

NetSDK_C#

Programming Manual



Foreword

Purpose

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

NetSDK, also known as network device SDK, is a development kit for developing 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 NetSDK interfaces and processes of the general function modules for Network Video Recorder (NVR), Enterprise Video Storage (EVS), and High Definition Composite Video Interface (HDCVI). For more function modules and data structures, refer to *NetSDK Development Manual*.



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

Reader

- NetSDK software development engineers
- Project managers
- Product managers

Signals

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

Signal Words	Meaning
 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
V1.0.1	Updated the dependent library information.	April 2021
V1.0.0	First release.	March 2020

Privacy Protection Notice

As the device user or data controller, you might collect personal data of others such as face, fingerprints, car plate number, email address, phone number, GPS and so on. You need to be in compliance with the local privacy protection laws and regulations to protect the legitimate rights

and interests of other people by implementing measures include but not limited to: providing clear and visible identification to inform data subject the existence of surveillance area and providing related contact.

About the Manual

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

Glossary

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

Term	Definition
NVR	Abbreviation for Network Video Recorder.
EVS	Abbreviation for Enterprise Video Storage.
HDCVI	Abbreviation for High Definition Composite Video Interface.
Main Stream	A type of video stream that usually has better resolution and clarity and provides a better experience if the network resource is not restricted.
Sub Stream	A type of video stream that usually has lower resolution and clarity than the main stream but demands less network resources. The user can choose the stream type according to the particular scenes.
Resolution	Resolution is consisted of display resolution and image resolution. Display resolution refers to the quantity of pixels in unit area, and the image resolution refers to information quantity (the quantity of pixels per inch) stored in the image.
Frame Rate	A measurement, usually in FPS and Hz, which shows the frames of video. The more the frame, more smooth the video. The frames over 24 FPS make the image feels coherent.
Video Channel	An abstract concept of the communication and video stream transmission between NetSDK and devices. For example, if a number of cameras (SD, IPC) are mounted on a storage device (NVR), the storage device manages the cameras as video channels which are numbered from 0. If NetSDK connects to the camera directly, the video channel is usually numbered as 0.
Alarm of Dynamic Detection	When detecting a moving object on the image, an alarm by dynamic detection will be uploaded.
Alarm of Hard Disk Failure	When detecting a hard disk failure, an alarm will be uploaded.
Alarm of Video Loss	This alarm is only for analog channel. When the record disappeared from the analog channel, an alarm will be uploaded. For the digital channel, refer to IPC disconnection alarm.
Alarm of Hard Disk Damage	When the hard disk is damaged, an alarm will be uploaded.
IPC Offline Alarm	When the IPC device is disconnected, an alarm will be uploaded.
External Alarm	NVR local alarm. When the NVR alarm terminal connects with alarm device, an alarm will be uploaded.
IPC External Alarm	When the alarm on IPC device connects with alarm device, an external alarm will be uploaded.

Table of Contents

Foreword	I
Glossary	III
1 Overview	1
1.1 General.....	1
1.2 Applicability	2
2 Function Modules	3
2.1 NetSDK Initialization	3
2.1.1 Introduction.....	3
2.1.2 Interface Overview	3
2.1.3 Process	3
2.1.4 Example Code	4
2.2 Device Login and Logout	5
2.2.1 Introduction.....	5
2.2.2 Interface Overview	5
2.2.3 Process	6
2.2.4 Example Code	7
2.3 Real-time Monitoring	7
2.3.1 Introduction.....	7
2.3.2 Interface Overview	8
2.3.3 Process	8
2.3.4 Example Code	12
2.4 Record Playback	13
2.4.1 Introduction.....	13
2.4.2 Interface Overview	14
2.4.3 Process	14
2.4.4 Example Code	16
2.5 Record Download	17
2.5.1 Introduction.....	17
2.5.2 Interface Overview	18
2.5.3 Process	18
2.5.4 Example Code	22
2.6 PTZ Control.....	27
2.6.1 Introduction.....	27
2.6.2 Interface Overview	27
2.6.3 Process	28
2.6.4 Example Code	30
2.7 Voice Talk.....	30
2.7.1 Introduction.....	30
2.7.2 Interface Overview	31
2.7.3 Process	31
2.7.4 Example Code	33
2.8 Video Snapshot	36
2.8.1 Introduction.....	36

2.8.2 Interface Overview	36
2.8.3 Process	37
2.8.4 Example Code	39
2.9 Subscribing Intelligent Event	40
2.9.1 Introduction.....	40
2.9.2 Interface Overview	40
2.9.3 Process	41
2.9.4 Example Code	42
2.10 Alarm Upload	43
2.10.1 Introduction	43
2.10.2 Interface Overview	43
2.10.3 Process.....	44
2.10.4 Example Code.....	45
2.11 Device Status and Information.....	46
2.11.1 Introduction	46
2.11.2 Interface Overview	46
2.11.3 Process.....	47
2.11.4 Example Code.....	51
3 Interface Definition	57
3.1 NetSDK Initialization	57
3.1.1 NetSDK Initialization.....	57
3.1.2 NetSDK Cleanup.....	57
3.1.3 Auto Reconnection Setting.....	57
3.1.4 Network Parameter Setting	58
3.2 Device Login	58
3.2.1 Login.....	58
3.2.2 Logout.....	59
3.3 Real-time Monitoring	59
3.3.1 Opening the Real-time Monitoring.....	59
3.3.2 Stopping the Real-time Monitoring.....	60
3.3.3 Saving the Real-time Monitoring Data	60
3.3.4 Stopping Saving the Real-time Monitoring Data	60
3.3.5 Setting Callback of Real-time Monitoring Data.....	61
3.4 Record Playback	62
3.4.1 Playback by Time.....	62
3.4.2 Setting the Work Mode.....	62
3.4.3 Stopping Playback.....	63
3.4.4 Getting the OSD Playback Time	63
3.5 Record Download	64
3.5.1 Querying Record Files within a Period.....	64
3.5.2 Opening the Record Query Handle	65
3.5.3 Finding the Record File	65
3.5.4 Closing the Record Query Handle	66
3.5.5 Downloading Record by File	66
3.5.6 Downloading Record by Time.....	67
3.5.7 Querying the record downloading progress.....	68
3.5.8 Stopping Record Downloading.....	68

3.6 PTZ Control.....	69
3.6.1 PTZ Control.....	69
3.7 Voice Talk.....	72
3.7.1 Opening Voice Talk.....	72
3.7.2 Stopping Voice Talk.....	73
3.7.3 Starting Local Recording.....	73
3.7.4 Stopping Local Recording	73
3.7.5 Talk Data Sending	74
3.7.6 Audio Decoding	74
3.8 Video Snapshot.....	75
3.8.1 Capturing Picture to File.....	75
3.8.2 Capturing Picture.....	75
3.9 Intelligent Event	76
3.9.1 Subscribing Intelligent Event.....	76
3.9.2 Unsubscribing Smart.....	76
3.10 Alarm Upload.....	77
3.10.1 Setting Alarm Callback.....	77
3.10.2 Subscribing to Alarm	77
3.10.3 Stopping Alarm Subscription	77
3.11 Device Status and Information.....	78
3.11.1 Querying Device State.....	78
3.11.2 Querying Device Information.....	78
3.11.3 Subscribing to State of Remote Device	79
3.11.4 Stopping Subscribing State of Remote Device.....	79
3.11.5 Getting Information of Remote Device.....	79
3.11.6 Getting Channel Name.....	80
4 Callback Function	81
4.1 fDisconnectCallBack.....	81
4.2 fHaveReConnectCallBack.....	81
4.3 fRealDataCallBackEx.....	81
4.4 fAudioDataCallBack.....	82
4.5 fDownloadPosCallBack.....	83
4.6 fDataCallBack.....	83
4.7 fTimeDownloadPosCallBack	83
4.8 fMessCallBackEx	84
4.9 fCameraStateCallBack	85
4.10 fAnalyzerDataCallBack.....	86
Appendix 1 Cybersecurity Recommendations	87

1 Overview

1.1 General

The manual introduces NetSDK interfaces reference information that includes main function modules, interface functions, and callback functions.

The following are the main functions:

NetSDK initialization, device login, real-time monitoring, record playback, record download, PTZ control, voice talk, video snapshot, IVS, alarm upload, and storage.

- For the files included in NetSDK library of C#, see Table 1-1.

Table 1-1 Files of NetSDK library

Library type	Library file name	Library file description
Function library	dhnetsdk.dll	Library file
	avnetsdk.dll	Library file
Configuration library	dhconfigsdk.dll	Library file
Playing auxiliary library (encoding and decoding)	dhplay.dll	Playing library
	fisheye.dll	Fisheye correction library
Auxiliary library of "dhnetsdk"	lvsDrawer.dll	Large displaying library
	StreamConvertor.dll	Transcoding library

- For the files included in the C# encapsulation project, see Table 1-2.

Table 1-2 Files of NetSDKCS project

File name	File description
NetSDK.cs	Encapsulate the C# interfaces which can be called by users
NetSDKStruct.cs	Store the structure enumerations
OriginalSDK.cs	Import the C interfaces in NetSDK library to C# project



- The function library and configuration library are necessary libraries.
- The function library is the main body of NetSDK, which is used for communication interaction between client and products. It remotely controls device, queries device data, configures device data information, as well as gets and handles the streams.
- NetSDK library is the foundation of NetSDKCS project, and the file "OriginalSDK.cs" defines the reference path of NetSDK library. Put the NetSDK library to the corresponding path when using it. Support to customize the reference path.
- Support to directly reference the encapsulation project in your own project, use the files of encapsulation project in your own project, or encapsulate by yourself referring to this encapsulation project.
- This manual mainly introduces the C# project which is used to encapsulate the interfaces of the C library. For more details, see the manuals of C NetSDK library.

1.2 Applicability

- Recommended memory: No less than 512 M
- System supported by NetSDK:
Windows 10/Windows 8.1/Windows 7/vista and Windows Server 2008/2003

2 Function Modules

2.1 NetSDK Initialization

2.1.1 Introduction

Initialization is the first step of NetSDK to conduct all the function modules. NetSDK does not have the surveillance function but can set some parameters that affect the NetSDK overall functions.

- Initialization occupies some memory.
- Only the first initialization is valid within one process.
- After using this function, call cleanup interface to release NetSDK resource.

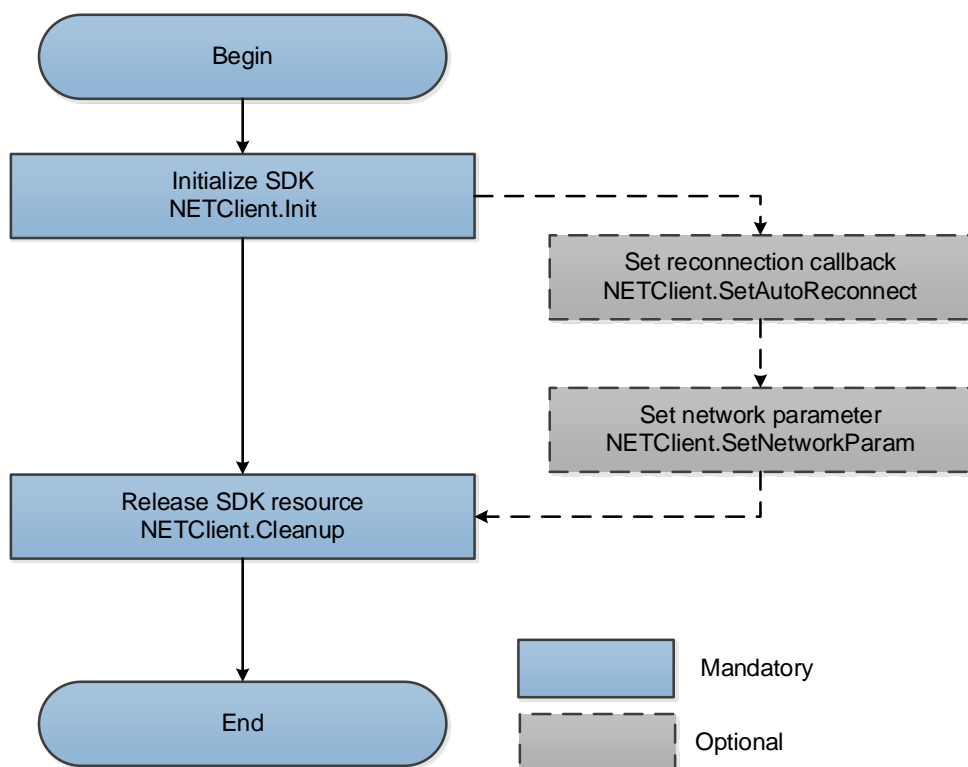
2.1.2 Interface Overview

Table 2-1 Interfaces of NetSDK initialization

Interface	Implication
NETClient.Init	NetSDK initialization
NETClient.SetAutoReconnect	Setting of reconnection after disconnection
NETClient.SetNetworkParam	Setting of network environment
NETClient.Cleanup	NetSDK cleaning up

2.1.3 Process

Figure 2-1 Initialization process



Process Description

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

Notes for Process

- Call **NETClient.Init** and **NETClient.Cleanup** in pairs. It supports multiple calling but it is suggested to call the pair for only one time overall.
- Initialization: Calling **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 off. NetSDK will keep logging until succeeded. Only the real-time monitoring, playback, smart event subscription and alarm subscription modules will be resumed after the connection is back.

2.1.4 Example Code

```
// Delegate a static callback (ordinary delegates may be released before callback)
private static fDisconnectCallBack m_DisConnectCallBack;    //Disconnect the callback
private static fHaveReConnectCallBack m_ReConnectCallBack; //Disconnect the callback

//Achieve to delegate
m_DisConnectCallBack = new fDisconnectCallBack(DisconnectCallBack);
m_ReConnectCallBack = new fHaveReConnectCallBack(ReConnectCallBack);

// Initialize NetSDK and disconnect callback during initialization
bool result = NETClient.Init(m_DisConnectCallBack, IntPtr.Zero, null);
if (!result)
{
    MessageBox.Show(NETClient.GetLastError());// Display wrong message
    return;
}

//Set auto reconnecting after disconnection
NETClient.SetAutoReconnect(m_ReConnectCallBack, IntPtr.Zero);
```

```
//Set network parameters
NET_PARAM param = new NET_PARAM()
{
    nWaittime = 10000, // Timeout of waiting (ms)
    nConnectTime = 5000, // Timeout of connection (ms)
};
NETClient.SetNetworkParam(param);

// Clean up the NetSDK resource
NETClient.Cleanup();
```

2.2 Device Login and Logout

2.2.1 Introduction

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

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

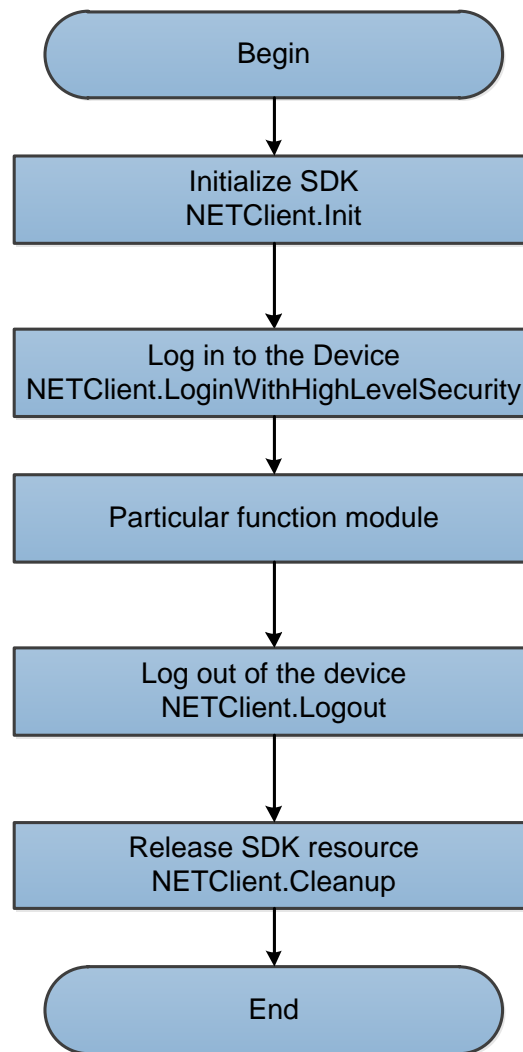
2.2.2 Interface Overview

Table 2-2 Interfaces of login and logout

Interface	Implication
NETClient.LoginWithHighLevelSecurity	Login
NETClient.Logout	Logout

2.2.3 Process

Figure 2-2 Log process



Process Description

- 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 NetSDK functions, call **NETClient.Cleanup** to release NetSDK resource.

Notes for Process

- Login handle: When the login is successful, the returned value is not 0 (even the handle is smaller than 0, the login is also successful). One device can login multiple times with different handle at each login. If there is not special function module, it is suggested to login only one time. The login handle can be repeatedly used on other function modules.
- Duplicate handles: It is normal that the login handle is the same as the existed handle. For example, log in to device A and get handle loginIDA. However, if you log out of loginIDA and

then log in, you may get LoginIDA again. But the duplicate handles do not occur throughout the lifetime of the handle.

