to log out of the device.
- Step 5 After using all SDK functions, call **NETClient. Cleanup** to release SDK resources.

2.2.6.2.4 Sample Code

```
#region Reboot Device restart the device
IntPtr inPtr = IntPtr.Zero;
bool ret = NETClient.ControlDevice(loginID, EM_CtrlType.REBOOT, inPtr,
10000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}
this.Hide();
#endregion
```

2.2.6.3 Restoring the Factory Settings

2.2.6.3.1 Introduction

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

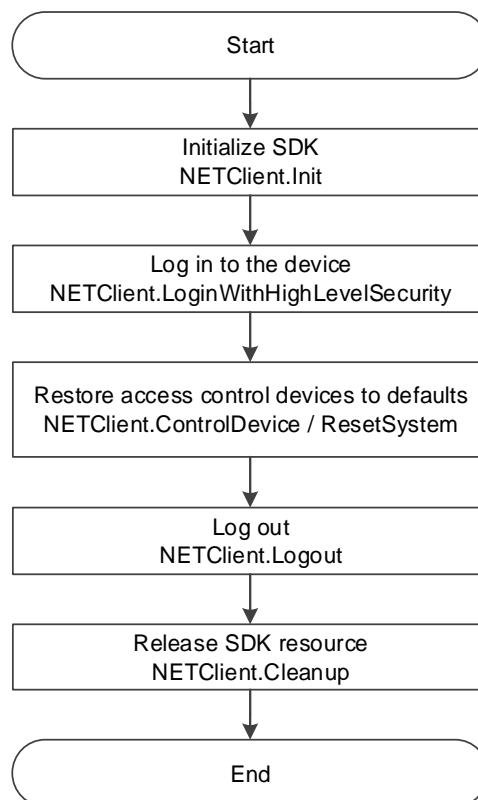
2.2.6.3.2 Interface Overview

Table 2-20 Description of interfaces for restoring factory defaults

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

2.2.6.3.3 Process Description

Figure 2-21 Factory defaults restoring



Process

Step 1 Call **NETClient.Init** to initialize SDK.

Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.

Step 3 Call **NETClient.ResetSystem** to control the device (controller or all-in-one fingerprint machine) to restore factory defaults. Or call **ControlDevice** function to control the device (controller or all-in-one fingerprint machine) to restore factory defaults.

Type: EM_CtrlType. RESTOREDEFAULT.

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

Step 5 After using all SDK functions, call **NETClient Cleanup** to release SDK resources.

2.2.6.3.4 Sample Code

```
#region Reset Device restore factory defaults
NET_IN_RESET_SYSTEM stuResetIn = new NET_IN_RESET_SYSTEM();
stuResetIn.dwSize
(uint)Marshal.SizeOf(typeof(NET_IN_USERINFO_START_FIND));

NET_OUT_RESET_SYSTEM stuResetOut = new
NET_OUT_RESET_SYSTEM();
stuResetOut.dwSize
(uint)Marshal.SizeOf(typeof(NET_OUT_USERINFO_START_FIND));
bool nRet = NETClient.ResetSystem(loginID, ref stuResetIn, ref stuResetOut,
5000);
if (!nRet)
{
    IntPtr inPtr = IntPtr.Zero;
    inPtr
    Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RESTORE_TEMPSTRUCT)));
    NET_RESTORE_TEMPSTRUCT temp = new
    NET_RESTORE_TEMPSTRUCT() { value = NET_RESTORE.ALL };
    Marshal.StructureToPtr(temp, inPtr, true);
    bool ret = NETClient.ControlDevice(loginID,
    EM_CtrlType.RESTOREDEFAULT, inPtr, 10000);
    if (!ret)
    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }
}
this.Hide();
#endregion
```

2.2.6.4 Device Upgrade

2.2.6.4.1 Introduction

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

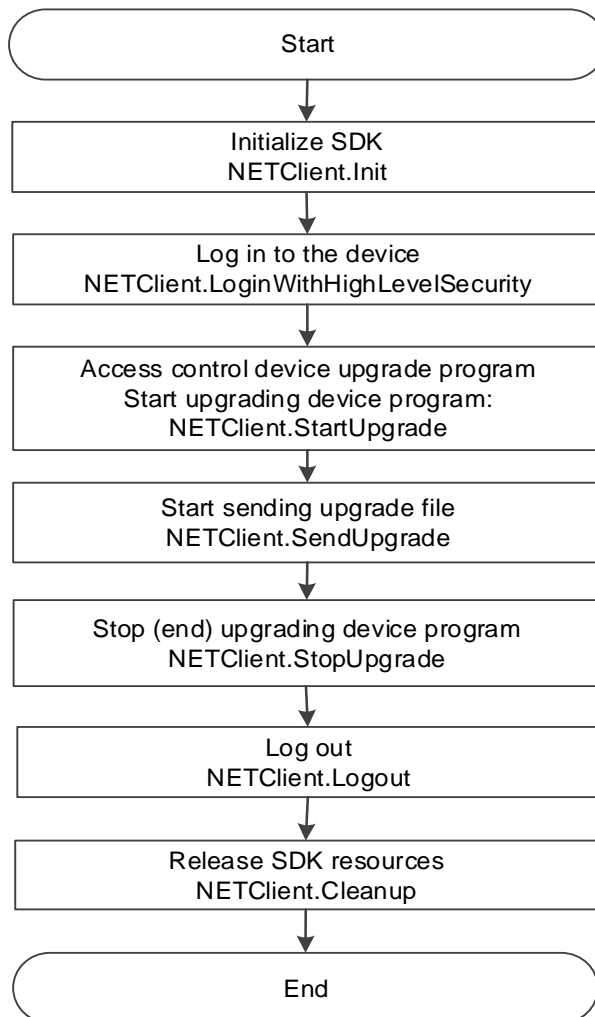
2.2.6.4.2 Interface Overview

Table 2-21 Description of device upgrade interfaces

Interface	Description
NETClient.StartUpgrade	Start upgrading device program—extension.
NETClient.SendUpgrade	Start sending upgrade file.
NETClient.StopUpgrade	Stop upgrading.

2.2.6.4.3 Process Description

Figure 2-22 Device upgrade



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.StartUpgrade** to start upgrading the device program.
- Step 4 Call **NETClient.SendUpgrade** to send the device upgrade file.
- Step 5 Call **NETClient.StopUpgrade** to stop/end upgrading the device program.
- Step 6 After completing this process, call **NETClient.Logout** to log out of the device.
- Step 7 After using all SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.2.6.4.4 Sample Code

```
#region upgrade device update device
if (textBox_Path.Text == null || textBox_Path.Text == "")
{
    MessageBox.Show("please choose a upgrade packet file.(please select
remote update package.)");
    return;
}

m_UpgradeID = NETClient.StartUpgrade(m_LoginID,
EM_UPGRADE_TYPE.BIOS_TYPE, textBox_Path.Text, m_UpgradeCallBack, IntPtr.Zero);
if (IntPtr.Zero != m_UpgradeID)
{
    bool bRet = NETClient.SendUpgrade(m_UpgradeID);
    if (!bRet)
    {
        MessageBox.Show(NETClient.GetLastError());
        button_Upgrade.Enabled = true;
    }
}
else
{
    MessageBox.Show(NETClient.GetLastError());
    button_Upgrade.Enabled = true;
}
#endregion

bool ret = NETClient.StopUpgrade(m_UpgradeID);
if (ret)
{
    m_UpgradeID = IntPtr.Zero;
    button_Upgrade.Enabled = true;
}
else
{
    MessageBox.Show(NETClient.GetLastError());
}
```

2.2.6.5 Auto Maintenance

2.2.6.5.1 Introduction

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

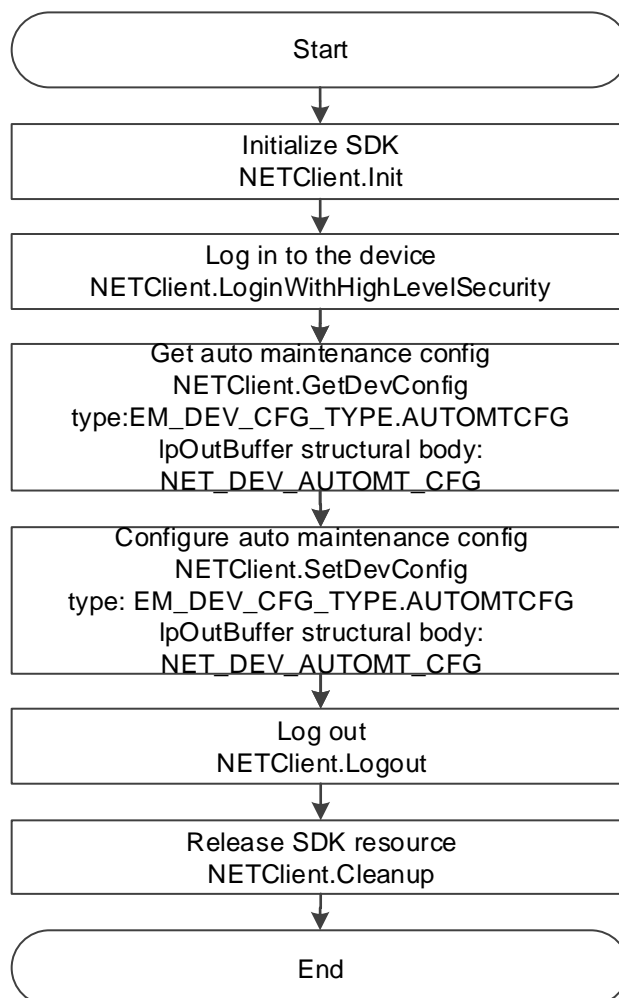
2.2.6.5.2 Interface Overview

Table 2-22 Description of auto maintenance interfaces

Interface	Description
NETClient.GetDevConfig	Query config information.
NETClient.SetDevConfig	Set config information.

2.2.6.5.3 Process Description

Figure 2-23 Auto maintenance



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.GetDevConfig** to query the access auto maintenance info.

- type: EM_DEV_CFG_TYPE.AUTOMTCFG.
- lpOutBuffer structural body: NET_DEV_AUTOMT_CFG.

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

- type is EM_DEV_CFG_TYPE.AUTOMTCFG.
- lpOutBuffer structural body NET_DEV_AUTOMT_CFG.

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

Step 6 After using all SDK functions, call **NETClient. Cleanup** to release SDK resources.

2.2.6.5.4 Sample Code

```

NET_DEV_AUTOMT_CFG cfg_AutoMT = new NET_DEV_AUTOMT_CFG();
public NET_DEV_AUTOMT_CFG GetDevConfig_AutoMT()
{
    uint ret = 0;
    IntPtr inPtr = IntPtr.Zero;
    try
    {
        inPtr = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_DEV_AUTOMT_CFG)));
        Marshal.StructureToPtr(cfg_AutoMT, inPtr, true);
        bool result = NETClient.GetDevConfig(loginID,
        EM_DEV_CFG_TYPE.AUTOMTCFG, 0, inPtr,
        (uint)Marshal.SizeOf(typeof(NET_DEV_AUTOMT_CFG)), ref ret, 5000);
        if (result && ret == (uint)Marshal.SizeOf(typeof(NET_DEV_AUTOMT_CFG)))
        {
            cfg_AutoMT = (NET_DEV_AUTOMT_CFG)Marshal.PtrToStructure(inPtr,
            typeof(NET_DEV_AUTOMT_CFG));
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        Marshal.FreeHGlobal(inPtr);
    }
    return cfg_AutoMT;
}

```

```

    }

    #region Set auto matrix config configure auto maintenance config
    GetDevConfig_AutoMT();
    cfg_AutoMT.byAutoRebootDay = (byte)comboBox_RebootDay.SelectedIndex;
    cfg_AutoMT.byAutoRebootTime = (byte)comboBox_RebootTime.SelectedIndex;
    IntPtr inPtr = IntPtr.Zero;
    inPtr
    Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_DEV_AUTOMT_CFG)));
    Marshal.StructureToPtr(cfg_AutoMT, inPtr, true);
    bool result = NETClient.SetDevConfig(loginID,
    EM_DEV_CFG_TYPE.AUTOMTCFG, 0, inPtr,
    (uint)Marshal.SizeOf(typeof(NET_DEV_AUTOMT_CFG)), 5000);
    if (!result)
    {
        MessageBox.Show(NETClient.GetLastError());
    }
    MessageBox.Show("Set successfully.(configured successfully)");
    #endregion

```

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).

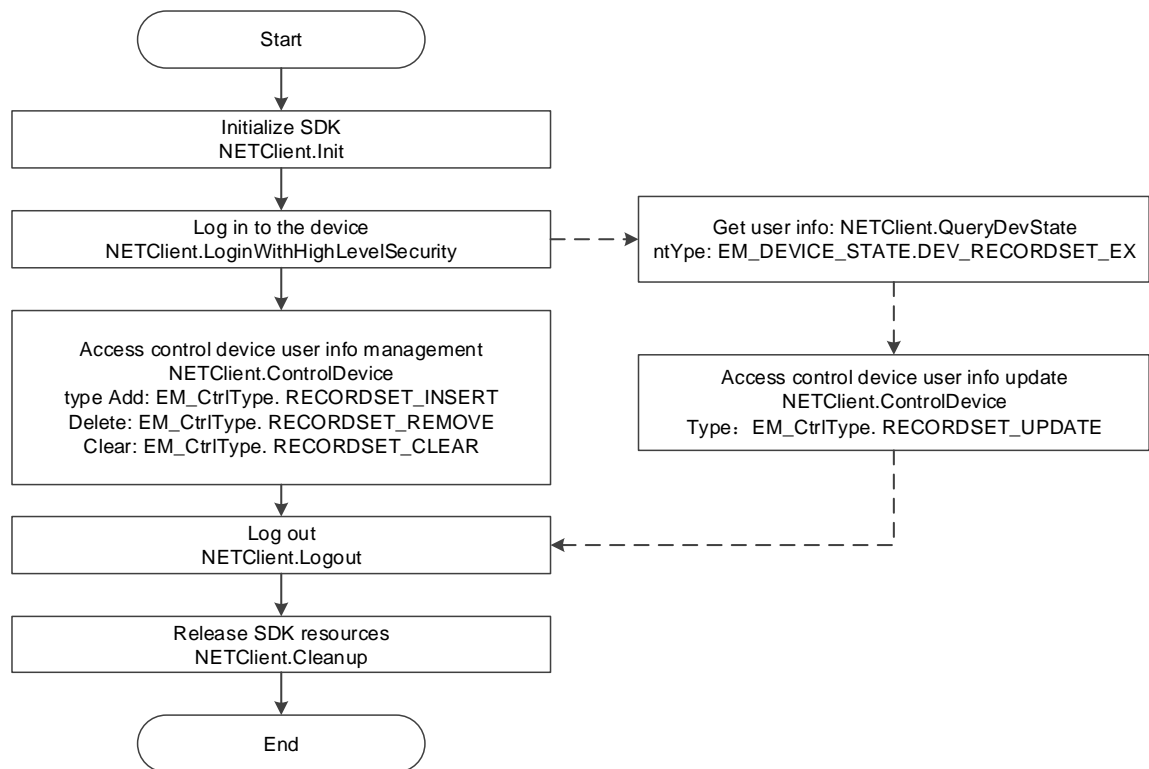
2.2.7.2 Interface Overview

Table 2-23 Description of personnel information interfaces

Interface	Description
NETClient.ControlDevice	Control device.
NETClient.QueryDevState	Query device status.

2.2.7.3 Process Description

Figure 2-24 User information management



Process

Step 1 Call **NETClient.Init** to initialize SDK.

Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.

Step 3 Call **NETClient.ControlDevice** to operate holiday information.

Table 2-24 Description and structure of user info management type

Type	Description	emType	Param
EM_CtrlType. RECORDSET_INSERT	Add user info	EM_NET_RE CORD_TYPE.	• NET_CTRL_RECORDSE T_INSERT_PARAM
EM_CtrlType. RECORDSET_INSERT EX	Add user info (with fingerprint)	ACCESSCTL CARD	• NET_RECORDSET_ACC ESS_CTL_CARD
EM_CtrlType. RECORDSET_REMOV E	Delete user info	EM_NET_RE CORD_TYPE.	• NET_CTRL_RECORDSE T_PARAM
		ACCESSCTL CARD	• NET_RECORDSET_ACC ESS_CTL_CARD
EM_CtrlType. RECORDSET_CLEAR	Clear user info	EM_NET_RE CORD_TYPE.	NET_CTRL_RECORDSET_PA RAM
		ACCESSCTL CARD	

Step 4 Call the **NETClient.QueryDevState** interface to get user information.

Table 2-25 Description and structure of user info getting type

Type	Description	emType	Param
EM_DEVICE_STATE. DEV_RECORDSET_EX	Get user info	ACCESSCTL CARD	NET_CTRL_RECORDSET_PA RAM NET_RECORDSET_ACCESS _CTL_CARD

Step 5 Call the **NETClient.ControlDevice** to update user information.

Table 2-26 Description and structure of user info updating type

Type	Description	emType	Param
EM_CtrlType. RECORDSET_UPDATE	Update user info	ACCESSCTL CARD	<ul style="list-style-type: none"> NET_CTRL_RECORDSE T_PARAM NET_RECORDSET_ACC ESS_CTL_CARD
EM_CtrlType. RECORDSET_UPDATE EX	Update user info (with fingerprint)		

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

Step 7 After using all SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.2.7.4 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. For details, see "2.2.10.5 Unlock Password."
- Valid number of times: Only guest users can set this field.
- Whether it is first card: Select as needed. For the configuration method of the first card, see "2.2.10.1 Unlock at Designated Intervals and First Card Unlock."

2.2.7.5 Sample Code

```
#region Get Card Info get card record
if (textBox_RecNo.Text == null || textBox_RecNo.Text == "")
{
    MessageBox.Show("Please input Rec Number (please enter recird set
number)");
    return;
}
NET_CTRL_RECORDSET_PARAM inp = new
NET_CTRL_RECORDSET_PARAM();
```

```

inp.dwSize = (uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM));
inp.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;

NET_RECORDSET_ACCESS_CTL_CARD info = new
NET_RECORDSET_ACCESS_CTL_CARD();

info.dwSize =
(uint)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));
info.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());
IntPtr infoPtr = IntPtr.Zero;

try
{
    infoPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD)));
    Marshal.StructureToPtr(info, infoPtr, true);
    inp.pBuf = infoPtr;
    inp.nBufLen =
Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));
    object objInp = inp;
    bool ret = NETClient.QueryDevState(loginID,
(int)EM_DEVICE_STATE.DEV_RECORDSET_EX, ref objInp,
typeof(NET_CTRL_RECORDSET_PARAM), 10000);
    if (!ret)
    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }
    inp = (NET_CTRL_RECORDSET_PARAM)objInp;
    info =
(NET_RECORDSET_ACCESS_CTL_CARD)Marshal.PtrToStructure(inp.pBuf,
typeof(NET_RECORDSET_ACCESS_CTL_CARD));

    // dateTimePicker_CreateTime.Value = info.stuCreateTime.ToDateTime();
    textBox_CardNo.Text = info.szCardNo;
    textBox_UserID.Text = info.szUserID;
    comboBox_CardStatus.SelectedIndex = (int)info.emStatus + 1;
    comboBox_CardType.SelectedIndex = (int)info.emType + 1;
    textBox_CardPwd.Text = info.szPsw;
    m_SelectDoorsAry = info.nNewDoors;
    m_SelectTimeAry = info.nNewTimeSectionNo;
    textBox_UseTimes.Text = info.nUseTime.ToString();
}

```

```

        dateTimePicker_ValidStart.Value = info.stuValidStartTime.ToDateTime();
        dateTimePicker_ValidEnd.Value = info.stuValidEndTime.ToDateTime();

        checkBox_First.Checked = info.bFirstEnter;

        int nCtlType = comboBox_OperateType.SelectedIndex;

        if (2 == nCtlType)
        {
            OnChangeUIState(7);
        }
        else
        {
            OnChangeUIState(8);
        }

        //  MessageBox.Show("Query Success");
    }
    finally
    {
        //  Marshal.FreeHGlobal(infoPtr);
    }
    #endregion

    #region Insert record add card record
    NET_CTRL_RECORDSET_INSERT_PARAM stuInfo = new
    NET_CTRL_RECORDSET_INSERT_PARAM();
    stuInfo.dwSize =
    (uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_PARAM));

    stuInfo.stuCtrlRecordSetInfo.dwSize =
    (uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_IN));
    stuInfo.stuCtrlRecordSetInfo.emType =
    EM_NET_RECORD_TYPE.ACCESSCTLCARD;

    stuInfo.stuCtrlRecordSetInfo.nBufLen =
    (int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));

    stuInfo.stuCtrlRecordSetResult.dwSize =
    (uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_OUT));

```

```

        m_stuInfo.stuCreateTime.dwYear = (uint)DateTime.Now.Year;
        m_stuInfo.stuCreateTime.dwMonth = (uint)DateTime.Now.Month;
        m_stuInfo.stuCreateTime.dwDay = (uint)DateTime.Now.Day;
        m_stuInfo.stuCreateTime.dwHour = (uint)DateTime.Now.Hour;
        m_stuInfo.stuCreateTime.dwMinute = (uint)DateTime.Now.Minute;
        m_stuInfo.stuCreateTime.dwSecond = (uint)DateTime.Now.Second;

        m_stuInfo.szCardNo = this.textBox_CardNo.Text.Trim();
        m_stuInfo.szUserID = this.textBox_UserID.Text.Trim();
        m_stuInfo.emStatus =
(EM_ACCESSCTLCARD_STATE)comboBox_CardStatus.SelectedIndex - 1;
        m_stuInfo.emType =
(EM_ACCESSCTLCARD_TYPE)comboBox_CardType.SelectedIndex - 1;
        m_stuInfo.szPsw = this.textBox_CardPwd.Text.Trim();
        m_stuInfo.nUseTime = Convert.ToInt32(textBox_UseTimes.Text.Trim());
        m_stuInfo.bNewDoor = true;

        if (m_selectDoorsList.Count > 0)
        {
            for (int i = 0; i < m_selectDoorsList.Count; i++)
            {
                m_stuInfo.nNewDoors[i] = m_selectDoorsList[i];
            }
        }
        m_stuInfo.nNewDoorNum = m_selectDoorsList.Count;

        if (m_selectTimesList.Count > 0)
        {
            for (int i = 0; i < m_selectTimesList.Count; i++)
            {
                m_stuInfo.nNewTimeSectionNo[i] = m_selectTimesList[i];
            }
        }
        m_stuInfo.nNewTimeSectionNum = m_selectTimesList.Count;

        m_stuInfo.stuValidStartTime.dwYear =
(uint)dateTimePicker_ValidStart.Value.Year;
        m_stuInfo.stuValidStartTime.dwMonth =
(uint)dateTimePicker_ValidStart.Value.Month;

```

```

        m_stuInfo.stuValidStartTime.dwDay =
(uint)dateTimePicker_ValidStart.Value.Day;
        m_stuInfo.stuValidStartTime.dwHour =
(uint)dateTimePicker_ValidStart.Value.Hour;
        m_stuInfo.stuValidStartTime.dwMinute =
(uint)dateTimePicker_ValidStart.Value.Minute;
        m_stuInfo.stuValidStartTime.dwSecond =
(uint)dateTimePicker_ValidStart.Value.Second;

        m_stuInfo.stuValidEndTime.dwYear =
(uint)dateTimePicker_ValidEnd.Value.Year;
        m_stuInfo.stuValidEndTime.dwMonth =
(uint)dateTimePicker_ValidEnd.Value.Month;
        m_stuInfo.stuValidEndTime.dwDay =
(uint)dateTimePicker_ValidEnd.Value.Day;
        m_stuInfo.stuValidEndTime.dwHour =
(uint)dateTimePicker_ValidEnd.Value.Hour;
        m_stuInfo.stuValidEndTime.dwMinute =
(uint)dateTimePicker_ValidEnd.Value.Minute;
        m_stuInfo.stuValidEndTime.dwSecond =
(uint)dateTimePicker_ValidEnd.Value.Second;
        m_stuInfo.bFirstEnter = this.checkBox_First.Checked;

        IntPtr inPtr = IntPtr.Zero;
        IntPtr ptr = IntPtr.Zero;
        try
        {
            inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD)));
            Marshal.StructureToPtr(m_stuInfo, inPtr, true);

            stuInfo.stuCtrlRecordSetInfo.pBuf = inPtr;
            stuInfo.stuCtrlRecordSetInfo.nBufLen =
(int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));
            ptr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_PARAM)));
            Marshal.StructureToPtr(stuInfo, ptr, true);
            bool ret = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_INSERT, ptr, 10000);
            if (!ret)
            {

```



```

        MessageBox.Show(NETClient.GetLastError());
        return;
    }
    stuInfo =
(NET_CTRL_RECORDSET_INSERT_PARAM)Marshal.PtrToStructure(ptr,
typeof(NET_CTRL_RECORDSET_INSERT_PARAM));
        MessageBox.Show("Execute Success\n RetNo="+
stuInfo.stuCtrlRecordSetResult.nRecNo.ToString());
    }
    finally
    {
        Marshal.FreeHGlobal(inPtr);
        Marshal.FreeHGlobal(ptr);
    }
#endregion

#region Update record
    NET_CTRL_RECORDSET_PARAM stuInfo = new
NET_CTRL_RECORDSET_PARAM();
    stuInfo.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM));
    stuInfo.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;

    m_stuInfo.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());
    m_stuInfo.stuCreateTime.dwYear = (uint)DateTime.Now.Year;
    m_stuInfo.stuCreateTime.dwMonth = (uint)DateTime.Now.Month;
    m_stuInfo.stuCreateTime.dwDay = (uint)DateTime.Now.Day;
    m_stuInfo.stuCreateTime.dwHour = (uint)DateTime.Now.Hour;
    m_stuInfo.stuCreateTime.dwMinute = (uint)DateTime.Now.Minute;
    m_stuInfo.stuCreateTime.dwSecond = (uint)DateTime.Now.Second;

    m_stuInfo.szCardNo = this.textBox_CardNo.Text.Trim();
    m_stuInfo.szUserID = this.textBox_UserID.Text.Trim();
    m_stuInfo.emStatus =
(EM_ACCESSCTLCARD_STATE)comboBox_CardStatus.SelectedIndex - 1;
    m_stuInfo.emType =
(EM_ACCESSCTLCARD_TYPE)comboBox_CardType.SelectedIndex - 1;
    m_stuInfo.szPsw = this.textBox_CardPwd.Text.Trim();
    m_stuInfo.nUseTime = Convert.ToInt32(textBox_UseTimes.Text.Trim());
    m_stuInfo.bNewDoor = true;
    if (m_selectDoorsList.Count > 0)

```

```

        {
            for (int i = 0; i < m_selectDoorsList.Count; i++)
            {
                m_stuInfo.nNewDoors[i] = m_selectDoorsList[i];
            }
        }
        m_stuInfo.nNewDoorNum = m_selectDoorsList.Count;

        if (m_selectTimesList.Count > 0)
        {
            for (int i = 0; i < m_selectTimesList.Count; i++)
            {
                m_stuInfo.nNewTimeSectionNo[i] = m_selectTimesList[i];
            }
        }
        m_stuInfo.nNewTimeSectionNum = m_selectTimesList.Count;
        m_stuInfo.stuValidStartTime.dwYear =
        (uint)dateTimePicker_ValidStart.Value.Year;
        m_stuInfo.stuValidStartTime.dwMonth =
        (uint)dateTimePicker_ValidStart.Value.Month;
        m_stuInfo.stuValidStartTime.dwDay =
        (uint)dateTimePicker_ValidStart.Value.Day;
        m_stuInfo.stuValidStartTime.dwHour =
        (uint)dateTimePicker_ValidStart.Value.Hour;
        m_stuInfo.stuValidStartTime.dwMinute =
        (uint)dateTimePicker_ValidStart.Value.Minute;
        m_stuInfo.stuValidStartTime.dwSecond =
        (uint)dateTimePicker_ValidStart.Value.Second;

        m_stuInfo.stuValidEndTime.dwYear =
        (uint)dateTimePicker_ValidEnd.Value.Year;
        m_stuInfo.stuValidEndTime.dwMonth =
        (uint)dateTimePicker_ValidEnd.Value.Month;
        m_stuInfo.stuValidEndTime.dwDay =
        (uint)dateTimePicker_ValidEnd.Value.Day;
        m_stuInfo.stuValidEndTime.dwHour =
        (uint)dateTimePicker_ValidEnd.Value.Hour;
        m_stuInfo.stuValidEndTime.dwMinute =
        (uint)dateTimePicker_ValidEnd.Value.Minute;
        m_stuInfo.stuValidEndTime.dwSecond =
        (uint)dateTimePicker_ValidEnd.Value.Second;
        m_stuInfo.bFirstEnter = this.checkBox_First.Checked;

```

```

        IntPtr inPtr = IntPtr.Zero;
        IntPtr ptr = IntPtr.Zero;
        try
        {
            inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD)));
            Marshal.StructureToPtr(m_stulInfo, inPtr, true);

            stulInfo.pBuf = inPtr;
            stulInfo.nBufLen =
(int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));

            ptr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM)));
            Marshal.StructureToPtr(stulInfo, ptr, true);
            bool ret = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_UPDATE, ptr, 10000);
            if (!ret)
            {
                MessageBox.Show(NETClient.GetLastError());
                return;
            }
            MessageBox.Show("Execute Success(operated successfully)" );
        }
        finally
        {
            Marshal.FreeHGlobal(inPtr);
            Marshal.FreeHGlobal(ptr);
        }
        OnChangeUIState(nCtlType);
    #endregion

    #region Remove record remove card record
    NET_CTRL_RECORDSET_PARAM stulInfo = new
NET_CTRL_RECORDSET_PARAM();
    stulInfo.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM));
    stulInfo.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;
    m_stulInfo.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());

```

```

        IntPtr inPtr = IntPtr.Zero;
        IntPtr ptr = IntPtr.Zero;
        try
        {
            inPtr = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(int)));
            Marshal.StructureToPtr(m_stulInfo.nRecNo, inPtr, true);

            stulInfo.pBuf = inPtr;
            stulInfo.nBufLen = (int)Marshal.SizeOf(typeof(int));

            ptr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM)));
            Marshal.StructureToPtr(stulInfo, ptr, true);
            bool ret = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_REMOVE, ptr, 10000);
            if (!ret)
            {
                MessageBox.Show(NETClient.GetLastError());
                return;
            }
            MessageBox.Show("Execute Success" );
        }
        finally
        {
            Marshal.FreeHGlobal(inPtr);
            Marshal.FreeHGlobal(ptr);
        }

        #endregion
        #region Clear card record clear card record
        NET_CTRL_RECORDSET_PARAM inParam = new
NET_CTRL_RECORDSET_PARAM();
        inParam.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM));
        inParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;
        IntPtr inPtr = IntPtr.Zero;
        try
        {