- Logout: The interface will release the opened functions internally, but it is not suggested to rely on the cleaning up function of logout. For example, if you opened the monitoring function, you should call the interface that stops the monitoring function when it is no longer required.
- Use login and logout in pairs: The login consumes some memory and socket information and release sources once logout.
- Login failure: It is suggested to check the failure through **NETClient.GetLastError**.
- After reconnection, the original login ID will be invalid. After the device is reconnected, the login ID will take effect again.

2.2.4 Example Code

```
// Login 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;
}
```

2.3 Real-time Monitoring

2.3.1 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.

NetSDK can get the main stream and sub stream from the device once it logged.

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

2.3.2 Interface Overview

Table 2-3 Interfaces of real-time monitoring

Interface	Implication
NETClient.RealPlay	Start real-time monitoring extension interface.
NETClient.StopRealPlay	Stop real-time monitoring extension interface.
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	Set real-time monitoring data callback function extension interface.

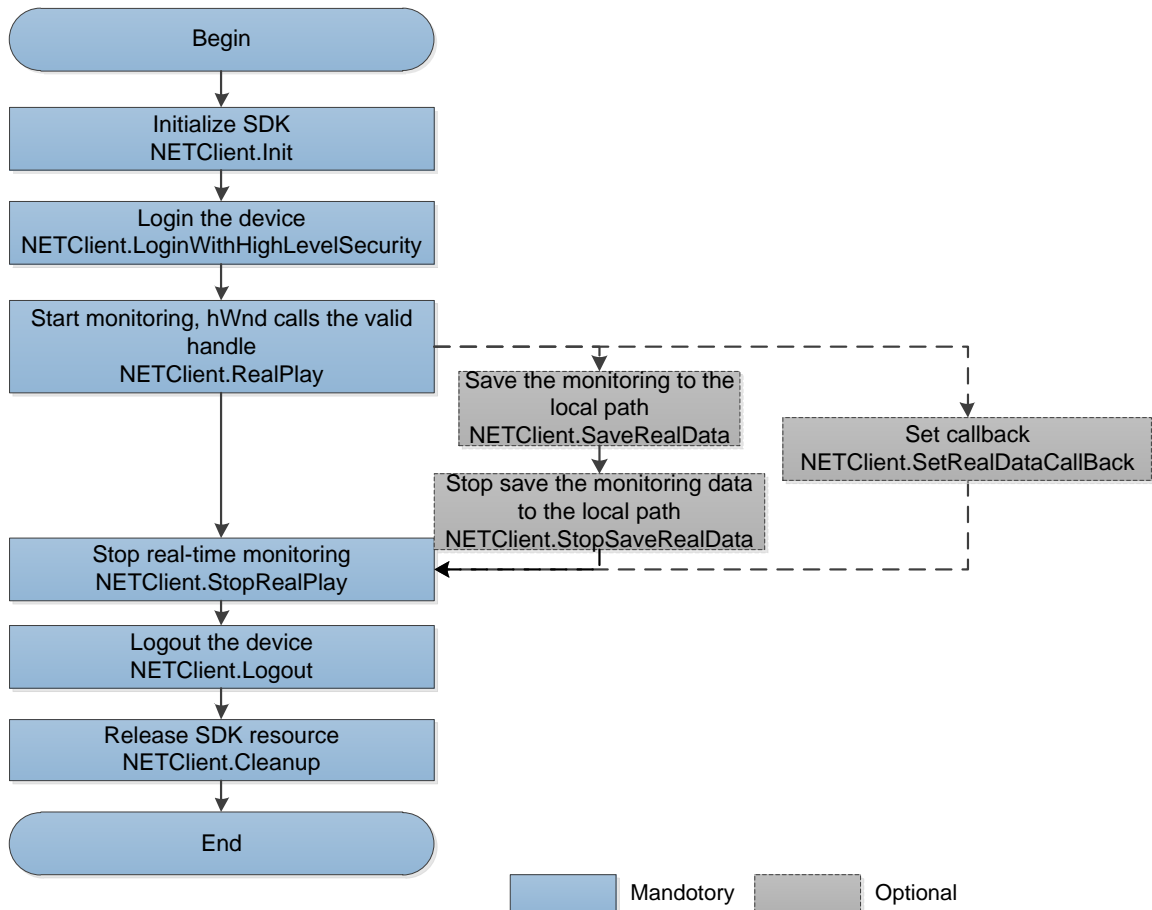
2.3.3 Process

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

2.3.3.1 NetSDK Decoding Library

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

Figure 2-3 Process of playing by NetSDK decoding library



Process Description

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- 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 the local video file.
- Step 6 (Optional) If you call **NETClient.SetRealDataCallBack**, you can choose to save or transmit the video stream. The saved video stream is same as the stream saved through step 4 and step 5.
- Step 7 After using the real-time function, call **NETClient.StopRealPlay** to stop real-time monitoring.
- 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 NetSDK resource.

Notes for Process

- NetSDK decoding play only supports Windows system. You need to call the decoding after getting the stream in other systems.

- In step 6, make sure that the parameter `hWnd` of `NETClient.RealPlay` introduces the valid window handle, when you want to perform the corresponding operations in the callback function of `NETClient.SetRealDataCallBack`,
- Timeout: The request on applying for monitoring resources should have made some agreement with the device before requiring the monitoring data. There are some timeout settings (see "NET_PARAM structure"), and the field about monitoring is `nGetConnInfoTime`. If there is timeout due to the reasons such as bad network connection, you can modify the value of `nGetConnInfoTime` bigger.

The example code is as follows. Call it for only one time after having called **NETClient.Init**.

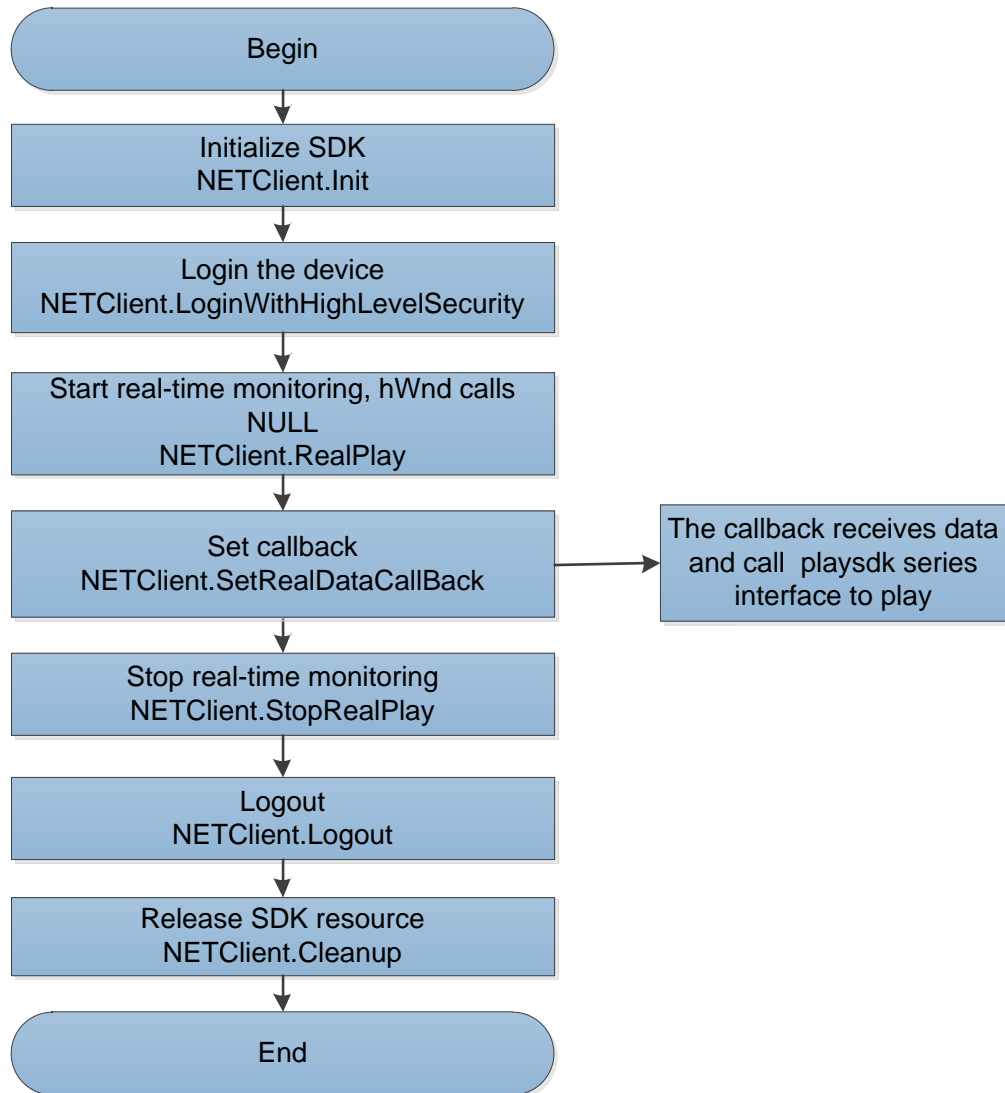
```
NET_PARAM param = new NET_PARAM()
{
    nGetConnInfoTime = 5000, //Timeout of getting connection information (ms)
};
NETClient.SetNetworkParam(param);
```

- Failed to repeat opening: For some models, the same channel cannot be opened for multiple times during a login. If you are trying to open it repeatedly, you will success in the first try but get failed afterwards. In this case, you can try the following:
 - ◇ Close the opened channel. For example, if you already opened the main stream video on the channel 1 and still want to open the sub stream video on the same channel, you can close the main stream first and then open the sub stream.
 - ◇ Login twice to obtain two login handles to deal with the main stream and sub stream respectively.
- Calling succeeded but no image: NetSDK decoding needs to use `dhplay.dll`. It is suggested to check if `dhplay.dll` and its auxiliary library are missing under the running directory. See Table 1-1 and Table 1-2.

2.3.3.2 Call Third Party Play Library

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

Figure 2-4 Process of calling the third party play library



Process Description

- 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 the 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 function, call **NETClient.StopRealPlay** to stop real-time monitoring.
- Step 7 After using the function module, call **NETClient.Logout** to log out of the device.
- Step 8 After using all NetSDK functions, call **NETClient.Cleanup** to release NetSDK resource.

Notes for Process

- Stream format: It is recommended to use PlaySDK for decoding.
- Lag image

- ◇ When using PlaySDK for decoding, there is a default channel buffer size (the PLAY_OpenStream interface in playsdk) for decoding. If the stream resolution value is big, it is recommended to modify the parameter value smaller such as 3 M.
- ◇ NetSDK callbacks can only moves into the next process after returning from you. It is not recommended for you to consume time for the unnecessary operations; otherwise the performance could be affected.

2.3.4 Example Code

2.3.4.1 NetSDK Decoding Play

```
// Take opening the main stream monitoring of channel 1 as an example. The parameter hWnd is a window handle.
IntPtr m_RealPlayID = NETClient.RealPlay(m_LoginID, 0, hWnd, EM_RealPlayType.Realplay);
if (IntPtr.Zero == m_RealPlayID)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

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

2.3.4.2 Call Play Library

```
// Take opening the main stream monitoring of channel 1 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;
}

//Set call function of real-time monitoring
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 lRealHandle, uint dwDataType, IntPtr pBuffer, uint dwBufSize, IntPtr
param, IntPtr dwUser)
{
    //Call PlaySDK interface to get the stream data from the device. See NetSDK monitoring demo source data
for more details.
    //Do some operations such as save data, send data, or change data to YUV.
    EM_REALDATA_FLAG type = (EM_REALDATA_FLAG)dwDataType;
    switch (type)
    {
        case EM_REALDATA_FLAG.RAW_DATA:
            //Process operation
            break;
    }
}

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

```

2.4 Record Playback

2.4.1 Introduction

Record playback function plays the videos of a particular period in some channels to find the target videos for check.

The playback includes the following functions: Start playback, pause Playback, resume playback, and stop playback.

2.4.2 Interface Overview

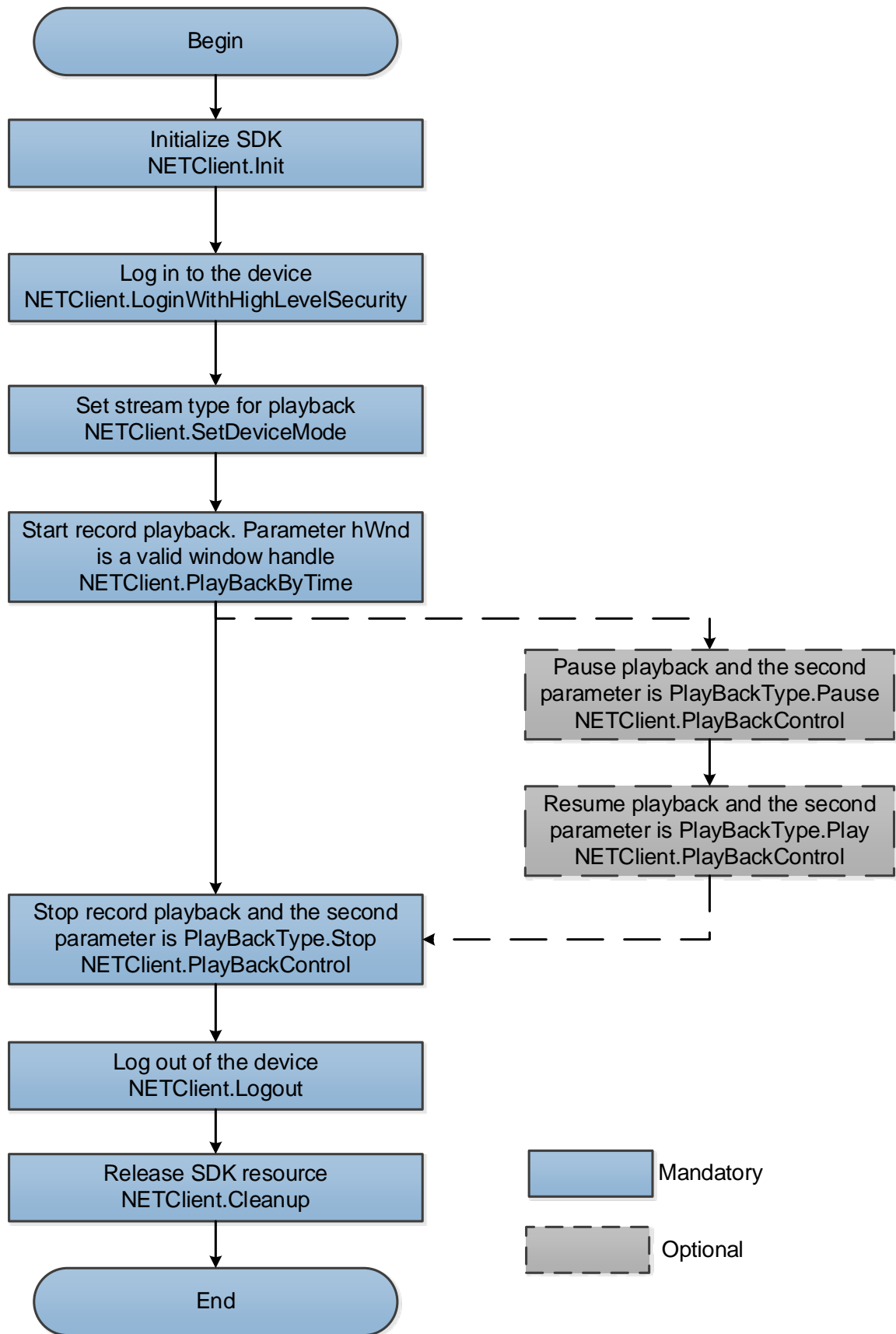
Table 2-4 Interfaces of record playback

Interface	Implication
NETClient.SetDeviceMode	Set the work mode such as voice talk, playback and authority.
NETClient.PlayBackByTime	Playback by time.
NETClient.PlayBackControl	Stop, fast forward, slow forward, pause, or resume record playback.
NETClient.GetPlayBackOsdTime	Get the playback OSD time.

2.4.3 Process

After NetSDK initialization, you need to enter channel number, start time, stop time, and valid window handle to realize the playback of the required record.

Figure 2-5 Process of record playback



Process Description

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

- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 (Optional) Call **NETClient.SetDeviceMode** twice and set the stream type parameter `emType` as `EM_USEDEV_MODE.RECORD_STREAM_TYPE`, and parameter `pValue` as `EM_STREAM_TYPE.MAIN`.
- Step 4 Call **NETClient.PlayBackByTime** to start playback. The parameter `hWnd` is a valid window handle value.
- Step 5 (Optional) Call **NETClient.PlayBackControl**. The playback will pause when the second parameter is `PlayBackType.Pause`.
- Step 6 (Optional) Call **NETClient.PlayBackControl**. The playback will resume when the second parameter is `PlayBackType.Play`.
- Step 7 After playback, call **NETClient.PlayBackControl**. The playback will stop when the second parameter is `PlayBackType.Stop`.
- 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 NetSDK resource.

2.4.4 Example Code

```
// Set the stream type of playback. Here is set as the main stream.
EM_STREAM_TYPE streamType = EM_STREAM_TYPE.MAIN;
IntPtr pStream = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(int)));
Marshal.StructureToPtr((int)streamType, pStream, true);
NETClient.SetDeviceMode(m_LoginID, EM_USEDEV_MODE.RECORD_STREAM_TYPE, pStream);

//Start record playback
NET_IN_PLAY_BACK_BY_TIME_INFO stuInfo = new NET_IN_PLAY_BACK_BY_TIME_INFO();
NET_OUT_PLAY_BACK_BY_TIME_INFO stuOut = new NET_OUT_PLAY_BACK_BY_TIME_INFO();
stuInfo.stStartTime = NET_TIME.FromDateTime(startTime);
stuInfo.stStopTime = NET_TIME.FromDateTime(endTime);
stuInfo.hWnd = playback_pictureBox.Handle;
stuInfo.cbDownloadPos = null;
stuInfo.dwPosUser = IntPtr.Zero;
stuInfo.fDownloadDataCallBack = null;
stuInfo.dwDataUser = IntPtr.Zero;
stuInfo.nPlayDirection = 0;
stuInfo.nWaittime = m_WaitTime;

m_PlayBackID = NETClient.PlayBackByTime(m_LoginID, nChannelID, stuInfo, ref stuOut);
if (IntPtr.Zero == m_PlayBackID)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}
```

```

// Pause playback
bool ret = NETClient.PlayBackControl(m_PlayBackID, PlayBackType.Pause);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}

// Resume playback
bool ret = NETClient.PlayBackControl(m_PlayBackID, PlayBackType.Play);
if (!ret)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}

// Stop playback
if (IntPtr.Zero != m_PlayBackID)
{
    NETClient.PlayBackControl(m_PlayBackID, PlayBackType.Stop);
    m_PlayBackID = IntPtr.Zero;
}

```

2.5 Record Download

2.5.1 Introduction

Video surveillance system widely applies to safe city, airport, metro, bank and factory. When and event occurs, you need to download the video records and report to the leaders, public security bureau, or mass media. Therefore, record download is an important function.

The record download function helps you obtain the records saved on the device through NetSDK and save into the local. It allows you to download from the selected channels and export to the local disk or external USB flash drive. The downloaded records are in the private format of Dahua. They can only be played with Dahua player or integrated Dahua playsdk.

2.5.2 Interface Overview

Table 2-5 Interfaces of record download

Interface	Implication
NETClient.SetDeviceMode	Set the work modes such as voice talk, playback, and authority.
NETClient.QueryRecordFile	Query all the record files within a period.
NETClient.FindFile	Open the record query handle.
NETClient.FindNextFile	Find the record file.
NETClient.FindClose	Close the record query handle.
NETClient.DownloadByRecordFile	Download the record by file.
NETClient.DownloadByTime	Download the record by time.
NETClient.GetDownloadPos	Get the record download progress.
NETClient.StopDownload	Stop the record download.

2.5.3 Process

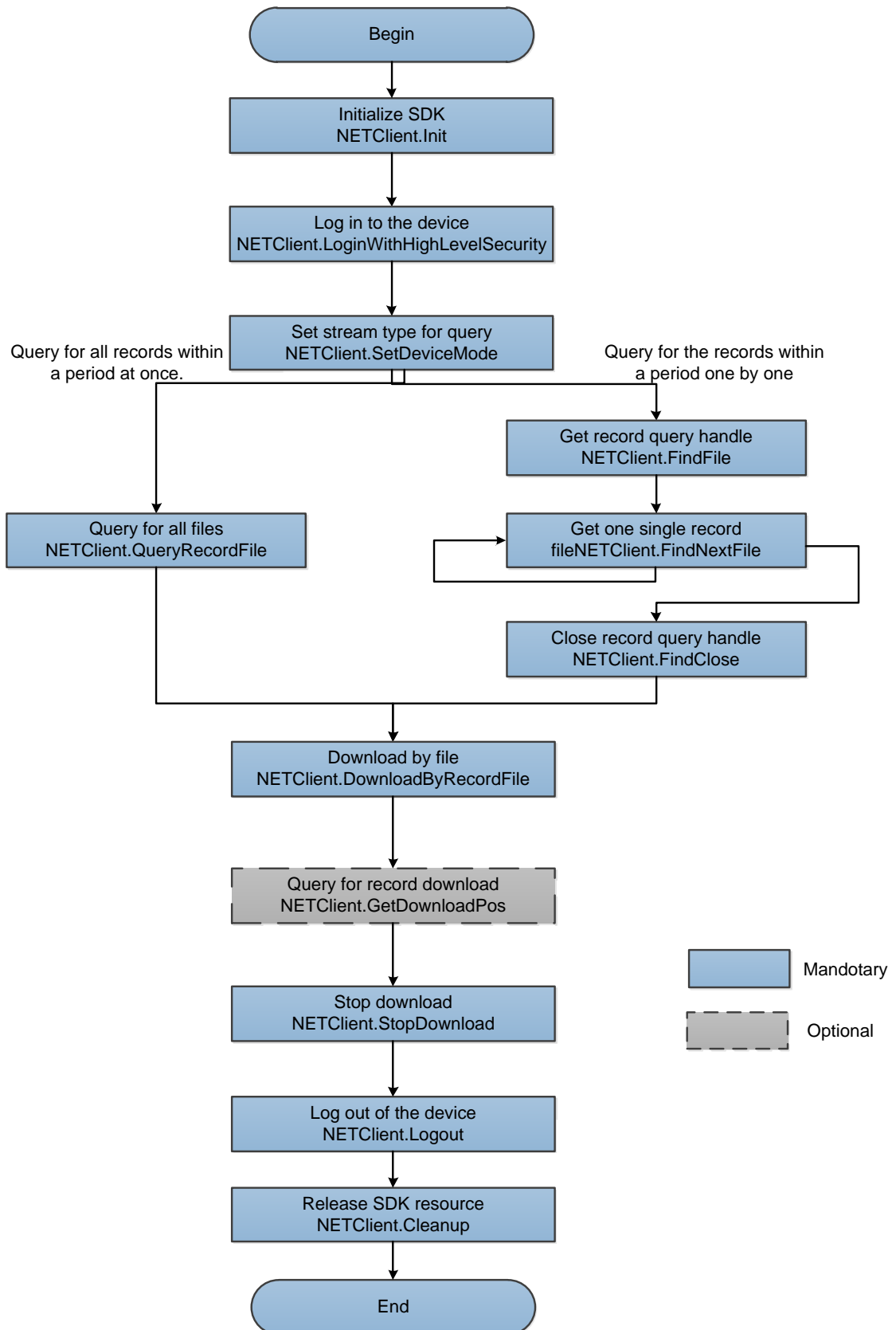
The record download is consisted of download by file and download by time.

2.5.3.1 Downloading by File

You need to import the record file information to be downloaded. NetSDK can download the specified record file and save to the required place.

You can also provide a callback pointer to NetSDK that calls back the specified record file to you for treatment.

Figure 2-6 Process of download by file



Process Description

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

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

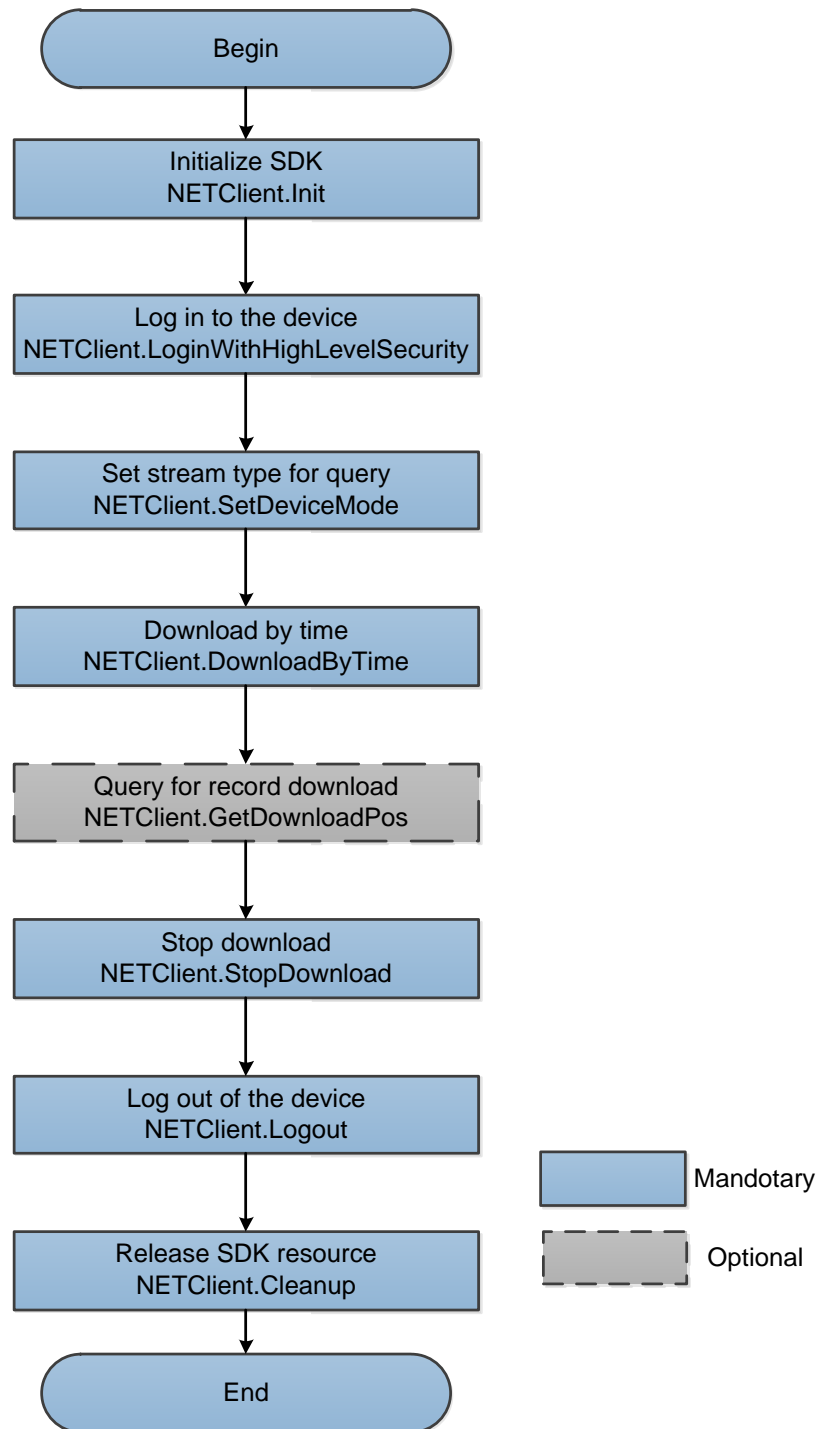
- Step 3 Call **NETClient.SetDeviceMode** twice and set the stream type parameter `emType` as `EM_USEDEV_MODE.RECORD_STREAM_TYPE`, and parameter `pValue` as `EM_STREAM_TYPE.MAIN`.
- Step 4 Query the record files by one of the following two ways:
- Call **NETClient.FindFile** to obtain the record query handle, and then call **NETClient.FindNextFile** several times to obtain the record file information and then call **NETClient.FindClose** to close the record query handle at last.
 - Call **NETClient.QueryRecordFile** to obtain all the record files information for a period one time.
- Step 5 After getting the record file information, call **NETClient.DownloadByRecordFile** to start downloading record files. You can decide whether to use `cbDownloadPos` to query the record downloading progress in real time, if not, set it as `NULL`.
- Step 6 (Optional) During downloading, call **NETClient.GetDownloadPos** to query the record downloading progress or use `cbDownloadPos` mentioned in step 5 to obtain the real-time download progress.
- Step 7 Call **NETClient.StopDownload** to stop download.
- 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 NetSDK resource.

2.5.3.2 Downloading by Time

You can import the start time and end time of download. NetSDK can download the specified record file and save to the required place.

You can also provide a callback pointer to NetSDK which calls back the specified record file to you for treatment.

Figure 2-7 Process of download by time



Process Description

- Step 1 Call **NETClient_Init** to initialize NetSDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.SetDeviceMode** to set the parameter emType as DH_RECORD_STREAM_TYPE, and pValue as EM_STREAM_TYPE.MAIN.
- Step 4 Call **NETClient.DownloadByTime** to start downloading by time. Either sSavedFileName or cbTimeDownloadPos is valid. You can decide whether to use cbDownloadPos to query the record downloading progress in real time, if not, set it as NULL.

- Step 5 (Optional) During downloading, call **NETClient.GetDownloadPos** to query the record downloading progress or use cbDownloadPos mentioned in step 4 to obtain the real-time download progress.
- Step 6 Call **NETClient.StopDownload** to stop download. You can close the download process after it is completed or it is just partially completed.
- Step 7 After using the function module, call **NETClient.Logout** to log out of the device.
- Step 8 After using all NetSDK functions, call **NETClient.Cleanup** to release NetSDK resource.

2.5.4 Example Code

2.5.4.1 Downloading by File

```
// Set the stream type of playback. Here is to set as the main stream.
EM_STREAM_TYPE streamType = EM_STREAM_TYPE.MAIN;
IntPtr pStream = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(int)));
Marshal.StructureToPtr((int)streamType, pStream, true);
NETClient.SetDeviceMode(m_LoginID, EM_USEDEV_MODE.RECORD_STREAM_TYPE, pStream);

// There are two ways of record query: 1. Get all the record files of a certain period one time; 2. Get all the
record files of a certain period by several times.

// Way 1: Get all the record files of a certain period one time
int nChannelID = 0; // Channel ID
int fileCount = 0;
NET_RECORDFILE_INFO[] recordFileArray = new NET_RECORDFILE_INFO[5000];
bool ret = NETClient.QueryRecordFile(m_LoginID, nChannelID, EM_QUERY_RECORD_TYPE.ALL, startTime,
endTime, null, ref recordFileArray, ref fileCount, m_WaitTime, false);
if (false == ret)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

// Way 2: Get all the record files of a certain period by several times
NET_IN_MEDIA_QUERY_FILE stuCondition = new NET_IN_MEDIA_QUERY_FILE();
stuCondition = (NET_IN_MEDIA_QUERY_FILE)obj;
stuCondition.dwSize = (uint)Marshal.SizeOf(stuCondition);
stuCondition.stuCardInfo.dwSize = (uint)Marshal.SizeOf(stuCondition.stuCardInfo);
stuCondition.nChannelID = 0; // -1 means all channels;
stuCondition.nMediaType = 0; // File type, 0: Query any type, 1: Query jpg picture
stuCondition.emFalgLists[0] = EM_RECORD_SNAP_FLAG_TYPE.TIMING;
stuCondition.nFalgCount = 1;
```

```

stuCondition.stuStartTime = NET_TIME.FromDateTime(DateTime.Now.AddDays(-0.99));//
stuCondition.stuEndTime = NET_TIME.FromDateTime(DateTime.Now);
// Step 1: Get record query handle
IntPtr fileHandle = IntPtr.Zero;
IntPtr pQueryCondition = IntPtr.Zero;
try
{
    pQueryCondition = Marshal.AllocHGlobal(Marshal.SizeOf(stuCondition));
    Marshal.StructureToPtr(stuCondition, pQueryCondition, true);
    fileHandle = NETClient.FindFile(m_LoginHandle, EM_FILE_QUERY_TYPE.FILE, (object)stuCondition,
stuCondition.GetType(), 5000);
    if (fileHandle == IntPtr.Zero)
    {
        Console.WriteLine(NETClient.GetLastError());
    }
    return fileHandle;
}
finally
{
    Marshal.FreeHGlobal(pQueryCondition);
}
// Step 2: Get single record file information
int nFilecount = 3;// max file count in one query
int nRetCount = 0;// return count in one query
NET_OUT_MEDIA_QUERY_FILE[] MediaFileInfos = new NET_OUT_MEDIA_QUERY_FILE[nFilecount];
List<object> infoList = new List<object>();
for (int i = 0; i < nFilecount; i++)
{
    MediaFileInfos[i].dwSize = (uint)Marshal.SizeOf(MediaFileInfos[i]);
    infoList.Add(MediaFileInfos[i]);
}

nRetCount = NETClient.FindNextFile(findHandle, nFilecount, infoList, MediaFileInfos[0].GetType(), 3000);
if (nRetCount < 0)
{
    Console.WriteLine("find failed,last error:" + NETClient.GetLastError());
    return;
}
else if (nRetCount == 0)

```