```

```

        IntPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM)));
        Marshal.StructureToPtr(inParam, IntPtr, true);
        bool ret = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_CLEAR, IntPtr, 10000);
        if (!ret)
        {
            MessageBox.Show(NETClient.GetLastError());
            return;
        }
        MessageBox.Show("Execute Success(operated successfully)");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        Marshal.FreeHGlobal(IntPtr);
    }
#endregion

#region Insert card record with finger added with fingerprint
NET_CTRL_RECORDSET_INSERT_PARAM stuInfo = new
NET_CTRL_RECORDSET_INSERT_PARAM();
    stuInfo.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_PARAM));

    stuInfo.stuCtrlRecordSetInfo.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_IN));
    stuInfo.stuCtrlRecordSetInfo.emType =
EM_NET_RECORD_TYPE.ACCESSCTLCARD;

    stuInfo.stuCtrlRecordSetInfo.nBufLen =
(int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));

    stuInfo.stuCtrlRecordSetResult.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_OUT));

    m_stuInfo.stuCreateTime.dwYear = (uint)DateTime.Now.Year;
    m_stuInfo.stuCreateTime.dwMonth = (uint)DateTime.Now.Month;

```

```

        m_stuInfo.stuCreateTime.dwDay = (uint)DateTime.Now.Day;
        m_stuInfo.stuCreateTime.dwHour = (uint)DateTime.Now.Hour;
        m_stuInfo.stuCreateTime.dwMinute = (uint)DateTime.Now.Minute;
        m_stuInfo.stuCreateTime.dwSecond = (uint)DateTime.Now.Second;

        m_stuInfo.szCardNo = this.textBox_CardNo.Text.Trim();
        m_stuInfo.szUserID = this.textBox_UserID.Text.Trim();
        m_stuInfo.emStatus =
(EM_ACCESSCTLCARD_STATE)comboBox_CardStatus.SelectedIndex - 1;
        m_stuInfo.emType =
(EM_ACCESSCTLCARD_TYPE)comboBox_CardType.SelectedIndex - 1;
        m_stuInfo.szPsw = this.textBox_CardPwd.Text.Trim();
        m_stuInfo.nUseTime = Convert.ToInt32(textBox_UseTimes.Text.Trim());
        m_stuInfo.bNewDoor = true;
        //m_stuInfo.nNewDoorNum;
        // m_stuInfo.sznDoors;
        // m_stuInfo.nNewTimeSectionNum;
        //m_stuInfo.nNewTimeSectionNo
        m_stuInfo.stuValidStartTime.dwYear =
(uint)dateTimePicker_ValidStart.Value.Year;
        m_stuInfo.stuValidStartTime.dwMonth =
(uint)dateTimePicker_ValidStart.Value.Month;
        m_stuInfo.stuValidStartTime.dwDay =
(uint)dateTimePicker_ValidStart.Value.Day;
        m_stuInfo.stuValidStartTime.dwHour =
(uint)dateTimePicker_ValidStart.Value.Hour;
        m_stuInfo.stuValidStartTime.dwMinute =
(uint)dateTimePicker_ValidStart.Value.Minute;
        m_stuInfo.stuValidStartTime.dwSecond =
(uint)dateTimePicker_ValidStart.Value.Second;

        m_stuInfo.stuValidEndTime.dwYear =
(uint)dateTimePicker_ValidEnd.Value.Year;
        m_stuInfo.stuValidEndTime.dwMonth =
(uint)dateTimePicker_ValidEnd.Value.Month;
        m_stuInfo.stuValidEndTime.dwDay =
(uint)dateTimePicker_ValidEnd.Value.Day;
        m_stuInfo.stuValidEndTime.dwHour =
(uint)dateTimePicker_ValidEnd.Value.Hour;
        m_stuInfo.stuValidEndTime.dwMinute =
(uint)dateTimePicker_ValidEnd.Value.Minute;

```

```

        m_stuInfo.stuValidEndTime.dwSecond =
(uint)dateTimePicker_ValidEnd.Value.Second;

        m_stuInfo.bFirstEnter = this.checkBox_First.Checked;

        m_stuInfo.bEnableExtended = true;
        m_stuInfo.stuFingerPrintInfoEx.nCount = 1;
        m_stuInfo.stuFingerPrintInfoEx.nLength = PacketLen;
        m_stuInfo.stuFingerPrintInfoEx.nPacketLen = PacketLen;
        m_stuInfo.stuFingerPrintInfoEx.pPacketData =
Marshal.AllocHGlobal(m_stuInfo.stuFingerPrintInfoEx.nPacketLen);
        Marshal.Copy(FingerPrintInfo, 0,
m_stuInfo.stuFingerPrintInfoEx.pPacketData, m_stuInfo.stuFingerPrintInfoEx.nPacketLen);

        IntPtr inPtr = IntPtr.Zero;
        IntPtr ptr = IntPtr.Zero;
        try
        {
            inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD)));
            Marshal.StructureToPtr(m_stuInfo, inPtr, true);

            stuInfo.stuCtrlRecordSetInfo.pBuf = inPtr;
            stuInfo.stuCtrlRecordSetInfo.nBufLen =
(int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));
            ptr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_PARAM)));
            Marshal.StructureToPtr(stuInfo, ptr, true);
            bool ret = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_INSERT, ptr, 10000);
            if (!ret)
            {
                MessageBox.Show(NETClient.GetLastError());
                return;
            }
            stuInfo =
(NET_CTRL_RECORDSET_INSERT_PARAM)Marshal.PtrToStructure(ptr,
typeof(NET_CTRL_RECORDSET_INSERT_PARAM));
            MessageBox.Show("Execute Success(operated successfully)\n
RetNo=" + stuInfo.stuCtrlRecordSetResult.nRecNo.ToString());
        }
        finally

```

```

        {
            Marshal.FreeHGlobal(m_stuInfo.stuFingerPrintInfoEx.pPacketData);
            Marshal.FreeHGlobal(inPtr);
            Marshal.FreeHGlobal(ptr);
        }
    #endregion

    #region Update record with finger update with fingerprint
    NET_CTRL_RECORDSET_PARAM stuInfo = new
NET_CTRL_RECORDSET_PARAM();
    stuInfo.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM));
    stuInfo.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;

    m_stuInfo.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());
    m_stuInfo.stuCreateTime.dwYear = (uint)DateTime.Now.Year;
    m_stuInfo.stuCreateTime.dwMonth = (uint)DateTime.Now.Month;
    m_stuInfo.stuCreateTime.dwDay = (uint)DateTime.Now.Day;
    m_stuInfo.stuCreateTime.dwHour = (uint)DateTime.Now.Hour;
    m_stuInfo.stuCreateTime.dwMinute = (uint)DateTime.Now.Minute;
    m_stuInfo.stuCreateTime.dwSecond = (uint)DateTime.Now.Second;

    m_stuInfo.szCardNo = this.textBox_CardNo.Text.Trim();
    m_stuInfo.szUserID = this.textBox_UserID.Text.Trim();
    m_stuInfo.emStatus =
(EM_ACCESSCTLCARD_STATE)comboBox_CardStatus.SelectedIndex - 1;
    m_stuInfo.emType =
(EM_ACCESSCTLCARD_TYPE)comboBox_CardType.SelectedIndex - 1;
    m_stuInfo.szPsw = this.textBox_CardPwd.Text.Trim();
    m_stuInfo.nUseTime = Convert.ToInt32(textBox_UseTimes.Text.Trim());
    m_stuInfo.bNewDoor = true;
    if (m_selectDoorsList.Count > 0)
    {
        for (int i = 0; i < m_selectDoorsList.Count; i++)
        {
            m_stuInfo.nNewDoors[i] = m_selectDoorsList[i];
        }
    }
    m_stuInfo.nNewDoorNum = m_selectDoorsList.Count;

```



```

        if (m_selectTimesList.Count > 0)
        {
            for (int i = 0; i < m_selectTimesList.Count; i++)
            {
                m_stuInfo.nNewTimeSectionNo[i] = m_selectTimesList[i];
            }
        }
        m_stuInfo.nNewTimeSectionNum = m_selectTimesList.Count;
        m_stuInfo.stuValidStartTime.dwYear =
(uint)dateTimePicker_ValidStart.Value.Year;
        m_stuInfo.stuValidStartTime.dwMonth =
(uint)dateTimePicker_ValidStart.Value.Month;
        m_stuInfo.stuValidStartTime.dwDay =
(uint)dateTimePicker_ValidStart.Value.Day;
        m_stuInfo.stuValidStartTime.dwHour =
(uint)dateTimePicker_ValidStart.Value.Hour;
        m_stuInfo.stuValidStartTime.dwMinute =
(uint)dateTimePicker_ValidStart.Value.Minute;
        m_stuInfo.stuValidStartTime.dwSecond =
(uint)dateTimePicker_ValidStart.Value.Second;

        m_stuInfo.stuValidEndTime.dwYear =
(uint)dateTimePicker_ValidEnd.Value.Year;
        m_stuInfo.stuValidEndTime.dwMonth =
(uint)dateTimePicker_ValidEnd.Value.Month;
        m_stuInfo.stuValidEndTime.dwDay =
(uint)dateTimePicker_ValidEnd.Value.Day;
        m_stuInfo.stuValidEndTime.dwHour =
(uint)dateTimePicker_ValidEnd.Value.Hour;
        m_stuInfo.stuValidEndTime.dwMinute =
(uint)dateTimePicker_ValidEnd.Value.Minute;
        m_stuInfo.stuValidEndTime.dwSecond =
(uint)dateTimePicker_ValidEnd.Value.Second;
        m_stuInfo.bFirstEnter = this.checkBox_First.Checked;

        m_stuInfo.bEnableExtended = true;
        m_stuInfo.stuFingerPrintInfoEx.nCount = 1;
        m_stuInfo.stuFingerPrintInfoEx.nLength = PacketLen;
        m_stuInfo.stuFingerPrintInfoEx.nPacketLen = PacketLen;
        m_stuInfo.stuFingerPrintInfoEx.pPacketData =
Marshal.AllocHGlobal(m_stuInfo.stuFingerPrintInfoEx.nPacketLen);

```

```

        Marshal.Copy(FingerPrintInfo, 0,
m_stuInfo.stuFingerPrintInfoEx.pPacketData, m_stuInfo.stuFingerPrintInfoEx.nPacketLen);

        IntPtr inPtr = IntPtr.Zero;
        IntPtr ptr = IntPtr.Zero;
        try
        {
            inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD)));
            Marshal.StructureToPtr(m_stuInfo, inPtr, true);

            stuInfo.pBuf = inPtr;
            stuInfo.nBufLen =
(int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));

            ptr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM)));
            Marshal.StructureToPtr(stuInfo, ptr, true);
            bool ret = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_UPDATEEX, ptr, 10000);
            if (!ret)
            {
                MessageBox.Show(NETClient.GetLastError());
                return;
            }
            MessageBox.Show("Execute Success(operated successfully)");
        }
        finally
        {
            Marshal.FreeHGlobal(m_stuInfo.stuFingerPrintInfoEx.pPacketData);
            Marshal.FreeHGlobal(inPtr);
            Marshal.FreeHGlobal(ptr);
        }
        OnChangeUIState(nCtlType);
    #endregion

    #region Update record with finger update with fingerprint
    NET_CTRL_RECORDSET_PARAM stuInfo = new
NET_CTRL_RECORDSET_PARAM();

```

```

        stuInfo.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM));
        stuInfo.emType = EM_NET_RECORD_TYPE.ACCESSCTLCARD;

        m_stuInfo.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());
        m_stuInfo.stuCreateTime.dwYear = (uint)DateTime.Now.Year;
        m_stuInfo.stuCreateTime.dwMonth = (uint)DateTime.Now.Month;
        m_stuInfo.stuCreateTime.dwDay = (uint)DateTime.Now.Day;
        m_stuInfo.stuCreateTime.dwHour = (uint)DateTime.Now.Hour;
        m_stuInfo.stuCreateTime.dwMinute = (uint)DateTime.Now.Minute;
        m_stuInfo.stuCreateTime.dwSecond = (uint)DateTime.Now.Second;

        m_stuInfo.szCardNo = this.textBox_CardNo.Text.Trim();
        m_stuInfo.szUserID = this.textBox_UserID.Text.Trim();
        m_stuInfo.emStatus =
(EM_ACCESSCTLCARD_STATE)comboBox_CardStatus.SelectedIndex - 1;
        m_stuInfo.emType =
(EM_ACCESSCTLCARD_TYPE)comboBox_CardType.SelectedIndex - 1;
        m_stuInfo.szPsw = this.textBox_CardPwd.Text.Trim();
        m_stuInfo.nUseTime = Convert.ToInt32(textBox_UseTimes.Text.Trim());
        m_stuInfo.bNewDoor = true;
        if (m_selectDoorsList.Count > 0)
        {
            for (int i = 0; i < m_selectDoorsList.Count; i++)
            {
                m_stuInfo.nNewDoors[i] = m_selectDoorsList[i];
            }
        }
        m_stuInfo.nNewDoorNum = m_selectDoorsList.Count;

        if (m_selectTimesList.Count > 0)
        {
            for (int i = 0; i < m_selectTimesList.Count; i++)
            {
                m_stuInfo.nNewTimeSectionNo[i] = m_selectTimesList[i];
            }
        }
        m_stuInfo.nNewTimeSectionNum = m_selectTimesList.Count;
        m_stuInfo.stuValidStartTime.dwYear =
(uint)dateTimePicker_ValidStart.Value.Year;

```

```

        m_stuInfo.stuValidStartTime.dwMonth =
(uint)dateTimePicker_ValidStart.Value.Month;
        m_stuInfo.stuValidStartTime.dwDay =
(uint)dateTimePicker_ValidStart.Value.Day;
        m_stuInfo.stuValidStartTime.dwHour =
(uint)dateTimePicker_ValidStart.Value.Hour;
        m_stuInfo.stuValidStartTime.dwMinute =
(uint)dateTimePicker_ValidStart.Value.Minute;
        m_stuInfo.stuValidStartTime.dwSecond =
(uint)dateTimePicker_ValidStart.Value.Second;

        m_stuInfo.stuValidEndTime.dwYear =
(uint)dateTimePicker_ValidEnd.Value.Year;
        m_stuInfo.stuValidEndTime.dwMonth =
(uint)dateTimePicker_ValidEnd.Value.Month;
        m_stuInfo.stuValidEndTime.dwDay =
(uint)dateTimePicker_ValidEnd.Value.Day;
        m_stuInfo.stuValidEndTime.dwHour =
(uint)dateTimePicker_ValidEnd.Value.Hour;
        m_stuInfo.stuValidEndTime.dwMinute =
(uint)dateTimePicker_ValidEnd.Value.Minute;
        m_stuInfo.stuValidEndTime.dwSecond =
(uint)dateTimePicker_ValidEnd.Value.Second;
        m_stuInfo.bFirstEnter = this.checkBox_First.Checked;

        m_stuInfo.bEnableExtended = true;
        m_stuInfo.stuFingerPrintInfoEx.nCount = 1;
        m_stuInfo.stuFingerPrintInfoEx.nLength = PacketLen;
        m_stuInfo.stuFingerPrintInfoEx.nPacketLen = PacketLen;
        m_stuInfo.stuFingerPrintInfoEx.pPacketData =
Marshal.AllocHGlobal(m_stuInfo.stuFingerPrintInfoEx.nPacketLen);
        Marshal.Copy(FingerPrintInfo, 0,
m_stuInfo.stuFingerPrintInfoEx.pPacketData, m_stuInfo.stuFingerPrintInfoEx.nPacketLen);

        IntPtr inPtr = IntPtr.Zero;
        IntPtr ptr = IntPtr.Zero;
        try
        {
            inPtr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD)));
            Marshal.StructureToPtr(m_stuInfo, inPtr, true);

```

```

        stuInfo.pBuf = inPtr;
        stuInfo.nBufLen =
(int)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARD));

        ptr =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_PARAM)));
        Marshal.StructureToPtr(stuInfo, ptr, true);
        bool ret = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_UPDATEEX, ptr, 10000);
        if (!ret)
        {
            MessageBox.Show(NETClient.GetLastError());
            return;
        }
        MessageBox.Show("Execute Success(operated successfully)");
    }
    finally
    {
        Marshal.FreeHGlobal(m_stuInfo.stuFingerPrintInfoEx.pPacketData);
        Marshal.FreeHGlobal(inPtr);
        Marshal.FreeHGlobal(ptr);
    }
    OnChangeUIState(nCtlType);
#endregion

```

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.

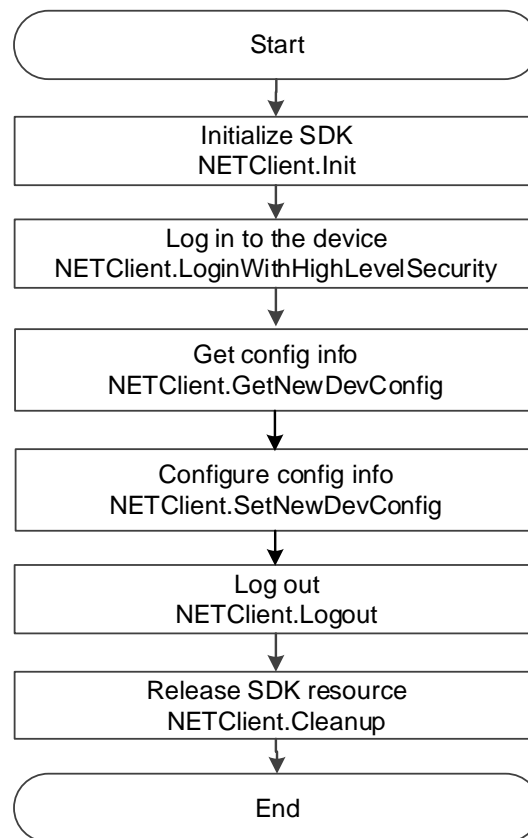
2.2.8.2 Interface Overview

Table 2-27 Description of door config information interfaces

Interface	Description
NETClient.GetNewDevConfig	Query config information.
NETClient.SetNewDevConfig	Set config information.

2.2.8.3 Process Description

Figure 2-25 Door config information



Process

- Step 1** Call **NETClient.Init** to initialize SDK.
- Step 2** Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3** Call **NETClient.GetNewDevConfig** to query the access door info.
- szCommand: AccessControl.
 - obj: NET_CFG_ACCESS_EVENT_INFO.

Table 2-28 Description of NET_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 **NETClient.SetNewDevConfig** to set the access door info.
- szCommand: AccessControl.
 - pBuf: NET_CFG_ACCESS_EVENT_INFO.
- Step 5** After completing this process, call **NETClient. Logout** to log out of the device.

Step 6 After using all SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.2.8.4 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 verification. For details, see "2.2.9.1 Period Config."

2.2.8.5 Sample Code

```
// get door config info
    NET_CFG_ACCESS_EVENT_INFO cfg = new
NET_CFG_ACCESS_EVENT_INFO();
    public NET_CFG_ACCESS_EVENT_INFO? GetConfig()
    {
        try
        {
            object objTemp = new object();
            bool bRet = NETClient.GetNewDevConfig(loginID,
cmbBox_DoorIndex.SelectedIndex, "AccessControl", ref objTemp,
typeof(NET_CFG_ACCESS_EVENT_INFO), 5000);
            cfg = (NET_CFG_ACCESS_EVENT_INFO)objTemp;
            if (!bRet)
            {
                MessageBox.Show(NETClient.GetLastError());
                return cfg;
            }
            cfg = (NET_CFG_ACCESS_EVENT_INFO)objTemp;
        }
        catch (NETClientExcetion nex)
        {
            MessageBox.Show(nex.Message);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        return cfg;
    }
```

```

public bool SetConfig(NET_CFG_ACCESS_EVENT_INFO cfg)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID,
cmbBox_DoorIndex.SelectedIndex, "AccessControl", (object)cfg,
typeof(NET_CFG_ACCESS_EVENT_INFO), 5000);
    }
    catch (NETClientExcetion nex)
    {
        Console.WriteLine(nex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return bRet;
}

```

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.

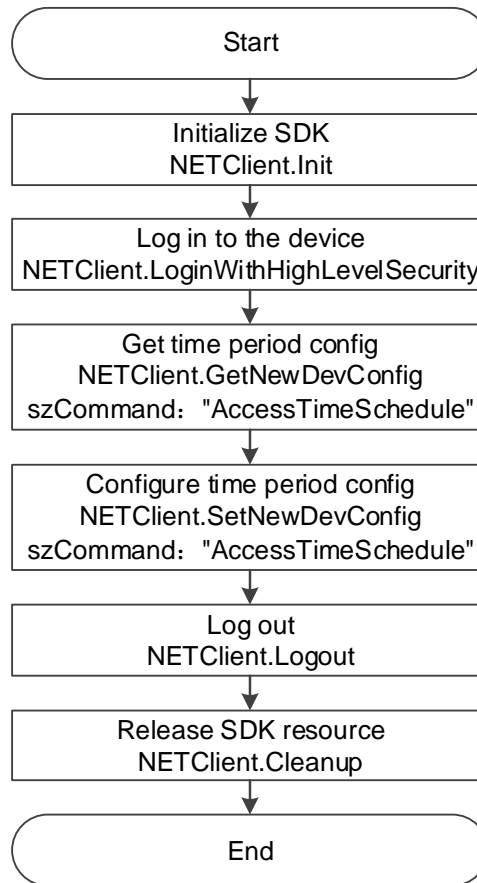
2.2.9.1.2 Interface Overview

Table 2-29 Description of period interfaces

Interface	Description
NETClient.GetNewDevConfig	Query config information.
NETClient.SetNewDevConfig	Configure config information.

2.2.9.1.3 Process Description

Figure 2-26 Period config



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.GetNewDevConfig** to query the access period info.
- szCommand: AccessTimeSchedule.
 - pBuf: CFG_ACCESS_TIMESCHEDULE_INFO.
- Step 4 Call **NETClient.SetNewDevConfig** to set the access period info.
- szCommand: AccessTimeSchedule.
 - pBuf: CFG_ACCESS_TIMESCHEDULE_INFO.
- Step 5 After completing this process, call **NETClient. Logout** to log out of the device.
- Step 6 After using all SDK functions, call **NETClient. Cleanup** to release SDK resources.

2.2.9.1.4 Sample Code

```
// Get time period config info
timeSchedule = new NET_CFG_ACCESS_TIMESCHEDULE_INFO();
object objInfo = timeSchedule;
bool ret = NETClient.GetNewDevConfig(loginID, comboBox_Index.SelectedIndex
+ 1, CFG_CMD_ACCESSTIMESCHEDULE, ref objInfo,
typeof(NET_CFG_ACCESS_TIMESCHEDULE_INFO), 10000);
if (!ret)
```

```

    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }
    timeSchedule = (NET_CFG_ACCESS_TIMESCHEDULE_INFO)objInfo;

    textBox_Name.Text = timeSchedule.szName;

    var temp = timeSchedule.stuTime[comboBox_Week.SelectedIndex * 4];
    dateTimePicker_Start1.Value = new DateTime(2020, 1, 1, temp.nBeginHour,
temp.nBeginMin, temp.nBeginSec);
    dateTimePicker_End1.Value = new DateTime(2020, 1, 1, temp.nEndHour,
temp.nEndMin, temp.nEndSec);

    temp = timeSchedule.stuTime[comboBox_Week.SelectedIndex * 4 + 1];
    dateTimePicker_Start2.Value = new DateTime(2020, 1, 1, temp.nBeginHour,
temp.nBeginMin, temp.nBeginSec);
    dateTimePicker_End2.Value = new DateTime(2020, 1, 1, temp.nEndHour,
temp.nEndMin, temp.nEndSec);

    temp = timeSchedule.stuTime[comboBox_Week.SelectedIndex * 4 + 2];
    dateTimePicker_Start3.Value = new DateTime(2020, 1, 1, temp.nBeginHour,
temp.nBeginMin, temp.nBeginSec);
    dateTimePicker_End3.Value = new DateTime(2020, 1, 1, temp.nEndHour,
temp.nEndMin, temp.nEndSec);

    temp = timeSchedule.stuTime[comboBox_Week.SelectedIndex * 4 + 3];
    dateTimePicker_Start4.Value = new DateTime(2020, 1, 1, temp.nBeginHour,
temp.nBeginMin, temp.nBeginSec);
    dateTimePicker_End4.Value = new DateTime(2020, 1, 1, temp.nEndHour,
temp.nEndMin, temp.nEndSec);

    MessageBox.Show("Get success(Get successfully)");

    object objInfo = timeSchedule;
    bool ret = NETClient.SetNewDevConfig(loginID, comboBox_Index.SelectedIndex
+ 1, CFG_CMD_ACCESSTIMESCHEDULE, objInfo,
typeof(NET_CFG_ACCESS_TIMESCHEDULE_INFO), 10000);
    if (!ret)
    {
        MessageBox.Show(NETClient.GetLastError());
    }

```

```

        return;
    }
    MessageBox.Show("Set success");

```

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.

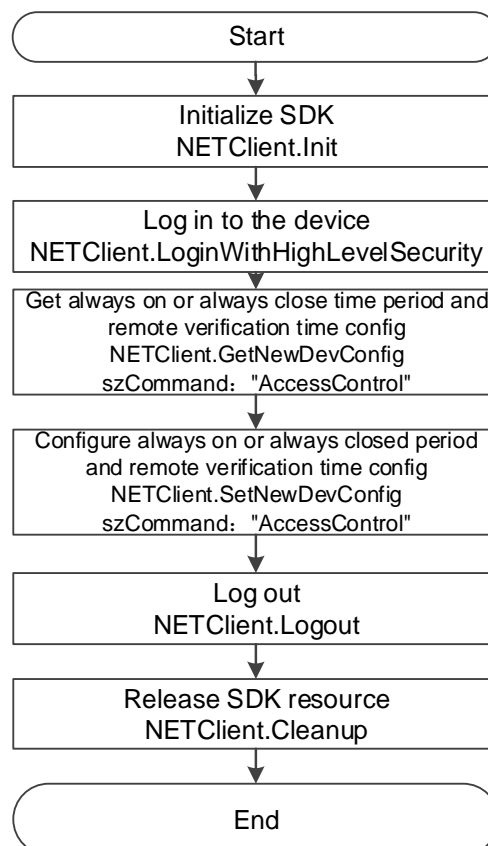
2.2.9.2.2 Interface Overview

Table 2-30 Description of always open and always closed period config interfaces

Interface	Description
NETClient.GetNewDevConfig	Query config information.
NETClient.SetNewDevConfig	Configure config information.

2.2.9.2.3 Process Description

Figure 2-27 Always open and always closed period config



Process

Step 1 Call **NETClient.Init** to initialize SDK.

Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.

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

- szCommand: AccessControl.
- pBuf: NET_CFG_ACCESS_EVENT_INFO.

Table 2-31 Description of NET_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 **NETClient.SetNewDevConfig** in pairs to set the access always open and always closed period info, and remote verification period.

- szCommand: AccessControl.
- pBuf: NET_CFG_ACCESS_EVENT_INFO.

Table 2-32 Description of NET_CFG_ACCESS_EVENT_INFO

NET_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 **NETClient. Logout** to log out of the device.

Step 6 After using all SDK functions, call **NETClient. Cleanup** to release SDK resources.

Note

Set the serial number of always open period, always close period and remote verification. For details, see "2.2.9.1 Period Config."

2.2.9.2.4 Sample Code

```
// get time period config of always on always closed and remote verification
    NET_CFG_ACCESS_EVENT_INFO cfg = new
NET_CFG_ACCESS_EVENT_INFO();
    public NET_CFG_ACCESS_EVENT_INFO? GetConfig()
    {
        try
        {
            object objTemp = new object();
            bool bRet = NETClient.GetNewDevConfig(loginID,
comboBox_DoorNo.SelectedIndex, "AccessControl", ref objTemp,
typeof(NET_CFG_ACCESS_EVENT_INFO), 5000);
            if (bRet)
```

```

        {
            cfg = (NET_CFG_ACCESS_EVENT_INFO)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return cfg;
}

public bool SetConfig(NET_CFG_ACCESS_EVENT_INFO cfg)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID,
comboBox_DoorNo.SelectedIndex, "AccessControl", (object)cfg,
typeof(NET_CFG_ACCESS_EVENT_INFO), 5000);
    }
    catch (NETClientExcetion nex)
    {
        Console.WriteLine(nex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return bRet;
}

```

2.2.9.3 Holiday Config

2.2.9.3.1 Introduction

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

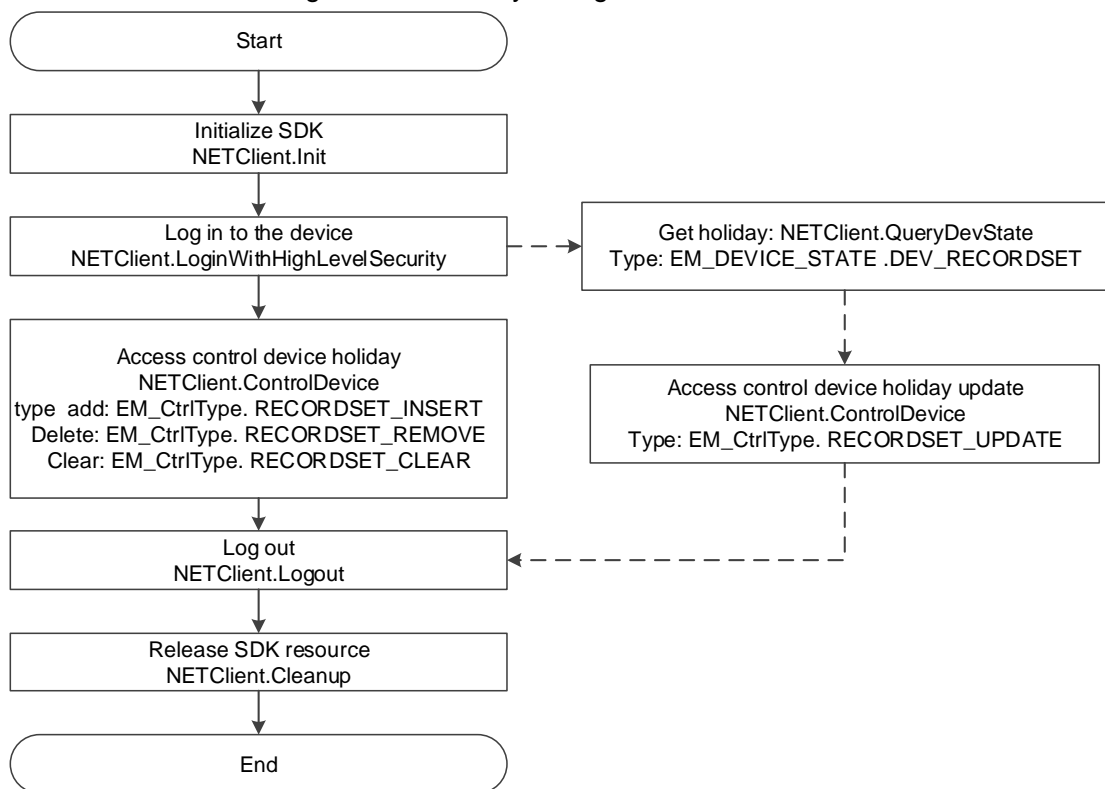
2.2.9.3.2 Interface Overview

Table 2-33 Description of holiday config interfaces

Interface	Description
NETClient.ControlDevice	Control device.
NETClient.QueryDevState	Query device status.

2.2.9.3.3 Process Description

Figure 2-28 Holiday configuration



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.ControlDevice** to operate holiday information.

Table 2-34 Description and structure of type

Type	Description	emType	Param
EM_CtrlType. RECORDSET_INSERT	Add holiday	EM_NET_RECORD_TYPE. ACCESSCTHOLIDAY	NET_CTRL_RECORDSET_INSERT_PARAM NET_RECORDSET_HOLIDAY
EM_CtrlType. RECORDSET_REMOVE	Delete holiday	EM_NET_RECORD_TYPE. ACCESSCTHOLIDAY	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_HOLIDAY
EM_CtrlType. RECORDSET_CLEAR	Clear holiday	EM_NET_RECORD_TYPE. ACCESSCTHOLIDAY	NET_CTRL_RECORDSET_PARAM

Step 4 Call the **NETClient.QueryDevState** interface to get holiday information.

Table 2-35 Description and structure of type

Type	Description	emType	Param
EM_DEVICE_STATE.DEV_RECORDSET	Get holiday	EM_NET_RECORD_TYPE.ACCESSCTHOLIDAY	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_HOLIDAY

Step 5 Call **NETClient.ControlDevice** to update holiday information.

Table 2-36 Description and structure of type

Type	Description	emType	Param
DH_CTRL_RECORDSET_UPDATE	Update holiday	EM_NET_RECORD_TYPE.ACCESSCTHOLIDAY	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_HOLIDAY

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

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

2.2.9.3.4 Sample Code

```
// Get holiday
IntPtr pBuf = IntPtr.Zero;

NET_RECORDSET_HOLIDAY result = new NET_RECORDSET_HOLIDAY();
NET_CTRL_RECORDSET_PARAM stuParam = new
NET_CTRL_RECORDSET_PARAM();

try
{
    pBuf = Marshal.AllocHGlobal(Marshal.SizeOf(result));

    //package for pwd
    result.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());
```

```

        result.dwSize = (uint)Marshal.SizeOf(result);
        Marshal.StructureToPtr(result, pBuf, true);

        //package stuParam
        stuParam.pBuf = pBuf;
        stuParam.nBufLen = Marshal.SizeOf(result);
        stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLHOLIDAY;
        stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);
        object obj = stuParam;

        bool bRet = NETClient.QueryDevState(loginID,
(int)EM_DEVICE_STATE.DEV_RECORDSET, ref obj,
typeof(NET_CTRL_RECORDSET_PARAM), 3000);
        if (bRet)
        {
            update_holiday =
(NET_RECORDSET_HOLIDAY)Marshal.PtrToStructure(pBuf,
typeof(NET_RECORDSET_HOLIDAY));

            dateTimePicker_StartTime.Value =
update_holiday.stuStartTime.ToDateTime();
            dateTimePicker_EndTime.Value =
update_holiday.stuEndTime.ToDateTime();
            textBox_HolidayNo.Text = update_holiday.szHolidayNo;
            MessageBox.Show("Get succeed");
            OnChangeUIState(5);
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        Marshal.FreeHGlobal(pBuf);
    }

```



```

// Add holiday
    IntPtr pParam = IntPtr.Zero;
    IntPtr pBuf = IntPtr.Zero;
    NET_CTRL_RECORDSET_INSERT_PARAM stuInsertParam = new
NET_CTRL_RECORDSET_INSERT_PARAM();
    NET_CTRL_RECORDSET_INSERT_PARAM stuOutParam = new
NET_CTRL_RECORDSET_INSERT_PARAM();

    NET_RECORDSET_HOLIDAY stuHoliday = new
NET_RECORDSET_HOLIDAY();
    object obj = stuHoliday;
    InitStruct(ref obj);
    stuHoliday = (NET_RECORDSET_HOLIDAY)obj;
    stuHoliday.dwSize = (uint)Marshal.SizeOf(stuHoliday);

    stuHoliday.stuStartTime.dwYear = (uint)dateTimePicker_StartTime.Value.Year;
    stuHoliday.stuStartTime.dwMonth =
(uint)dateTimePicker_StartTime.Value.Month;
    stuHoliday.stuStartTime.dwDay = (uint)dateTimePicker_StartTime.Value.Day;

    stuHoliday.stuEndTime.dwYear = (uint)dateTimePicker_EndTime.Value.Year;
    stuHoliday.stuEndTime.dwMonth = (uint)dateTimePicker_EndTime.Value.Month;
    stuHoliday.stuEndTime.dwDay = (uint)dateTimePicker_EndTime.Value.Day;

    stuHoliday.szHolidayNo = textBox_HolidayNo.Text;

    if (m_selectDoorsList.Count > 0)
    {
        for (int i = 0; i < m_selectDoorsList.Count; i++)
        {
            stuHoliday.sznDoors[i] = m_selectDoorsList[i];
        }
    }
    stuHoliday.nDoorNum = m_selectDoorsList.Count;

    try
    {
        pParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_PARAM)));
        pBuf =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_HOLIDAY)));

```

```

        Marshal.StructureToPtr(stuHoliday, pBuf, true);

        //package stuInsertParam
        stuInsertParam.stuCtrlRecordSetInfo.pBuf = pBuf;
        stuInsertParam.stuCtrlRecordSetInfo.nBufLen = Marshal.SizeOf(stuHoliday);
        stuInsertParam.dwSize = (uint)Marshal.SizeOf(stuInsertParam);
        stuInsertParam.stuCtrlRecordSetInfo.dwSize =
        (uint)Marshal.SizeOf(stuInsertParam.stuCtrlRecordSetInfo);
        stuInsertParam.stuCtrlRecordSetInfo.emType =
        EM_NET_RECORD_TYPE.ACCESSCTLHOLIDAY;
        stuInsertParam.stuCtrlRecordSetResult.dwSize =
        (uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_OUT));
        Marshal.StructureToPtr(stuInsertParam, pParam, true);

        bool bRet = NETClient.ControlDevice(loginID,
        EM_CtrlType.RECORDSET_INSERT, pParam, 3000);

        stuOutParam =
        (NET_CTRL_RECORDSET_INSERT_PARAM)Marshal.PtrToStructure(pParam,
        typeof(NET_CTRL_RECORDSET_INSERT_PARAM));
        if (bRet && stuOutParam.stuCtrlRecordSetResult.nRecNo > 0)
        {
            MessageBox.Show("Inster succeed。 RecNO:" +
        stuOutParam.stuCtrlRecordSetResult.nRecNo);
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    //free resource
    finally
    {
        Marshal.FreeHGlobal(pParam);
    }

```

```

        Marshal.FreeHGlobal(pBuf);
    }

// update holiday
    update_holiday.stuEndTime.dwDay = (uint)dateTimePicker_EndTime.Value.Day;

    update_holiday.szHolidayNo = textBox_HolidayNo.Text;
    if (m_selectDoorsList.Count > 0)
    {
        for (int i = 0; i < m_selectDoorsList.Count; i++)
        {
            update_holiday.sznDoors[i] = m_selectDoorsList[i];
        }
    }
    update_holiday.nDoorNum = m_selectDoorsList.Count;

    bool bRet = false;
    IntPtr pParam = IntPtr.Zero;
    IntPtr pBuf = IntPtr.Zero;
    NET_CTRL_RECORDSET_PARAM stuParam = new
NET_CTRL_RECORDSET_PARAM();
    try
    {
        pParam = Marshal.AllocHGlobal(Marshal.SizeOf(stuParam));
        pBuf = Marshal.AllocHGlobal(Marshal.SizeOf(update_holiday));

        Marshal.StructureToPtr(update_holiday, pBuf, true);
        stuParam.pBuf = pBuf;
        stuParam.nBufLen = Marshal.SizeOf(update_holiday);
        stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLHOLIDAY;
        stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);
        Marshal.StructureToPtr(stuParam, pParam, true);

        bRet = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_UPDATE, pParam, 3000);
        if (bRet)
        {
            MessageBox.Show("Update succeed。");
        }
    }

```

```

        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        Marshal.FreeHGlobal(pParam);
        Marshal.FreeHGlobal(pBuf);
    }

// remove holiday
    IntPtr pParam = IntPtr.Zero;
    IntPtr pBuf = IntPtr.Zero;
    NET_CTRL_RECORDSET_PARAM stuParam = new
NET_CTRL_RECORDSET_PARAM();
    stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLHOLIDAY;
    stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);
    stuParam.pBuf = IntPtr.Zero;
    stuParam.nBufLen = 0;
    try
    {
        pParam = Marshal.AllocHGlobal(Marshal.SizeOf(stuParam));
        pBuf =
Marshal.AllocHGlobal(Marshal.SizeOf(int.Parse(textBox_RecNo.Text)));
        Marshal.StructureToPtr(int.Parse(textBox_RecNo.Text), pBuf, true);
        stuParam.pBuf = pBuf;
        stuParam.nBufLen = Marshal.SizeOf(int.Parse(textBox_RecNo.Text));
        Marshal.StructureToPtr(stuParam, pParam, true);

        result = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_REMOVE, pParam, 3000);
        if (result)
        {
            MessageBox.Show("Remove succeed");

```

```

        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        Marshal.FreeHGlobal(pBuf);
        Marshal.FreeHGlobal(pParam);
    }

// clear holiday
    IntPtr pParam = IntPtr.Zero;
    NET_CTRL_RECORDSET_PARAM stuParam = new
NET_CTRL_RECORDSET_PARAM();
    stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLHOLIDAY;
    stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);

    pParam = Marshal.AllocHGlobal(Marshal.SizeOf(stuParam));
    Marshal.StructureToPtr(stuParam, pParam, true);

    bool result = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_CLEAR, pParam, 3000);
    if (result)
    {
        MessageBox.Show("Clear succeed");
    }
    else
    {
        MessageBox.Show(NETClient.GetLastError());
    }
}

```

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.

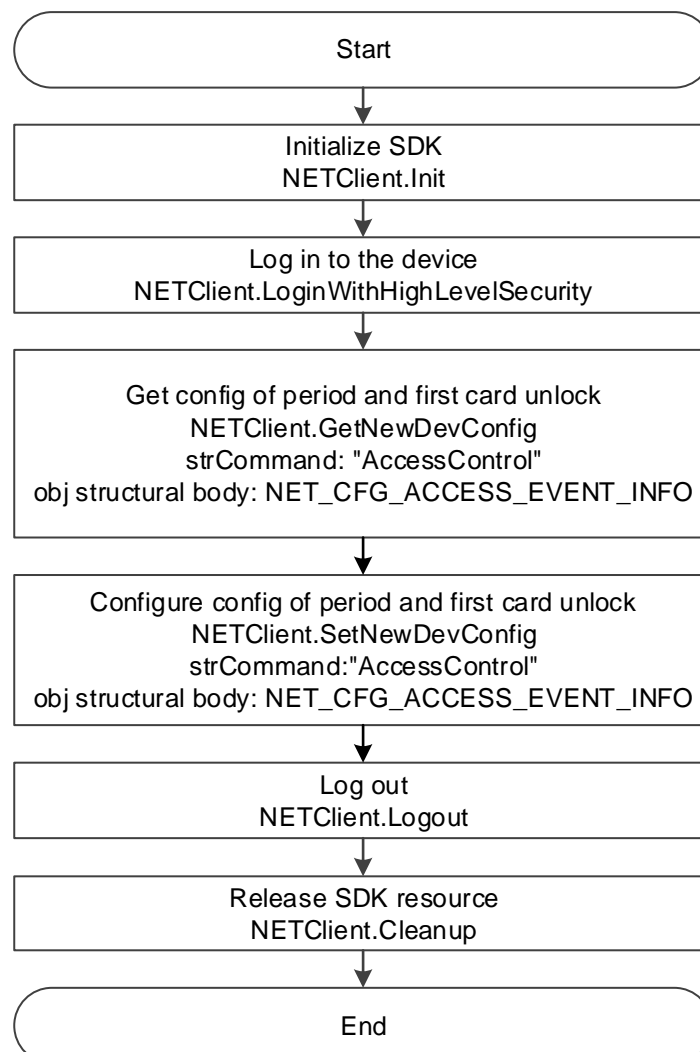
2.2.10.1.2 Interface Overview

Table 2-37 Description of interfaces for unlock at designated intervals and first card unlock

Interface	Description
NETClient. GetNewDevConfig	Query config information.
NETClient. SetNewDevConfig	Set config information.

2.2.10.1.3 Process Description

Figure 2-29 Unlock at designated intervals and first card unlock



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.GetNewDevConfig** to query the access info of unlock at designated intervals and first card unlock.
- strCommand: AccessControl.
 - Obj structural body: NET_CFG_ACCESS_EVENT_INFO.

Table 2-38 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 **NETClient.SetNewDevConfig** to set the access info of unlock at designated intervals and first card unlock.
- strCommand: AccessControl.
 - Obj structural body: NET_CFG_ACCESS_EVENT_INFO.
- Step 5 After completing this process, call **NETClient. Logout** to log out of the device.
- Step 6 After using all SDK functions, call **NETClient. 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 Sample Code

```
// get config of unlock by period and config of first card/user unlock
public bool GetConfig()
{
    bool bRet = false;
    try
    {
        object objTemp = new object();
        bRet = NETClient.GetNewDevConfig(loginID,
comboBox_Channel.SelectedIndex, CFG_CMD_ACCESS_EVENT, ref objTemp,
typeof(NET_CFG_ACCESS_EVENT_INFO), 5000);
        if (bRet)
        {
            cfg = (NET_CFG_ACCESS_EVENT_INFO)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
}
```

```

        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return bRet;
}

public bool SetConfig(NET_CFG_ACCESS_EVENT_INFO cfg)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID,
comboBox_Channel.SelectedIndex, CFG_CMD_ACCESS_EVENT, (object)cfg,
typeof(NET_CFG_ACCESS_EVENT_INFO), 5000);
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return bRet;
}

```

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.

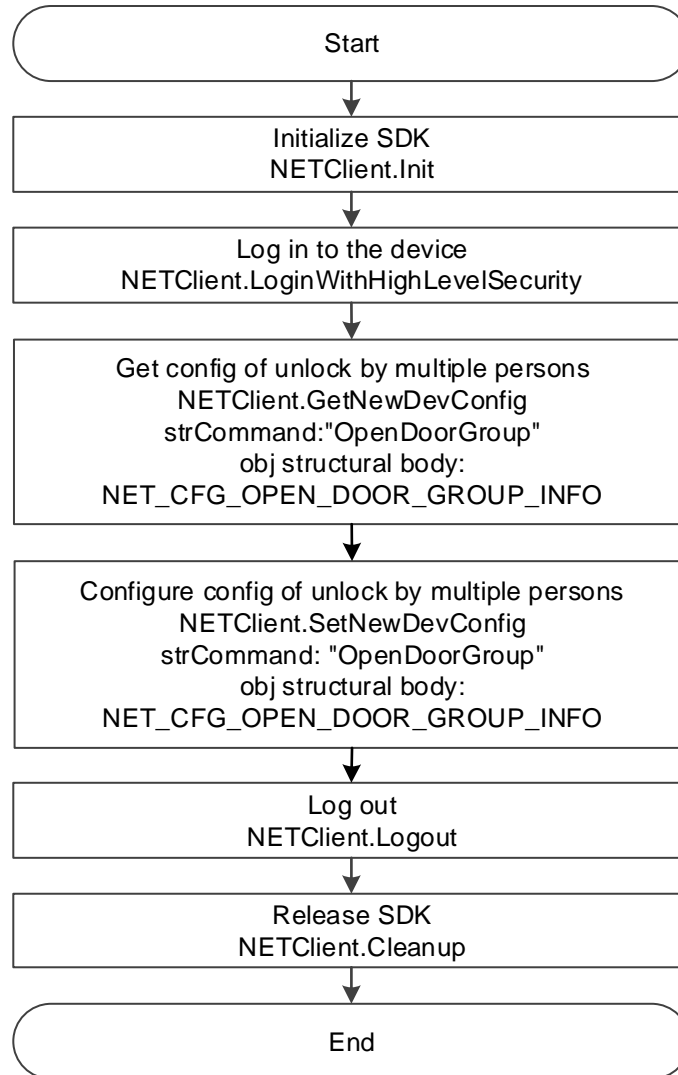
2.2.10.2.2 Interface Overview

Table 2-39 Description of interfaces for combination unlock by multiple persons

Interface	Description
NETClient.GetNewDevConfig	Query config information.
NETClient.SetNewDevConfig	Set config information.

2.2.10.2.3 Process Description

Figure 2-30 Combination unlock by multiple persons



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.GetNewDevConfig** to query the access info of combination unlock by multiple persons. The parameter strCommand is "OpenDoorGroup", and the corresponding obj structural body is NET_CFG_OPEN_DOOR_GROUP_INFO.
- Step 4 Call **NETClient.SetNewDevConfig** to set the access info of combination unlock by multiple persons. The parameter strCommand is "OpenDoorGroup", and the obj corresponding structural body is NET_CFG_OPEN_DOOR_GROUP_INFO.

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

Step 6 After using all SDK functions, call **NETClient.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 either card or fingerprint.

2.2.10.2.4 Sample Code

```
public bool GetConfig()
{
    bool bRet = false;
    try
    {
        cfg_info.stuGroupInfo = new NET_CFG_OPEN_DOOR_GROUP[4];
        for (int i = 0; i < cfg_info.stuGroupInfo.Length; i++)
        {
            cfg_info.stuGroupInfo[i].stuGroupDetail = new
NET_CFG_OPEN_DOOR_GROUP_DETAIL[64];

            object objTemp = cfg_info;
            bRet = NETClient.GetNewDevConfig(loginID,
comboBox_Door.SelectedIndex, CFG_CMD_OPEN_DOOR_GROUP, ref objTemp,
typeof(NET_CFG_OPEN_DOOR_GROUP_INFO), 10000);
            if (bRet)
            {
                cfg_info = (NET_CFG_OPEN_DOOR_GROUP_INFO)objTemp;
            }
            else
            {
                MessageBox.Show(NETClient.GetLastError());
            }
        }
    }
    catch (NETClientExcetion nex)
```

```

        {
            MessageBox.Show(nex.Message);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        return bRet;
    }

    public bool SetConfig(NET_CFG_OPEN_DOOR_GROUP_INFO cfg)
    {
        bool bRet = false;
        try
        {
            bRet = NETClient.SetNewDevConfig(loginID,
            comboBox_Door.SelectedIndex, CFG_CMD_OPEN_DOOR_GROUP, (object)cfg,
            typeof(NET_CFG_OPEN_DOOR_GROUP_INFO), 5000);
            if (!bRet)
            {
                MessageBox.Show(NETClient.GetLastError());
            }
        }
        catch (NETClientExcetion nex)
        {
            MessageBox.Show(nex.Message);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        return bRet;
    }
}

```

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.

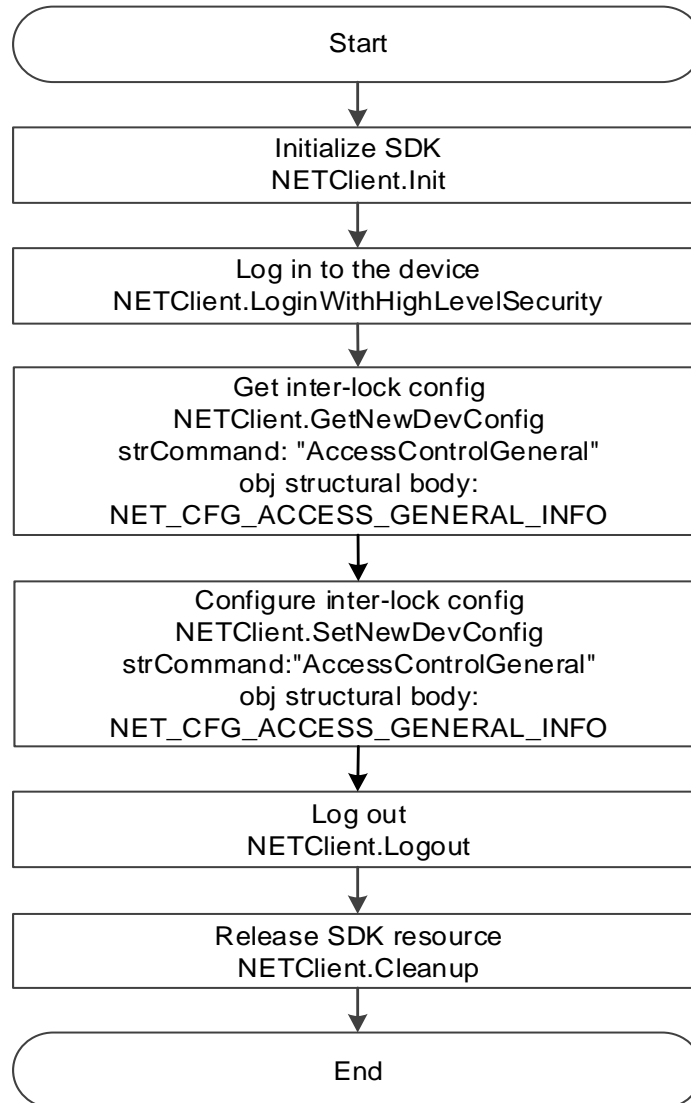
2.2.10.3.2 Interface Overview

Table 2-40 Description of inter-door lock interfaces

Interface	Description
NETClient.GetNewDevConfig	Query config information.
NETClient.SetNewDevConfig	Set config information.

2.2.10.3.3 Process Description

Figure 2-31 Inter-door lock config



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.GetNewDevConfig** to query the access inter-door lock info. The parameter strCommand is "AccessControlGeneral", and the corresponding obj structural body is NET_CFG_ACCESS_GENERAL_INFO.
- Step 4 Call **NETClient.SetNewDevConfig** to set the access inter-door lock info. The parameter strCommand is "AccessControlGeneral", and the corresponding obj structural body is NET_CFG_ACCESS_GENERAL_INFO.

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

Step 6 After using all SDK functions, call **NETClient. Cleanup** to release SDK resources.

Note

One device supports only one inter-door lock scheme.

2.2.10.3.4 Sample Code

```
//get inter-lock config
public bool GetConfig()
{
    bool bRet = false;
    try
    {
        object objTemp = new object();
        bRet = NETClient.GetNewDevConfig(loginID, -1,
CFG_CMD_ACCESS_GENERAL, ref objTemp,
typeof(NET_CFG_ACCESS_GENERAL_INFO), 5000);
        if (bRet)
        {
            cfg = (NET_CFG_ACCESS_GENERAL_INFO)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return bRet;
}

public bool SetConfig(NET_CFG_ACCESS_GENERAL_INFO cfg)
{
    bool bRet = false;
```

```

        try
        {
            bRet = NETClient.SetNewDevConfig(loginID, -1,
CFG_CMD_ACCESS_GENERAL, (object)cfg, typeof(NET_CFG_ACCESS_GENERAL_INFO),
5000);

            if (!bRet)
            {
                MessageBox.Show(NETClient.GetLastError());
            }
        }
        catch (NETClientExcetion nex)
        {
            MessageBox.Show(nex.Message);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        return bRet;
    }

```

2.2.10.4 Anti-passback

2.2.10.4.1 Introduction

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

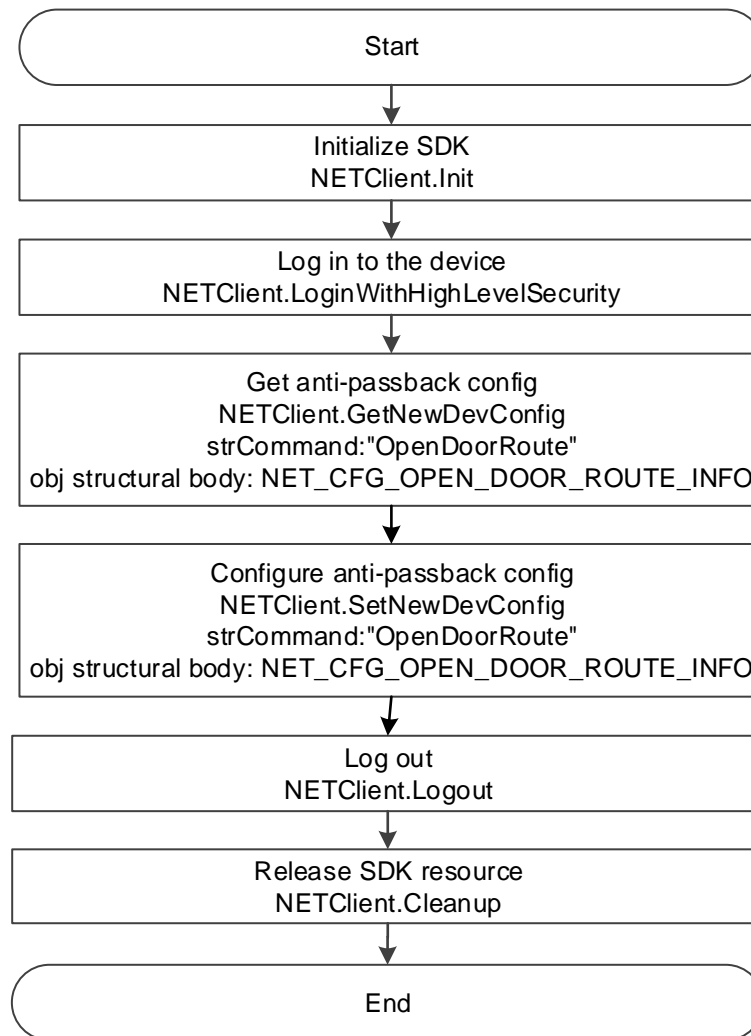
2.2.10.4.2 Interface Overview

Table 2-41 Description of anti-passback interfaces

Interface	Description
NETClient.GetNewDevConfig	Query config information.
NETClient.SetNewDevConfig	Set config information.

2.2.10.4.3 Process Description

Figure 2-32 Anti-passback config



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.GetNewDevConfig** to query the access anti-passback info. The parameter strCommand is "OpenDoorRoute", and the corresponding obj structure body is NET_CFG_OPEN_DOOR_ROUTE_INFO.
- Step 4 Call **NETClient.SetNewDevConfig** to set the access anti-passback info. T The parameter strCommand is "OpenDoorRoute", and the corresponding obj structure body is NET_CFG_OPEN_DOOR_ROUTE_INFO.
- Step 5 After completing this process, call **NETClient.Logout** to log out of the device.
- Step 6 After using all SDK functions, call **NETClient.Cleanup** to release SDK resources.

Note

One device supports only one anti-passback scheme.

2.2.10.4.4 Sample Code

```
//get anti-passback config
public bool GetConfig()
{
    bool bRet = false;
    try
    {
        object objTemp = new object();
        bRet = NETClient.GetNewDevConfig(loginID, -1,
CFG_CMD_OPEN_DOOR_ROUTE, ref objTemp,
typeof(NET_CFG_OPEN_DOOR_ROUTE_INFO), 5000);
        if (bRet)
        {
            cfg = (NET_CFG_OPEN_DOOR_ROUTE_INFO)objTemp;
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return bRet;
}

public bool SetConfig(NET_CFG_OPEN_DOOR_ROUTE_INFO cfg)
{
    bool bRet = false;
    try
    {
        bRet = NETClient.SetNewDevConfig(loginID, -1,
CFG_CMD_OPEN_DOOR_ROUTE, (object)cfg,
typeof(NET_CFG_OPEN_DOOR_ROUTE_INFO), 5000);
        if (!bRet)
```



```

        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (NETClientExcetion nex)
    {
        MessageBox.Show(nex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    return bRet;
}

```

2.2.10.5 Unlock Password

2.2.10.5.1 Introduction

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

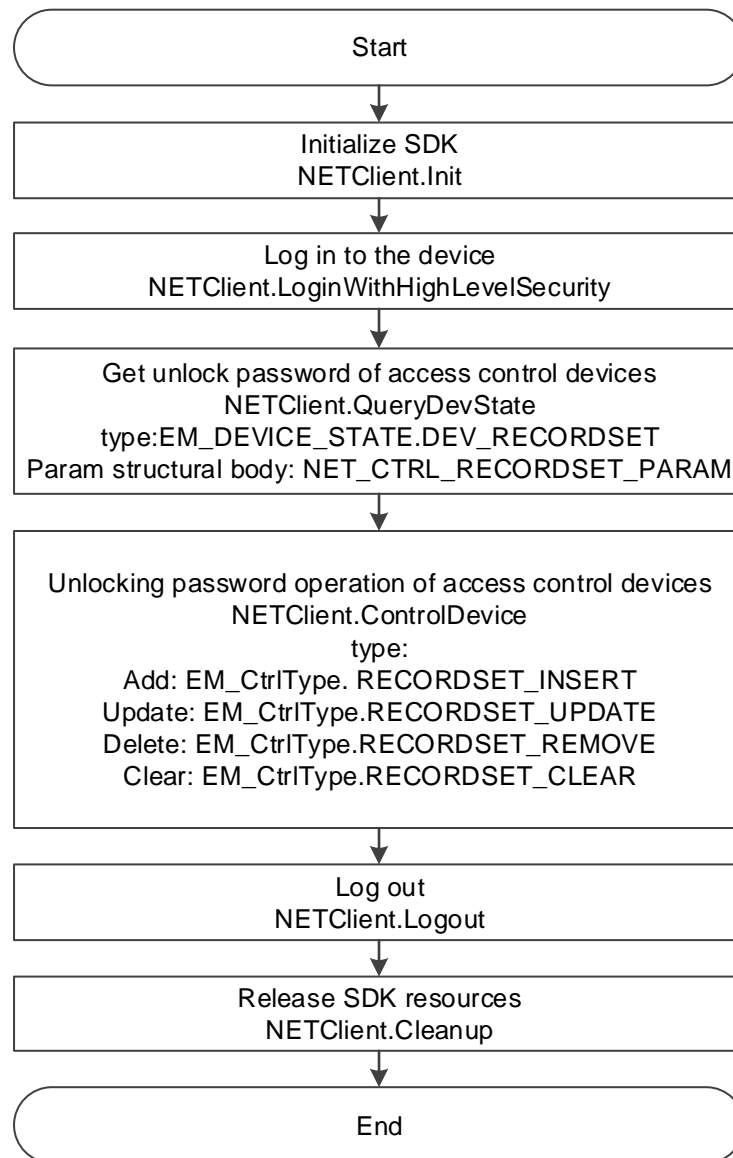
2.2.10.5.2 Interface Overview

Table 2-42 Description of unlock password interface

Interface	Description
NETClient.ControlDevice	Device control.

2.2.10.5.3 Process Description

Figure 2-33 Unlock password config



Process

Step 1 Call **NETClient.Init** to initialize SDK.

Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.

Step 3 Call **NETClient.QueryDevState** interface to get unlock password information.

Table 2-43 Description and structure of type (1)

Type	Description	Param
EM_DEVICE_STATE.D EV_RECORDSET	Get unlock password	NET_CTRL_RECORDSET_PARAM

Step 4 Call **NETClient.ControlDevice** to update unlock password information.

Table 2-44 Description and structure of type (2)

Type	Description	Param
EM_CtrlType.RECORDSET_INSERT	Add unlock password	NET_CTRL_RECORDSET_INSERT_PARAM NET_RECORDSET_ACCESS_CTL_PWD
EM_CtrlType.RECORDSET_UPDATE	Update unlock password	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_ACCESS_CTL_PWD
EM_CtrlType.RECORDSET_REMOVE	Delete unlock password	NET_CTRL_RECORDSET_PARAM
EM_CtrlType.RECORDSET_Clear	Clear unlock password	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_ACCESS_CTL_PWD

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

Step 6 After using all SDK functions, call **NETClient. Cleanup** to release SDK resources.