```

{
    Console.WriteLine("find file 0");
    return;
}

// Step 3: Close record query handle
if(IntPtr.Zero != findHandle)
{
    bool ret = NETClient.FindClose(findHandle);
    if (!ret)
    {
        MessageBox.Show(NETClient.GetLastError());
    }
    findHandle = IntPtr.Zero;
}

// Callback declaration
// Playback/download progress callback
// It is not recommended to call NetSDK interface in this callback
// dwDownloadSize: "-1" that represents current playback/download has completed. "-2" represents
writing file failed. Other values represent valid data.
// Set this callback through NETClient.DownloadByRecordFile. When NetSDK receives playback/download
data, it will call this function.
private static fDownloadPosCallBack m_DownloadPosHandle;
m_DownloadPosHandle = new fDownloadPosCallBack(DownloadPosCallBack);

// Record download
// Start record download
// Either sSavedFileName or cbTimeDownloadPos is valid.
// In the application, save to sSavedFileName or callback to handle the data.
// stuFile is the file information queried before
NET_RECORDFILE_INFO stuRecordFile = new NET_RECORDFILE_INFO();
stuRecordFile.ch = (uint)stuFile.nChannelID;
stuRecordFile.bHint = stuFile.byPartition;
stuRecordFile.nRecordFileType = 255;
stuRecordFile.starttime = stuFile.stuStartTime;
stuRecordFile.endtime = stuFile.stuEndTime;
stuRecordFile.driveno = stuFile.nDriveNo;
stuRecordFile.size = stuFile.nFileSize/1024;

```

```

stuRecordFile.filename = stuFile.szFilePath;
stuRecordFile.startcluster = stuFile.nCluster;
IntPtr m_DownloadHandle = NETClient.DownloadByRecordFile(m_LoginHandle, ref stuRecordFile,
storeFile, m_DownloadPosHandle, IntPtr.Zero);
    if (m_DownloadHandle == IntPtr.Zero)
    {
        MessageBox.Show(NETClient.GetLastError());
    }

// Close download which can be called after download or in process.
    if (IntPtr.Zero != m_DownloadHandle)
    {
        bool ret = NETClient.StopDownload(m_DownloadHandle);
        if (!ret)
        {
            MessageBox.Show(NETClient.GetLastError());
        }
        m_DownloadHandle = IntPtr.Zero;
    }

// Callback definition
private void DownloadPosCallBack(IntPtr IPlayHandle, uint dwTotalSize, uint dwDownloadSize, IntPtr dwUser)
{
    if ((int)dwDownloadSize == -1)
    {
        Console.WriteLine("download over");
    }
    else if ((int)dwDownloadSize == -2)
    {
        Console.WriteLine("write file error");
    }
    else
    {
        value = (int)(dwDownloadSize * 100 / dwTotalSize);
        Console.WriteLine("Download" + value);
    }
}

```


2.5.4.2 Downloading by Time

```
// Callback declaration

    // Playback/download progress callback
    // It is not recommended to call NetSDK interface in this callback
    // dwDownloadSize: "-1" represents current playback/download has completed. "-2" represents writing
file failed. Other values represent valid data.
    // Set this callback through NETClient. DownloadByTime. When NetSDK receives playback/download data,
it will call this function.
private static fTimeDownloadPosCallBack m_DownloadPosCallBack;
m_DownloadPosCallBack = new fTimeDownloadPosCallBack(DownloadPosCallBack);

// Set the stream type of playback. Here is to set as the main stream.
EM_STREAM_TYPE streamType = EM_STREAM_TYPE.MAIN;
IntPtr pStream = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(int)));
Marshal.StructureToPtr((int)streamType, pStream, true);
NETClient.SetDeviceMode(m_LoginID, EM_USEDEV_MODE.RECORD_STREAM_TYPE, pStream);

// Download record file
m_DownloadID = NETClient.DownloadByTime(m_LoginID, nChannelId, EM_QUERY_RECORD_TYPE.ALL,
startTime, endTime, sSavedFileName, m_DownloadPosCallBack, IntPtr.Zero, null, IntPtr.Zero, IntPtr.Zero);
if (IntPtr.Zero == m_DownloadID)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

// Close download which can be called after download or in process.
if (IntPtr.Zero != m_DownloadHandle)
{
    bool ret = NETClient.StopDownload(m_DownloadHandle);
    if (!ret)
    {
        MessageBox.Show(NETClient.GetLastError());
    }
    m_DownloadHandle = IntPtr.Zero;
}

// Callback definition
```

```

private void DownloadPosCallBack(IntPtr IPlayHandle, uint dwTotalSize, uint dwDownloadSize, int index,
NET_RECORDFILE_INFO recordfileinfo, IntPtr dwUser)
{
    if (IPlayHandle == m_DownloadID)
    {
        int value = 0;
        if (-1 == (int)dwDownloadSize)
        {
            value = DOWNLOAD_END;
        }
        else if (-2 == (int)dwDownloadSize)
        {
            value = DOWNLOAD_FAILED;
        }
        else
        {
            value = (int)(dwDownloadSize * 100 / dwTotalSize);
            Console.WriteLine("Download" + value);
        }
    }
}

```

2.6 PTZ Control

2.6.1 Introduction

PTZ is a mechanical platform that carries the device and the protective enclosure and performs remote control in all directions.

PTZ is consisted of two motors that can perform horizontal and vertical movement to provide the all-around vision.

This section provides guidance to you about how to control directions (there are eight directions: upper, lower, left, right, upper left, upper right, bottom left, and bottom right), focus, zoom, iris, fast positioning, and 3-dimensional positioning through NetSDK.

2.6.2 Interface Overview

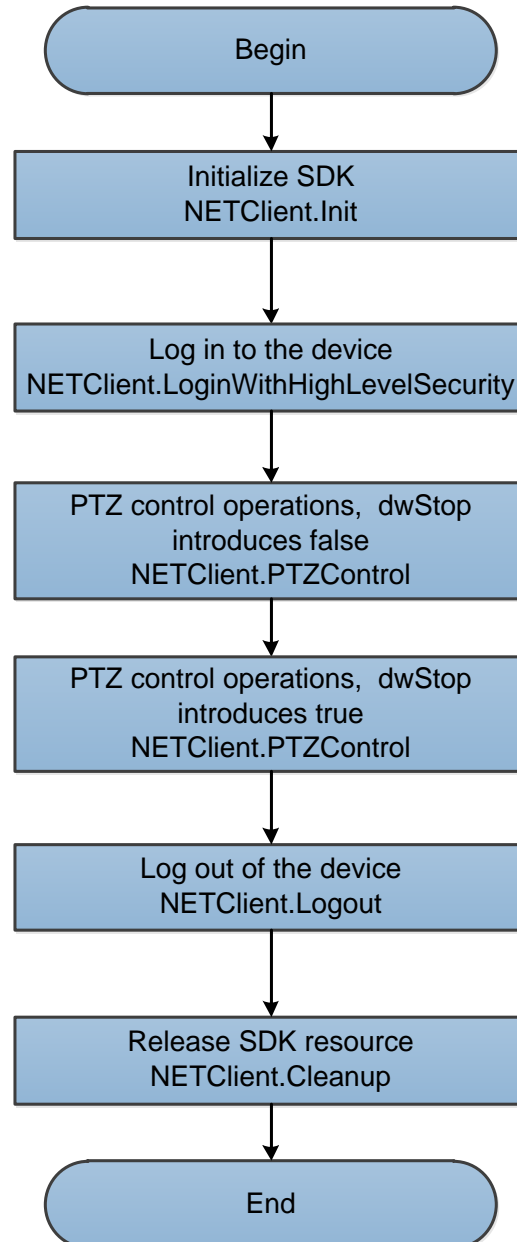
Table 2-6 Interface of PTZ control

Interface	Implication
NETClient.PTZControl	PTZ control extension interface

2.6.3 Process

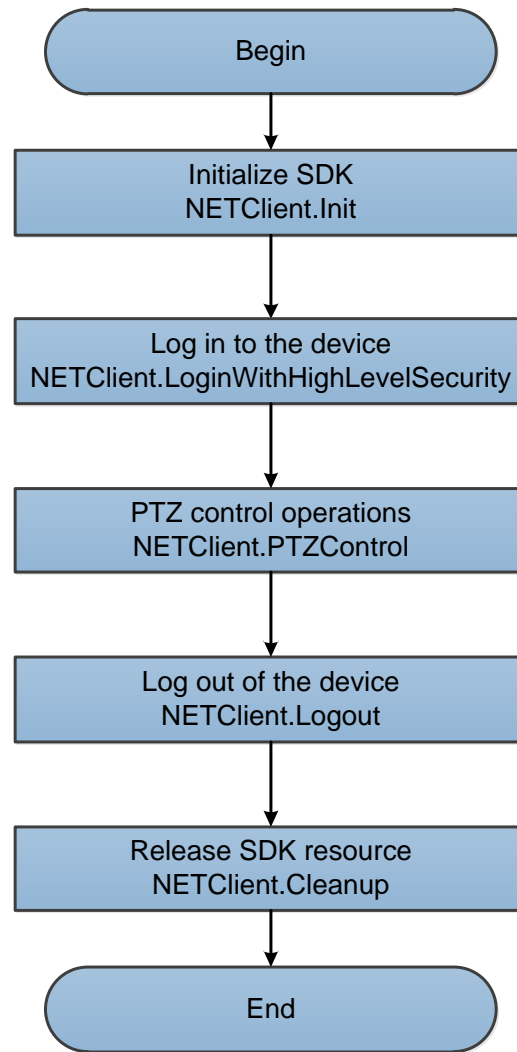
Direction control, focus, zoom, and aperture are the continuous operations. NetSDK provides start and stop interfaces to you for timing control.

Figure 2-8 Process of PTZ control (continuous)



Both fast positioning and 3-dimensional positioning belong to one-time action, which needs to call the PTZ control interface just one time.

Figure 2-9 Process of PTZ control (one-time)



Process Description

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.PTZControl** to operate the PTZ according to the situation. Different PTZ command might need different parameters, and part of commands need to call the corresponding stop command, such as moving left and moving right. For details, see "2.6.4 Example Code."
- Step 4 After using the function module, call **NETClient.Logout** to log out of the device.
- Step 5 After using all NetSDK functions, call **NETClient.Cleanup** to release NetSDK resource.

Notes for Process

- Fast positioning: For the SD, take the current monitoring image center as origin, and the valid range of horizontal and vertical coordinates is [-8191, 8191]. For example, if the horizontal coordinate is 2000 and the vertical is 2000, the SD moves toward upper right and gets a new origin, which means the coordinate specified every time is only relative to the current location.

- 3-dimensional positioning: For the SD, there is an initial position first. The horizontal coordinate is [0, 3600] and the vertical is [-1800, 1800]. The coordinate specified each time is the absolute coordinate and is irrelevant to the location of the SD image last time.
- For more example code, see the NetSDK package on the website.

2.6.4 Example Code

```
int hSpeedValue = 4; // Rotating speed in horizontal direction.
int vSpeedValue = 4; // Rotating speed in vertical direction.
int IParam3 = 0;
// Continuous operation: take moving upward as an example.
// Start moving upward.
bool bRet = NETClient.PTZControl(m_LoginID, nChannelId, EM_EXTPTZ_ControlType.UP_CONTROL, 0,
vSpeedValue, 0, false, IntPtr.Zero);
// Stop moving forward.
bRet = NETClient.PTZControl(m_LoginID, nChannelId, EM_EXTPTZ_ControlType.UP_CONTROL, 0, vSpeedValue,
0, true, IntPtr.Zero);

// One-time operation movement: Take fast positioning as an example.
int IParam1 = 2000; // Horizontal coordinate, valid range[-8191,8191]
int IParam2 = 2000; // Vertical coordinate, valid range [-8191,8191]
int IParam3 = 1; // Zoom, valid range (-16 ~ 16), 1 indicates rotating without zooming
bRet = NETClient.PTZControl((m_LoginID, nChannelId, EM_EXTPTZ_ControlType.FASTGOTO, IParam1, IParam2,
IParam3, false, IntPtr.Zero);
```

2.7 Voice Talk

2.7.1 Introduction

Voice talk realizes the voice interaction between the local platform and the environment where front-end devices are located.

This section introduces how to use NetSDK to realize the voice talk with the front-end devices.

2.7.2 Interface Overview

Table 2-7 Interfaces of voice talk

Interface	Implication
NETClient.StartTalk	Start voice talk.
NETClient.StopTalk	Stop voice talk.
NETClient.RecordStart	Start NETClient record (valid only in Windows system).
NETClient.RecordStop	Stop NETClient record (valid only in Windows system).
NETClient.TalkSendData	Send voice data to the device.
NETClient.AudioDec	Decode audio data (valid only in Windows system).

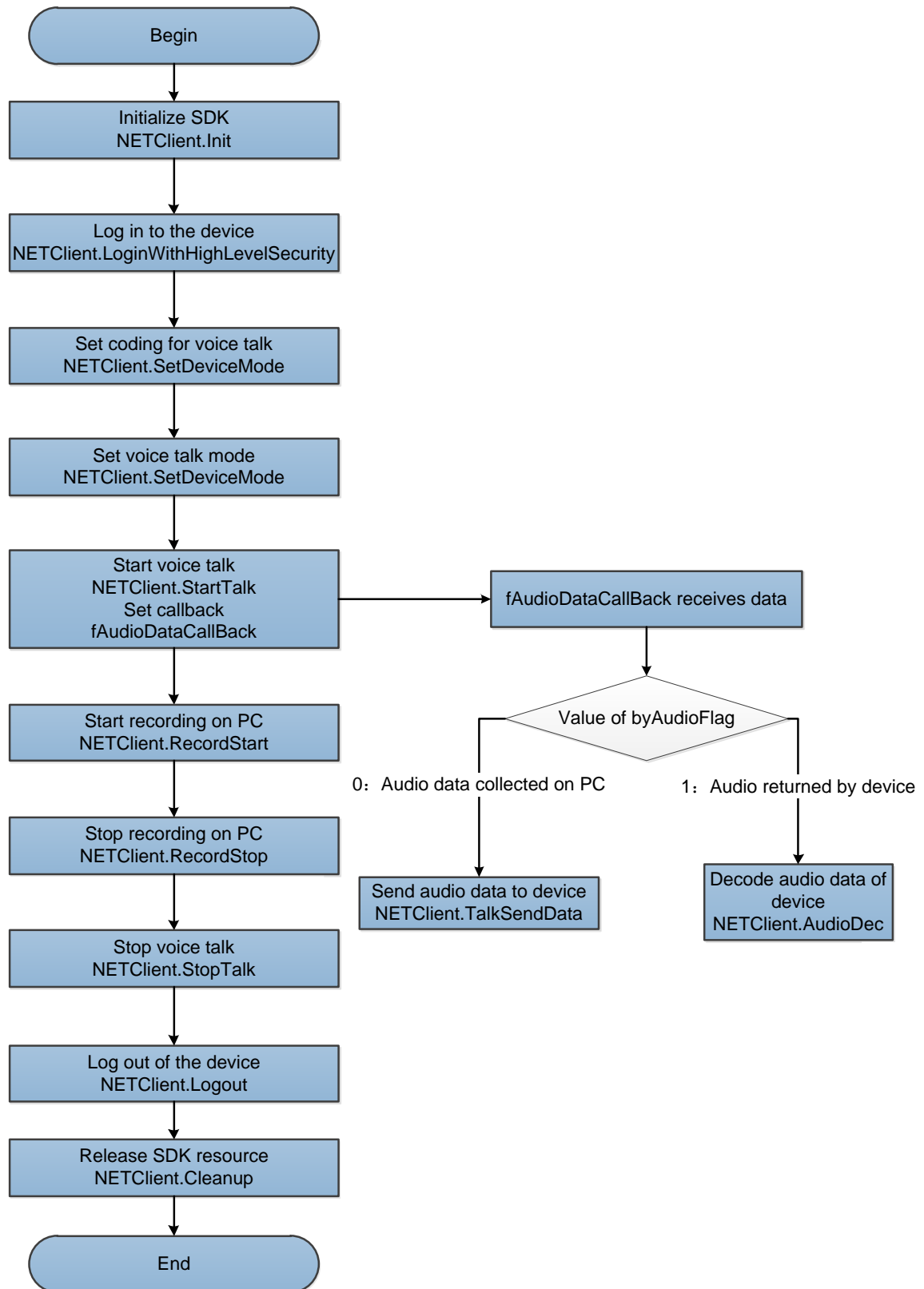
2.7.3 Process

When NetSDK has collected the audio data from the local audio card, or NetSDK has received the audio data from the front-end devices, NetSDK will call the callback of audio data.

You can call the NetSDK interface in the callback parameters to send the local audio data to the front-end devices, or call NetSDK interface to decode and playback the audio data received from the front-end devices.

This process is valid only in Windows system.

Figure 2-10 Process of voice talk



Process Description

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.SetDeviceMode** to set decoding information of voice talk. Set parameter emType as EM_USEDEV_MODE.TALK_ENCODE_TYPE.

- Step 4 Call **NETClient.SetDeviceMode** to set mode of voice talk. Set parameter emType as EM_USEDEV_MODE.TALK_SPEAK_PARAM.
- Step 5 Call **NETClient.SetDeviceMode** 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 of the PC end to the device.
- Step 6 Call **NETClient.RecordStart** to start recording at PC. After this interface is called, the voice talk callback in 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 After using the function module, call **NETClient.Logout** to log out of the device.
- Step 10 After using all NetSDK functions, call **NETClient.Cleanup** to release NetSDK resource.

Notes from Process

- Voice encoding format: The example uses the common PCM format. NetSDK supports accessing the voice encoding format supported by the device. For more details of the example code, see the NetSDK package on the website. If the default PCM can satisfy the requirement, it is not recommended to obtain the voice encoding format from the device.
- No sound at the device: The audio data needs to be collected by the device such as microphone. It is recommended to check if the microphone or other equivalent device is plugged in and if the NETClient.RecordStart succeeded in returning.

2.7.4 Example Code

```
// Set the voice talk encoding data, and take PCM as an example.
IntPtr talkEncodePointer = 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);
NETClient.SetDeviceMode(m_LoginID, EM_USEDEV_MODE.TALK_ENCODE_TYPE, talkEncodePointer);

// Set mode of voice talk
IntPtr talkSpeakPointer = IntPtr.Zero;
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);
NETClient.SetDeviceMode(m_LoginID, EM_USEDEV_MODE.TALK_SPEAK_PARAM, talkSpeakPointer);

// Set callback of voice talk
private static fAudioDataCallBack m_AudioDataCallBack;
m_AudioDataCallBack = new fAudioDataCallBack(AudioDataCallBack);

// Start voice talk
IntPtr m_TalkID = NETClient.StartTalk(m_LoginID, m_AudioDataCallBack, IntPtr.Zero);
if(IntPtr.Zero == m_TalkID)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

// Achieve callback function
private void AudioDataCallBack(IntPtr lTalkHandle, IntPtr pDataBuf, uint dwBufSize, byte byAudioFlag, IntPtr dwUser)
{
    if (lTalkHandle == m_TalkID)
    {
        if (SendAudio == byAudioFlag)
        {
            //send talk data
            NETClient.TalkSendData(lTalkHandle, 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);
            }
        }
    }
}

```

```

    }
}

// Start voice recording of PC
bool ret = NETClient.RecordStart(m_LoginID);
if(!ret)
{
    NETClient.StopTalk(m_TalkID);
    m_TalkID = IntPtr.Zero;
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

// Stop voice recording of PC
NETClient.RecordStop(m_LoginID);

// Stop voice talk
NETClient.StopTalk(m_TalkID);
m_TalkID = IntPtr.Zero;

DHDEV_TALKDECODE_INFO curTalkMode;
curTalkMode.encodeType = DH_TALK_PCM;
curTalkMode.nAudioBit = 16;
curTalkMode.dwSampleRate = 8000;
curTalkMode.nPacketPeriod = 25;
NETClient.SetDeviceMode(ILoginHandle, DH_TALK_ENCODE_TYPE, &curTalkMode);
// Start voice talk
ITalkHandle = NETClient.StartTalk(ILoginHandle, AudioDataCallBack, (DWORD)NULL);
if(0 != ITalkHandle)
{
    BOOL bSuccess = NETClient.RecordStart(ILoginHandle);
}

// Stop local recording
if (!NETClient.RecordStop(ILoginHandle))
{
    printf("CLIENT_RecordStop Failed!Last Error[%x]\n", CLIENT_GetLastError());
}

// Stop voice talk

```

```

if (0 != ITalkHandle)
{
    NETClient.StopTalk(ITalkHandle);
}

void CALLBACK AudioDataCallBack(LLONG ITalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE byAudioFlag,
DWORD dwUser)
{
    if(0 == byAudioFlag)
    {
        // Send the sound data checked by the PC to the device
        LONG lSendLen = NETClient.TalkSendData(ITalkHandle, pDataBuf, dwBufSize);
        if(lSendLen != (LONG)dwBufSize)
        {
            printf("NETClient.TalkSendData Failed!Last Error[%x]\n" , CLIENT_GetLastError());
        }
    }
    else if(1 == byAudioFlag)
    {
        // Send the voice data of the device to NetSDK encode and play.
        NETClient.AudioDec(pDataBuf, dwBufSize);
    }
}

```

2.8 Video Snapshot

2.8.1 Introduction

Video snapshot obtains the picture data of the current video. This section introduces the following two snapshot ways:

- Network snapshot: Call the NetSDK interface to send the capturing command to the device that captures the current image and send to NetSDK through network, and then NetSDK returns the image data to you.
- Local snapshot: When the monitoring is opened, you can save the monitoring data to the picture format that is the frame information that does not have an interaction with the device.

2.8.2 Interface Overview

Table 2-8 Interfaces of video snapshot

Interface	Implication
NETClient.SnapPictureToFile	Snapshot and directly returns the picture data to the user.
NETClient.CapturePicture	Local snapshot with the parameters that could be monitoring

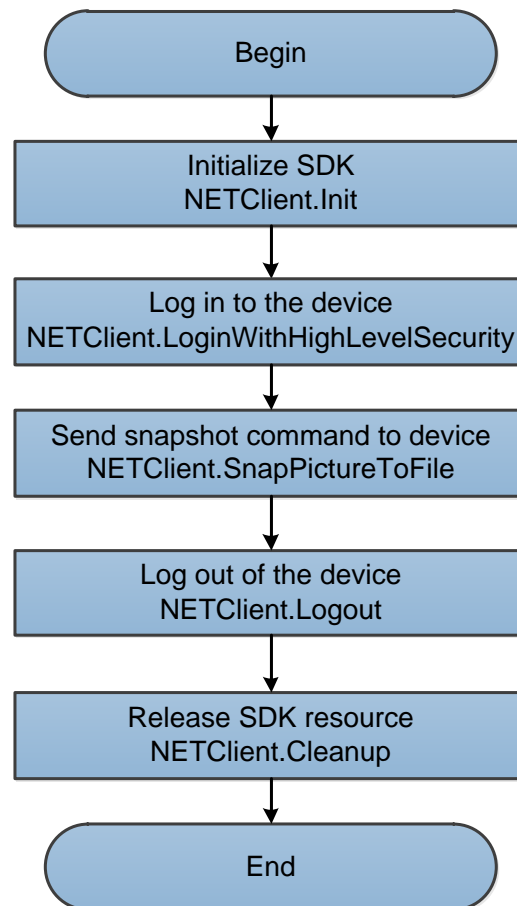
Interface	Implication
	handle or playback handle.

2.8.3 Process

Video snapshot is consisted of network snapshot and local snapshot.

2.8.3.1 Network Snapshot

Figure 2-11 Process of network snapshot



Process Description

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device after initialization.
- Step 3 Call **NETClient.SnapPictureToFile** to obtain the picture data.
- Step 4 After using the function module, call **NETClient.Logout** to log out of the device.
- Step 5 After using all NetSDK functions, call **NETClient.Cleanup** to release NetSDK resource.

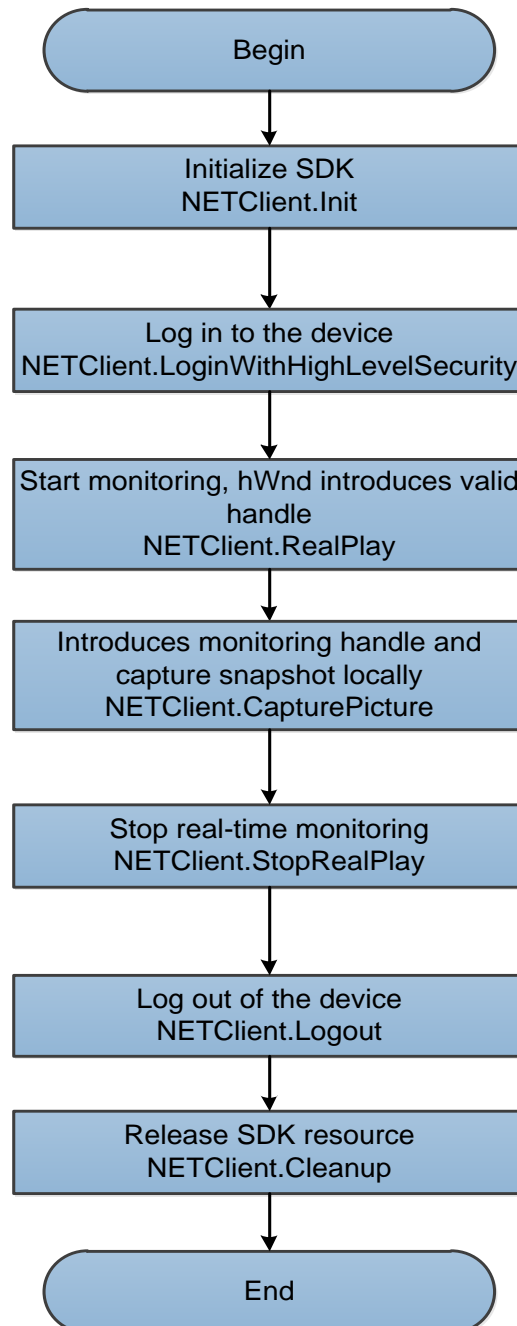
Notes for Process

- Multi-thread calling: Multi-thread calling is not supported for the functions within the same login session.

- Snapshot configuration: You can configure the items such as quality and definition for the snapshot. However, it is not recommended to modify if the default configurations are satisfactory.
- Picture save: The picture data is returned as memory and the interface supports saving the picture data as file (the precondition is that you have set the szFilePath field of NET_IN_SNAP_PIC_TO_FILE_PARAM).

2.8.3.2 Local Snapshot

Figure 2-12 Process of local snapshot



Process Description

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device after initialization.
- Step 3 Call **NETClient.RealPlay** to open monitoring and obtain the monitoring handle.
- Step 4 After the monitoring screen is displayed, call **NETClient.CapturePicture** to introduce the monitoring handle.
- Step 5 Call **NETClient.StopRealPlay** to close the real-time monitoring.
- Step 6 After using the function module, call **NETClient.Logout** to log out of the device.
- Step 7 After using all NetSDK functions, call **NETClient.Cleanup** to release NetSDK resource.

2.8.4 Example Code

```
// Network snapshot example
NET_SNAP_PARAMS stu_snap_param = new NET_SNAP_PARAMS()
{
    Channel = 0,
    Quality = 2,
    mode = 0
};

NET_IN_SNAP_PIC_TO_FILE_PARAM inParam = new NET_IN_SNAP_PIC_TO_FILE_PARAM()
{
    dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_SNAP_PIC_TO_FILE_PARAM)),
    stuParam = stu_snap_param,
    szFilePath = savepath + string.Format("\\{0}_{1}_image.jpg", nameprefix, i + 1)
};

NET_OUT_SNAP_PIC_TO_FILE_PARAM outParam = new NET_OUT_SNAP_PIC_TO_FILE_PARAM()
{
    dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_SNAP_PIC_TO_FILE_PARAM)),
    dwPicBufLen = 1024000,
    szPicBuf = Marshal.AllocHGlobal(1024000),
};

bool result = NETClient.SnapPictureToFile(m_LoginID, ref inParam, ref outParam, timeout);
if (!result)
{
    MessageBox.Show("Snap picture failed");
}

// Example of local capturing, and m_RealPlayID is the handle for opening monitoring.
string filePath = path + "\\" + "client" + m_SnapSerialNum.ToString() + ".jpg";
bool result = NETClient.CapturePicture(m_RealPlayID, filePath, EM_NET_CAPTURE_FORMATS.JPEG);
```

```

if (!result)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

```

2.9 Subscribing Intelligent Event

2.9.1 Introduction

Smart event upload is that smart devices analyze real-time streams. When finding the events which have been set in advance, the events will be sent to users. Smart events correspond to EM_EVENT_IVS_TYPE enumeration. For details, see the value of enumeration.

NetSDK connects to the device and subscribes to smart event function. When the device gets the smart events, they will be sent to NetSDK.

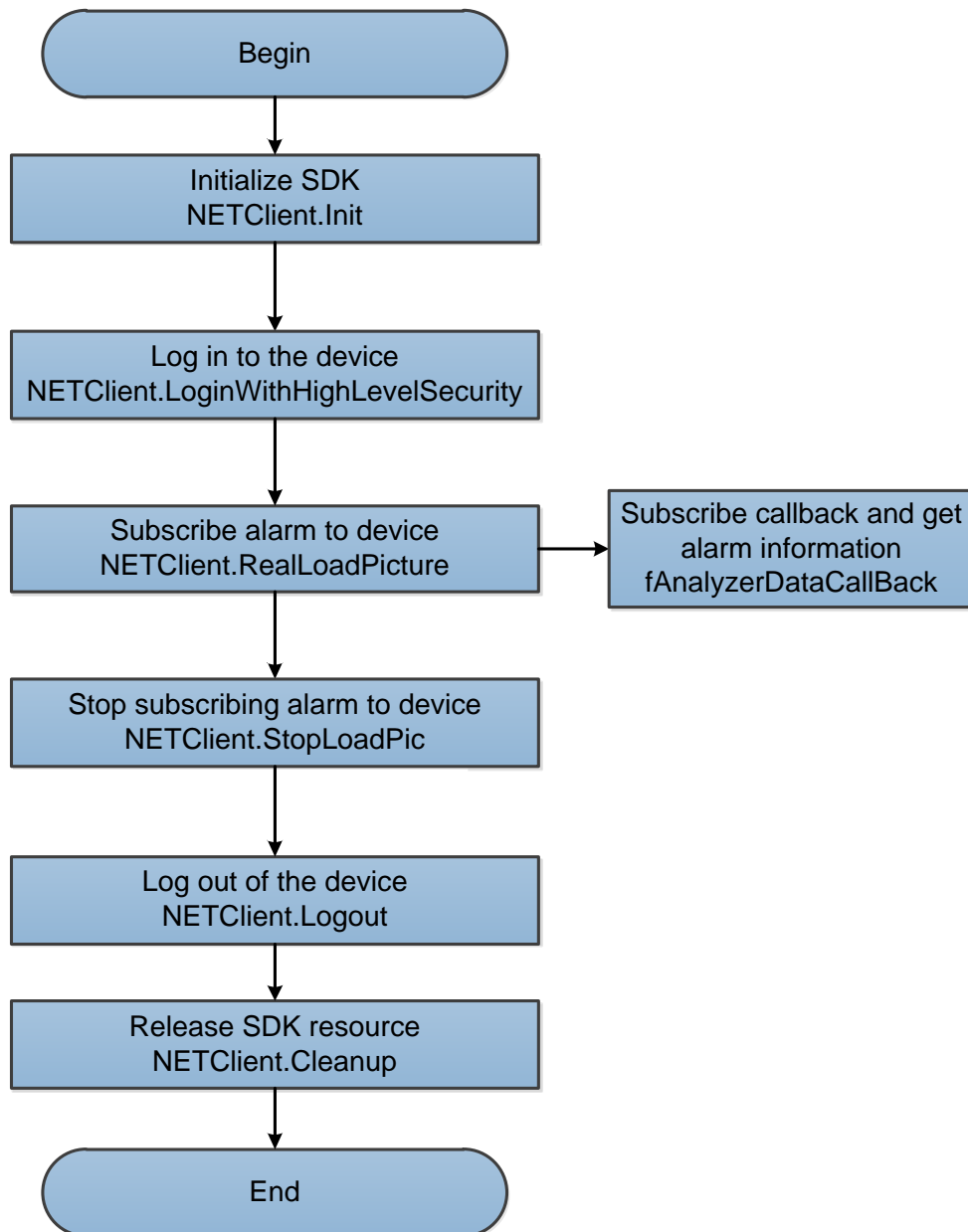
2.9.2 Interface Overview

Table 2-9 Interfaces of smart event upload

Interface	Implication
NETClient.RealLoadPicture	Subscribe to alarm events.
NETClient.StopLoadPic	Stop subscribing to smart events.

2.9.3 Process

Figure 2-13 Process of smart event upload



Process Description

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.RealLoadPicture** to subscribe intelligent events from devices.
- Step 4 After subscribing, SDK gets the uploaded intelligent event by fAnalyzerDataCallBack and inform users.
- Step 5 After uploading intelligent event, call **NETClient.StopLoadPic** to stop subscribing intelligent event.
- Step 6 Call **NETClient.Logout** to log out of the device.
- Step 7 After using all NetSDK functions, call **NETClient.Cleanup** to release resources.

Notes for Process

- Event type: Subscribe to all smart events (EM_EVENT_IVS_TYPE.ALL) if different smart events need to be uploaded. Also support to subscribe to a single smart event.
- Image receiving or not: The network environment of some devices is 3G or 4G. When the NetSDK is connected to the device, set bNeedPicFile parameter in the NETClient.RealLoadPicture to false if images are not needed. Then only receive information about smart traffic event, without images.
- Pass the channel number -1 for full channel subscription. Some intelligent transportation products do not support full-channel subscription. If the -1 subscription fails, try a single-channel subscription.
- In the case of multi-device subscription, there are two main ways to distinguish which event is reported by one device.
 - ◇ Establish the mapping relationship between the device IP, login handle, and subscription handle. Locate the login handle through the subscription handle returned by the callback function, and then find the device IP;
 - ◇ Use dwUser in the callback function. The dwUser passed in by the subscription function will be returned in the callback function, thereby locating which device subscribes the event that triggered the callback.

2.9.4 Example Code