Note

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

2.2.10.5.4 Sample Code

```
// Get password
IntPtr pBuf = IntPtr.Zero;

NET_RECORDSET_ACCESS_CTL_PWD result = new
NET_RECORDSET_ACCESS_CTL_PWD();
NET_CTRL_RECORDSET_PARAM stuParam = new
NET_CTRL_RECORDSET_PARAM();

try
{
    pBuf = Marshal.AllocHGlobal(Marshal.SizeOf(result));

    //package for pwd
    result.nRecNo = Convert.ToInt32(textBox_RecNo.Text.Trim());

    result.dwSize = (uint)Marshal.SizeOf(result);
    Marshal.StructureToPtr(result, pBuf, true);

    //package stuParam
    stuParam.pBuf = pBuf;
    stuParam.nBufLen = Marshal.SizeOf(result);
    stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLPWD;
    stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);
```

```

        object obj = stuParam;

        bool bRet = NETClient.QueryDevState(loginID,
(int)EM_DEVICE_STATE.DEV_RECORDSET, ref obj,
typeof(NET_CTRL_RECORDSET_PARAM), 3000);
        if (bRet)
        {
            update_pwd =
(NET_RECORDSET_ACCESS_CTL_PWD)Marshal.PtrToStructure(pBuf,
typeof(NET_RECORDSET_ACCESS_CTL_PWD));
            m_SelectDoorsAry = update_pwd.sznDoors;
            textBox_OpenPwd.Text =
Encoding.UTF8.GetString(update_pwd.szDoorOpenPwd);
            textBox_UserID.Text =
Encoding.UTF8.GetString(update_pwd.szUserID);
            MessageBox.Show("Get succeed");
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        Marshal.FreeHGlobal(pBuf);
    }

// Add password
    IntPtr pParam = IntPtr.Zero;
    IntPtr pBuf = IntPtr.Zero;
    NET_CTRL_RECORDSET_INSERT_PARAM stuInsertParam = new
NET_CTRL_RECORDSET_INSERT_PARAM();
    NET_CTRL_RECORDSET_INSERT_PARAM stuOutParam = new
NET_CTRL_RECORDSET_INSERT_PARAM();

    NET_RECORDSET_ACCESS_CTL_PWD stuPwd = new
NET_RECORDSET_ACCESS_CTL_PWD();

```

```

        object obj = stuPwd;
        InitStruct(ref obj);
        stuPwd = (NET_RECORDSET_ACCESS_CTL_PWD)obj;
        stuPwd.dwSize = (uint)Marshal.SizeOf(stuPwd);

        Encoding.Default.GetBytes( textBox_UserID.Text, 0, textBox_UserID.Text.Length,
        stuPwd.szUserID, 0);

        Encoding.Default.GetBytes(textBox_OpenPwd.Text, 0,
        textBox_OpenPwd.Text.Length, stuPwd.szDoorOpenPwd, 0);
        if (m_selectDoorsList.Count > 0)
        {
            for (int i = 0; i < m_selectDoorsList.Count; i++)
            {
                stuPwd.sznDoors[i] = m_selectDoorsList[i];
            }
        }
        stuPwd.nDoorNum = m_selectDoorsList.Count;

        try
        {
            pParam =
            Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_PARAM)));
            pBuf =
            Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_PWD)));

            Marshal.StructureToPtr(stuPwd, pBuf, true);

            //package stuInsertParam
            stuInsertParam.stuCtrlRecordSetInfo.pBuf = pBuf;
            stuInsertParam.stuCtrlRecordSetInfo.nBufLen = Marshal.SizeOf(stuPwd);
            stuInsertParam.dwSize = (uint)Marshal.SizeOf(stuInsertParam);
            stuInsertParam.stuCtrlRecordSetInfo.dwSize =
            (uint)Marshal.SizeOf(stuInsertParam.stuCtrlRecordSetInfo);
            stuInsertParam.stuCtrlRecordSetInfo.emType =
            EM_NET_RECORD_TYPE.ACCESSCTLPWD;
            stuInsertParam.stuCtrlRecordSetResult.dwSize =
            (uint)Marshal.SizeOf(typeof(NET_CTRL_RECORDSET_INSERT_OUT));
            Marshal.StructureToPtr(stuInsertParam, pParam, true);

```

```

        bool bRet = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_INSERT, pParam, 3000);

        stuOutParam =
(NET_CTRL_RECORDSET_INSERT_PARAM)Marshal.PtrToStructure(pParam,
typeof(NET_CTRL_RECORDSET_INSERT_PARAM));
        if (bRet && stuOutParam.stuCtrlRecordSetResult.nRecNo > 0)
        {
            MessageBox.Show("Inster succeed. RecNO:" +
stuOutParam.stuCtrlRecordSetResult.nRecNo);
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    //free resource
    finally
    {
        Marshal.FreeHGlobal(pParam);
        Marshal.FreeHGlobal(pBuf);
    }

// Get and then update
        Encoding.Default.GetBytes(textBox_UserID.Text, 0, textBox_UserID.Text.Length,
update_pwd.szUserID, 0);
        Encoding.Default.GetBytes(textBox_OpenPwd.Text, 0,
textBox_OpenPwd.Text.Length, update_pwd.szDoorOpenPwd, 0);
        if (m_selectDoorsList.Count > 0)
        {
            for (int i = 0; i < m_selectDoorsList.Count; i++)
            {
                update_pwd.sznDoors[i] = m_selectDoorsList[i];
            }
        }
        update_pwd.nDoorNum = m_selectDoorsList.Count;

```

```

        bool bRet = false;
        IntPtr pParam = IntPtr.Zero;
        IntPtr pBuf = IntPtr.Zero;
        NET_CTRL_RECORDSET_PARAM stuParam = new
NET_CTRL_RECORDSET_PARAM();
        try
        {
            pParam = Marshal.AllocHGlobal(Marshal.SizeOf(stuParam));
            pBuf = Marshal.AllocHGlobal(Marshal.SizeOf(update_pwd));

            Marshal.StructureToPtr(update_pwd, pBuf, true);
            stuParam.pBuf = pBuf;
            stuParam.nBufLen = Marshal.SizeOf(update_pwd);
            stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLPWD;
            stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);
            Marshal.StructureToPtr(stuParam, pParam, true);

            bRet = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_UPDATE, pParam, 3000);
            if (bRet)
            {
                MessageBox.Show("Update succeed");
            }
            else
            {
                MessageBox.Show(NETClient.GetLastError());
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        finally
        {
            Marshal.FreeHGlobal(pParam);
            Marshal.FreeHGlobal(pBuf);
        }

```

```

// Remove password

    bool result = false;
    IntPtr pParam = IntPtr.Zero;
    IntPtr pBuf = IntPtr.Zero;
    NET_CTRL_RECORDSET_PARAM stuParam = new
NET_CTRL_RECORDSET_PARAM();
    stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLPWD;
    stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);
    stuParam.pBuf = IntPtr.Zero;
    stuParam.nBufLen = 0;
    try
    {
        pParam = Marshal.AllocHGlobal(Marshal.SizeOf(stuParam));
        pBuf =
Marshal.AllocHGlobal(Marshal.SizeOf(int.Parse(textBox_RecNo.Text)));
        Marshal.StructureToPtr(int.Parse(textBox_RecNo.Text), pBuf, true);
        stuParam.pBuf = pBuf;
        stuParam.nBufLen = Marshal.SizeOf(int.Parse(textBox_RecNo.Text));
        Marshal.StructureToPtr(stuParam, pParam, true);

        result = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_REMOVE, pParam, 3000);
        if (result)
        {
            MessageBox.Show("Remove succeed");
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        Marshal.FreeHGlobal(pBuf);
        Marshal.FreeHGlobal(pParam);
    }

```



```

    }

// Clear password
    IntPtr pParam = IntPtr.Zero;
    NET_CTRL_RECORDSET_PARAM stuParam = new
NET_CTRL_RECORDSET_PARAM();
    stuParam.emType = EM_NET_RECORD_TYPE.ACCESSCTLPWD;
    stuParam.dwSize = (uint)Marshal.SizeOf(stuParam);

    pParam = Marshal.AllocHGlobal(Marshal.SizeOf(stuParam));
    Marshal.StructureToPtr(stuParam, pParam, true);

    bool result = NETClient.ControlDevice(loginID,
EM_CtrlType.RECORDSET_CLEAR, pParam, 3000);
    if (result)
    {
        MessageBox.Show("Clear succeed");
    }
    else
    {
        MessageBox.Show(NETClient.GetLastError());
    }

//

```

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

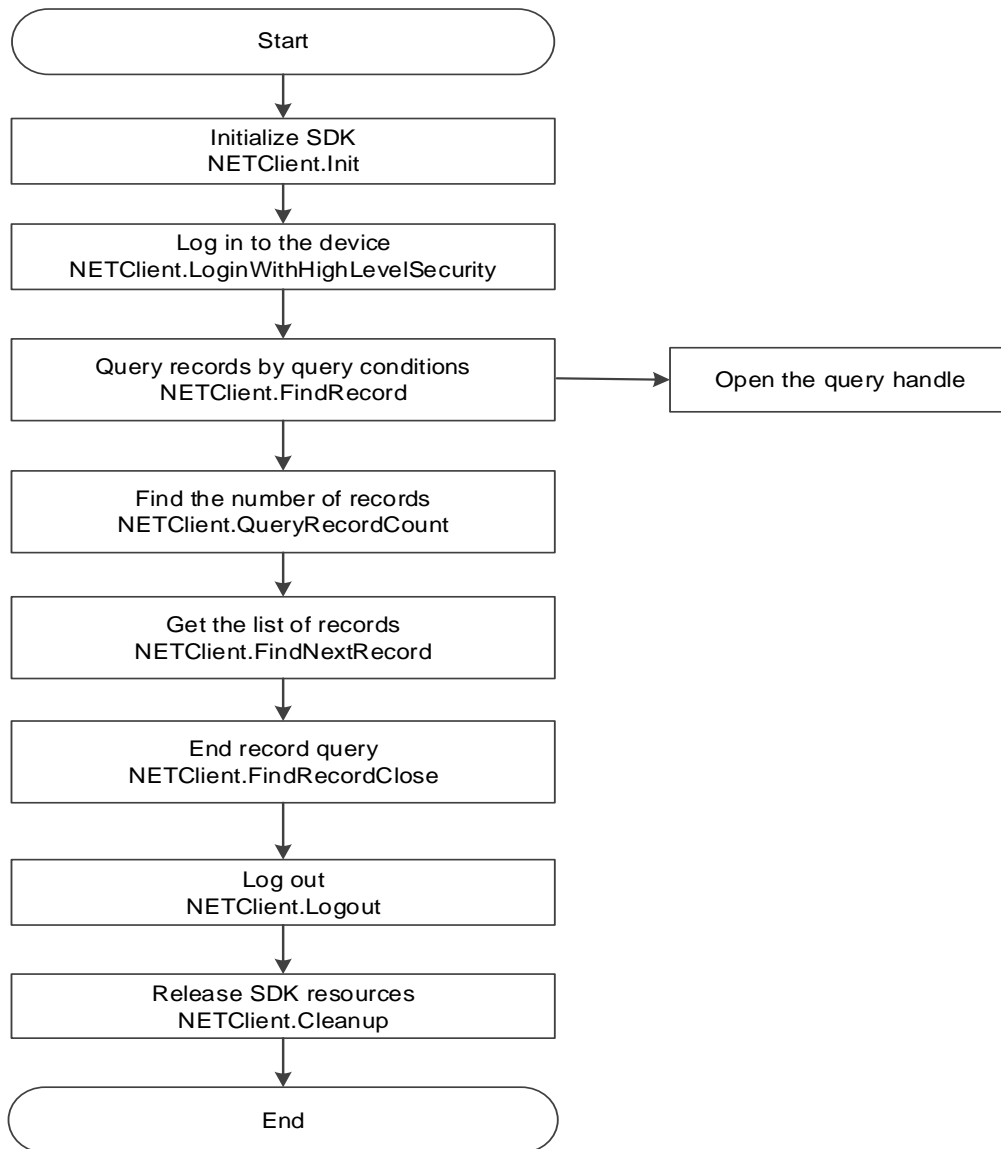
Table 2-45 Description of record query interfaces

Interface	Description
NETClient.QueryRecordCount	Find the count of records.
NETClient.FindRecord	Query records by query conditions.

NETClient.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.
NETClient.FindRecordClose	End record query.

2.2.11.1.3 Process Description

Figure 2-34 Record query



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.FindRecord** to get the query handle.
emType unlock record: EM_NET_RECORD_TYPE .ACCESSCTLCARDREC.
- Step 4 Call **NETClient.QueryRecordCount** to find the count of records.
- Step 5 Call **NETClient.FindNextRecord** to get the list of records.
- Step 6 After query, call **NETClient.FindRecordClose** to close the query handle.
- Step 7 After completing this process, call **NETClient.Logout** to log out of the device.

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

2.2.11.1.4 Sample Code

```
// Start query
m_LogNum = 0;
if (IntPtr.Zero == m_FindDoorRecordID)
{
    NET_FIND_RECORD_ACCESSCTLCARDREC_CONDITION_EX condition
= new NET_FIND_RECORD_ACCESSCTLCARDREC_CONDITION_EX();
    condition.dwSize =
(uint)Marshal.SizeOf(typeof(NET_FIND_RECORD_ACCESSCTLCARDREC_CONDITION_EX
));
    condition.bTimeEnable = true;
    condition.stStartTime =
NET_TIME.FromDateTime(dateTimePicker_Start.Value);
    condition.stEndTime =
NET_TIME.FromDateTime(dateTimePicker_End.Value);
    object obj = condition;

    bool ret = NETClient.FindRecord(loginID,
EM_NET_RECORD_TYPE.ACCESSCTLCARDREC_EX, obj,
typeof(NET_FIND_RECORD_ACCESSCTLCARDREC_CONDITION_EX), ref
m_FindDoorRecordID, 10000);
    if (!ret)
    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }
    btn_StartQuery.Text = "StopQuery";
    btn_NextFind.Enabled = true;
    btn_GetRecordCount.Enabled = true;
    dateTimePicker_Start.Enabled = false;
    dateTimePicker_End.Enabled = false;
}
else
{
    NETClient.FindRecordClose(m_FindDoorRecordID);
    m_FindDoorRecordID = IntPtr.Zero;
    btn_StartQuery.Text = "StartQuery";
    btn_NextFind.Enabled = false;
    btn_GetRecordCount.Enabled = false;
}
```

```

        dateTimePicker_Start.Enabled = true;
        dateTimePicker_End.Enabled = true;
        textBox_Count.Text = "";

        listView_DoorRecord.Items.Clear();
    }

// Get number of query
    if (IntPtr.Zero == m_FindDoorRecordID)
    {
        return;
    }

    int nCount = 0;
    try
    {
        if (NETClient.QueryRecordCount(m_FindDoorRecordID, ref nCount, 3000))
        {
            textBox_Count.Text = nCount.ToString();
        }
        else
        {
            MessageBox.Show(NETClient.GetLastError());
            return;
        }
    }
    catch (NETClientExcetion ex)
    {
        MessageBox.Show(NETClient.GetLastError());
        return;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }

// Query the next record
    listView_DoorRecord.Items.Clear();
    int max = 10;

```

```

        int retNum = 0;
        List<object> ls = new List<object>();
        for (int i = 0; i < max; i++)
        {
            NET_RECORDSET_ACCESS_CTL_CARDREC cardrec = new
NET_RECORDSET_ACCESS_CTL_CARDREC();
            cardrec.dwSize =
(uint)Marshal.SizeOf(typeof(NET_RECORDSET_ACCESS_CTL_CARDREC));
            ls.Add(cardrec);
        }
        NETClient.FindNextRecord(m_FindDoorRecordID, max, ref retNum, ref ls,
typeof(NET_RECORDSET_ACCESS_CTL_CARDREC), 10000);
        BeginInvoke(new Action(() =>
        {
            foreach (var item in ls)
            {
                NET_RECORDSET_ACCESS_CTL_CARDREC info =
(NET_RECORDSET_ACCESS_CTL_CARDREC)item;
                var listitem = new ListViewItem();
                listitem.Text = info.nRecNo.ToString();
                listitem.SubItems.Add(info.stuTime.ToString());
                listitem.SubItems.Add(info.szCardNo);
                listitem.SubItems.Add(info.bStatus.ToString());
                listitem.SubItems.Add(info.nDoor.ToString());
                listitem.SubItems.Add(info.emMethod.ToString());
                if (listView_DoorRecord != null)
                {
                    listView_DoorRecord.BeginUpdate();
                    listView_DoorRecord.Items.Add(listitem);
                    listView_DoorRecord.EndUpdate();
                }
            }
        }));
    }
}

```

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.

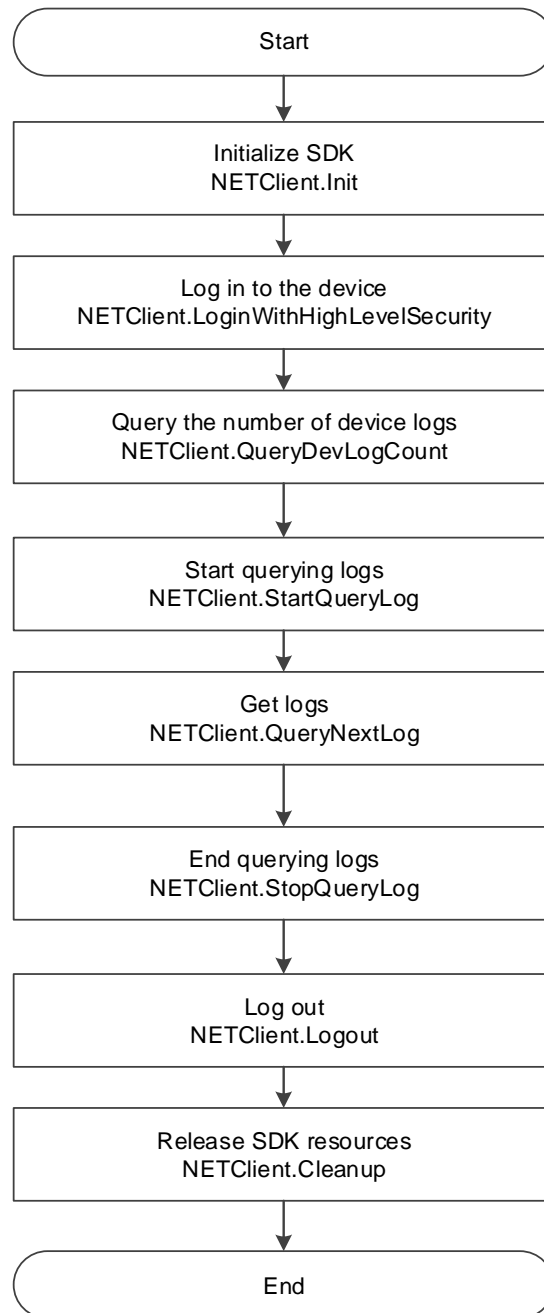
2.2.11.2.2 Interface Overview

Table 2-46 Description of device log interfaces

Interface	Description
NETClient.QueryDevLogCount	Query the count of device logs.
NETClient.StartQueryLog	Start querying logs.
NETClient.QueryNextLog	Get logs.
NETClient.StopQueryLog	Stop querying logs.

2.2.11.2.3 Process Description

Figure 2-35 Device log



Process

Step 1 Call **NETClient.Init** to initialize SDK.

- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.QueryDevLogCount** to set the number of queried logs.
- Step 4 Call **NETClient.StartQueryLog** to start querying log information.
- pInParam: NET_IN_START_QUERYLOG.
 - pOutParam: NET_OUT_START_QUERYLOG.
- Step 5 Call **NETClient.QueryNextLog** to get log information.
- pInParam: NET_IN_QUERYNEXTLOG.
 - pOutParam: NET_OUT_QUERYNEXTLOG.
- Step 6 Call **NETClient.StopQueryLog** to stop querying logs.
- Step 7 After completing this process, call **NETClient.Logout** to log out of the device.
- Step 8 After using all SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.2.11.2.4 Sample Code

```
// Start querying log info
    m_LogNum = 0;
    if (IntPtr.Zero == m_FindLogID)
    {
        NET_IN_START_QUERYLOG stuIn = new NET_IN_START_QUERYLOG();
        stuIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_START_QUERYLOG));
        NET_OUT_START_QUERYLOG stuOut = new
NET_OUT_START_QUERYLOG();
        stuOut.dwSize =
(uint)Marshal.SizeOf(typeof(NET_OUT_START_QUERYLOG));

        m_FindLogID = NETClient.StartQueryLog(loginID, ref stuIn, ref stuOut,
5000); //CLIENT_StartQueryLog(m_LoginID, &stuIn, &stuOut, SDK_API_WAIT);
        if (IntPtr.Zero == m_FindLogID)
        {
            MessageBox.Show(NETClient.GetLastError());
            return;
        }

        btn_StartQuery.Text = "StopQuery";

        btn_NextLog.Enabled = true;
        btn_GetLogCount.Enabled = true;
    }
    else
    {
        NETClient.StopQueryLog(m_FindLogID);
        m_FindLogID = IntPtr.Zero;
        btn_StartQuery.Text = "StartQuery";
```

```

        btn_NextLog.Enabled = false;
        btn_GetLogCount.Enabled = false;
        textBox_LogCount.Text = "";

        listView_Log.Items.Clear();
    }

//Get number of records
    NET_IN_GETCOUNT_LOG_PARAM stuIn = new
NET_IN_GETCOUNT_LOG_PARAM();
    stuIn.dwSize =
(uint)Marshal.SizeOf(typeof(NET_IN_GETCOUNT_LOG_PARAM));
    NET_OUT_GETCOUNT_LOG_PARAM stuOut = new
NET_OUT_GETCOUNT_LOG_PARAM();
    stuOut.dwSize =
(uint)Marshal.SizeOf(typeof(NET_OUT_GETCOUNT_LOG_PARAM));
    if (NETClient.QueryDevLogCount(loginID, ref stuIn, ref stuOut, 5000))
    {
        textBox_LogCount.Text = stuOut.nLogCount.ToString();
    }
    else
    {
        MessageBox.Show(NETClient.GetLastError());
    }

// Query the next record
    listView_Log.Items.Clear();
    int max = 10;
    NET_IN_QUERYNEXTLOG stuIn = new NET_IN_QUERYNEXTLOG();
    stuIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_QUERYNEXTLOG));
    stuIn.nGetCount = max;

    NET_OUT_QUERYNEXTLOG stuOut = new NET_OUT_QUERYNEXTLOG();
    stuOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_QUERYNEXTLOG));
    stuOut.nMaxCount = max;
    stuOut.pstuLogInfo =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_LOG_INFO)) * stuOut.nMaxCount);

    NET_LOG_INFO[] logInfo = new NET_LOG_INFO[stuOut.nMaxCount];
    for (int i = 0; i < stuOut.nMaxCount; i++)
    {
        logInfo[i].dwSize = (uint)Marshal.SizeOf(typeof(NET_LOG_INFO));
    }

```



```

        logInfo[i].stuLogMsg.dwSize =
(uint)Marshal.SizeOf(typeof(NET_LOG_MESSAGE));
        IntPtr pDst = IntPtr.Add(stuOut.pstuLogInfo,
Marshal.SizeOf(typeof(NET_LOG_INFO)) * i);
        Marshal.StructureToPtr(logInfo[i], pDst, true);
    }

    if (NETClient.QueryNextLog(m_FindLogID, ref stuIn, ref stuOut, 5000))
    {
        if (stuOut.nRetCount > 0)
        {
            BeginInvoke(new Action(() =>
            {
                for (int i = 0; i < stuOut.nRetCount; i++)
                {
                    IntPtr pDst = IntPtr.Add(stuOut.pstuLogInfo,
Marshal.SizeOf(typeof(NET_LOG_INFO)) * i);
                    NET_LOG_INFO retInfo =
(NET_LOG_INFO)Marshal.PtrToStructure(pDst, typeof(NET_LOG_INFO));

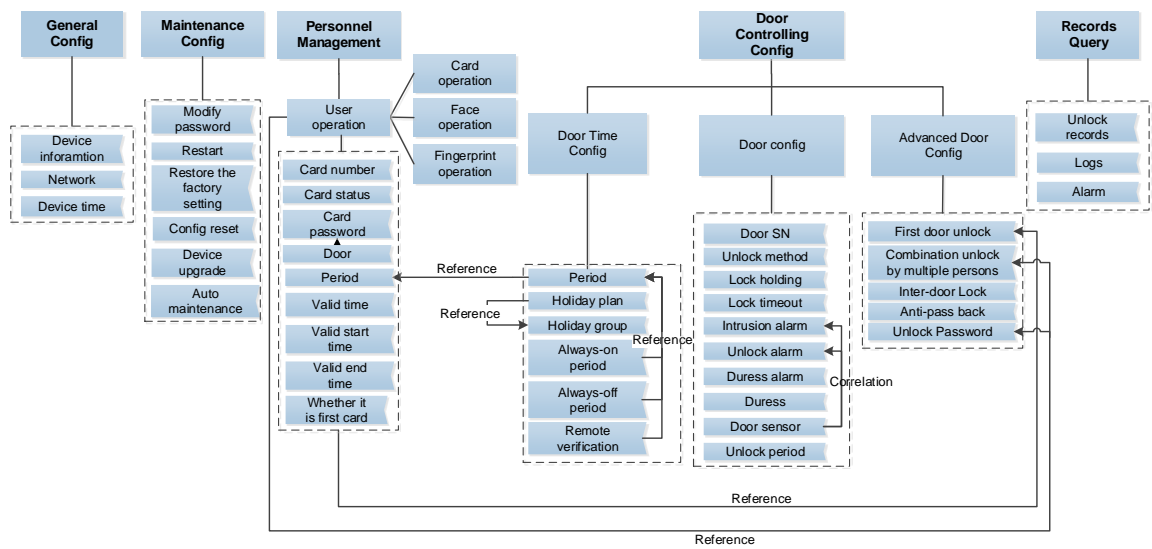
                    m_LogNum += 1;
                    var listitem = new ListViewItem();
                    listitem.Text = m_LogNum.ToString();
                    listitem.SubItems.Add(retInfo.stuTime.ToString());
                    listitem.SubItems.Add(retInfo.szUserName);
                    listitem.SubItems.Add(retInfo.szLogType);
                    listitem.SubItems.Add(retInfo.stuLogMsg.szLogMessage);
                    if (listView_Log != null)
                    {
                        listView_Log.BeginUpdate();
                        listView_Log.Items.Add(listitem);
                        listView_Log.EndUpdate();
                    }
                }
            }));
        }
    }
    else

```

```
{
    MessageBox.Show(NETClient.GetLastError());
}
```

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

Figure 2-36 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 Viewing Device Information

2.3.3.1 Capability Set Query

2.3.3.1.1 Introduction

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

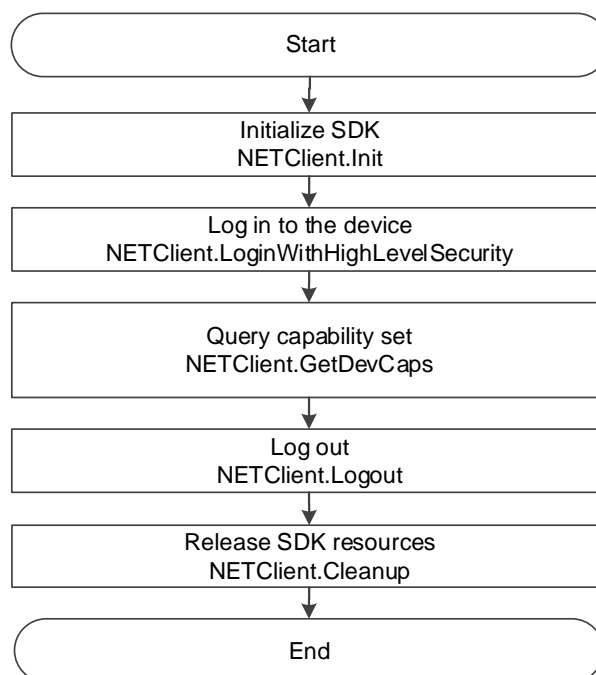
2.3.3.1.2 Interface Overview

Table 2-47 Description of capability set query interface

Interface	Description
NETClient.GetDevCaps	Get the access control capability (such as access control, user, card, face, and fingerprint).

2.3.3.1.3 Process Description

Figure 2-37 Device information viewing



Process

- Step 1 Call **NETClient. Init** to initialize SDK.
- Step 2 Call **NETClient. LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient. GetDevCaps** and assign **NET_ACCESSCONTROL_CAPS** to nType, to get the access control.
- Step 4 After completing this process, call **NETClient. Logout** to log out of the device.
- Step 5 After using all SDK functions, call **NETClient. Cleanup** to release SDK resources.

2.3.3.1.4 Sample Code

```
NET_IN_AC_CAPS stuIn = new NET_IN_AC_CAPS();
stuIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_AC_CAPS));
NET_OUT_AC_CAPS stuOut = new NET_OUT_AC_CAPS();
stuOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_AC_CAPS));
stuOut.stuACCaps = new NET_AC_CAPS();
stuOut.stuUserCaps = new NET_ACCESS_USER_CAPS();
stuOut.stuCardCaps = new NET_ACCESS_CARD_CAPS();
stuOut.stuFingerprintCaps = new NET_ACCESS_FINGERPRINT_CAPS();
stuOut.stuFaceCaps = new NET_ACCESS_FACE_CAPS();

IntPtr ptrIn = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_IN_AC_CAPS)));
IntPtr ptrOut = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_OUT_AC_CAPS)));
Marshal.StructureToPtr(stuIn, ptrIn, true);
Marshal.StructureToPtr(stuOut, ptrOut, true);
bool bRet = NETClient.GetDevCaps(m_LoginID,
EM_DEVCAP_TYPE.ACCESSCONTROL_CAPS, ptrIn, ptrOut, 5000);
if (bRet)
{
    stuOut = (NET_OUT_AC_CAPS)Marshal.PtrToStructure(ptrOut,
typeof(NET_OUT_AC_CAPS));
    m_AccessCount = stuOut.stuACCaps.nChannels;
}
else
{
    MessageBox.Show(NETClient.GetLastError());
}
```

2.3.3.2 Viewing Device Version and MAC

See "2.2.3.2 Viewing Device Version and MAC."

2.3.4 Network Setting

See "2.2.4 Network Setting."

2.3.5 Setting the Device Time

See "2.2.5 Device Time Getting and Setting."

2.3.6 Maintenance Config

See "2.2.6 Maintenance Config."

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).