```
// Delegate the intelligent event callback
private static fAnalyzerDataCallBack m_AnalyzeHandle;
m_AnalyzeHandle = new fAnalyzerDataCallBack(AnalyzerDataCallBack);

// Upload the subscribed smart traffic events
// The example subscribes to the event of crowd density detection with 0 channel. Here, m_LoginID is the
// login handle. It will return event picture.
IntPtr realLoadPictureHandle = NETClient.RealLoadPicture(m_LoginID, 0,
(uint)EM_EVENT_IVS_TYPE.CROWDDETECTION, true, m_AnalyzeHandle, IntPtr.Zero, IntPtr.Zero);
if (IntPtr.Zero == realLoadPictureHandle)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

// Cancel the subscription of smart traffic events
NETClient.StopLoadPic(realLoadPictureHandle);

// Upload callback of intelligent event
private int AnalyzerDataCallBack(IntPtr IAnalyzerHandle, uint dwAlarmType, IntPtr pAlarmInfo, IntPtr pBuffer,
uint dwBufSize, IntPtr dwUser, int nSequence, IntPtr reserved)
```

```

{
switch ((EM_EVENT_IVS_TYPE)dwAlarmType)
{
    // Event of crowd density detection
case EM_EVENT_IVS_TYPE.CROWDDETECTION:
{
    NET_DEV_EVENT_CROWD_DETECTION_INFO info =
    (NET_DEV_EVENT_CROWD_DETECTION_INFO)Marshal.PtrToStructure(pAlarmInfo,
    typeof(NET_DEV_EVENT_CROWD_DETECTION_INFO));
    this.BeginInvoke(new Action(() =>
    {
        Console.WriteLine("Event: Crowd density detection;");
        Console.WriteLine("Channel number: " + info.nChannelID + ";");
        Console.WriteLine("Time: " + info.UTC.ToString() + ";");
        /* .....
        **Other operations of event information **
        .....*/
    }));
    }
    break;
default:
    break;
    }
    return 0;
}

```

2.10 Alarm Upload

2.10.1 Introduction

Alarm upload can be realized through NetSDK login the device and subscription of the alarm function to the device which will send the detected alarm event to NetSDK. The alarm information can be obtained through callback.

2.10.2 Interface Overview

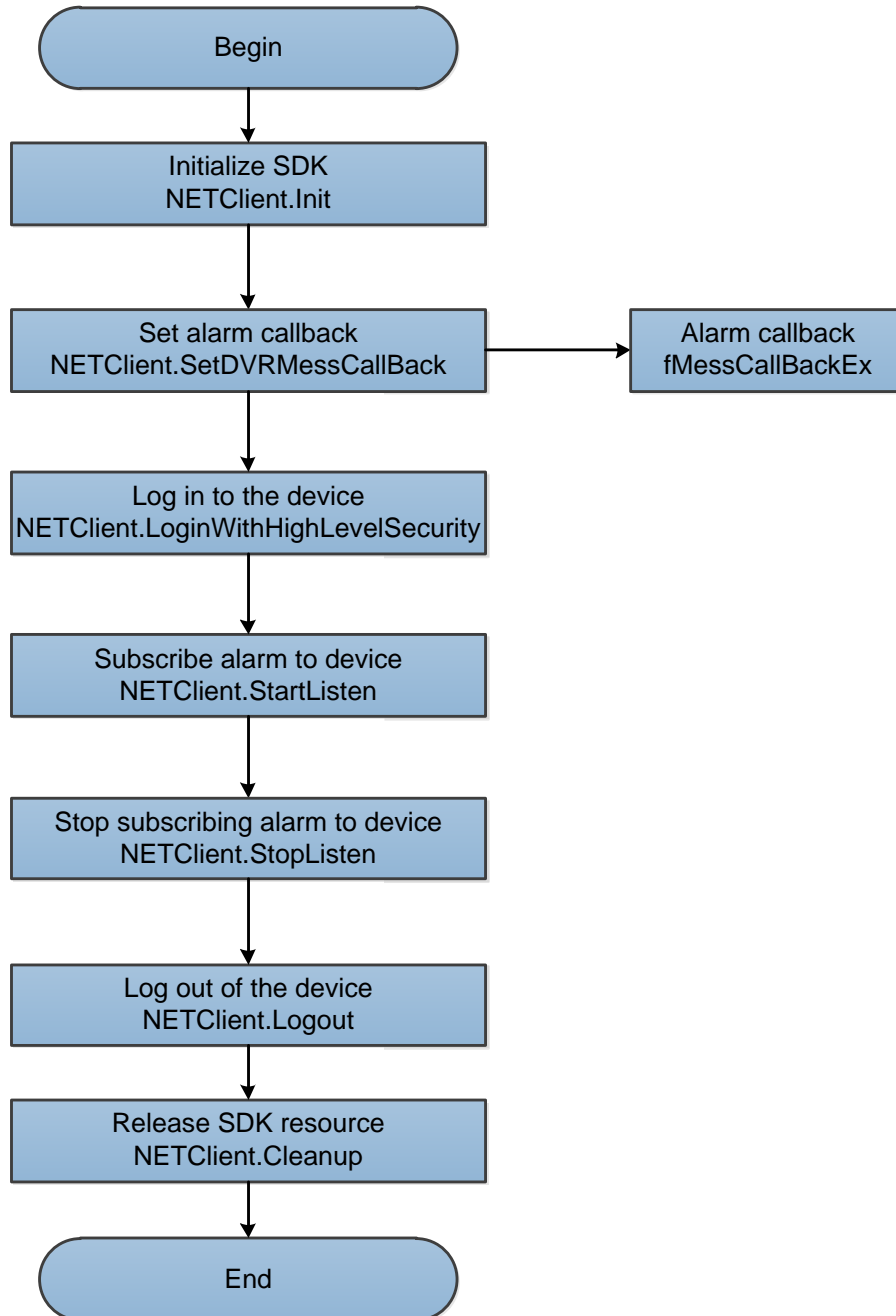
Table 2-10 Interfaces of alarm upload

Interface	Implication
NETClient.SetDVRMessCallBack	Set the alarm callback.
NETClient.StartListen	Subscribe to alarm.

Interface	Implication
NETClient.StopListen	Stop subscribing to alarm.

2.10.3 Process

Figure 2-14 Process of alarm upload



Process Description

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.SetDVRMessCallBack** to set alarm callback which should be called before subscribing alarm.
- Step 3 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.

- Step 4** Call **NETClient.StartListen** to subscribe to alarm to the device. If succeeded, the alarm event uploaded by the device will be informed to you through the callback set by NETClient.SetDVRMessCallBack.
- Step 5** After using the alarm upload function, call **NETClient.StopListen** to stop subscribing alarm to the device.
- Step 6** After using the function module, call **NETClient.Logout** to log out of the device.
- Step 7** After using all NetSDK functions, call **NETClient.Cleanup** to release NetSDK resource.

2.10.4 Example Code

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

// Deal with 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
                {
                    //Customize
                }
                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));
```

```

        // Customize
    }

    break;
default:
    Console.WriteLine(lCommand.ToString("X"));
    break;
}

return true;
}

// Subscribe to alarm
bool ret = NETClient.StartListen(m_LoginID);
if (!ret)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

// Stop alarm subscription
bool ret = NETClient.StopListen(m_LoginID);
if (!ret)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

```

2.11 Device Status and Information

2.11.1 Introduction

The storage interface mainly includes access to remote device information, query subscription on connection state of remote device, and modification of remote channel name.

2.11.2 Interface Overview

Table 2-11 Interfaces of storage

Interface	Implication
NETClient.QueryDevState	Directly get the connection state of remote device for each channel individually. Set the parameter type as EM_DEVICE_STATE.VIRTUALCAMERA.
NETClient.QueryDevInfo	Directly get the connection state of remote device for all the

Interface	Implication
	channels at the same time. Set the parameter nQueryType as EM_QUERY_DEV_INFO.GET_CAMERA_STATE.
NETClient.AttachCameraState	Subscribe to the remote device state. When the state changes, the corresponding information will be reported.
NETClient.DetachCameraState	Stop subscribing the remote device state. Used with NETClient.AttachCameraState in match.
NETClient.MatrixGetCameras	Get the information of remote device, such as device type and IP.
NETClient.GetNewDevConfig	Get the channel name. Set the parameter strCommand as ChannelTitle.

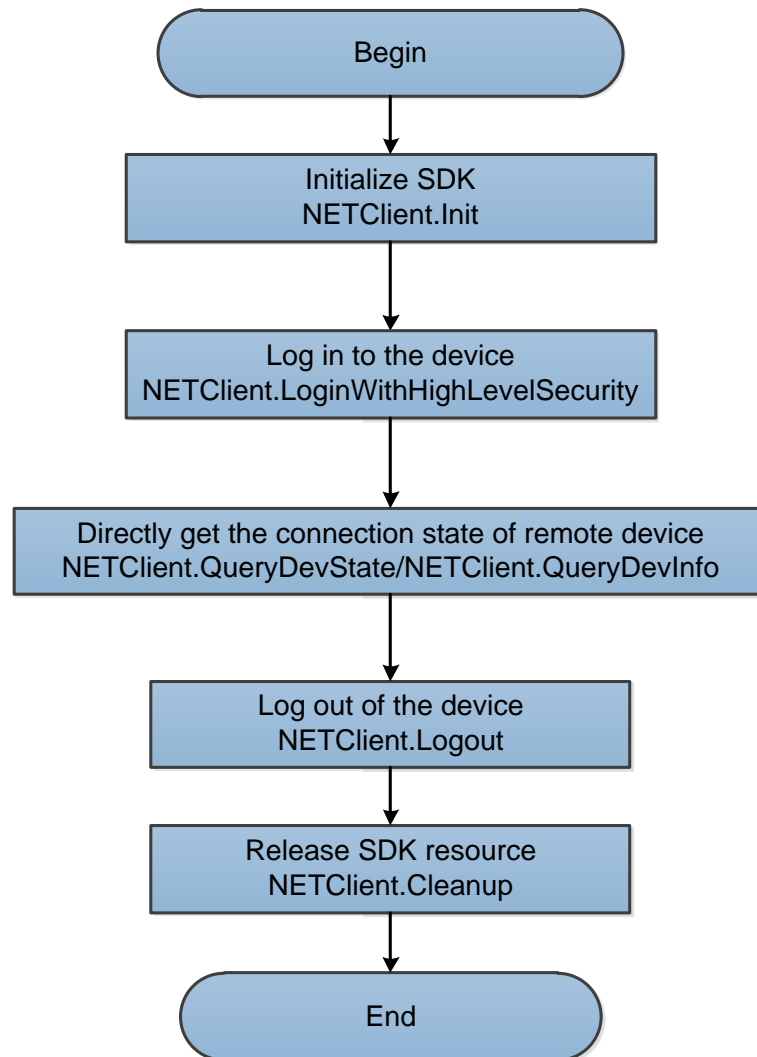
2.11.3 Process

The storage module has the following processes:

- Direct access to connection state of remote device
- Subscription to connection state of remote device
- Access to the information of remote device
- Access to channel name of remote device

2.11.3.1 Direct Access to Connection State of Remote Device

Figure 2-15 Process of directly accessing to connection state of remote device



Process Description

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

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

Step 3 Directly access to connection state of remote device.

There are two interfaces that are NETClient.QueryDevState (nType is EM_DEVICE_STATE.VIRTUALCAMERA) and NETClient.QueryDevInfo (emQueryType is EM_QUERY_DEV_INFO.GET_CAMERA_STATE) respectively.



The two interfaces are different depending on the device. It is suggested to have a test prior to use, and then select the proper interface.

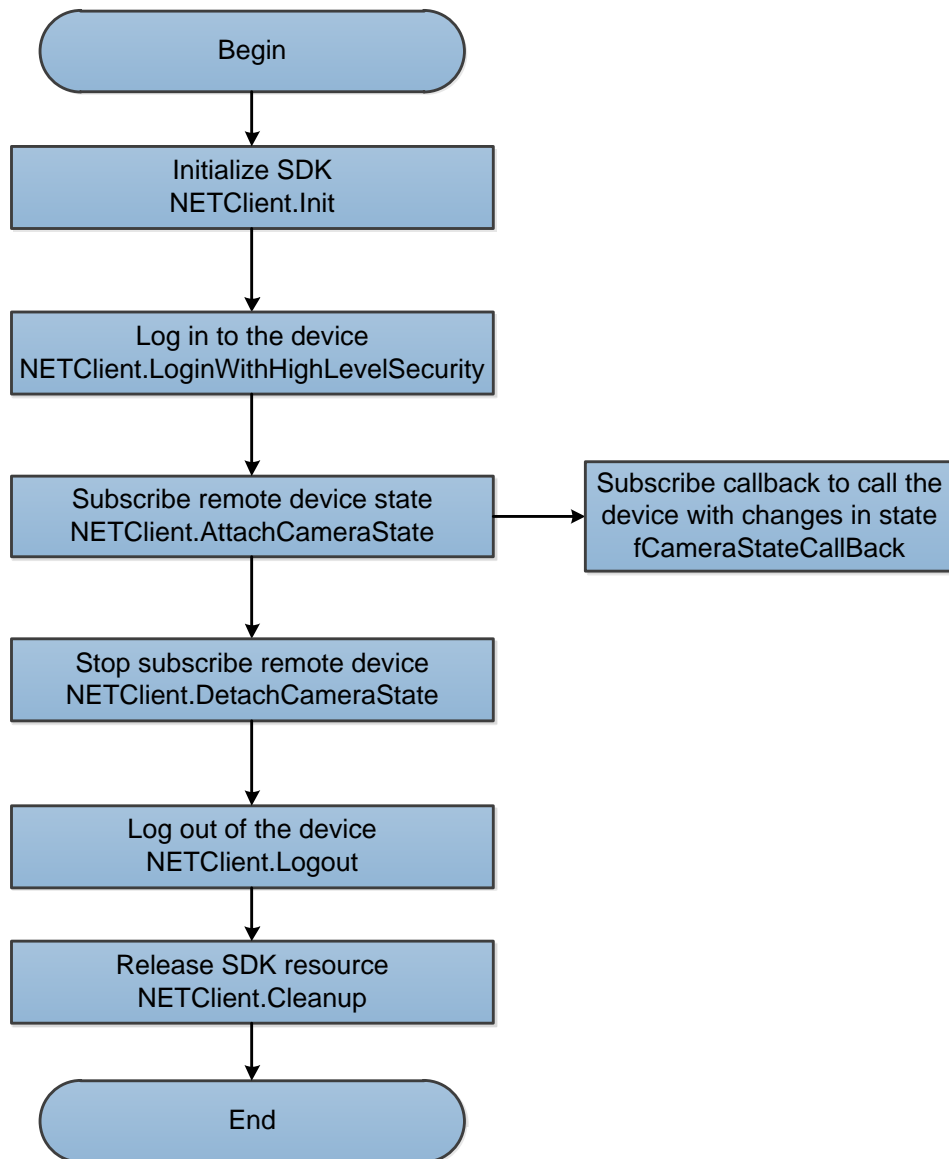
Step 4 After using the function module, call **NETClient.Logout** to log out of the device.

Step 5 After using all NetSDK functions, call **NETClient.Cleanup** to release NetSDK resource.

2.11.3.2 Subscription to Connection State of Remote Device

For the process of subscription to connection state of remote device, see Figure 2-16.