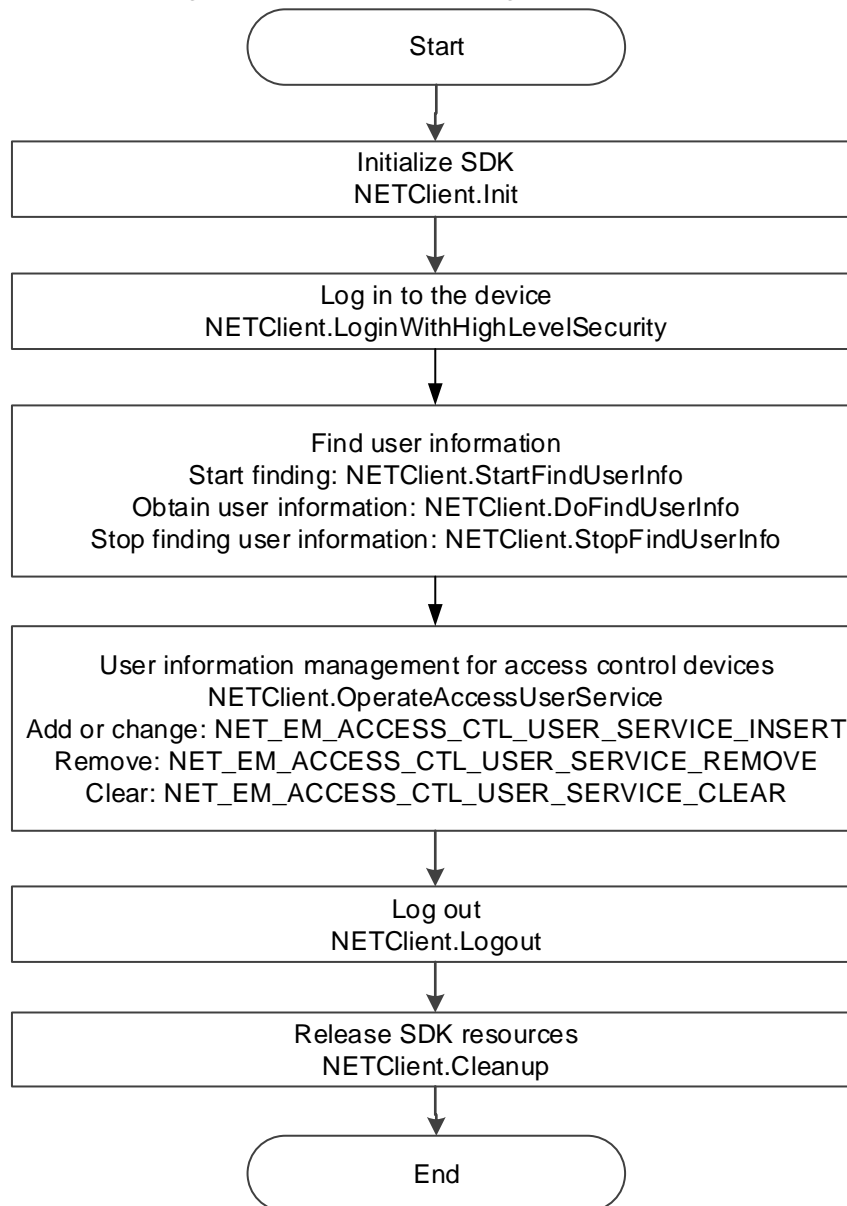
2.3.7.1.2 Interface Overview

Table 2-48 Description of user information interface

Interface	Description
NETClient.StartFindUserInfo	Start query user information.
NETClient.DoFindUserInfo	Stop getting user information.
NETClient.StopFindUserInfo	Stop querying user information.
NETClient.InsertOperateAccessUserService	Add or change access control user information.
NETClient.RemoveOperateAccessUserService	Delete access control user information.
NETClient.ClearOperateAccessUserService	Clear access control user information.

2.3.7.1.3 Process Description

Figure 2-38 User info management



Process

- Step 1 Call **NETClient. Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.StartFindUserInfo** to start finding the user information.
- Step 4 Call **NETClient.DoFindUserInfo** to obtain the user information.
- Step 5 Call **NETClient.StopFindUserInfo** to stop finding the user information.
- Step 6 Call **NETClient.InsertOperateAccessUserService** to add or change user information; call **NETClient.RemoveOperateAccessUserService** to delete user info; call **NETClient.ClearOperateAccessUserService** to clear user info.
- Step 7 After completing this process, call **NETClient. Logout** to log out of the device.
- Step 8 After using all SDK functions, call **NETClient. Cleanup** to release SDK resources.

2.3.7.1.4 Sample Code

```
private IntPtr m_FindUserID = IntPtr.Zero;
```

```

private List<NET_ACCESS_USER_INFO> userInfoList = new
List<NET_ACCESS_USER_INFO>();

//Get all user info
NET_IN_USERINFO_START_FIND stuStartIn = new NET_IN_USERINFO_START_FIND();
stuStartIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_USERINFO_START_FIND));

NET_OUT_USERINFO_START_FIND stuStartOut = new
NET_OUT_USERINFO_START_FIND();
stuStartOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_USERINFO_START_FIND));
stuStartOut.nTotalCount = 0;
stuStartOut.nCapNum = 50;
m_FindUserID = NETClient.StartFindUserInfo(m_LoginID, ref stuStartIn, ref stuStartOut,
5000);
if (IntPtr.Zero == m_FindUserID)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}

userInfoList.Clear();

NET_IN_USERINFO_DO_FIND stuFindIn = new NET_IN_USERINFO_DO_FIND();
stuFindIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_USERINFO_DO_FIND));
stuFindIn.nCount = QueryNum;

NET_OUT_USERINFO_DO_FIND stuFindOut = new NET_OUT_USERINFO_DO_FIND();
stuFindOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_USERINFO_DO_FIND));
stuFindOut.nMaxNum = QueryNum;

NET_ACCESS_USER_INFO[] stuOutUserInfo = new
NET_ACCESS_USER_INFO[stuFindOut.nMaxNum];
IntPtr outInfo = IntPtr.Zero;
outInfo = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_ACCESS_USER_INFO)) *
stuFindOut.nMaxNum);
for (int index = 0; index < stuFindOut.nMaxNum; index++)
{
    IntPtr outInfoIndex = outInfo + index *
Marshal.SizeOf(typeof(NET_ACCESS_USER_INFO));

```

```

        if (stuOutUserInfo[index].GetType() == typeof(NET_ACCESS_USER_INFO))
//if obj is boxed type of typeName, some param(ex. dwsize) need trans to unmanaged
memory
        {
            Marshal.StructureToPtr(stuOutUserInfo[index], outInfoIndex, true);
        }
        else
        {
            for (int i = 0; i < Marshal.SizeOf(typeof(NET_ACCESS_USER_INFO)); i++)
            {
                Marshal.WriteByte(outInfoIndex, i, 0);
            }
        }
    }
    stuFindOut.pstuInfo = outInfo;

    int startNum = 0;
    while (true)
    {
        stuFindIn.nStartNo = startNum;

        bool result = NETClient.DoFindUserInfo(m_FindUserID, ref stuFindIn, ref stuFindOut,
5000);
        if (!result)
        {
            break;
        }

        if (stuFindOut.nRetNum > 0)
        {
            startNum += stuFindOut.nRetNum;
            for (int i = 0; i < stuFindOut.nRetNum; i++)
            {
                var userinfo =
(NET_ACCESS_USER_INFO)Marshal.PtrToStructure(IntPtr.Add(stuFindOut.pstuInfo,
Marshal.SizeOf(typeof(NET_ACCESS_USER_INFO)) * i),
typeof(NET_ACCESS_USER_INFO));
                userInfoList.Add(userinfo);
            }
        }
    }
}

```



```

NETClient.StopFindUserInfo(m_FindUserID);

//Add or change
private NET_ACCESS_USER_INFO m_UserInfo = new NET_ACCESS_USER_INFO();
m_UserInfo.szUserID = txt_UserID.Text.Trim();
m_UserInfo.szName = txt_Name.Text.Trim();
m_UserInfo.szPsw = txt_Pwd.Text.Trim();
bool result = false;
NET_ACCESS_USER_INFO[] stuInArray = new NET_ACCESS_USER_INFO[1]
{ m_UserInfo };
NET_EM_FAILCODE[] stuOutErrArray = new NET_EM_FAILCODE[1];
result = NETClient.InsertOperateAccessUserService(m_LoginID, stuInArray, out
stuOutErrArray, 5000);
if (!result)
{
    for (int i = 0; i < stuOutErrArray.Length; i++)
    {
        MessageBox.Show(stuOutErrArray[i].emCode.ToString());
    }
}
//Delete
NET_EM_FAILCODE[] stuOutErrArray = new NET_EM_FAILCODE[1];
string[] InUserid = new string[] { szUserID }; // szUserID: szUserID of users to be deleted
bool result = NETClient.RemoveOperateAccessUserService(m_LoginID, InUserid, out
stuOutErrArray, 3000);
if (!result)
{
    for (int i = 0; i < stuOutErrArray.Length; i++)
    {
        MessageBox.Show(stuOutErrArray[i].emCode.ToString());
    }
}

```

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).

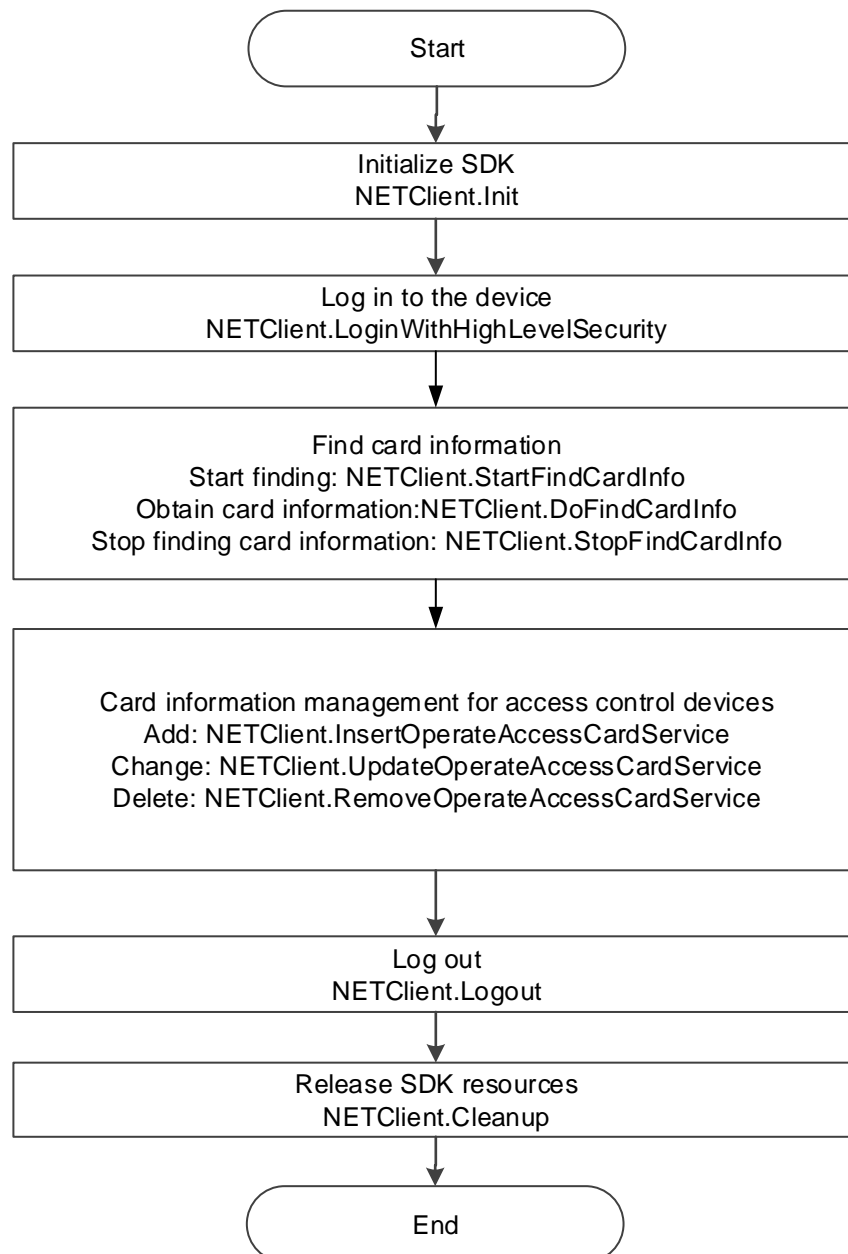
2.3.7.2.2 Interface Overview

Table 2-49 Description of card information interface

Interface	Description
NETClient.StartFindCardInfo	Start to find the card information.
NETClient.DoFindCardInfo	Obtain the card information.
NETClient.StopFindCardInfo	Stop finding the card information.
NETClient.InsertOperateAccessCardService	Add access control card information.
NETClient.RemoveOperateAccessCardService	Delete access control card information.
NETClient.UpdateOperateAccessCardService	Update access control card information.

2.3.7.2.3 Process Description

Figure 2-39 Management of card information



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.StartFindCardInfo** to start finding the card information.
- Step 4 Call **NETClient.DoFindCardInfo** to obtain the card information.
- Step 5 Call **NETClient.StopFindCardInfo** to stop finding the card information.
- Step 6 Call **NETClient.InsertOperateAccessCardService** to add the card information, call **NETClient.UpdateOperateAccessCardService** to update the card information, call **NETClient.RemoveOperateAccessCardService** to delete the card information.
- Step 7 After completing this process, call **NETClient.Logout** to log out of the device.
- Step 8 After using all SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.3.7.2.4 Sample Code

```
//Get
private List<NET_ACCESS_CARD_INFO> cardInfoList = new
List<NET_ACCESS_CARD_INFO>();
NET_IN_CARDINFO_START_FIND stuStartIn = new NET_IN_CARDINFO_START_FIND();
stuStartIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_CARDINFO_START_FIND));
stuStartIn.szUserID = m_UserInfo.szUserID;

NET_OUT_CARDINFO_START_FIND stuStartOut = new
NET_OUT_CARDINFO_START_FIND();
stuStartOut.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_CARDINFO_START_FIND));
stuStartOut.nTotalCount = 0;
stuStartOut.nCapNum = 10;

IntPtr cardFindId = NETClient.StartFindCardInfo(m_LoginID, ref stuStartIn, ref stuStartOut,
5000);
if (IntPtr.Zero != cardFindId)
{
    int nStartNo = 0;
    bool m_blsDoFindNextCard = true;
    while (m_blsDoFindNextCard)
    {
        int nRecordNum = 0;

        NET_IN_CARDINFO_DO_FIND stuFindIn = new NET_IN_CARDINFO_DO_FIND();
        stuFindIn.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_CARDINFO_DO_FIND));
        stuFindIn.nStartNo = nStartNo;
        stuFindIn.nCount = 10;
```

```

        NET_OUT_CARDINFO_DO_FIND stuFindOut = new
NET_OUT_CARDINFO_DO_FIND();
        stuFindOut.dwSize =
(uint)Marshal.SizeOf(typeof(NET_OUT_CARDINFO_DO_FIND));
        stuFindOut.nMaxNum = 10;
        stuFindOut.pstuInfo =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_ACCESS_CARD_INFO)) *
stuFindOut.nMaxNum);

        NET_ACCESS_CARD_INFO[] pCardInfo = new
NET_ACCESS_CARD_INFO[stuFindOut.nMaxNum];
        for (int i = 0; i < stuFindOut.nMaxNum; i++)
        {
            IntPtr pDst = IntPtr.Add(stuFindOut.pstuInfo,
Marshal.SizeOf(typeof(NET_ACCESS_CARD_INFO)) * i);
            Marshal.StructureToPtr(pCardInfo[i], pDst, true);
        }

        bool ret = NETClient.DoFindCardInfo(cardFindId, ref stuFindIn, ref stuFindOut, 5000);
        if (ret)
        {
            if (stuFindOut.nRetNum > 0)
            {
                nRecordNum = stuFindOut.nRetNum;
                for (int i = 0; i < nRecordNum; i++)
                {
                    IntPtr pDst = IntPtr.Add(stuFindOut.pstuInfo,
Marshal.SizeOf(typeof(NET_ACCESS_CARD_INFO)) * i);
                    NET_ACCESS_CARD_INFO stuInfo =
(NET_ACCESS_CARD_INFO)Marshal.PtrToStructure(pDst,
typeof(NET_ACCESS_CARD_INFO));
                    cardInfoList.Add(stuInfo);
                }
            }

            if (nRecordNum < 10)
            {
                break;
            }
            else
            {

```

```

        nStartNo += nRecordNum;
    }
}
else
{
    break;
}
}

NETClient.StopFindCardInfo(cardFindId);
}
else
{
    MessageBox.Show(NETClient.GetLastError());
}

//Add
NET_ACCESS_CARD_INFO[] stuInArray = new NET_ACCESS_CARD_INFO[1]
{ m_CardInfo };
NET_EM_FAILCODE[] stuOutErrArray = new NET_EM_FAILCODE[1];
stuInArray[0].emType = (EM_ACCESSCTLCARD_TYPE)cmb_CardType.SelectedIndex;
stuInArray[0].szCardNo = txt_CardNum.Text;
result = NETClient.InsertOperateAccessCardService(m_LoginID, stuInArray, out
stuOutErrArray, 5000);
if (!result)
{
    for (int i = 0; i < stuOutErrArray.Length; i++)
    {
        MessageBox.Show(stuOutErrArray[i].emCode.ToString());
    }
}
//Update
NET_ACCESS_CARD_INFO[] stuInArray = new NET_ACCESS_CARD_INFO[1]
{ m_CardInfo };
NET_EM_FAILCODE[] stuOutErrArray = new NET_EM_FAILCODE[1];

stuInArray[0].emType = (EM_ACCESSCTLCARD_TYPE)cmb_CardType.SelectedIndex;
result = NETClient.UpdateOperateAccessCardService(m_LoginID, stuInArray, out
stuOutErrArray, 3000);
if (!result)

```

```

{
    for (int i = 0; i < stuOutErrArray.Length; i++)
    {
        MessageBox.Show(stuOutErrArray[i].emCode.ToString());
    }
}

//Delete
NET_EM_FAILCODE[] stuOutErrArray = new NET_EM_FAILCODE[1];
string[] InCardid = new string[] { szCardNo };// szCardNo: szCardNo of users to be deleted.
bool result = NETClient.RemoveOperateAccessCardService(m_LoginID, InCardid, out
stuOutErrArray, 3000);
if (!result)
{
    for (int i = 0; i < stuOutErrArray.Length; i++)
    {
        MessageBox.Show(stuOutErrArray[i].emCode.ToString());
    }
}
}

```

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).

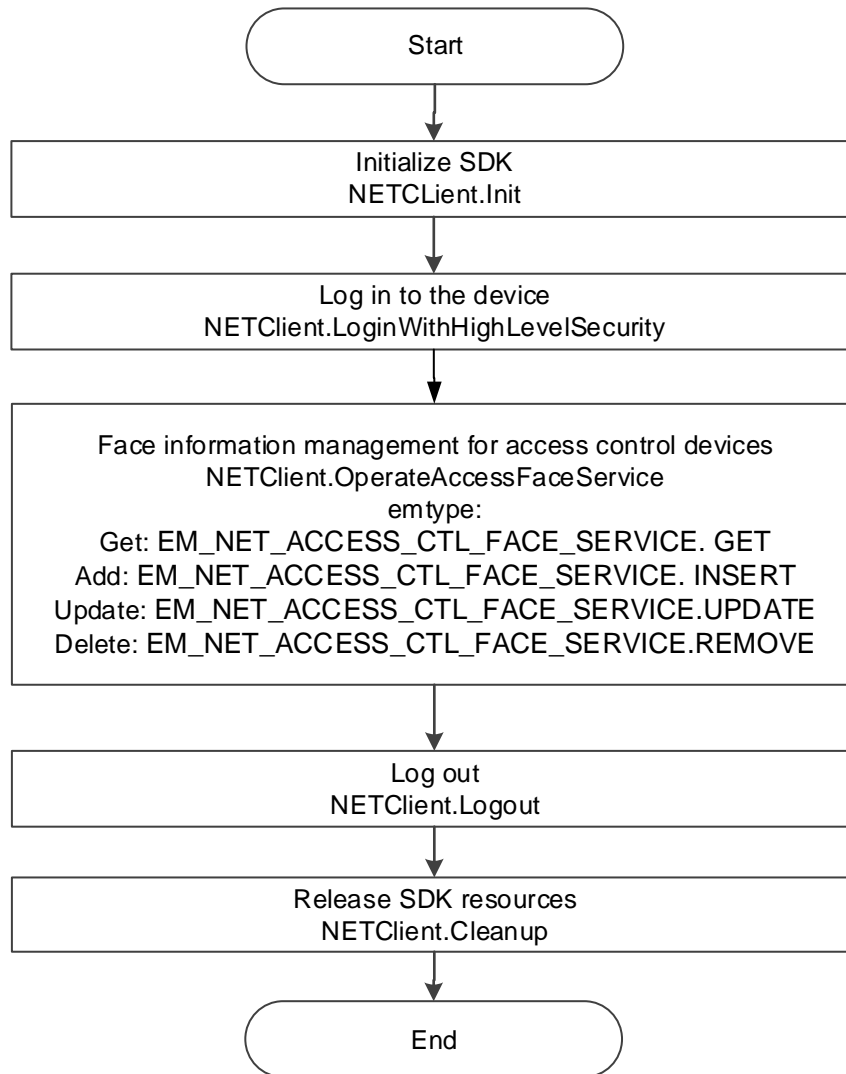
2.3.7.3.2 Interface Overview

Table 2-50 Description of face information interface

Interface	Description
NETClient.OperateAccessFaceService	Face information management interface for access control devices

2.3.7.3.3 Process Description

Figure 2-40 Management of face information



Process

Step 1 Call **NETClient.Init** to initialize SDK.

Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.

Step 3 Call **NETClient.OperateAccessFaceService** to add, get, update, and delete the face information.

Table 2-51 EM_NET_ACCESS_CTL_FACE_SERVICE parameter discription

Parameter	Definition
INSERT	Add
GET	Get
UPDATE	Update
REMOVE	Delete

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

Step 5 After using all SDK functions, call **NETClient.Cleanup** to release SDK resources.

2.3.7.3.4 Sample Code

```
//Get
```

```

NET_IN_ACCESS_FACE_SERVICE_GET stuFaceGetIn = new
NET_IN_ACCESS_FACE_SERVICE_GET();

stuFaceGetIn.dwSize =
(uint)Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_GET));
stuFaceGetIn.nUserNum = 1;
stuFaceGetIn.szUserID = new NET_IN_ACCESS_FACE_SERVICE_UserID[100];
stuFaceGetIn.szUserID[0] = new NET_IN_ACCESS_FACE_SERVICE_UserID() { userID =
m_UserInfo.szUserID };//m_UserInfo.szUserID;

NET_OUT_ACCESS_FACE_SERVICE_GET stuFaceGetOut = new
NET_OUT_ACCESS_FACE_SERVICE_GET();

stuFaceGetOut.dwSize =
(uint)Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_GET));
stuFaceGetOut.nMaxRetNum = 1;
stuFaceGetOut.pFaceInfo = IntPtr.Zero;
stuFaceGetOut.pFaceInfo =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_ACCESS_FACE_INFO)));
stuFaceGetOut.pFailCode = IntPtr.Zero;
stuFaceGetOut.pFailCode =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_EM_FAILCODE)));

NET_ACCESS_FACE_INFO stuFaceInfo = new NET_ACCESS_FACE_INFO();
stuFaceInfo.nInFacePhotoLen = new int[5];
stuFaceInfo.pFacePhoto = new IntPtr[5];
for (int i = 0; i < 5; i++)
{
    stuFaceInfo.nInFacePhotoLen[i] = 100 * 1024;
    IntPtr tempPtr = IntPtr.Zero;
    tempPtr = Marshal.AllocHGlobal(100 * 1024);
    for (int j = 0; j < 100 * 1024; j++)
    {
        Marshal.WriteByte(tempPtr, j, 0);
    }
    stuFaceInfo.pFacePhoto[i] = tempPtr;
}
Marshal.StructureToPtr(stuFaceInfo, stuFaceGetOut.pFaceInfo, true);

NET_EM_FAILCODE stuFailCode = new NET_EM_FAILCODE();
Marshal.StructureToPtr(stuFailCode, stuFaceGetOut.pFailCode, true);

IntPtr pstInParam = IntPtr.Zero;

```



```

pstInParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_GET)));
Marshal.StructureToPtr(stuFaceGetIn, pstInParam, true);

IntPtr pstOutParam = IntPtr.Zero;
pstOutParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_GET)));
Marshal.StructureToPtr(stuFaceGetOut, pstOutParam, true);

bool result = NETClient.OperateAccessFaceService(m_LoginID,
EM_NET_ACCESS_CTL_FACE_SERVICE.GET, pstInParam, pstOutParam, 5000);
var get_face_service =
(NET_OUT_ACCESS_FACE_SERVICE_GET)Marshal.PtrToStructure(pstOutParam,
typeof(NET_OUT_ACCESS_FACE_SERVICE_GET));
if (!result)
{
    stuFailCode = (NET_EM_FAILCODE)Marshal.PtrToStructure(get_face_service.pFailCode,
typeof(NET_EM_FAILCODE));

    if (stuFailCode.emCode == EM_FAILCODE.NOERROR)
    {
        MessageBox.Show(NETClient.GetLastError());
    }
    else if (stuFailCode.emCode != EM_FAILCODE.UNKNOWN)
    {
        MessageBox.Show(stuFailCode.emCode.ToString());
    }
}
else
{
    stuFaceInfo =
(NET_ACCESS_FACE_INFO)Marshal.PtrToStructure(get_face_service.pFaceInfo,
typeof(NET_ACCESS_FACE_INFO));
    if (stuFaceInfo.nFacePhoto > 0)
    {
        m_ImageData = new byte[stuFaceInfo.nOutFacePhotoLen[0]];
        Marshal.Copy(stuFaceInfo.pFacePhoto[0], m_ImageData, 0,
stuFaceInfo.nOutFacePhotoLen[0]);
        using (MemoryStream stream = new MemoryStream(m_ImageData))
        {
            Image image = Image.FromStream(stream);

```

```

        pictureBox_face.Image = image;
        pictureBox_face.Refresh();
    }
}
}
//Add
NET_IN_ACCESS_FACE_SERVICE_INSERT stuFaceInsertIn = new
NET_IN_ACCESS_FACE_SERVICE_INSERT();
stuFaceInsertIn.dwSize =
(uint)Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_INSERT));
stuFaceInsertIn.nFaceInfoNum = 1;
stuFaceInsertIn.pFaceInfo = IntPtr.Zero;
stuFaceInsertIn.pFaceInfo =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_ACCESS_FACE_INFO)));

NET_ACCESS_FACE_INFO stuFaceInfo = new NET_ACCESS_FACE_INFO();
stuFaceInfo.szUserID = m_UserInfo.szUserID;
stuFaceInfo.nFacePhoto = 1;
stuFaceInfo.nInFacePhotoLen = new int[5];
stuFaceInfo.nOutFacePhotoLen = new int[5];
stuFaceInfo.nInFacePhotoLen[0] = stuFaceInfo.nOutFacePhotoLen[0] = m_ImageData.Length;
stuFaceInfo.pFacePhoto = new IntPtr[5];
stuFaceInfo.pFacePhoto[0] = Marshal.AllocHGlobal(stuFaceInfo.nInFacePhotoLen[0]);
Marshal.Copy(m_ImageData, 0, stuFaceInfo.pFacePhoto[0],
stuFaceInfo.nInFacePhotoLen[0]);

Marshal.StructureToPtr(stuFaceInfo, stuFaceInsertIn.pFaceInfo, true);

NET_OUT_ACCESS_FACE_SERVICE_INSERT stuFaceInsertOut = new
NET_OUT_ACCESS_FACE_SERVICE_INSERT();
stuFaceInsertOut.dwSize =
(uint)Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_INSERT));
stuFaceInsertOut.nMaxRetNum = 1;
stuFaceInsertOut.pFailCode = IntPtr.Zero;
stuFaceInsertOut.pFailCode =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_EM_FAILCODE)));

NET_EM_FAILCODE stuFailCodeR = new NET_EM_FAILCODE();
Marshal.StructureToPtr(stuFailCodeR, stuFaceInsertOut.pFailCode, true);

IntPtr pstInParam = IntPtr.Zero;

```

```

pstInParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_INSERT)))
;
Marshal.StructureToPtr(stuFaceInsertIn, pstInParam, true);

IntPtr pstOutParam = IntPtr.Zero;
pstOutParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_INSERT
)));
Marshal.StructureToPtr(stuFaceInsertOut, pstOutParam, true);

bool result = NETClient.OperateAccessFaceService(m_LoginID,
EM_NET_ACCESS_CTL_FACE_SERVICE.INSERT, pstInParam, pstOutParam, 5000);
var faceinfo =
(NET_OUT_ACCESS_FACE_SERVICE_INSERT)Marshal.PtrToStructure(pstOutParam,
typeof(NET_OUT_ACCESS_FACE_SERVICE_INSERT));
if (!result)
{
    var failcode = (NET_EM_FAILCODE)Marshal.PtrToStructure(faceinfo.pFailCode,
typeof(NET_EM_FAILCODE));
    if (failcode.emCode == EM_FAILCODE.NOERROR)
    {
        MessageBox.Show(NETClient.GetLastError());
    }
    else
    {
        MessageBox.Show(failcode.emCode.ToString());
    }
}

//Update
NET_IN_ACCESS_FACE_SERVICE_UPDATE stuFaceUpdateIn = new
NET_IN_ACCESS_FACE_SERVICE_UPDATE();
stuFaceUpdateIn.dwSize =
(uint)Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_UPDATE));
stuFaceUpdateIn.nFaceInfoNum = 1;
stuFaceUpdateIn.pFaceInfo = IntPtr.Zero;
stuFaceUpdateIn.pFaceInfo =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_ACCESS_FACE_INFO)));

NET_ACCESS_FACE_INFO stuFaceInfo = new NET_ACCESS_FACE_INFO();

```

```

stuFaceInfo.szUserID = m_UserInfo.szUserID;
stuFaceInfo.nFacePhoto = 1;
stuFaceInfo.nInFacePhotoLen = new int[5];
stuFaceInfo.nOutFacePhotoLen = new int[5];
stuFaceInfo.nInFacePhotoLen[0] = stuFaceInfo.nOutFacePhotoLen[0] = m_ImageData.Length;
stuFaceInfo.pFacePhoto = new IntPtr[5];
stuFaceInfo.pFacePhoto[0] = Marshal.AllocHGlobal(stuFaceInfo.nInFacePhotoLen[0]);
Marshal.Copy(m_ImageData, 0, stuFaceInfo.pFacePhoto[0],
stuFaceInfo.nInFacePhotoLen[0]);

Marshal.StructureToPtr(stuFaceInfo, stuFaceUpdateIn.pFaceInfo, true);

NET_OUT_ACCESS_FACE_SERVICE_UPDATE stuFaceUpdateOut = new
NET_OUT_ACCESS_FACE_SERVICE_UPDATE(); //{ sizeof(stuFaceUpdateOut) };
stuFaceUpdateOut.dwSize =
(uint)Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_UPDATE));
stuFaceUpdateOut.nMaxRetNum = 1;
stuFaceUpdateOut.pFailCode = IntPtr.Zero;
stuFaceUpdateOut.pFailCode =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_EM_FAILCODE)));

NET_EM_FAILCODE stuFailCodeR = new NET_EM_FAILCODE();
Marshal.StructureToPtr(stuFailCodeR, stuFaceUpdateOut.pFailCode, true);

IntPtr pstInParam = IntPtr.Zero;
pstInParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_UPDATE))
);
Marshal.StructureToPtr(stuFaceUpdateIn, pstInParam, true);

IntPtr pstOutParam = IntPtr.Zero;
pstOutParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_UPDAT
E)));
Marshal.StructureToPtr(stuFaceUpdateOut, pstOutParam, true);

bool result = NETClient.OperateAccessFaceService(m_LoginID,
EM_NET_ACCESS_CTL_FACE_SERVICE.UPDATE, pstInParam, pstOutParam, 5000);
var faceinfo =
(NET_OUT_ACCESS_FACE_SERVICE_UPDATE)Marshal.PtrToStructure(pstOutParam,
typeof(NET_OUT_ACCESS_FACE_SERVICE_UPDATE));

```

```

if (!result)
{
    var failcode = (NET_EM_FAILCODE)Marshal.PtrToStructure(faceinfo.pFailCode,
    typeof(NET_EM_FAILCODE));
    if (failcode.emCode == EM_FAILCODE.NOERROR)
    {
        MessageBox.Show(NETClient.GetLastError());
    }
    else
    {
        MessageBox.Show(failcode.emCode.ToString());
    }
}

//Delete
NET_IN_ACCESS_FACE_SERVICE_REMOVE stuFaceRemoveIn = new
NET_IN_ACCESS_FACE_SERVICE_REMOVE();
stuFaceRemoveIn.dwSize =
(uint)Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_REMOVE));
stuFaceRemoveIn.nUserNum = 1;
stuFaceRemoveIn.szUserID = new NET_IN_ACCESS_FACE_SERVICE_UserID[100];
stuFaceRemoveIn.szUserID[0] = new NET_IN_ACCESS_FACE_SERVICE_UserID() { userID
= m_UserInfo.szUserID };

NET_OUT_ACCESS_FACE_SERVICE_REMOVE stuFaceRemoveOut = new
NET_OUT_ACCESS_FACE_SERVICE_REMOVE(); //{ sizeof(stuFaceROut) };
stuFaceRemoveOut.dwSize =
(uint)Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_REMOVE));
stuFaceRemoveOut.nMaxRetNum = 1;
stuFaceRemoveOut.pFailCode = IntPtr.Zero;
stuFaceRemoveOut.pFailCode =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_EM_FAILCODE)));

NET_EM_FAILCODE stuFailCodeR = new NET_EM_FAILCODE();
Marshal.StructureToPtr(stuFailCodeR, stuFaceRemoveOut.pFailCode, true);

IntPtr pstInParam = IntPtr.Zero;
pstInParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_IN_ACCESS_FACE_SERVICE_REMOVE
)));

```

```

Marshal.StructureToPtr(stuFaceRemoveIn, pstInParam, true);

IntPtr pstOutParam = IntPtr.Zero;
pstOutParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_OUT_ACCESS_FACE_SERVICE_REMOVE)));
Marshal.StructureToPtr(stuFaceRemoveOut, pstOutParam, true);

bool result = NETClient.OperateAccessFaceService(m_LoginID,
EM_NET_ACCESS_CTL_FACE_SERVICE.REMOVE, pstInParam, pstOutParam, 5000);
var faceinfo =
(NET_OUT_ACCESS_FACE_SERVICE_REMOVE)Marshal.PtrToStructure(pstOutParam,
typeof(NET_OUT_ACCESS_FACE_SERVICE_REMOVE));

if (!result)
{
    var failcode = (NET_EM_FAILCODE)Marshal.PtrToStructure(faceinfo.pFailCode,
typeof(NET_EM_FAILCODE));
    if (failcode.emCode == EM_FAILCODE.NOERROR)
    {
        MessageBox.Show(NETClient.GetLastError());
    }
    else
    {
        MessageBox.Show(failcode.emCode.ToString());
    }
}

```

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).