Figure 2-16 Process of subscription to connection state of remote device

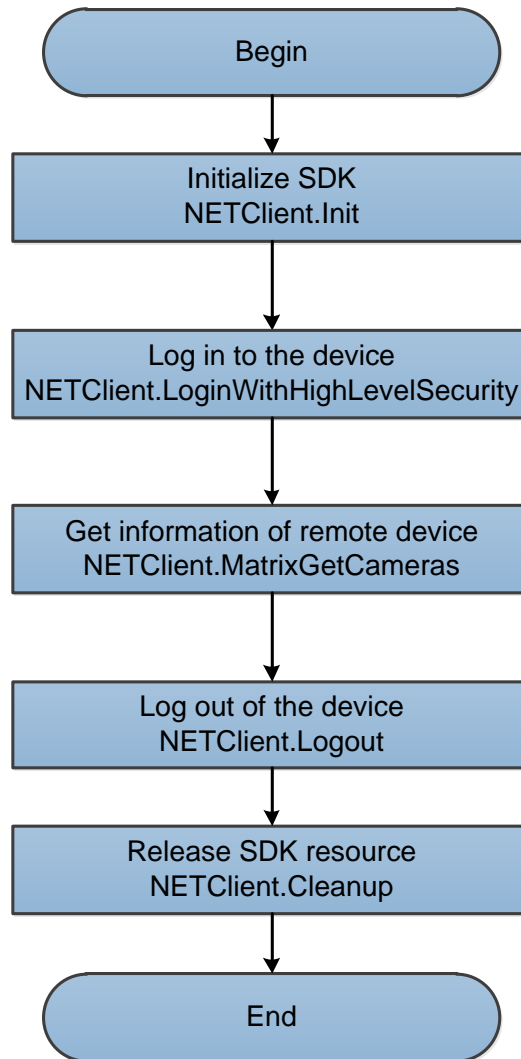


Process Description

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.AttachCameraState** to subscribe to the connection state of remote device. If the state changes, the report will be sent through fCameraStateCallBack.
- Step 4 Call **NETClient.DetachCameraState** to cancel the subscription. There will be no upload even the remote device state is changed, when you cancel subscription.
- Step 5 After using the function module, call **NETClient.Logout** to log out of the device.
- Step 6 After using all NetSDK functions, call **NETClient.Cleanup** to release NetSDK resource.

2.11.3.3 Access to the Information of Remote Device

Figure 2-17 Process of accessing to the information of remote device

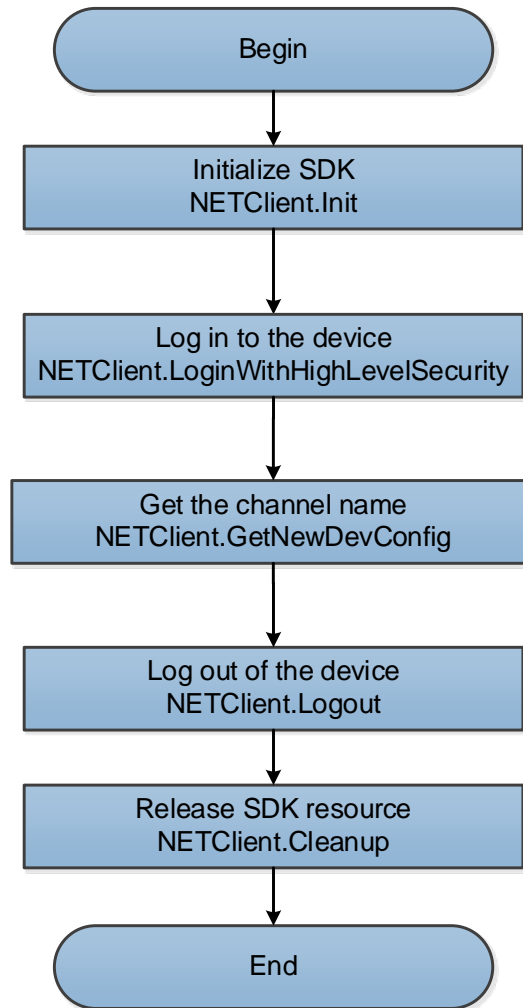


Process Description

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.MatrixGetCameras** to get the remote device information such as device type and IP.
- Step 4 After using the function module, call **NETClient.Logout** to log out of the device.
- Step 5 After using all NetSDK functions, call **NETClient.Cleanup** to release NetSDK resource.

2.11.3.4 Access to channel name of remote device

Figure 2-18 Process of accessing to channel name of remote device



Process Description

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **NETClient.GetNewDevConfig** to get the channel name. There are two available commands, including CFG_CMD_VIDEOIN and CFG_CMD_CHANNELTITLE.
- Step 4 After using the function module, call **NETClient.Logout** to log out of the device.
- Step 5 After using all NetSDK functions, call **NETClient.Cleanup** to release NetSDK resource.

2.11.4 Example Code

2.11.4.1 Direct Access to Connection State of Remote Device

```
// Interface 1 for getting the connection state of remote device  
bool result = false;
```

```

NET_VIRTUALCAMERA_STATE_INFO info = new NET_VIRTUALCAMERA_STATE_INFO()
{
    nStructSize = (uint)Marshal.SizeOf(typeof(NET_VIRTUALCAMERA_STATE_INFO)),
};
for (int i = 0; i < device.nChanNum; ++i)
{
    info.nChannelID = i;
    object obj = (object)info;
    result = NETClient.QueryDevState(ILoginID, EM_DEVICE_STATE.VIRTUALCAMERA, ref obj, typeof(NET_VIRTUALCAMERA_STATE_INFO), 3000);
    if (result)
    {
        info = (NET_VIRTUALCAMERA_STATE_INFO)obj;
        Console.WriteLine("QueryDevState_ChannelState channel:{0}, state: {1}", info.nChannelID, info.emConnectState.ToString());
    }
    else
    {
        Console.WriteLine("QueryDevState_ChannelState fail, {0}", NETClient.GetLastError());
    }
}

// Interface 2 for getting the connection state of remote device
IntPtr inPtr = IntPtr.Zero;
IntPtr outPtr = IntPtr.Zero;
NET_IN_GET_CAMERA_STATEINFO inParam = new NET_IN_GET_CAMERA_STATEINFO();
NET_OUT_GET_CAMERA_STATEINFO outParam = new NET_OUT_GET_CAMERA_STATEINFO();

inParam.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_GET_CAMERA_STATEINFO));
inParam.bGetAllFlag = true;
outParam.dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_GET_CAMERA_STATEINFO));
outParam.nMaxNum = DeviceInfo.nChanNum; // Total channel number
outParam.pCameraStateInfo = Marshal.AllocHGlobal(outParam.nMaxNum *
    Marshal.SizeOf(typeof(NET_CAMERA_STATE_INFO)));

inPtr = Marshal.AllocHGlobal((int)inParam.dwSize);
Marshal.StructureToPtr(inParam, inPtr, true);
outPtr = Marshal.AllocHGlobal((int)outParam.dwSize);
Marshal.StructureToPtr(outParam, outPtr, true);

```

```

bool res = NETClient.QueryDevInfo(m_LoginID, EM_QUERY_DEV_INFO.GET_CAMERA_STATE, inPtr, outPtr,
3000);
if (!res)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}
outParam = (NET_OUT_GET_CAMERA_STATEINFO)Marshal.PtrToStructure(outPtr,
typeof(NET_OUT_GET_CAMERA_STATEINFO));
if (outParam.nValidNum > 0)
{
    var status = new NET_CAMERA_STATE_INFO[outParam.nValidNum];
    for (int i = 0; i < outParam.nValidNum; i++)
    {
        status[i] = (NET_CAMERA_STATE_INFO)Marshal.PtrToStructure(outParam.pCameraStateInfo + i *
Marshal.SizeOf(typeof(NET_CAMERA_STATE_INFO)), typeof(NET_CAMERA_STATE_INFO));
    }
}
}

```

2.11.4.2 Subscription to Connection State of Remote Device

```

// Callback declaration of remote device state. Any change in state will return through this callback.
private static fCameraStateCallBack cameraStateCallBack = new fCameraStateCallBack(CameraStateCallBack);

// Subscription to the state of remote device
IntPtr intPtr = IntPtr.Zero;
intPtr = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(IntPtr)));
int[] channels = new int[1] { -1 };
Marshal.Copy(channels, 0, intPtr, channels.Length);
NET_IN_CAMERASTATE inParam = new NET_IN_CAMERASTATE()
{
    dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_CAMERASTATE)),
    nChannels = 1,
    pChannels = intPtr,
    cbCamera = cameraStateCallBack
};

NET_OUT_CAMERASTATE outParam = new NET_OUT_CAMERASTATE()
{
    dwSize = (uint)Marshal.SizeOf(typeof(NET_OUT_CAMERASTATE)),

```

```

};

IAttachHandle = NETClient.AttachCameraState(ILLoginID, inParam, ref outParam);
if (IntPtr.Zero == IAttachHandle)
{
    Console.WriteLine("Attach camera state fail");
}
else
{
    Console.WriteLine("Attach camera state success");
}

// Stop subscription of remote device state
if (IntPtr.Zero == IAttachHandle)
{
    return;
}
bool result = NETClient.DetachCameraState(IAttachHandle);
if (result)
{
    Console.WriteLine("Detach camera state success");
}
else
{
    Console.WriteLine("Detach camera state fail");
}

// Definition of state callback of remote device
private static void CameraStateCallback(IntPtr ILoginID, IntPtr IAttachHandle, IntPtr pBuf, int nBufLen, IntPtr dwUser)
{
    if (ILoginID == IntPtr.Zero || IAttachHandle == IntPtr.Zero)
    {
        return;
    }
    NET_CB_CAMERASTATE state = (NET_CB_CAMERASTATE)Marshal.PtrToStructure(pBuf,
    typeof(NET_CB_CAMERASTATE));
    Console.WriteLine("channel: {0} state: {1}", state.nChannel, state.emConnectState.ToString());
}

```

2.11.4.3 Access to Information of Remote Device

```
// Access to information of remote device. This demo obtains the information of online remotedevices. You
// have to call NETClient.QueryDevInfo at first, to get the online device number and device channel. And then call
// NETClient.MatrixGetCameras, to get remote device information of the corresponding channel.
// nChanNum Channel number
NET_MATRIX_CAMERA_INFO[] infos = new NET_MATRIX_CAMERA_INFO[nChanNum];
NET_CAMERA_STATE_INFO[] status = new NET_CAMERA_STATE_INFO[nChanNum];
for (int i = 0; i < nChanNum; i++)
{
    infos[i].dwSize = (uint)Marshal.SizeOf(typeof(NET_MATRIX_CAMERA_INFO));
}
bool ret = NETClient.MatrixGetCameras(m_LoginID, out infos, nChanNum, 5000);
if(!ret)
{
    MessageBox.Show(NETClient.GetLastError());
    return;
}
for (int i = 0; i < nChanNum; i++)
{
    Console.WriteLine(string.Format("Channel{0}device name:{1}", i, infos[i].stuRemoteDevice.szDevType));
    Console.WriteLine(string.Format("Channel{0}device name:{1}", i, infos[i].stuRemoteDevice.szDevName));
}
```

2.11.4.4 Access to Channel Name of Remote Device

```
// Access to channel name of remote device
AV_CFG_ChannelName[] infos = new AV_CFG_ChannelName[nChanNum];
object[] objs = new object[nChanNum];
for (int i = 0; i < infos.Length; i++)
{
    infos[i].nStructSize = Marshal.SizeOf(typeof(AV_CFG_ChannelName));
    objs[i] = infos[i];
}

bool ret = GetNewDevConfig(loginID, -1, "ChannelTitle", ref objs, typeof(AV_CFG_ChannelName), 5000);
if (!ret)
{
    Console.WriteLine(NETClient.GetLastError().ToString("X"));
    break;
}
```

```
}  
for (int i = 0; i < infos.Length; i++)  
{  
    infos[i] = (AV_CFG_ChannelName)objs[i];  
    Console.WriteLine(infos[i].szName);  
}
```

3 Interface Definition

3.1 NetSDK Initialization

3.1.1 NetSDK Initialization

Item	Description	
Name	Initialize NetSDK.	
Function	bool Init(fDisconnectCallBack cbDisconnect, IntPtr dwUser, NETSDK_INIT_PARAM? stulnitParam);	
Parameter	[in] cbDisconnect	Disconnection callback.
	[in] dwUser	User parameter of disconnection callback.
	[in] stulnitParam	NetSDK parameter initialization.
Return value	<ul style="list-style-type: none">• Success: true• Failure: false.	
Note	<ul style="list-style-type: none">• The precondition for calling other function modules of NetSDK.• The callback will not send to the user after the device is disconnected if the callback is set as NULL.• The dwUser parameter passed by Init will be returned with the same field dwUser in the callback function cbDisconnect, which is convenient for customers to locate. The other functions are the same.	

3.1.2 NetSDK Cleanup

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

3.1.3 Auto Reconnection Setting

Item	Description
Name	Set auto reconnection for callback.
Function	void SetAutoReconnect(fHaveReConnectCallBack cbAutoConnect, IntPtr dwUser);
Parameter	[in] cbAutoConnect Reconnection callback.

Item	Description	
	[in] dwUser	User parameter of disconnection callback.
Return value	None.	
Note	Set the reconnection callback interface. If the callback is set as NULL, it will not connect automatically.	

3.1.4 Network Parameter Setting

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

3.2 Device Login

3.2.1 Login

Item	Description	
Name	Log in to the device.	
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	User name.
	[in] pchPassword	Password.
	[in] emSpecCap	Login category.
	[in] pCapParam	Login category parameter.
	[out] deviceInfo	Device information.
Return value	<ul style="list-style-type: none"> Success: Returns a non-zero value. Failure: 0. 	
Note	None.	

3.2.2 Logout

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

3.3 Real-time Monitoring

3.3.1 Opening the Real-time Monitoring

Item	Description	
Name	Open the real-time monitoring	
Function	LLONG NETCLIENT_RealPlayEx(LLONG ILoginID, int nChannelID, HWND hWnd, DH_RealPlayType rType);	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[in] nChannelID	Video channel number is a round number starting from 0.
	[in] hWnd	Window handle valid only under Windows system
	[in] rType	Preview type
Return value	<ul style="list-style-type: none">• Success: Returns a non-zero value.• Failure: 0	
Note	Windows system: <ul style="list-style-type: none">• When hWnd is valid, the corresponding window displays picture.• When hWnd is NULL, get the video data through setting a callback and send to user for handle.	

The following table shows information about live type:

Table 3-1 Information of live type

Live Type	Meaning
DH_RType_Realplay	Real-time live.
DH_RType_Multiplay	Multi-picture live.
DH_RType_Realplay_0	Real-time monitoring—main stream, equivalent to DH_RType_Realplay.
DH_RType_Realplay_1	Real-time monitoring—sub stream 1.

Live Type	Meaning
DH_RType_Realplay_2	Real-time monitoring—sub stream 2.
DH_RType_Realplay_3	Real-time monitoring—sub stream 3.
DH_RType_Multiplay_1	Multi-picture live—1 picture.
DH_RType_Multiplay_4	Multi-picture live—4 pictures.
DH_RType_Multiplay_8	Multi-picture live—8 pictures.
DH_RType_Multiplay_9	Multi-picture live—9 pictures.
DH_RType_Multiplay_16	Multi-picture live—16 pictures.
DH_RType_Multiplay_6	Multi-picture live—6 pictures.
DH_RType_Multiplay_12	Multi-picture live—12 pictures.
DH_RType_Multiplay_25	Multi-picture live—25 pictures.
DH_RType_Multiplay_36	Multi-picture live—36 pictures.

3.3.2 Stopping the Real-time Monitoring

Item	Description		
Name	Stop the real-time monitoring.		
Function	<pre>bool StopRealPlay(IntPtr IRealHandle);</pre>		
Parameter	<table border="1"> <tr> <td>[in] IRealHandle</td><td>Return value of NETClient.RealPlay.</td></tr> </table>	[in] IRealHandle	Return value of NETClient.RealPlay.
[in] IRealHandle	Return value of NETClient.RealPlay.		
Return value	<ul style="list-style-type: none"> • Success: true • Failure: false. 		
Note	None.		

3.3.3 Saving the Real-time Monitoring Data

Item	Description				
Name	Save the real-time monitoring data as file.				
Function	<pre>bool SaveRealData(IntPtr IRealHandle, string pchFileName);</pre>				
Parameter	<table border="1"> <tr> <td>[in] IRealHandle</td><td>Return value of NETClient.RealPlay.</td></tr> <tr> <td>[in] pchFileName</td><td>Save path.</td></tr> </table>	[in] IRealHandle	Return value of NETClient.RealPlay.	[in] pchFileName	Save path.
[in] IRealHandle	Return value of NETClient.RealPlay.				
[in] pchFileName	Save path.				
Return value	<ul style="list-style-type: none"> • Success: true • Failure: false. 				
Note	None.				

3.3.4 Stopping Saving the Real-time Monitoring Data

Item	Description
Name	Stop saving the real-time monitoring data as file.

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

3.3.5 Setting Callback of Real-time Monitoring Data

Item	Description	
Name	Set the callback of real-time monitoring data.	
Function	bool SetRealDataCallBack(IntPtr IRealHandle, fRealDataCallBackEx2 cbRealData, IntPtr dwUser, EM_REALDATA_FLAG dwFlag);	
Parameter	[in] IRealHandle	Return value of NETClient.RealPlay.
	[in] cbRealData	Callback of monitoring data flow.
	[in] dwUser	Parameter of callback 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.	

The following table shows information about parameter dwFlag:

Table 3-2 Type and meaning of dwFlag

dwFlag	Meaning
0x00000001	Initial data of device.
0x00000004	Data converted to YUV format.