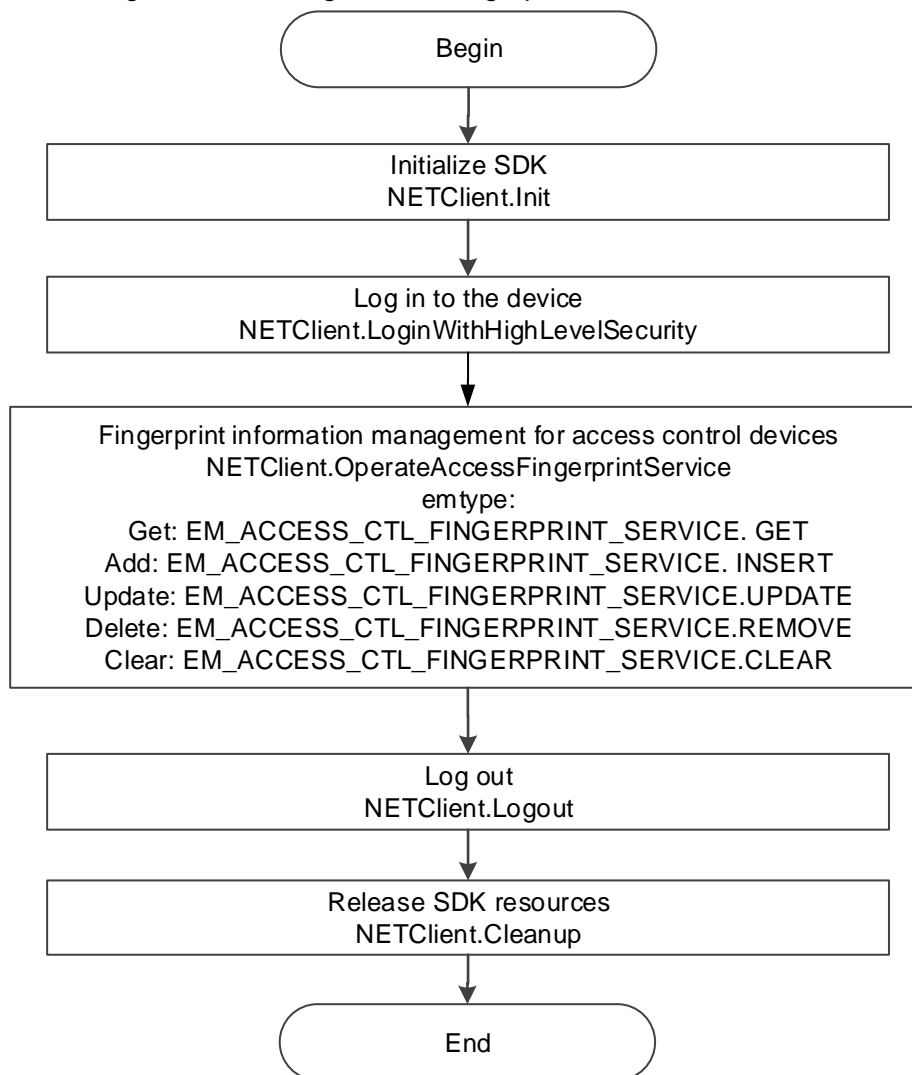
2.3.7.4.2 Interface Overview

Table 2-52 Description of fingerprint information interface

Interface	Description
NETClient.OperateAccessFingerprintService	Fingerprint information management interface.

2.3.7.4.3 Process Description

Figure 2-41 Management of fingerprint information



Process

Step 1 Call **NETClient.Init** to initialize SDK.

Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.

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

Table 2-53 Operations and structural body of type

emtype	Definition	Param
INSERT	Add fingerprint information	NET_IN_ACCESS_FINGERPRINT_SERVICE_INSERT NET_OUT_ACCESS_FINGERPRINT_SERVICE_INSERT
GET	Get fingerprint information	NET_IN_ACCESS_FINGERPRINT_SERVICE_GET NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET
UPDATE	Update fingerprint information	NET_IN_ACCESS_FINGERPRINT_SERVICE_UPDATE NET_OUT_ACCESS_FINGERPRINT_SERVICE_UPDATE
REMOVE	Delete fingerprint information	NET_IN_ACCESS_FINGERPRINT_SERVICE_REMOVE NET_OUT_ACCESS_FINGERPRINT_SERVICE_REMOVE
CLEAR	Clear fingerprint	NET_IN_ACCESS_FINGERPRINT_SERVICE_CLEAR

emtype	Definition	Param
	information	NET_OUT_ACCESS_FINGERPRINT_SERVICE_CLEAR

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

Step 5 After using all SDK functions, call **NETClient. Cleanup** to release SDK resources.

2.3.7.4.4 Sample Code

```
private NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET m_FingerprintInfo = new
NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET();

//Get
NET_IN_ACCESS_FINGERPRINT_SERVICE_GET stuFingerPrintGetIn = new
NET_IN_ACCESS_FINGERPRINT_SERVICE_GET();
stuFingerPrintGetIn.dwSize =
(uint)Marshal.SizeOf(typeof(NET_IN_ACCESS_FINGERPRINT_SERVICE_GET));
stuFingerPrintGetIn.szUserID = m_UserInfo.szUserID;

NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET stuFingerPrintGetOut = new
NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET();
stuFingerPrintGetOut.dwSize =
(uint)Marshal.SizeOf(typeof(NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET));
stuFingerPrintGetOut.nMaxFingerDataLength = 10000;
stuFingerPrintGetOut.pbyFingerData = IntPtr.Zero;
stuFingerPrintGetOut.pbyFingerData = Marshal.AllocHGlobal(10000);

IntPtr pstInParam = IntPtr.Zero;
pstInParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_IN_ACCESS_FINGERPRINT_SERVICE_
GET)));
Marshal.StructureToPtr(stuFingerPrintGetIn, pstInParam, true);

IntPtr pstOutParam = IntPtr.Zero;
pstOutParam =
Marshal.AllocHGlobal(Marshal.SizeOf(typeof(NET_OUT_ACCESS_FINGERPRINT_SERVICE_
_GET)));
Marshal.StructureToPtr(stuFingerPrintGetOut, pstOutParam, true);

bool result = NETClient.OperateAccessFingerprintService(m_LoginID,
EM_ACCESS_CTL_FINGERPRINT_SERVICE.GET, pstInParam, pstOutParam, 5000);
m_FingerprintInfo =
(NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET)Marshal.PtrToStructure(pstOutParam,
typeof(NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET));

//Add
```



```

m_FingerprintInfo.nPacketNum = 1;
m_FingerprintInfo.nPacketLen = m_PacketLen;
m_FingerprintInfo.szFingerPrintInfo = Marshal.AllocHGlobal(m_PacketLen);
for (int i = 0; i < m_PacketLen; i++)
{
    Marshal.WriteByte(m_FingerprintInfo.szFingerPrintInfo, i, FingerPrintInfo[i]);
}
m_FingerprintInfo.nDuressIndex = 0;
if (ckb_Duress.Checked)
{
    m_FingerprintInfo.nDuressIndex = 1;
}

NET_ACCESS_FINGERPRINT_INFO[] stuInArray = new
NET_ACCESS_FINGERPRINT_INFO[1] { m_FingerprintInfo };
NET_EM_FAILCODE[] stuOutArray;

bRet = NETClient.InsertOperateAccessFingerprintService(m_LoginID, stuInArray, out
stuOutArray, 3000);
if (!bRet)
{
    for (int i = 0; i < stuOutArray.Length; i++)
    {
        MessageBox.Show(stuOutArray[i].emCode.ToString());
    }
}

//Update
for (int i = 0; i < m_PacketLen; i++)
{
    Marshal.WriteByte(m_FingerprintInfo.szFingerPrintInfo, (m_FingerprintNum - 1) *
m_PacketLen + i, FingerPrintInfo[i]);
}
if (ckb_Duress.Checked)
{
    m_FingerprintInfo.nDuressIndex = m_FingerprintNum;
}

NET_ACCESS_FINGERPRINT_INFO[] stuInArray = new
NET_ACCESS_FINGERPRINT_INFO[1] { m_FingerprintInfo };

```

```

NET_EM_FAILCODE[] stuOutArray;

bRet = NETClient.UpdateOperateAccessFingerprintService(m_LoginID, stuInArray, out
stuOutArray, 3000);
if (!bRet)
{
    for (int i = 0; i < stuOutArray.Length; i++)
    {
        MessageBox.Show(stuOutArray[i].emCode.ToString());
    }
}
//Delete
NET_EM_FAILCODE[] stuOutErrArray;
string[] userid = new string[] { m_UserInfo.szUserID };
result = NETClient.RemoveOperateAccessFingerprintService(m_LoginID, userid, out
stuOutErrArray, 3000);
if (!result)
{
    for (int i = 0; i < stuOutErrArray.Length; i++)
    {
        MessageBox.Show(stuOutErrArray[i].emCode.ToString());
    }
}

```

2.3.8 Door Config

See "2.2.8 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.

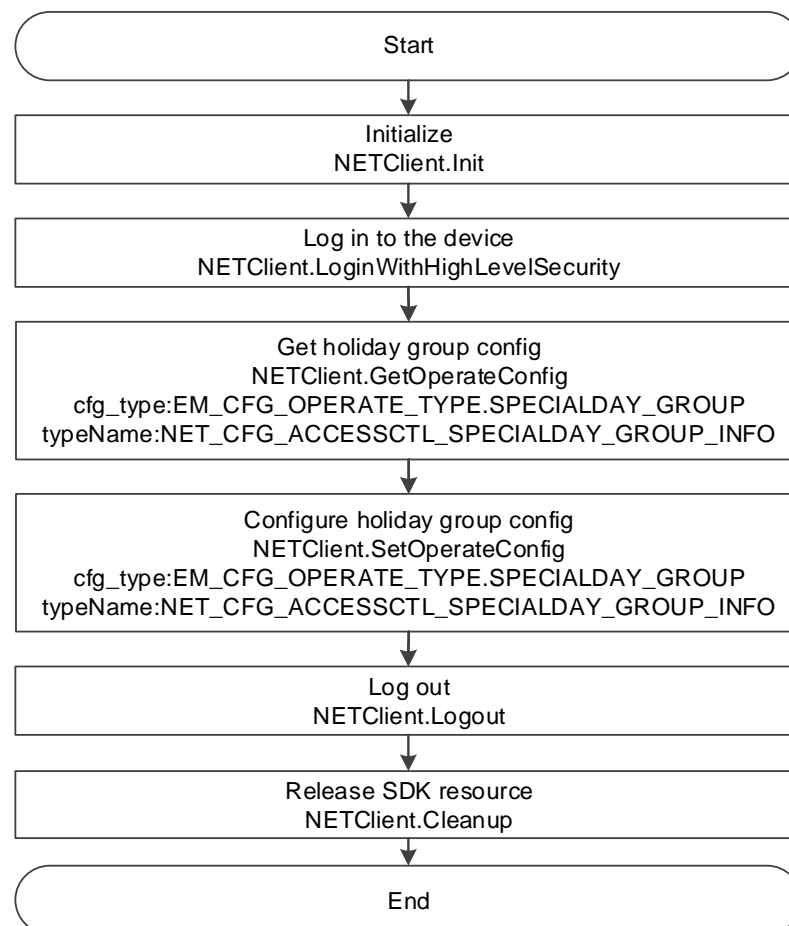
2.3.9.3.2 Interface Overview

Table 2-54 Description of holiday group interface

Interface	Description
NETClient.GetOperateConfig	Query config information.
NETClient.SetOperateConfig	Set config information.

2.3.9.3.3 Process Description

Figure 2-42 Holiday group



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.GetOperateConfig** to query the holiday group config info for the access control device.

Table 2-55 Description of cfg_type

emCfgOpType	Description	Structural body
EM_CFG_OPERATE_TYPE.SPECIALDAY_GROUP	Get information of holiday group	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO

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

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

Step 6 After using all SDK functions, call **NETClient. Cleanup** to release SDK resources.

2.3.9.3.4 Sample Code

```
//Get
NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO stuln = new
NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO();
stuln.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO));
object obj = stuln;
bool ret = NETClient.GetOperateConfig(m_LoginID,
EM_CFG_OPERATE_TYPE.SPECIALDAY_GROUP, cmb_Index.SelectedIndex, ref obj,
typeof(NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO), 5000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
}
m_SpecialdayGroupInfo = (NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO)obj;

//Set
m_SpecialdayGroupInfo.bGroupEnable = chb_Enable.Checked;
m_SpecialdayGroupInfo.szGroupName = txt_Name.Text;
object obj = m_SpecialdayGroupInfo;
bool ret = NETClient.SetOperateConfig(m_LoginID,
EM_CFG_OPERATE_TYPE.SPECIALDAY_GROUP, cmb_Index.SelectedIndex, obj,
typeof(NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO), 5000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
}
```

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.

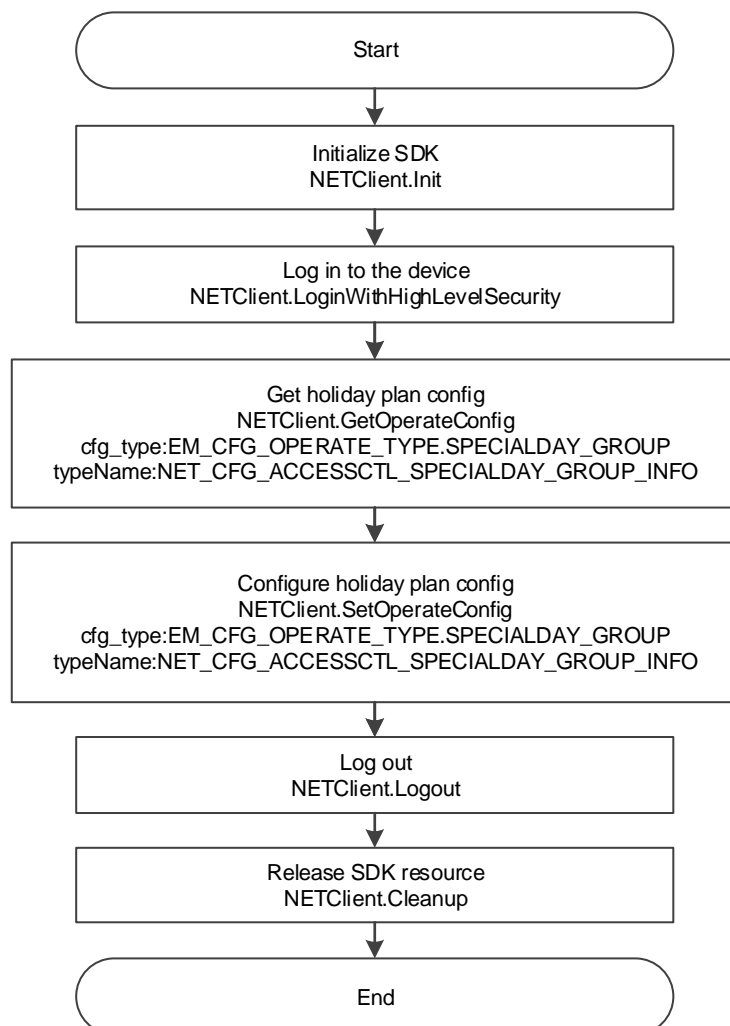
2.3.9.4.2 Interface Overview

Table 2-56 Description of holiday plan interface

Interface	Description
NETClient.GetOperateConfig	Query config information.
NETClient.SetOperateConfig	Set config information.

2.3.9.4.3 Process Description

Figure 2-43 Holiday plan



Process

Step 1 Call **NETClient.Init** to initialize SDK.

Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.

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

Table 2-57 Description of Cfg_Type

emCfgOpType	Description	Structural body
EM_CFG_OPERATE_TYPE.SPECIALDAYS_SCHEDULE	Get holiday information	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO

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

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

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

2.3.9.4.4 Sample Code

```
// Get
NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO stuln = new
NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO();
stuln.dwSize =
(uint)Marshal.SizeOf(typeof(NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO));
object obj = stuln;
bool ret = NETClient.GetOperateConfig(m_LoginID,
EM_CFG_OPERATE_TYPE.SPECIALDAYS_SCHEDULE,
cmb_ScheduleGroup.SelectedIndex, ref obj,
typeof(NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO), 5000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
}
m_ScheduleInfo = (NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO)obj;

//Set
m_ScheduleInfo.szSchdName = txt_ScheduleName.Text;
m_ScheduleInfo.bSchd = chb_ScheduleEnable.Checked;
m_ScheduleInfo.nGroupNo = int.Parse(txt_GroupNum.Text);

object obj = m_ScheduleInfo;
bool ret = NETClient.SetOperateConfig(m_LoginID,
EM_CFG_OPERATE_TYPE.SPECIALDAYS_SCHEDULE,
cmb_ScheduleGroup.SelectedIndex, obj,
typeof(NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO), 5000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
}
```

2.3.10 Advanced Config of Door

See "2.2.10 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."

2.3.11.3 Alarm Records

2.3.11.3.1 Introduction

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

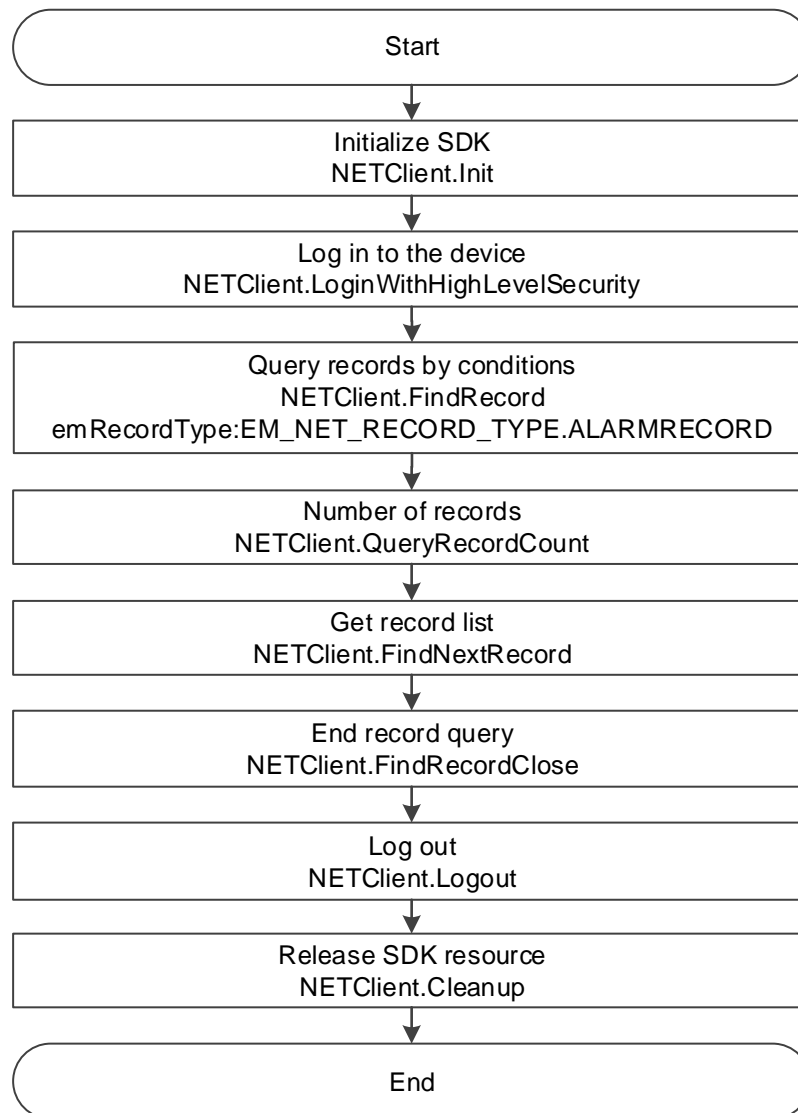
2.3.11.3.2 Interface Overview

Table 2-58 Description of record query interfaces

Interface	Description
NETClient.QueryRecordCount	Find the number of records.
NETClient.FindRecord	Query records by query conditions.
NETClient.FindNextRecord	Query records: nFilecount: the number of files to be queried. nRetRecordNum in output parameters is the number of queried records. When the number of queried records is less than nFilecount, the query will be stopped.
NETClient.FindRecordClose	End record query

2.3.11.3.3 Process Description

Figure 2-44 Record query



Process

- Step 1 Call **NETClient.Init** to initialize SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.FindRecord** to get the query handle.
Assign **NET_RECORD_ACCESS_ALARMRECORD** to **emRecordType**.
- Step 4 Call **NETClient.QueryRecordCount** to find the count of records.
- Step 5 Call **NETClient.FindNextRecord** to get the list of records.
- Step 6 Call **NETClient. FindRecordClose** to close the query handle.
- Step 7 After completing this process, call **NETClient. Logout** to log out of the device.
- Step 8 After using all SDK functions, call **NETClient. Cleanup** to release SDK resource.

2.3.11.3.4 Sample Code

```
//Start query
```

```

NET_FIND_NET_RECORD_ACCESS_ALARMRECORD_INFO_CONDITION condition = new
NET_FIND_NET_RECORD_ACCESS_ALARMRECORD_INFO_CONDITION();
condition.dwSize =
(uint)Marshal.SizeOf(typeof(NET_FIND_NET_RECORD_ACCESS_ALARMRECORD_INFO_
CONDITION));
condition.stStartTime = NET_TIME.FromDateTime(dateTimePicker_DoorStart.Value);
condition.stEndTime = NET_TIME.FromDateTime(dateTimePicker_DoorEnd.Value);
object obj = condition;

bool ret = NETClient.FindRecord(m_LoginID,
EM_NET_RECORD_TYPE.ACCESS_ALARMRECORD, obj,
typeof(NET_FIND_NET_RECORD_ACCESS_ALARMRECORD_INFO_CONDITION), ref
m_FindAlarmRecordID, 10000);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}

//number of records
int nCount = 0;
if (NETClient.QueryRecordCount(m_FindAlarmRecordID, ref nCount, 3000))
{
    txt_AlarmRecordCount.Text = nCount.ToString();
}
else
{
    MessageBox.Show(NETClient.GetLastError());
}

//query records
int max = 20;
int retNum = 0;
List<object> ls = new List<object>();
for (int i = 0; i < max; i++)
{
    NET_RECORD_ACCESS_ALARMRECORD_INFO alarm_rec = new
NET_RECORD_ACCESS_ALARMRECORD_INFO();
    alarm_rec.dwSize =
(uint)Marshal.SizeOf(typeof(NET_RECORD_ACCESS_ALARMRECORD_INFO));
    ls.Add(alarm_rec);
}

```

```
NETClient.FindNextRecord(m_FindAlarmRecordID, max, ref retNum, ref ls,  
typeof(NET_RECORD_ACCESS_ALARMRECORD_INFO), 10000);  
foreach (var item in ls)  
{  
    NET_RECORD_ACCESS_ALARMRECORD_INFO info =  
    (NET_RECORD_ACCESS_ALARMRECORD_INFO)item;  
}  
//End record query  
NETClient.FindRecordClose(m_FindAlarmRecordID);  
m_FindAlarmRecordID = IntPtr.Zero;
```

3 Interface Function

3.1 Common Interface

3.1.1 SDK Initialization

3.1.1.1 SDK Initialization

Table 3-1 NetSDK initialization description

Item	Description	
Description	Initialize the NetSDK.	
Function	bool Init(fDisconnectCallBack cbDisconnect, IntPtr dwUser, NETSDK_INIT_PARAM? stulnitParam);	
Parameter	[in]cbDisconnect	Disconnection callback.
	[in]dwUser	User parameters for disconnection callback.
	[in]stulnitParam	Initialize NetSDK parameters
Return Value	<ul style="list-style-type: none">• Success: TRUE• Failure: FALSE	
Note	<ul style="list-style-type: none">• 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.• dwUser parameter introduced by Init will be returned by dwUser in the callback function cbDisconnect.	

3.1.1.2 SDK Cleaning up

Table 3-2 Description of SDK cleaning up

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

3.1.1.3 Configuring Reconnection Callback SetAutoReconnect

Table 3-3 Description of setting reconnection callback

Item	Description
Description	Configure auto reconnection callback.

Item	Description	
Function	void SetAutoReconnect(fHaveReConnectCallBack cbAutoConnect, IntPtr dwUser);	
Parameter	[in]cbAutoConnect	Reconnection callback.
	[in]dwUser	User parameters for reconnection callback.
Return Value	None.	
Note	Configure reconnection callback interface. If the callback is set as NULL, the device will not be reconnected automatically.	

3.1.1.4 Configuring Network Parameter SetNetworkParam

Table 3-4 Description of device network parameter

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

3.1.2 Device Initialization

3.1.2.1 Querying Device StartSearchDevices

Table 3-5 Description of Querying device

Item	Description	
Description	Query device information.	
Function	IntPtr StartSearchDevice(fSearchDevicesCB cbSearchDevice, IntPtr pUserData, IntPtr szLocallp)	
Parameter	[in] cbSearchDevice	Input parameter of async Querying.
	[in] pUserData	User info
	[in] szLocallp	Single NIC: szLocallp is optional Multiple NICs: szLocallp enter hostIP
Return Value	Failure: 0; success: non 0.	
Note	Multi-thread calling is not supported.	

3.1.2.2 Device Initialization

Table 3-6 Description of device initialization

Item	Description	
Description	Initialize Device.	
Function	<pre>bool InitDevAccount(NET_IN_INIT_DEVICE_ACCOUNT pInitAccountIn, ref NET_OUT_INIT_DEVICE_ACCOUNT pInitAccountOut, uint dwWaitTime, string szLocalIp)</pre>	
Parameter	[in]pInitAccountIn	Input parameter, corresponding to NET_IN_INIT_DEVICE_ACCOUNT structure.
	[out]pInitAccountOut	Output parameter, corresponding to NET_OUT_INIT_DEVICE_ACCOUNT structure.
	[in]dwWaitTime	Timeout period.
	[in]szLocalIp	<ul style="list-style-type: none"> In the case of single network adapter, szLocalIp can be left empty. In the case of multiple network adapters, fill the host IP in szLocalIp.
Return Value	<ul style="list-style-type: none"> Success: TRUE Failure: FALSE 	
Note	None.	

3.1.2.3 Stopping Querying Device

Table 3-7 Description of stopping Querying device

Item	Description	
Description	Stop Querying device information.	
Function	<pre>bool StopSearchDevice(IntPtr ISearchHandle)</pre>	
Parameter	[in] ISearchHandle	Input parameter, Query handle.
Return Value	<ul style="list-style-type: none"> Success: TRUE Failure: FALSE 	
Note	Multi-thread calling is not supported.	

3.1.3 Device Login

3.1.3.1 Logging in to the Device LoginWithHighLevelSecurity

Table 3-8 Description of user logging in to the device

Item	Description
Description	Log in to the device.

Item	Description	
Function	IntPtr LoginWithHighLevelSecurity(string pchDVRIP, ushort wDVRPort, string pchUserName, string pchPassword, EM_LOGIN_SPAC_CAP_TYPE emSpecCap, IntPtr pCapParam, ref NET_DEVICEINFO_Ex deviceInfo);	
Parameter	[in]pchDVRIP	Device IP
	[in]wDVRPort	Device port
	[in]pchUserName	Username
	[in]pchPassword	Password
	[in]emSpecCap	Login type
	[in]pCapParam	Login type parameter
	[out]deviceInfo	Device info
Return Value	<ul style="list-style-type: none"> • Success: Non-0 • Failure: 0 	
Note	High security level login interface. You can still use LoginEx2, but there is a security risk. Therefore, it is highly recommended to use the latest interface LoginWithHighLevelSecurity to log into the device.	

3.1.3.2 User Logging Out of the Device Logout

Table 3-9 Description of user logging out of the device

Item	Description	
Description	Log out of the device.	
Function	bool Logout(IntPtr ILoginID);	
Parameter	[in]ILoginID	Return value of LoginWithHighLevelSecurity.
Return Value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Description	None.	

3.1.4 Realtime Monitor

3.1.4.1 Opening the Monitoring RealPlay

Table 3-10 Description of opening the monitoring

Item	Description
Description	Open the real-time monitoring.

Item	Description	
Function	<pre>IntPtr RealPlay(IntPtr ILoginID, int nChannelID, IntPtr hWnd, EM_RealPlayType rType = EM_RealPlayType.Realplay);</pre>	
Parameter	[in]ILoginID	Return value of LoginWithHighLevelSecurity.
	[in]nChannelID	Video channel number, an integer increasing from 0.
	[in]hWnd	Window handle, only valid in Windows system.
	[in]rType	Live view type.
Return Value	<ul style="list-style-type: none"> • Success: Non-0 • Failure: 0 	
Note	<p>In Windows environment:</p> <ul style="list-style-type: none"> • When hWnd is valid, the picture is displayed in the corresponding window. • When hWnd is NULL, the way of getting stream is to get video data by setting callback function, and then submit the data to users for processing. 	

Table 3-11 Description of live view types

Live view type	Meanings
Realplay	Live View
Multiplay	Multi-picture preview
Realplay_0	Real-time monitoring—main stream, equivalent to DH_RType_Realplay
Realplay_1	Real-time monitoring—sub stream 1
Realplay_2	Real-time monitoring—sub stream 2
Realplay_3	Real-time monitoring—sub stream 3
Multiplay_1	Multi-picture preview—1 picture
Multiplay_4	Multi-picture preview—4 pictures
Multiplay_8	Multi-picture preview—8 pictures
Multiplay_9	Multi-picture preview—9 pictures
Multiplay_16	Multi-picture preview—16 pictures
Multiplay_6	Multi-picture preview—6 pictures
Multiplay_12	Multi-picture preview—12 pictures
Multiplay_25	Multi-picture preview—25 pictures
Multiplay_36	Multi-picture preview—36 pictures

3.1.4.2 Closing the Monitoring StopRealPlay

Table 3-12 Description of closing the monitoring

Item	Description
Description	Close the real-time monitoring.

Item	Description	
Function	bool StopRealPlay(IntPtr IRealHandle);	
Parameter	[in]IRealHandle	Return value of RealPlay.
Return Value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.1.4.3 Saving the Monitoring Data SaveRealData

Table 3-13 Description of saving the monitoring data

Item	Description	
Description	Save the real-time monitoring data as a file.	
Function	bool SaveRealData(IntPtr IRealHandle, string pchFileName);	
Parameter	[in]IRealHandle	Return value of RealPlay.
	[in]pchFileName	Path of the file to be saved.
Return Value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.1.4.4 Stopping Saving the Monitoring Data StopSaveRealData

Table 3-14 Description of stopping saving the monitoring data

Item	Description	
Description	Stop saving the real-time monitoring data as a file.	
Function	bool StopSaveRealData(IntPtr IRealHandle);	
Parameter	[in]IRealHandle	Return value of RealPlay.
Return Value	Success: TRUE;Failure: FALSE	
Note	None.	

3.1.4.5 Setting Monitoring Data Callback SetRealDataCallback

Table 3-15 Description of setting monitoring data callback

Item	Description
Description	Set real-time monitoring data callback.

Item	Description	
Function	<pre>bool SetRealDataCallBack(IntPtr IRealHandle, fRealDataCallBackEx2 cbRealData, IntPtr dwUser, EM_REALDATA_FLAG dwFlag);</pre>	
Parameter	[in]IRealHandle	Return value of RealPlay.
	[in]cbRealData	Callback function for monitoring data flow.
	[in]dwUser	Parameters of the callback function for monitoring data flow.
	[in]dwFlag	Type of monitoring data in callback.
Return Value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

Table 3-16 dwFlag types and meanings

dwFlag	Meanings
0x00000001	Device original data
0x00000004	Data transformed to YUV format.

3.1.5 Device Control

3.1.5.1 Device Controlling ControlDevice

Table 3-17 Device control description

Item	Description	
Description	Device control.	
Function	<pre>bool ControlDevice(IntPtr ILoginID, EM_CtrlType type, IntPtr param, int waittime)</pre>	
Parameter	[in]ILoginID	Return value of LoginWithHighLevelSecurity.
	[in]type	Control type.
	[in]param	Input parameters, which vary by emType.
	[in]waittime	Timeout period, 1000 ms by default, which can be set as needed.
Return Value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.1.6 Alarm Listening

3.1.6.1 Setting Alarm Callback Function SetDVRMessCallBack

Table 3-18 Description of setting alarm callback function

Item	Description	
Description	Set alarm callback function.	
Function	void SetDVRMessCallBack(fMessCallBackEx cbMessage, IntPtr dwUser);	
Parameter	[in]cbMessage	Message callback function <ul style="list-style-type: none">• Status in which devices can be called back, such as alarm status.• When the value is set as 0, it means callback is forbidden.
	[in]dwUser	User-defined data.
Return Value	None	
Note	<ul style="list-style-type: none">• Set device message callback function to get the current device status information; this function is independent of the calling sequence, and the NetSDK is not called back by default.• The callback function fMessCallBack must call the alarm message subscription interface StartListen first before it takes effect.	

3.1.6.2 Subscribing to Alarm StartListen

Table 3-19 Description of subscribing to alarm

Item	Description	
Description	Subscribing alarms.	
Function	bool StartListen(IntPtr ILoginID);	
Parameter	[in]ILoginID	Return value of LoginWithHighLevelSecurity.
Return Value	<ul style="list-style-type: none">• Success: TRUE• Failure: FALSE	
Note	Subscribe to device message, and the message received is called back from the set value of SetDVRMessCallBack.	

3.1.6.3 Stopping Subscribing to Alarm StopListen

Table 3-20 Description of stopping subscribing to alarm

Item	Description
Description	Stop subscribing alarms.

Item	Description	
Function	<pre>bool StopListen(IntPtr ILoginID);</pre>	
Parameter	[in]ILoginID	Return value of LoginWithHighLevelSecurity.
Return Value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.1.7 Getting Device Status

3.1.7.1 Getting Device Status QueryDevState

Table 3-21 Description of getting device status

Item	Description	
Description	Directly get the connection status of remote devices.	
Function	<pre>bool QueryDevState(IntPtr ILoginID, int nType, ref object obj, Type typeName, int waittime);</pre>	
Parameter	[in]ILoginID	Return value of LoginWithHighLevelSecurity.
	[in]nType	Query information type. When getting remote device connection status, nType is EM_DEVICE_STATE.VIRTUALCAMERA.
	[out]obj	Used to receive data cache returned by querying. The structural body is NET_VIRTUALCAMERA_STATE_INFO.
	[in]typeName	Type of query structural body.
	[in]waittime	Waiting time in query status.
Return Value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.1.8 Voice Talk

3.1.8.1 Setting Voice Talk Mode SetDeviceMode

Table 3-22 Description of setting device voice talk mode

Item	Description
Description	Set device voice talk mode.

Item	Description	
Function	<pre>bool SetDeviceMode(IntPtr ILoginID, EM_USEDEV_MODE emType, IntPtr pValue);</pre>	
Parameter	[in]ILoginID	Return value of LoginWithHighLevelSecurity.
	[in]emType	Enumerated value.
	[in]pValue	For structure data pointers corresponding to the enumerated values.
Return Value	<ul style="list-style-type: none"> Success: TRUE Failure: FALSE 	
Note	None.	

Table 3-23 Enumeration of working modes and structural body

Emulated emType	Definition	Structural body
EM_USEDEV_MODE. TALK_ENCODE_TYPE	Select a designated format to realize voice call.	NET_DEV_TALKDECODE_INFO
EM_USEDEV_MODE. TALK_CLIENT_MODE	Configure client method for voice call.	None
EM_USEDEV_MODE. TALK_SPEAK_PARAM	Configure speaking parameter for voice call.	NET_SPEAK_PARAM
EM_USEDEV_MODE. TALK_MODE3	Configure voice call parameter for the 3 rd generation video door phone.	NET_TALK_EX

3.1.8.2 Starting Talk StartTalk

Table 3-24 Description of starting talk

Item	Description	
Description	Start voice talk.	
Function	<pre>IntPtr StartTalk(IntPtr ILoginID, fAudioDataCallBack pfcb, IntPtr dwUser);</pre>	
Parameter	[in]ILoginID	Return value of LoginWithHighLevelSecurity.
	[in]pfcb	Audio data callback function.
	[in]dwUser	Parameters of audio data callback function.
Return Value	<ul style="list-style-type: none"> Success: Non-0 Failure: 0 	
Note	None.	

3.1.8.3 Stopping Talk StopTalk

Table 3-25 Description of stopping talk

Item	Description	
Description	Stop voice talk.	
Function	bool StopTalk(IntPtr ITalkHandle);	
Parameter	[in]ITalkHandle	Return value of StartTalk
Return Value	<ul style="list-style-type: none">• Success: TRUE• Failure: FALSE	
Note	None.	

3.1.8.4 Enabling the Recording RecordStart

Table 3-26 Description of enabling the recording

Item	Description	
Description	Open the local recording.	
Function	bool RecordStart(IntPtr ILoginID);	
Parameter	[in]ILoginID	Return value of LoginWithHighLevelSecurity.
Return Value	<ul style="list-style-type: none">• Success: TRUE• Failure: FALSE	
Note	This interface is only valid in Windows.	

3.1.8.5 Disabling the Recording RecordStop

Table 3-27 Description of disabling the recording

Item	Description	
Description	Stop the local recording.	
Function	bool RecordStop(IntPtr ILoginID);	
Parameter	[in]ILoginID	Return value of LoginWithHighLevelSecurity.
Return Value	<ul style="list-style-type: none">• Success: TRUE• Failure: FALSE	
Note	This interface is only valid in Windows.	

3.1.8.6 Sending Voice TalkSendData

Table 3-28 Description of sending voice

Item	Description
Description	Send audio data to the device.

Item	Description	
Function	<pre>int TalkSendData(IntPtr ITalkHandle, IntPtr pSendBuf, uint dwBufSize);</pre>	
Parameter	[in]ITalkHandle	Return value of StartTalk.
	[in]pSendBuf	Pointer of audio data blocks to be sent.
	[in]dwBufSize	Length of audio data blocks to be sent, in bytes.
Return Value	<ul style="list-style-type: none"> Length of audio data blocks successfully returned. Return -1 if failed. 	
Note	None.	

3.1.8.7 Decoding Voice AudioDec

Table 3-29 Description of decoding voice

Item	Description	
Description	Decode audio data.	
Function	<pre>void AudioDec(IntPtr pAudioDataBuf, uint dwBufSize);</pre>	
Parameter	[in]pAudioDataBuf	Pointer of audio data blocks to be decoded.
	[in]dwBufSize	Length of audio data blocks to be decoded, in bytes.
Return Value	None.	
Note	None.	

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

3.2.1 Access Control

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

For details of the door sensor status interface, see "3.2.3.3 Querying Device Status QueryDevState."

3.2.2 Alarm Event

See "3.1.6 Alarm Listening."

3.2.3 Viewing Device Information

3.2.3.1 Querying System Capability Information

Table 3-30 Description of Querying system capability information

Item	Description	
Description	Query system capability information in string format.	
Function	bool QueryNewSystemInfo(IntPtr ILoginID, Int32 IChannel, string strCommand, ref object obj, Type typeName, int waittime)	
Parameter	[in] ILoginID	Return value of LoginWithHighLevelSecurity.
	[in] IChannel	Channel number.
	[in] szCommand	Command parameter.
	[out] obj	Received protocol buffer.
	[in] typeName	Structural body.
	[in] waittime	Timeout period, 1000ms by default, which can be set as needed.
Return Value	<ul style="list-style-type: none">• Success: TRUE• Failure: FALSE	
Note	None.	

3.2.3.2 Getting Device Capabilities GetDevCaps

Table 3-31 Description of getting device capabilities

Item	Description	
Description	Get device capabilities.	
Function	bool GetDevCaps(IntPtr ILoginID, EM_DEVCAP_TYPE nType, IntPtr pInBuf, IntPtr pOutBuf, int nWaitTime)	
Parameter	[in] ILoginID	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.

Item	Description
Return Value	<ul style="list-style-type: none"> Success: TRUE, Failure: FALSE
Note	None.

Table 3-32 Comparison of nType, plnBuf and pOutBuf

nType	Description	plnBuf	pOutBuf
EM_DEVCAP_TYPE. FACEINFO_CAPS	Obtain the capability set for face access controller	NET_IN_GET_FAC EINFO_CAPS	NET_OUT_GET_FAC EINFO_CAPS

3.2.3.3 Querying Device Status QueryDevState

Table 3-33 Description of querying device status

Item	Description	
Description	Directly get the current working status of the remote device.	
Function	bool QueryDevState(IntPtr ILoginID, int nType, ref object obj, Type typeName, int waittime);	
Parameter	[in] ILoginID	Return value of LoginWithHighLevelSecurity.
	[in]nType	Information type. When getting status of remote devices, nType: EM_DEVICE_STATE.VIRTUALCAMERA
	[out]obj	Output parameter, used to receive the returned data buffer in query. Corresponding to NET_VIRTUALCAMERA_STATE_INFO structure
	[in]typeName	Structural body type
	[in] waittime	Status waiting period.
Return Value	<ul style="list-style-type: none">● Success: TRUE,● Failure: FALSE	
Note	None.	

Table 3-34 Correspondence between nType, Query type and structure

nType	Description	pBuf
EM_DEVICE_STATE.SOFT WARE	Query device software version information	NET_DEV_VERSION_INFO
EM_DEVICE_STATE.NETIN TERFACE	Query network port information	NET_DEV_NETINTERFACE_I NFO
EM_DEVICE_STATE.RECO RDSET	Query device record set information	NET_CTRL_RECORDSET_PA RAM
EM_DEVICE_STATE.DOOR_ STATE	Query access control status (door sensor)	NET_DOOR_STATUS_INFO

3.2.4 Network Setting

3.2.4.1 IP Settings

3.2.4.1.1 Querying Config Information GetNewDevConfig

Table 3-35 Description of Querying config information

Item	Description	
Description	Get config in string format.	
Function	bool GetNewDevConfig(IntPtr ILoginID, Int32 IChannel, string strCommand, ref object obj, Type typeName, int waittime);	
Parameter	[in] ILoginID	Return value of LoginWithHighLevelSecurity.
	[in] IChannel	Device channel number, starting from 0.
	[in] strCommand	Command parameter. Channel name strCommand is ChannelTitle.
	[out] obj	Information array that is found.
	[in] typeName	Structural body type.
	[in] waittime	Timeout period for waiting.
Return Value	<ul style="list-style-type: none">• Success: TRUE,• Failure: FALSE	
Note	None.	

3.2.4.1.2 Setting Config Information SetNewDevConfig

Table 3-36 Description of setting config information

Item	Description	
Description	Get config in string format.	
Function	bool SetNewDevConfig(IntPtr ILoginID, int IChannel, string strCommand, object obj, Type typeName, int waittime)	
Parameter	[in] ILoginID	Return value of LoginWithHighLevelSecurity.
	[in] szCommand	Command parameter information.
	[in] nChannelID	Channel number.

Item	Description	
	[in]obj	The content you configured.
	[in]typeName	Structural body type.
	[in] waittime	Timeout period for waiting.
Return Value	<ul style="list-style-type: none"> • Success: TRUE, • Failure: FALSE 	
Note	None.	

3.2.4.2 Auto Register Config

3.2.4.2.1 Querying Config Information GetNewDevConfig

See "3.2.4.1.1 Querying Config Information GetNewDevConfig."

3.2.4.2.2 Setting Config Information SetNewDevConfig

See "3.2.4.1.2 Setting Config Information SetNewDevConfig."

3.2.5 Time Settings

3.2.5.1 Time Settings

Table 3-37 Description of time settings

Item	Description	
Description	Set the current time of the device.	
Function	<pre>bool SetupDeviceTime(IntPtr ILoginID, NET_TIME DeviceTime)</pre>	
Parameter	[in] ILoginID	Login handle.
	[in] DeviceTime	Set device time.
Return Value	<ul style="list-style-type: none"> • 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.	

3.2.5.2 NTP Time Sync, Time Zone Config

3.2.5.2.1 Querying Config Information GetNewDevConfig

See "3.2.4.1.1 Querying Config Information GetNewDevConfig."

3.2.5.2.2 Setting Config Information SetNewDevConfig

See "3.2.4.1.2 Setting Config Information SetNewDevConfig."

3.2.5.3 DST Setting

3.2.5.3.1 Querying Config Information GetNewDevConfig

See "3.2.4.1.1 Querying Config Information GetNewDevConfig."

3.2.5.3.2 Setting Config Information SetNewDevConfig

See "3.2.4.1.2 Setting Config Information SetNewDevConfig."

3.2.6 Maintenance Config

3.2.6.1 Changing Login Password

3.2.6.1.1 Operating Device User OperateUserInfoNew

Table 3-38 Description of operating device user

Item	Description	
Description	Operate device user, supporting up to 64-channel device.	
Function	bool OperateUserInfoNew(IntPtr ILoginID, EM_OPERATE_USER_TYPE nOperateType, IntPtr opParam, IntPtr subParam, int waittime)	
Parameter	[in] ILoginID	Return value of LoginWithHighLevelSecurity.
	[in] nOperateType	For operation types, see Table 3-39 for details.
	[in] opParam	Set the input buffer for user information. See Table 3-39 for details.
	[in] subParam	Set the auxiliary input buffer for user information. When the set type is modified information, part of the original user information shall be passed in here. See Table 3-39 for details.
	[in] waittime	Timeout period, 1000ms by default, which can be set as needed.
Return Value	<ul style="list-style-type: none">• Success: TRUE• Failure: FALSE	
Note	To implement the required function, set user information for changed devices.	

Table 3-39 Correspondence between nOperateType, opParam and subParam

nOperateType	opParam	subParam
EM_OPERATE_USER_TYPE.MODIFY_PASSWORD	NET_USER_INFO_NEW	NET_USER_INFO_NEW

3.2.6.2 Restart

3.2.6.2.1 Device Control ControlDevice

Table 3-40 Device control description

Item	Description	
Description	Device control.	
Function	bool ControlDevice(IntPtr ILoginID, EM_CtrlType type, IntPtr param, int waittime)	
Parameter	[in]ILoginID	Return value of LoginWithHighLevelSecurity.
	[in]type	Control type.
	[in]param	Control parameters vary by type.
	[in]waittime	Timeout period, 1000ms by default, which can be set as needed.
Return Value	<ul style="list-style-type: none"> Success: TRUE Failure: FALSE 	
Note	None.	

Table 3-41 Comparison of type and param

Type	Description	Param
REBOOT	Restart	None
RECORDSET_INSERT	Add records to get the record set number	NET_CTRL_RECORDSET_INSERT_PARAM
RECORDSET_INSERTEX	Add fingerprint records to get the record set number	NET_CTRL_RECORDSET_INSERTEX_PARAM
RECORDSET_REMOVE	Delete a record according to the record set number	NET_CTRL_RECORDSET_PARAM
RECORDSET_CLEAR	Clear information of all record sets	NET_CTRL_RECORDSET_PARAM
RECORDSET_UPDATE	Update records of a record set number	NET_CTRL_RECORDSET_PARAM
RECORDSET_UPDATEEX	Update records of a fingerprint record set number	NET_CTRL_RECORDSET_PARAM
ACCESS_OPEN	Access control—open	NET_CTRL_ARM_DISARM_PARAM
RESTOREDEFAULT	Restore the device to factory default	NET_RESTORE_COMMON

3.2.6.3 Restoring to Factory Defaults

3.2.6.3.1 Restoring to Factory Defaults ControlDevice, ResetSystem

- For details of ControlDevice, see "3.2.6.2.1 Device Control."
- For details of ResetSystem, see Table 3-42.

Table 3-42 Description of restoring to factory defaults

Item	Description	
Description	Restoring to factory defaults.	
Function	bool ResetSystem(IntPtr ILoginID, ref NET_IN_RESET_SYSTEM pInParam, ref NET_IN_RESET_SYSTEM pOutParam, int nWaitTime)	
Parameter	[in] ILoginID	Return value of LoginWithHighLevelSecurity.
	[in] pstInParam	Input parameter for restoring to factory defaults.
	[out] pstOutParam	Output parameter for restoring to factory defaults.
	[in] nWaitTime	Timeout period.
Return Value	<ul style="list-style-type: none">• Success: TRUE• Failure: FALSE	

3.2.6.4 Device Upgrade

3.2.6.4.1 Starting Upgrading StartUpgrade

Table 3-43 Description of start upgrading device program

Item	Description	
Description	Start upgrading device program—extension.	
Function	IntPtr StartUpgrade(IntPtr ILoginID, EM_UPGRADE_TYPE emType, string pchFileName, fUpgradeCallBack cbUpgrade, IntPtr dwUser)	
Parameter	[in] ILoginID	Return value of LoginWithHighLevelSecurity.
	[in] emType	Enumerated value. See Table 3-44 for details.
	[in] pchFileName	Name of file to be upgraded.
	[in] cbUpgrade	Upgrade progress callback function. See "4.8 Upgrade Progress Callback fUpgradeCallBackEx" for details.
	[in] dwUser	User-defined data.
Return Value	<ul style="list-style-type: none">• Success: Upgrade handle ID• Failure: 0	

Item	Description
Note	Set the upgrade of remote programs to return the program upgrade handle. Calling this interface has not sent upgrade program data, which will be sent by calling the SendUpgrade interface.

Table 3-44 Enumerated value

emType	Meanings
BIOS_TYPE	BIOS upgrade
WEB_TYPE	WEB upgrade
BOOT_YPE	BOOT upgrade
CHARACTER_TYPE	Chinese character library
LOGO_TYPE	LOGO
EXE_TYPE	EXE, such as player
DEVCONSTINFO_TYPE	Inherent device information settings (such as hardware ID, MAC, SN)
PERIPHERAL_TYPE	Peripheral access slave chip (such as vehicle chip)
GEOINFO_TYPE	Geographic information positioning chip
MENU	Menu (pictures in the device operating interface)
ROUTE	Route file (such as bus routes)
ROUTE_STATE_AUTO	Bus stop announcement audio (matching with routes)
SCREEN	Dispatch screen (such as bus operating screen)

3.2.6.4.2 Starting Sending SendUpgrade

Table 3-45 Description of starting sending upgrade file

Item	Description
Description	Start sending upgrade file.
Function	bool SendUpgrade(IntPtr IUpgradeID)
Parameter	[in] IUpgradeID Upgrade handle ID.
Return Value	<ul style="list-style-type: none"> Success: TRUE Failure: FALSE
Note	Send upgrade program data.

3.2.6.4.3 Stop Upgrading StopUpgrade

Table 3-46 Description of stopping upgrading

Item	Description
Description	Stopping upgrading.
Function	bool StopUpgrade(IntPtr IUpgradeID)
Parameter	[in] IUpgradeID Upgrade handle ID.
Return Value	<ul style="list-style-type: none"> Success: TRUE Failure: FALSE
Note	Do not call this interface in callback function.

3.2.6.5 Auto Maintenance

3.2.6.5.1 Querying Config Information GetDevConfig

Table 3-47 Description of querying config information

Item	Description	
Description	Read device config information.	
Function	bool GetDevConfig(IntPtr ILoginID, EM_DEV_CFG_TYPE type, int IChannel, IntPtr lpOutBuffer, uint dwOutBufferSize, ref uint bytesReturned, int waittime)	
Parameter	[in] ILoginID	Device login handle.
	[in] type	Device config command. See EM_DEV_CFG_TYPE enumeration.
	[in] IChannel	Channel number. If all channel data obtained is 0xFFFFFFFF and the command does not require channel number, this parameter is invalid.
	[out] lpOutBuffer	Pointer of received data buffer.
	[in] dwOutBufferSize	Length of received data buffer (in bytes).
	[out] BytesReturned	Length of data actually received.
	[in] waittime	Timeout period for waiting.
Return Value	<ul style="list-style-type: none">• Success: TRUE• Failure: FALSE	
Note	None.	

Table 3-48 Correspondence between dwCommand and lpOutBuffer

dwCommand	Query type	Corresponding structure lpOutBuffer
DST_CFG	DST configuration	NET_CFG_NTP_INFO
AUTOMTCFG	Auto maintenance config	NET_DEV_AUTOMT_CFG

3.2.6.5.2 Configuring Config Information SetDevConfig

Table 3-49 Description of configuring config information

Item	Description
Description	Set device config information.

Item	Description	
Function	<pre>bool SetDevConfig(IntPtr ILoginID, EM_DEV_CFG_TYPE type, int IChannel, IntPtr lpInBuffer, uint dwInBufferSize, int waittime)</pre>	
Parameter	[in] ILoginID	Device login handle.
	[in] type	Device config commands.
	[in] IChannel	Channel number. If all channel data obtained is 0xFFFFFFFF and the command does not require channel number, this parameter is invalid.
	[in] lpInBuffer	Data buffer pointer.
	[in] dwInBufferSize	Data buffer length (in bytes).
	[in] waittime	Timeout period for waiting.
Return Value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.2.7 Personnel Management

3.2.7.1 Collection of Personnel Information Fields

See "3.2.6.2.1 Device Control ControlDevice" and "3.2.3.3 Querying Device Status QueryDevState."

3.2.8 Door Config

3.2.8.1 Door Config Information

3.2.8.1.1 Querying Config Information GetNewDevConfig

See "3.2.4.1.1 Querying Config Information GetNewDevConfig."

3.2.8.1.2 Setting Config Information SetNewDevConfig

See "3.2.4.1.2 Setting Config Information SetNewDevConfig."

3.2.9 Door Time Config

3.2.9.1 Period Config

3.2.9.1.1 Querying Config Information GetNewDevConfig

See "3.2.4.1.1 Querying Config Information GetNewDevConfig."

3.2.9.1.2 Setting Config Information SetNewDevConfig

See "3.2.4.1.2 Setting Config Information SetNewDevConfig."

3.2.9.2 Always Open and Always Closed Period Config

3.2.9.2.1 Querying Config Information GetNewDevConfig

See "3.2.4.1.1 Querying Config Information GetNewDevConfig."

3.2.9.2.2 Setting Config Information SetNewDevConfig

See "3.2.4.1.2 Setting Config Information SetNewDevConfig"

3.2.9.3 Holiday Config

See "3.2.6.2.1 Device Control ControlDevice" and "3.2.3.3 Querying Device Status QueryDevState."

3.2.10 Advanced Config of Door

3.2.10.1 Unlock at Designated Intervals and First Card Unlock

3.2.10.1.1 Querying Config Information GetNewDevConfig

See "3.2.4.1.1 Querying Config Information GetNewDevConfig."

3.2.10.1.2 Setting Config Information SetNewDevConfig

See "3.2.4.1.2 Setting Config Information SetNewDevConfig."

3.2.10.2 Combination Unlock by Multiple Persons

3.2.10.2.1 Querying Config Information GetNewDevConfig

See "3.2.4.1.1 Querying Config Information GetNewDevConfig."

3.2.10.2.2 Setting Config Information SetNewDevConfig

See "3.2.4.1.2 Setting Config Information SetNewDevConfig."

3.2.10.3 Inter-door Lock

3.2.10.3.1 Querying Config Information GetNewDevConfig

See "3.2.4.1.1 Querying Config Information GetNewDevConfig."

3.2.10.3.2 Setting Config Information SetNewDevConfig

See "3.2.4.1.2 Setting Config Information SetNewDevConfig."

3.2.10.4 Anti-passback

3.2.10.4.1 Querying Config Information GetNewDevConfig

See "3.2.4.1.1 Querying Config Information GetNewDevConfig."

3.2.10.4.2 Setting Config Information SetNewDevConfig

See "3.2.4.1.2 Setting Config Information SetNewDevConfig."

3.2.10.5 Unlock Password

See "3.2.6.2.1 Device Control ControlDevice."

3.2.10.6 Device Log

3.2.10.6.1 Querying the Count of Device Logs QueryDevLogCount

Table 3-50 Description of Querying the count of device logs

Item	Description	
Description	Query the count of device logs.	
Function	bool QueryDevLogCount(IntPtr ILoginID, ref NET_IN_GETCOUNT_LOG_PARAM pInParam, ref NET_OUT_GETCOUNT_LOG_PARAM pOutParam, int nWaitTime)	
Parameter	[in] ILoginID	Device login handle.
	[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.

Item	Description	
	[in] waittime	Timeout period in query.
Return Value	Return the queried log count.	
Note	None.	

3.2.10.6.2 Starting Querying Logs

Table 3-51 Description of starting Querying logs

Item	Description	
Description	Start Querying device logs.	
Function	<pre>IntPtr StartQueryLog(IntPtr ILoginID, ref NET_IN_START_QUERYLOG pInParam, ref NET_OUT_START_QUERYLOG pOutParam, int nWaitTime)</pre>	
Parameter	[in] ILoginID	Device login handle.
	[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	<ul style="list-style-type: none"> Success: non 0 Failure: 0 	
Note	None.	

3.2.10.6.3 Getting Logs

Table 3-52 Description of getting logs

Item	Description	
Description	Get logs.	
Function	<pre>bool QueryNextLog(IntPtr IFindLogID, ref NET_IN_QUERYNEXTLOG pInParam, ref NET_OUT_QUERYNEXTLOG pOutParam, int nWaitTime)</pre>	
Parameter	[in] IFindLoginID	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	<ul style="list-style-type: none"> Success: TRUE, Failure: FALSE 	
Note	None.	

3.2.10.6.4 Ending Querying Logs

Table 3-53 Description of ending querying logs

Item	Description	
Description	Stop querying device logs.	
Function	bool StopQueryLog(IntPtr IFindLoginID)	
Parameter	[in] IFindLoginID	Query log handle.
Return Value	<ul style="list-style-type: none">• Success: TRUE,• Failure: FALSE	
Description	None.	

3.2.11 Records Query

3.2.11.1 Unlock Records

3.2.11.1.1 Querying Record Count

Table 3-54 Description of Querying record count

Item	Description	
Description	Query the count of records.	
Function	bool QueryRecordCount(IntPtr IFindHandle, ref int nRecordCount, int waittime)	
Parameter	[in]IFindHandle	Handle of Querying records.
	[out]nRecordCount	Number of querying records
	[in]waittime	Timeout period in query.
Return Value	<ul style="list-style-type: none">• Success: TRUE• Failure: FALSE	
Note	Before calling this interface, you should call FindRecord first to open the query handle.	

3.2.11.1.2 Querying Records by Query Conditions

Table 3-55 Description of Querying records by Query conditions

Item	Description
Description	Query records by Query conditions.

Item	Description	
Function	<pre>bool FindRecord(IntPtr ILoginID, EM_NET_RECORD_TYPE emRecordType, object oCondition, Type tyCondition, ref IntPtr IFindID, int waittime)</pre>	
Parameter	[in] ILoginID	Device login handle.
	[in] emRecordType	Video type.
	[in] oCondition	Parameter for querying records.
	[in] tyCondition	Structural body.
	[out] IFindID	Return querying handle.
	[in] waittime	Timeout period for waiting.
Return Value	<ul style="list-style-type: none"> • Success: TRUE, • Failure: FALSE 	
Note	You can call this interface first to get the query handle, then call the FindNextRecord function to get the list of records. After the query is completed, you can call FindRecordClose to close the query handle.	

Table 3-56 Description of unlock records parameter

emRecordType value	Structural body	Description
EM_NET_RECORD_TYPE. ACCESSCTLCARDREC_EX	NET_FIND_RECORD_ACCESS CTLCARDREC_CONDITION_E X	Query door unlock records.

3.2.11.1.3 Querying Records

Table 3-57 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	<pre>int FindNextRecord(IntPtr IFindHandle, int nMaxNum, ref int nRetNum, ref List<object> ls, Type tyRecord, int waittime)</pre>	
Parameter	[in] IFindHandle	Device query handle
	[in] nMaxNum	The max number of devices that can be Queried.
	[out] nRetNum	The max number of devices found.
	[out] ls	Structural body list Queried.
	[in] tyRecord	Structural body type.

Item	Description	
	[in] waittime	Timeout period for waiting.
Return Value	<ul style="list-style-type: none"> • Success: 1 • Failure: 0 	
Note	None.	

3.2.11.1.4 Ending Record Query

Table 3-58 Description of ending record Query

Item	Description	
Description	Stop record Query.	
Function	bool FindRecordClose(IntPtr IFindHandle)	
Parameter	[in] IFindHandle	Return value of FindRecord.
Return Value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	Call FindRecord to open the query handle; after the Query is completed, you should call this function to close the Query for handles.	

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

3.3.1 Access Control

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

For details of the door contact status interface, see 3.2.3.3 Querying Device Status QueryDevState"

3.3.2 Alarm Event

See "3.1.6 Alarm Listening."

3.3.3 Viewing Device Information

3.3.3.1 Getting Device Capabilities QueryDevState

Table 3-59 Description of getting device capabilities

Item	Description
Description	Get device capabilities.

Item	Description	
Function	<pre> bool GetDevCaps(IntPtr ILoginID, EM_DEVCAP_TYPE nType, IntPtr pInBuf, IntPtr pOutBuf, int nWaitTime) </pre>	
Parameter	[in] ILoginID	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	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Description	None.	

Table 3-60 Comparison of nType, pInBuf and pOutBuf

nType	Description	pInBuf	pOutBuf
EM_DEVCAP_TYPE.ACCESSCONTROL_CAPS	Get the access control capability	NET_IN_AC_CAPS	NET_OUT_AC_CAPS

3.3.3.2 Querying for Device Status

For details about QueryDevState, see "3.2.3.3 Querying Device Status QueryDevState."

3.3.4 Network Setting

See "3.2.4 Network Setting."

3.3.5 Time Settings

See "3.2.5 Time Settings."

3.3.6 Maintenance Config

See "3.2.6 Maintenance Config."

3.3.7 Personnel Management

3.3.7.1 User Management

3.3.7.1.1 User Information Querying Interface StartFindUserInfo

Table 3-61 Description of user information Querying interface

Item	Description	
Description	Personnel information Querying interface.	
Function	IntPtr StartFindUserInfo(IntPtr ILoginID, ref NET_IN_USERINFO_START_FIND pstIn, ref NET_OUT_USERINFO_START_FIND pstOut, int nWaitTime)	
Parameter	[in] ILoginID	Login handle.
	[in] pstIn	User information management (input parameter).
	[out] pstOut	User information management (output parameter).
	[in] nWaitTime	Timeout period.
Return value	<ul style="list-style-type: none"> • Success: non 0 • Failure: 0 	
Note	None.	

3.3.7.1.2 Getting Personnel Information Interface DoFindUserInfo

Table 3-62 Description of getting personnel information interface

Item	Description	
Description	Getting personnel information interface	
Function	bool DoFindUserInfo(IntPtr IFindHandle, ref NET_IN_USERINFO_DO_FIND pstIn, ref NET_OUT_USERINFO_DO_FIND pstOut, int nWaitTime)	
Parameter	[in] IFindHandle	Return value of StartFindUserInfo.
	[in] pstIn	Getting personnel information interface (input parameter).
	[out] pstOut	Getting personnel information interface (output parameter).
	[in] nWaitTime	Timeout period.
Return value	<ul style="list-style-type: none"> • Success: true. • Failure: false. 	
Note	None	

3.3.7.1.3 Stopping Getting Personnel Information Interface StopFindUserInfo

Table 3-63 Stopping getting personnel information interface

Item	Description	
Description	Stopping getting personnel information interface.	
Function	bool StopFindUserInfo(IntPtr IFindHandle)	
Parameter	[in] IFindHandle	StartFindUserInfo return value.

Item	Description
Return value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE
Note	None.

3.3.7.1.4 Access Control User Info Getting Interface GetOperateAccessUserService

Table 3-64 Description of access control user info getting interface

Item	Description
Description	Access control user info getting interface
Function	<pre>bool GetOperateAccessUserService(IntPtr ILoginID, string[] userid, out NET_ACCESS_USER_INFO[] stOutParam1, out NET_EM_FAILCODE[] stOutParam2, int nWaitTime)</pre>
Parameter Return value	[in] ILoginID Login handle
	[in] userid Userid of the user to be Queried for.
	[in] stOutParam1 User info management (output parameter).
	[out] stOutParam2 User info Querying error type (input parameter).
	[in] nWaitTime Timeout period.
Note	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE
Description	None.

3.3.7.1.5 Access Control User Info Adding Interface InsertOperateAccessUserService

Table 3-65 Description of access control user info adding interface

Item	Description
Description	Access control user info adding interface
Function	<pre>bool InsertOperateAccessUserService(IntPtr ILoginID, NET_ACCESS_USER_INFO[] stInParam, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)</pre>
Parameter	[in] ILoginID Login handle
	[in] stInParam User info management (input parameter)
	[out] stOutParam User info management (output parameter)
	[in] nWaitTime Timeout duration
Return value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE
Note	None.

3.3.7.1.6 Access Control User Info Deleting Interface RemoveOperateAccessUserService

Table 3-66 Description of access control user info deleting interface

Item	Description	
Description	Access control user info deleting interface	
Function	bool RemoveOperateAccessUserService(IntPtr ILoginID, string[] userid, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)	
Parameter	[in] ILoginID	Login handle
	[in] userid	User ID of users to be deleted.
	[out] stOutParam	User info query error type (output parameter).
	[in] nWaitTime	Timeout duration
Return value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.3.7.1.7 Access Control User Info Clearing Interface ClearOperateAccessUserService

Table 3-67 Description of access control user info clearing interface

Item	Description	
Description	Access control user info cclearinginterface.	
Function	bool ClearOperateAccessUserService(IntPtr ILoginID, int nWaitTime)	
Parameter	[in] ILoginID	Login handle
	[in] nWaitTime	Timeout duration
Return value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.3.7.2 Card Management

3.3.7.2.1 Card Information Management Interface for Access Control Devices StartFindCardInfo

Table 3-68 Description of card information management interface for access control devices

Item	Description	
Description	Card information management interface for access control devices.	
Function	IntPtr StartFindCardInfo(IntPtr ILoginID, ref NET_IN_CARDINFO_START_FIND pstIn, ref NET_OUT_CARDINFO_START_FIND pstOut, int nWaitTime)	
Parameter	[in] ILoginID	Login handle.

Item	Description	
	[in] pstIn	Start Querying for card information interface (input parameter).
	[out] pstOut	Start Querying for card information interface (output parameter).
	[in] nWaitTime	Timeout duration
Return value	<ul style="list-style-type: none"> • Success: Query handle. • Failure: 0. 	
Description	None	

3.3.7.2.2 Finding Card Information Interface DoFindCardInfo

Table 3-69 Description of finding the card information interface

Item	Description	
Description	Finding the card information interface.	
Function	<pre>bool DoFindCardInfo(IntPtr IFindHandle, ref NET_IN_CARDINFO_DO_FIND pstIn, ref NET_OUT_CARDINFO_DO_FIND pstOut, int nWaitTime)</pre>	
Parameter	[in] IFindHandle	Return value of 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	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Description	None.	

3.3.7.2.3 Stopping Finding Card Information Interface StopFindUserInfo

Table 3-70 Description of stopping finding card information interface

Item	Description	
Description	Stopping finding card information interface.	
Function	<pre>bool StopFindCardInfo(IntPtr IFindHandle)</pre>	
Parameter	[in] IFindHandle	Return value of StartFindCardInf.
Return value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.3.7.2.4 Access Control Card Info Getting Interface GetOperateAccessCardService

Table 3-71 Description of access control card info getting interface

Item	Description	
Description	Access control card info getting interface	
Function	<pre>bool GetOperateAccessCardService(IntPtr ILoginID, string[] Cardid, out NET_ACCESS_CARD_INFO[] stOutParam1, out NET_EM_FAILCODE[] stOutParam2, int nWaitTime)</pre>	
Parameter	[in] ILoginID	Login handle.
	[in] Cardid	Card ID of cards to be Queried for.
	[in] stOutParam1	Card info management (output parameter)
	[out] stOutParam2	Card info querying error type (output parameter)
	[in] nWaitTime	Timeout period.
Return value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.3.7.2.5 Access Control Card Info Adding Interface InsertOperateAccessCardService

Table 3-72 Description of access control card info adding interface

Item	Description	
Description	Access control card info adding interface	
Function	<pre>bool InsertOperateAccessCardService(IntPtr ILoginID, NET_ACCESS_CARD_INFO[] stInParam, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)</pre>	
Parameter	[in] ILoginID	Login handle.
	[in] stInParam	Card info management (input parameter).
	[out] stOutParam	Card info management (output parameter)
	[in] nWaitTime	Timeout period.
Return value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.3.7.2.6 Access Control Card Info Deleting Interface RemoveOperateAccessCardService

Table 3-73 Description of access control card info deleting interface

Item	Description
Description	Access control card info deleting interface

Item	Description	
Function	bool RemoveOperateAccessCardService(IntPtr ILoginID, string[] Cardid, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)	
Parameter	[in] ILoginID	Login handle.
	[in] Cardid	Card ID of cards to be deleted.
	[out] stOutParam	Card info querying error type (output parameter)
	[in] nWaitTime	Timeout period.
Return value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.3.7.2.7 Access Control Card Info Update Interface ClearOperateAccessCardService

Table 3-74 Description of access control card info update interface

Item	Description	
Description	Access control card info update interface	
Function	bool UpdateOperateAccessCardService(IntPtr ILoginID, NET_ACCESS_CARD_INFO[] stInParam, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)	
Parameter	[in] ILoginID	Login handle.
	[in] stInParam	Card info management (input parameter)
	[out] stOutParam	Card info management (output parameter)
	[in] nWaitTime	Timeout period.
Return value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.3.7.3 Face Management

3.3.7.3.1 Face Information Management Interface for Access Control Devices OperateAccessFaceService

Table 3-75 Description of face information management interface for access control devices

Item	Description
Description	Face information management interface for access control devices.

Item	Description	
Function	<pre>bool OperateAccessFaceService(IntPtr ILoginID, EM_NET_ACCESS_CTL_FACE_SERVICE emtype, IntPtr pstInParam, IntPtr pstOutParam, int nWaitTime)</pre>	
Parameter	[in] ILoginID	Login handle.
	[in] emtype	Face information operation type.
	[in] pstInParam	Face information management (input parameter).
	[out] pstOutParam	Face information management (output parameter).
	[in] nWaitTime	Timeout period.
Return value	<ul style="list-style-type: none"> Success: TRUE Failure: FALSE 	
Note	None.	

Table 3-76 Comparison of emtype, pInBuf and pOutBuf

emtype	Description	pInBuf	pOutBuf
EM_NET_ACCESS_CTL_FACE_SERVICE .INSERT	Add the face information	NET_IN_ACCESS_FACE_SERVICE_INSERT	NET_OUT_ACCESS_FACE_SERVICE_INSERT
EM_NET_ACCESS_CTL_FACE_SERVICE .GET	Find the face information	NET_IN_ACCESS_FACE_SERVICE_GET	NET_OUT_ACCESS_FACE_SERVICE_GET
EM_NET_ACCESS_CTL_FACE_SERVICE .UPDATE	Update the face information	NET_IN_ACCESS_FACE_SERVICE_UPDATE	NET_OUT_ACCESS_FACE_SERVICE_UPDATE
EM_NET_ACCESS_CTL_FACE_SERVICE .REMOVE	Delete the face information	NET_IN_ACCESS_FACE_SERVICE_REMOVE	NET_OUT_ACCESS_FACE_SERVICE_REMOVE
EM_NET_ACCESS_CTL_FACE_SERVICE .CLEAR	Clear the face information	NET_IN_ACCESS_FACE_SERVICE_CLEAR	NET_OUT_ACCESS_FACE_SERVICE_CLEAR

3.3.7.4 Fingerprint Management

3.3.7.4.1 Fingerprint Information Management Interface for Access Control Devices GetOperateAccessFingerprintService

Table 3-77 Description of fingerprint information management interface for access control devices

Item	Description
Description	Fingerprint information management interface for access control devices.

Item	Description	
Function	bool GetOperateAccessFingerprintService(IntPtr ILoginID, string userid, IntPtr pFingerprintData, int dataLen, out NET_ACCESS_FINGERPRINT_INFO stOutParam1, int nWaitTime)	
Parameter	[in] ILoginID	Login handle.
	[in] userid	User ID of users to be queried.
	[out] pFingerprintData	Fingerprint information data (output parameter).
	[out] dataLen	Fingerprint information data length (output parameter).
	[out] stOutParam1	Fingerprint info query error type (output parameter).
	[in] nWaitTime	Timeout period.
Return value	<ul style="list-style-type: none"> Success: TRUE Failure: FALSE 	
Note	None.	

3.3.7.4.2 Access Control Fingerprint Info Adding Interface InsertOperateAccessFingerprintService

Table 3-78 Description of access control fingerprint info adding interface

Item	Description	
Description	Access control fingerprint info management interface.	
Function	bool InsertOperateAccessFingerprintService(IntPtr ILoginID, NET_ACCESS_FINGERPRINT_INFO[] stInParam, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)	
Parameter	[in] ILoginID	Login handle.
	[in] stInParam	Fingerprint info management (input parameter)
	[out] stOutParam	Fingerprint info management (output parameter)
	[in] nWaitTime	Timeout period.
Return value	<ul style="list-style-type: none"> Success: TRUE Failure: FALSE 	
Note	None.	

3.3.7.4.3 Access Control Fingerprint Info Deleting Interface RemoveOperateAccessFingerprintService

Table 3-79 Description of access control card info deleting interface

Item	Description
Description	Access control fingerprint info management interface.

Item	Description	
Function	bool RemoveOperateAccessFingerprintService(IntPtr ILoginID, string[] userid, out NET_EM_FAILCODE[] stOutParam, int nWaitTime)	
Parameter	[in] ILoginID	Login handle.
	[in] userid	User ID of users to be deleted.
	[out] stOutParam	Fingerprint info query error type (output parameter).
	[in] nWaitTime	Timeout period.
Return value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.3.7.4.4 Access Control Fingerprint Info Clearing Interface ClearOperateAccessFingerprintService

Table 3-80 Description of access control fingerprint info clearing interface

Item	Description	
Description	Access control fingerprint info management interface.	
Function	bool ClearOperateAccessFingerprintService(IntPtr ILoginID, int nWaitTime)	
Parameter	[in] ILoginID	Login handle.
	[in] nWaitTime	Timeout period.
Return value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE 	
Note	None.	

3.3.8 Door Config

3.3.8.1 Door Config Information

3.3.8.1.1 Querying Config Information

See "3.2.4.1.1 Querying Config Information."

3.3.8.1.2 Setting Config Information

See "3.2.4.1.2 Setting Config Information."

3.3.9 Door Time Config

3.3.9.1 Period Config

3.3.9.1.1 Querying Config Information

See "3.2.4.1.1 Querying Config Information."

3.3.9.1.2 Setting Config Information

See "3.2.4.1.2 Setting Config Information."

3.3.9.2 Always Open and Always Closed Period Config

3.3.9.2.1 Querying Config Information

See "3.2.4.1.1 Querying Config Information."

3.3.9.2.2 Setting Config Information

See "3.2.4.1.2 Setting Config Information."

3.3.9.3 Holiday group

3.3.9.3.1 Getting the Holiday Group Interface GetOperateConfig

Table 3-81 Description of getting the holiday group interface

Item	Description	
Description	Getting the holiday group interface.	
Function	bool GetOperateConfig(IntPtr ILoginID, EM_CFG_OPERATE_TYPE cfg_type, int IChannel, ref object obj, Type typeName, int waittime)	
Parameter	[in] ILoginID	Login handle.
	[in] cfg_type	Set the type of configuration info.
	[in] IChannel	Channel number.
	[out] obj	Returned data structural body.
	[in] typeName	Structural body type.
	[in] waittime	Timeout period.
Return value	<ul style="list-style-type: none">Success: TRUEFailure: FALSE	

Item	Description
Description	None.

Table 3-82 Description of cfg_type

cfg_type	Description	typeName
NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP	Get the holiday group info	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO

3.3.9.3.2 Setting the Holiday Group Interface SetOperateConfig

Table 3-83 Description of setting the holiday group interface

Item	Description
Description	Setting the holiday group interface.
Function	bool SetOperateConfig(IntPtr ILoginID, EM_CFG_OPERATE_TYPE cfg_type, int IChannel, object obj, Type typeName, int waittime)
Parameter	[in] ILoginID Login handle.
	[in] cfg_type Set the type of configuration info.
	[in] IChannel Channel number.
	[in] obj Returned data structural body.
	[in] typeName Structural body type.
	[in] waittime Timeout period.
Return value	<ul style="list-style-type: none"> • Success: TRUE • Failure: FALSE
Description	None.

Table 3-84 Description of cfg_type

cfg_type	Description	szInBuffer
EM_CFG_OPERATE_TYPE.SPECIALDAY_GROUP	Setting the holiday group info	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO

3.3.9.4 Holiday Plan

For details, see "3.3.9.3 Holiday group."

Table 3-85 Description of getting emCfgOpType

cfg_type	Description	typeName
EM_CFG_OPERATE_TYPE.SPECIALDAYS_SCHEDULE	Get holiday plan info	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO

Table 3-86 Description of configuring emCfgOpType

cfg_type	Description	typeName
EM_CFG_OPERATE _TYPE. SPECIALDAYS_SCH EDULE	Configure holiday plan info	NET_CFG_ACCESSCTL_SPECIALDAYS_SC HEDULE_INFO

3.3.10 Advanced Config of Door

See "3.2.10 Advanced Config of Door."

3.3.11 Records Query

3.3.11.1 Unlock Records

See "3.2.11.1 Unlock Records."

3.3.11.2 Alarm Records

3.3.11.2.1 Querying Record Count

See "3.2.11.1.1 Querying Record Count."

3.3.11.2.2 Querying Records by Query Conditions

See "3.2.11.1.2 Querying Records by Query Conditions."

Table 3-87 Description of unlocking record query input parameter

emRecordType	Structural Body	Description
EM_NET_RECORD_TYP E. ACCESS_ALARMRECOR D	NET_RECORD_ACCESS_ALARMRECORD _INFO	Used to query access and alarm records.

3.3.11.2.3 Querying Records

See "3.2.11.1.3 Querying Records."

3.3.11.2.4 Ending Record Query

See "3.2.11.1.4 Ending Record Query"

4 Callback Function

4.1 Device Querying Callback fSearchDevicesCB

Table 4-1 Description of callback function for Querying device

Item	Description	
Description	Callback function for Querying device.	
Function	public delegate void fQueryDevicesCB(IntPtr pDevNetInfo, IntPtr pUserData);	
Parameter	[out]pDevNetInfo	Device information.
	[out]pUserData	User data.
Return Value	None.	
Note	None.	

4.2 Device Querying Callback fSearchDevicesCBEx

Table 4-2 Callback of Querying devices

Item	Description	
Name	Callback of Querying devices.	
Function	public delegate void fQueryDevicesCBEx(IntPtr IQueryHandle, IntPtr pDevNetInfo, IntPtr pUserData);	
Parameter	[out] IQueryHandle	Query Handle
	[out]pDevNetInfo	Device information.
	[out]pUserData	User data.
Return value	None.	
Note	None.	

4.3 Disconnection Callback ffDisConnectCallBack

Table 4-3 Description of disconnecting callback function

Item	Description	
Description	Disconnection callback.	
Function	public delegate void fDisConnectCallBack(IntPtr ILoginID, IntPtr pchDVRIP, Int nDVRPort, IntPtr dwUser);	
Parameter	[out]ILoginID	Return value of LoginWithHighLevelSecurity.
	[out]pchDVRIP	Disconnected device IP.

Item	Description	
	[out]nDVRPort	Disconnected device port.
	[out]dwUser	User parameters for callback function.
Return Value	None.	
Note	None.	

4.4 Reconnection Callback fHaveReConnectCallBack

Table 4-4 Description of reconnecting callback function

Item	Description	
Description	Reconnection callback.	
Function	<pre>public delegate void fHaveReConnectCallBack(IntPtr ILoginID, IntPtr pchDVRIP, int nDVRPort, IntPtr dwUser);</pre>	
Parameter	[out]ILoginID	Return value of LoginWithHighLevelSecurity.
	[out]pchDVRIP	Reconnected device IP.
	[out]nDVRPort	Reconnected device port.
	[out]dwUser	User parameters for callback function.
Return Value	None.	
Note	None.	

4.5 Callback for Real-time Monitoring Data fRealDataCallBackEx2

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

Item	Description	
Description	Callback function for real-time monitoring data.	
Function	<pre>public delegate void fRealDataCallBackEx2(IntPtr IRealHandle, uint dwDataType, IntPtr pBuffer, uint dwBufSize, IntPtr param, IntPtr dwUser);</pre>	
Parameter	[out]IRealHandle	Return value of RealPlay.
	[out]dwDataType	Data type <ul style="list-style-type: none"> 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.

Item	Description	
	[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. <ul style="list-style-type: none"> dwDataType is 0, param is null pointer dwDataType is 1, param is NET_VideoFrameParam structural body. dwDataType is 3, param is NET_CBPCMDDataParam structural body.
	[out]dwUser	User parameters for callback function.
Return Value	None.	
Note	None.	

4.6 Audio Data Callback pfAudioDataCallback

Table 4-6 Description of audio data callback function

Item	Description	
Description	Audio data callback for voice talk.	
Function	<pre>public delegate void fAudioDataCallback(IntPtr ITalkHandle, IntPtr pDataBuf, uint dwBufSize, byte byAudioFlag, IntPtr dwUser);</pre>	
Parameter	[out]ITalkHandle	Return value of NETClient. StartTalkEx.
	[out]pDataBuf	Audio data block address.
	[out]dwBufSize	Length of audio data block, in bytes.
	[out]byAudioFlag	Flag of data type <ul style="list-style-type: none"> 0 means that the data is locally collected. 1 means that the data is sent from the device.
	[out]dwUser	User parameters for callback function.
Return Value	None.	
Note	None.	

4.7 Alarm Callback fMessCallBackEx

Table 4-7 Description of alarm callback function

Item	Description	
Description	Alarm callback function.	
Function	<pre>public delegate bool fMessCallBackEx(int ICommand, IntPtr ILoginID, IntPtr pBuf, uint dwBufLen, IntPtr pchDVRIP,</pre>	

Item	Description	
	int nDVRPort, bool bAlarmAckFlag, int nEventID, IntPtr dwUser);	
Parameter	[out]ICommand	Alarm type. See Table 4-8 for details.
	[out]ILoginID	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 ICommand values.
	[out]dwBufLen	Length of pBuf, in bytes.
	[out]pchDVRIP	Device IP.
	[out]nDVRPort	Port.
	[out]bAlarmAckFlag	TRUE, the event can be confirmed. FALSE, the event cannot be confirmed.
	[out]nEventID	Used to assign value to input parameter AlarmAck. When bAlarmAckFlag is TRUE, the data is valid.
	[out]dwUser	User-defined data.
Return Value	<ul style="list-style-type: none"> Success: TRUE Failure: FALSE 	
Note	Usually, call the set callback function during application initialization, and process properly in the callback function according to different device ID and command values.	

Table 4-8 Correspondence between alarm type and structure

Alarm business	Alarm type name	ICommand	pBuf
Alarm host	Local alarm event	ALARM_ALARM_EX2	NET_ALARM_ALARM_INFO_EX2
	Power failure event	ALARM_POWERFAULT	NET_ALARM_POWERFAULT_INFO
	Dismantlement prevention event	ALARM_CHASSISINTRUDE	NET_ALARM_CHASSISINTRUDED_INFO
	Extended alarm input channel event	ALARM_ALARMEXTENDED	NET_ALARM_ALARMEXTENDED_INFO
	Emergency event	URGENCY_ALARM_EX	The data is a 16-byte array, and each byte indicates the status of a channel: <ul style="list-style-type: none"> 1: There is alarm 0: There is no alarm
	Low battery voltage event	ALARM_BATTERYLOWPOWER	NET_ALARM_BATTERYLOWPOWER_INFO

Alarm business	Alarm type name	ICommand	pBuf
	Device inviting platform to talk event	ALARM_TALKING_INVITE	NET_ALARM_TALKING_INVITE_INFO
	Device arming mode change event	ALARM_ARMMODE_CHANGE_EVENT	NET_ALARM_ARMMODE_CHANGE_INFO
	Protection zone bypass status change event	ALARM_BYPASSMODE_CHANGE_EVENT	NET_ALARM_BYPASSMODE_CHANGE_INFO
	Alarm input source signal event	ALARM_INPUT_SOURCE_SIGNAL	NET_ALARM_INPUT_SOURCE_SIGNAL_INFO
	Alarm clearing event	ALARM_ALARMCLEAR	NET_ALARM_ALARMCLEAR_INFO
	Sub-system status change event	ALARM_SUBSYSTEM_STATE_CHANGE	NET_ALARM_SUBSYSTEM_STATE_CHANGE_INFO
	Extension module offline event	ALARM_MODULE_LOST	NET_ALARM_MODULE_LOST_INFO
	PSTN offline event	ALARM_PSTN_BREAK_LINE	NET_ALARM_PSTN_BREAK_LINE_INFO
	Analog quantity alarm event	ALARM_ANALOG_PULSE	NET_ALARM_ANALOGPULSE_INFO
	Alarm transmission event	ALARM_PROFILE_ALARM_TRANSMIT	NET_ALARM_PROFILE_ALARM_TRANSMIT_INFO
	Wireless device low battery alarm event	ALARM_WIRELESSDEV_LOWPOWER	NET_ALARM_WIRELESSDEV_LOWPOWER_INFO
	Protection zone arming and disarming status change event	ALARM_DEFENCE_ARMMODE_CHANGE	NET_ALARM_DEFENCE_ARMMODECHANGE_INFO

Alarm business	Alarm type name	ICommand	pBuf
	Sub-system arming and disarming status change event	ALARM_SUBSYSTEM_ARMMODE_CHANGE	NET_ALARM_SUBSYSTEM_ARMMODECHANGE_INFO
	Detector abnormality alarm	ALARM_SENSOR_ABNORMAL	NET_ALARM_SENSOR_ABNORMAL_INFO
	Patient activity status alarm event	ALARM_PATIENTDETECTION	NET_ALARM_PATIENTDETECTION_INFO
Access Control	Access control event	ALARM_ACCESS_CTL_EVENT	NET_ALARM_ACCESS_CTL_EVENT_INFO
	Details of access control unlocking event	ALARM_ACCESS_CTL_NOT_CLOSE	NET_ALARM_ACCESS_CTL_NOT_CLOSE_INFO
	Details of intrusion event	ALARM_ACCESS_CTL_BREAK_IN	NET_ALARM_ACCESS_CTL_BREAK_IN_INFO
	Details of repeated entry event	ALARM_ACCESS_CTL_REPEAT_ENTER	NET_ALARM_ACCESS_CTL_REPEAT_ENTER_INFO
	Malicious unlocking event	ALARM_ACCESS_CTL_MALICIOUS	NET_ALARM_ACCESS_CTL_MALICIOUS
	Details of forced card swiping event	ALARM_ACCESS_CTL_DURESS	NET_ALARM_ACCESS_CTL_DURESS_INFO
	Combination unlocking by multiple persons event	ALARM_OPENDOORGROUP	NET_ALARM_OPEN_DOOR_GROUP_INFO
	Dismantlement prevention event	ALARM_CHASSISINTRUDE	NET_ALARM_CHASSISINTRUDED_INFO
	Local alarm event	ALARM_ALARM_EX2	NET_ALARM_ALARM_INFO_EX2
	Access control status event	ALARM_ACCESS_CTL_STATUS	NET_ALARM_ACCESS_CTL_STATUS_INFO

Alarm business	Alarm type name	ICommand	pBuf
	Bolt alarm	ALARM_ACCESS_CTL_STATUS	NET_ALARM_ACCESS_CTL_STATUS_INFO
	Fingerprint acquisition event	ALARM_FINGER_PRINT	NET_ALARM_CAPTURE_FINGER_PRINT_INFO
Video Intercom	No response to the call in direct connection event	ALARM_CALL_NO_ANSWERED	NET_ALARM_CALL_NO_ANSWERED_INFO
	Mobile phone number report event	ALARM_TELEPHONE_CHECK	NET_ALARM_TELEPHONE_CHECK_INFO
	VTs status report	ALARM_VTSTATE_UPDATE	NET_ALARM_VTSTATE_UPDATE_INFO
	VTO face recognition	ALARM_ACCESSIDENTIFY	NET_ALARM_ACCESSIDENTIFY
	Device inviting another party to start talk event	ALARM_TALKING_INVITE	NET_ALARM_TALKING_INVITE_INFO
	Device canceling talk request event	ALARM_TALKING_IGNORE_INVITE	NET_ALARM_TALKING_IGNORE_INVITE_INFO
	Device actively hanging up talk event	ALARM_TALKING_HANGUP	NET_ALARM_TALKING_HANGUP_INFO
	Radar monitoring overspeed alarm event	ALARM_RADAR_HIGH_SPEED	NET_ALARM_RADAR_HIGH_SPEED_INFO

4.8 Upgrade Progress Callback fUpgradeCallbackEx

Table 4-9 Description of upgrade progress callback function

Item	Description
Description	Upgrade progress callback function. Update files of G level and above are supported.
Function	public delegate void fUpgradeCallbackEx(IntPtr ILoginID, IntPtr IUpgradechannel, long nTotalSize,

Item	Description	
	long nSendSize, IntPtr dwUser);	
Parameter	[out] ILoginID	Return value of login interface.
	[out] IUpgradechannel	Update handle ID returned by StartUpgrade.
	[out] nTotalSize	Total length of update file, in bytes.
	[out] nSendSize	Sent file length, in bytes; when it is -1, it means the sending of update file has ended.
	[out] dwUser	User-defined data.
Return Value	None.	
Note	Device upgrade program callback function prototype supports upgrade files above G. nTotalSize = 0, nSendSize = -1 means that upgrade is completed. nTotalSize = 0, nSendSize = -2 means upgrade error. nTotalSize = 0, nSendSize = -3 means that the user has no upgrade permission. nTotalSize = 0, nSendSize = -4 means that the upgrade program version is too low. nTotalSize = -1, nSendSize = XX means upgrade progress. nTotalSize = XX, nSendSize = XX means the progress of sending upgrade files.	

Appendix 1 Cybersecurity Recommendations

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

Mandatory actions to be taken for basic equipment network security:

1. Use Strong Passwords

Please refer to the following suggestions to set passwords:

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

2. Update Firmware and Client Software in Time

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

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

1. Physical Protection

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

2. Change Passwords Regularly

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

3. Set and Update Passwords Reset Information Timely

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

4. Enable Account Lock

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

5. Change Default HTTP and Other Service Ports

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

6. Enable HTTPS

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

7. 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.

8. Assign Accounts and Privileges Reasonably

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

9. 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.

10. 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.

11. 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.

12. 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.

13. 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.

- Enable IP/MAC address filtering function to limit the range of hosts allowed to access the device.