3.4 Record Playback

3.4.1 Playback by Time

Item	Description	
Name	Playback by time.	
Function	IntPtr PlayBackByTime(IntPtr ILoginID, int nChannelID, NET_IN_PLAY_BACK_BY_TIME_INFO pstNetIn, ref NET_OUT_PLAY_BACK_BY_TIME_INFO pstNetOut);	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[in] nChannelID	Device channel number.
	[in] pstNetIn	Query on input condition.
	[out] pstNetOut	Query on output information.
Return value	<ul style="list-style-type: none">• Success: Returns a non-zero value.• Failure: 0.	
Note	<ul style="list-style-type: none">• For the callback declaration fDataCallBack and fDownloadPosCallBackfDataCallBack in NET_IN_PLAY_BACK_BY_TIME_INFO, see "Chapter 4 Callback."• The parameters hWnd and fDownloadDataCallBack in pstNetIn cannot be NULL at the same time; otherwise the interface calling will be failed returned.	

3.4.2 Setting the Work Mode

Item	Description	
Name	Set the work mode.	
Function	bool SetDeviceMode(IntPtr ILoginID, EM_USEDEV_MODE emType, IntPtr pValue);	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[in] emType	Work mode enumeration.
	[in] pValue	The structure corresponding to work mode.
Return value	<ul style="list-style-type: none">• Success: true• Failure: false.	
Note	None.	

The following table shows information about work mode enumeration and structure:

Table 3-3 Work mode and corresponding structure

emType Enumeration	Meaning	Structure
RECORD_STREAM_TYPE	Set the record stream type to query and playback by time.	None

emType Enumeration	Meaning	Structure
	<ul style="list-style-type: none"> 0: Main and sub stream 1: Main stream 2: Sub stream 	
RECORD_TYPE	Set the record file type to playback and download by time.	EM_RECORD_TYPE

3.4.3 Stopping Playback

Item	Description	
Name	Stop video playback.	
Function	<pre>bool PlayBackControl(IntPtr IPlayHandle, PlayBackType type);</pre>	
Parameter	[in] IPlayHandle	Return value of playback interface.
	[in] type	Controlling type.
Return value	<ul style="list-style-type: none"> Success: true Failure: false. 	
Note	None.	

3.4.4 Getting the OSD Playback Time

Item	Description	
Name	Get the OSD playback time.	
Function	<pre>bool GetPlayBackOsdTime(IntPtr IPlayHandle, ref NET_TIME lpOsdTime, ref NET_TIME lpStartTime, ref NET_TIME lpEndTime);</pre>	
Parameter	[in] IPlayHandle	Return value of playback interface.
	[out] lpOsdTime	OSD time.
	[out] lpStartTime	The start time of current playback file.
	[out] lpEndTime	The end time of current playback file.
Return value	<ul style="list-style-type: none"> Success: true Failure: false. 	
Note	None.	

Table 3-4 Control type enumeration

PlayBackType enumeration	Meaning
Play	Play
Pause	Pause
Stop	Stop
Fast	Fast play
Slow	Slow play

PlayBackType enumeration	Meaning
Normal	Normal play

3.5 Record Download

3.5.1 Querying Record Files within a Period

Item	Description	
Name	Query all record files within a period.	
Function	<pre>bool QueryRecordFile(IntPtr lLoginID, int nChannelId, EM_QUERY_RECORD_TYPE nRecordFileType, DateTime tmStart, DateTime tmEnd, string pchCardid, ref NET_RECORDFILE_INFO[] nriFileinfo, ref int filecount, int waittime, bool bTime);</pre>	
Parameter	[in] lLoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[in] nChannelId	Device channel number starting from 0.
	[in] nRecordFileType	Record file type.
	[in] tmStart	Record start time.
	[in] tmEnd	Record end time.
	[in] pchCardid	Card ID.
	[out] nriFileinfo	The returned record file is a NET_RECORDFILE_INFO structured data.
	[out] filecount	The number of returned files. The output parameters can only check the recordings until the buffer is full.
	[in] waittime	Waiting time.
	[in] bTime	Invalid currently.
Return value	<ul style="list-style-type: none"> Success: true Failure: false. 	
Note	Before playback, call this interface to query the records. When the queried records within the input time are larger than the buffer size, it will only return the records that can be stored by buffer. Continue with the query if needed.	

The following table shows information about record file type and card ID:

Table 3-5 Record file type and card ID

EM_QUERY_RECORD_TYPE Enumeration	Record File Type	Card ID
ALL	All record files	NULL

EM_QUERY_RECORD_TYPE Enumeration	Record File Type	Card ID
ALARM	External alarm	NULL
MOTION_DETECT	Alarm by dynamical detection	NULL
ALARM_ALL	All the alarms	NULL
CARD	Card IP query	Card ID
CONDITION	Combined conditions query	Card ID && Transaction type && Transaction amount (If you want to skip a field, set as blank)
JOIN	Record location and deviation length	NULL
CARD_PICTURE	Pictures queried by card ID (Only supported by some models of HB-U and NVS)	Card ID
PICTURE	Query pictures (Only supported by some models of HB-U and NVS)	NULL
FIELD	Query by field	FELD1&&FELD2&&FELD3&& (If you want to skip a field, set as blank)

3.5.2 Opening the Record Query Handle

Item	Description	
Name	Open the record query handle.	
Function	<pre>IntPtr FindFile(IntPtr ILoginID, EM_FILE_QUERY_TYPE emType, object oQueryCondition, Type tyCondition, int waittime);</pre>	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[in] emType	Record file type. For details, see Table 3-5.
	[in] oQueryCondition	Query condition.
	[in] waittime	Query timeout.
Return value	<ul style="list-style-type: none"> Success: Returns a non-zero value. Failure: 0. 	
Note	None.	

3.5.3 Finding the Record File

Item	Description
Name	Find the record file.

Item	Description	
Function	<pre>int FindNextFile(IntPtr IFindHandle, int nFilecount, List<object> IsOMediaFileInfo, Type tyFile , int waittime);</pre>	
Parameter	[in] IFindHandle	Return value of NETClient.FindFile (open the record query handle).
	[in] nFilecount	Number of the found files.
	[out] IsOMediaFileInfo	Information of the found files.
	[in] tyFile	Information type of the found files.
	[in] waittime	Query timeout. The unit is millisecond.
Return value	Number of the found file information.	
Note	Before calling this interface, call NETClient.FindFile to open the query handle.	

3.5.4 Closing the Record Query Handle

Item	Description	
Name	Close the record query handle.	
Function	<pre>bool FindClose(IntPtr IFindHandle);</pre>	
Parameter	[in] IFindHandle	Return value of NETClient.FindFile (open the record query handle).
Return value	<ul style="list-style-type: none"> Success: true Failure: false. 	
Note	Call NETClient.FindFile to open the query handle. After query is finished, call this function to close the query handle.	

3.5.5 Downloading Record by File

Item	Description	
Name	Download record by file.	
Function	<pre>IntPtr DownloadByRecordFile(IntPtr ILoginID, ref NET_RECORDFILE_INFO lpRecordFile, string sSavedFileName, fDownLoadPosCallBack cbDownLoadPos, IntPtr dwUserData);</pre>	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[out] lpRecordFile	The information pointer of record file.
	[in] sSavedFileName	The record file name and full save path.

Item	Description	
	[in] cbDownloadPos	Download progress callback.
	[in] dwUserData	Download progress callback customized data.
Return value	<ul style="list-style-type: none"> Success: Returns a non-zero value. Failure: 0. 	
Note	<ul style="list-style-type: none"> For callback declaration of fDownloadPosCallBack, see "Chapter 4 Callback." sSavedFileName is not blank, and the record data is input into the file corresponding with the path. fDownloadDataCallBack is not blank, and the record data is returned through callback function. 	

3.5.6 Downloading Record by Time

Item	Description	
Name	Download record by time.	
Function	<pre>IntPtr DownloadByTime(IntPtr lLoginID, int nChannelId, EM_QUERY_RECORD_TYPE nRecordFileType, DateTime tmStart, DateTime tmEnd, string sSavedFileName, fTimeDownloadPosCallBack cbTimeDownloadPos, IntPtr dwUserData, fDataCallBack fDownloadDataCallBack, IntPtr dwDataUser, IntPtr pReserved);</pre>	
Parameter	[in] lLoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[in] nChannelId	The device channel number starting from 0.
	[in] nRecordFileType	Record file type. For details, see Table 3-5.
	[in] tmStart	Start time of download.
	[in] tmEnd	End time of download.
	[in] sSavedFileName	The record file name and full save path.
	[in] cbTimeDownloadPos	Download progress callback.
	[in] dwUserData	Download progress callback customized data.
	[in] fDownloadDataCallBack	Download data callback.
	[in] dwDataUser	Download data callback customized data.
	[in] pReserved	Parameter reserved and the default is NULL.
Return value	<ul style="list-style-type: none"> Success: Returns a non-zero value. Failure: 0. 	

Item	Description
Note	<ul style="list-style-type: none"> For callback declaration of fDataCallBack and fDownloadPosCallBack, see "Chapter 4 Callback." sSavedFileName is not blank, and the record data is input into the file corresponding with the path. fDownloadDataCallBack is not blank, and the record data is returned through callback.

3.5.7 Querying the record downloading progress

Item	Description
Name	Query the record downloading progress.
Function	<pre>bool GetDownloadPos(IntPtr IFileHandle, ref int nTotalSize, ref int nDownloadSize);</pre>
Parameter	[in] IFileHandle Return value of download interface.
	[out] nTotalSize The total length of download and the unit is KB.
	[out] nDownloadSize The downloaded length and the unit is KB.
Return value	<ul style="list-style-type: none"> Success: true Failure: false.
Note	<ul style="list-style-type: none"> Get the current location of the record to be downloaded to apply to display interface that does not need to display real-time download progress. It is similar to the function of download callback. Calculate the progress without using the callback. Call this interface regularly to get the current progress.

3.5.8 Stopping Record Downloading

Item	Description
Name	Stop record downloading.
Function	<pre>bool StopDownloadMediaFile(IntPtr IFileHandle);</pre>
Parameter	[in] IFileHandle Return value of download interface
Return value	<ul style="list-style-type: none"> Success: true Failure: false.
Note	Stop downloading after it is completed or partially completed according to particular situation.

3.6 PTZ Control

3.6.1 PTZ Control

Item	Description	
Name	PTZ control.	
Function	<pre>bool PTZControl(IntPtr ILoginID, int nChannelID, EM_EXTPTZ_ControlType dwPTZCommand, int IParam1, int IParam2, int IParam3, bool dwStop, IntPtr param4);</pre>	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[in] nChannelID	Video channel number that is a whole number and starts from 0.
	[in] dwPTZCommand	Control command type.
	[in] IParam1	Parameter 1.
	[in] IParam2	Parameter 2.
	[in] IParam3	Parameter 3.
	[in] dwStop	Stop mark, which is valid for operations of eight directions. When performing other operations, enter FALSE for this parameter.
	[in] param4	Support the following extension command: EM_EXTPTZ_ControlType.MOVE_ABSOLUTELY EM_EXTPTZ_ControlType.MOVE_CONTINUOUSLY EM_EXTPTZ_ControlType.GOTOPRESET EM_EXTPTZ_ControlType.SET_VIEW_RANGE EM_EXTPTZ_ControlType.FOCUS_ABSOLUTELY EM_EXTPTZ_ControlType.HORSECTORSCAN EM_EXTPTZ_ControlType.VERSECTORSCAN EM_EXTPTZ_ControlType.SET_FISHEYE_EPTZ EM_EXTPTZ_ControlType.SET_TRACK_START/SET_TRACK_STOP
Return value	<ul style="list-style-type: none"> ● Success: true ● Failure: false. 	
Note	For the relationship between dwPTZCommand and Param1, Param2 and Param3, see Table 3-6.	

Table 3-6 Relationship between command and parameters

EM_EXTPTZ_ControlType Enumeration	Function	param1	param2	param3
UP_CONTROL	Up	None	Vertical speed (1–8)	None
DOWN_CONTROL	Down	None	Vertical speed (1–8)	None
LEFT_CONTROL	Left	None	Horizontal speed (1–8)	None
RIGHT_CONTROL	Right	None	Horizontal speed (1–8)	None
ZOOM_ADD_CONTROL	Zoom+	None	Multi-speed	None
ZOOM_DEC_CONTROL	Zoom-	None	Multi-speed	None
FOCUS_ADD_CONTROL	Focus+	None	Multi-speed	None
FOCUS_DEC_CONTROL	Focus-	None	Multi-speed	None
APERTURE_ADD_CONTROL	Aperture+	None	Multi-speed	None
APERTURE_DEC_CONTROL	Aperture -	None	Multi-speed	None
POINT_MOVE_CONTROL	Move to preset	None	Value of preset	None
POINT_SET_CONTROL	Set	None	Value of preset	None
POINT_DEL_CONTROL	Delete	None	Value of preset	None
POINT_LOOP_CONTROL	Cruise among presets	Cruise route	None	76: On 99: Auto 96: Off
LAMP_CONTROL	Lamp wiper	0x01: On x00: Off	None	None
LEFTTOP	Left top	Vertical speed (1–8)	Horizontal speed (1–8)	None
RIGHTTOP	Right top	Vertical speed (1–8)	Horizontal speed (1–8)	None
LEFTDOWN	Left bottom	Vertical speed (1–8)	Horizontal speed (1–8)	None
RIGHTDOWN	Right bottom	Vertical speed (1–8)	Horizontal speed (1–8)	None
ADDTOLOOP	Add preset to cruise	Cruise route	Value of preset	None
DELFROMLOOP	Delete preset point in cruise	Cruise route	Value of preset	None
CLOSELOOP	Delete cruise	Cruise route	None	None
STARTPANCRUISE	Start horizontal rotation	None	None	None
STOPPANCRUISE	Stop horizontal rotation	None	None	None

EM_EXTPTZ_ControlType Enumeration	Function	param1	param2	param3
SETLEFTBORDER	Set left border	None	None	None
SETRIGHTBORDER	Set right border	None	None	None
STARTLINESCAN	Start line scan	None	None	None
CLOSELINESCAN	Stop line scan	None	None	None
SETMODESTART	Set mode start	Mode route	None	None
SETMODESTOP	Set mode stop	Mode route	None	None
RUNMODE	Running mode	Mode route	None	None
STOPMODE	Stop mode	Mode route	None	None
DELETEMODE	Delete mode	Mode route	None	None
REVERSECOMM	Reverse command	None	None	None
FASTGOTO	Fast positioning	Horizontal coordinate (0–8192)	Vertical coordinate (0–8192)	Zoom (4)
AUXIOPEN	Open auxiliary switch	Auxiliary point	None	None
AUXICLOSE	Close auxiliary switch	Auxiliary point	None	None
OPENMENU	Open menu	None	None	None
CLOSEMENU	Close menu	None	None	None
MENUOK	Menu confirm	None	None	None
MENUCANCEL	Menu cancel	None	None	None
MENUUP	Menu up	None	None	None
MENUDOWN	Menu down	None	None	None
MENULEFT	Menu left	None	None	None
MENURIGHT	Menu right	None	None	None
ALARMHANDLE	Alarm action with PTZ	Alarm input channel	Alarm action type: Preset Line scan Cruise	Linkage value, such as preset number
MATRIXSWITCH	Matrix switch	Monitor device number (video output number)	Video input number	Matrix number
LIGHTCONTROL	Light controller	Refer to DH_PTZ_LAMP_CONTROL	None	None
EXACTGOTO	3D positioning	Horizontal angle (0–3600)	Vertical coordinate (0–900)	Zoom (1–128)
RESETZERO	Reset to zero	None	None	None
UP_TELE	Up +TELE	Speed (1–8)	None	None

EM_EXTPTZ_ControlType Enumeration	Function	param1	param2	param3
DOWN_TELE	Down +TELE	Speed (1–8)	None	None
LEFT_TELE	Left +TELE	Speed (1–8)	None	None
RIGHT_TELE	Right+TELE	Speed (1–8)	None	None
LEFTUP_TELE	Leftup +TELE	Speed (1–8)	None	None
LEFTDOWN_TELE	Leftdown +TELE	Speed (1–8)	None	None
TIGHTUP_TELE	Rightup+TELE	Speed (1–8)	None	None
RIGHTDOWN_TELE	Rightdown +TELE	Speed (1–8)	None	None
UP_WIDE	Up +WIDE	Speed (1–8)	None	None
DOWN_WIDE	Down+WIDE	Speed (1–8)	None	None
LEFT_WIDE	Left +WIDE	Speed (1–8)	None	None
RIGHT_WIDE	Right+WIDE	Speed (1–8)	None	None
LEFTUP_WIDE	Leftup+WIDE	Speed (1–8)	None	None
LEFTDOWN_WIDE	Leftdown+WIDE	Speed (1–8)	None	None
TIGHTUP_WIDE	Rightup +WIDE	Speed (1–8)	None	None
RIGHTDOWN_WIDE	Rightdown +WIDE	Speed (1–8)	None	None

3.7 Voice Talk

3.7.1 Opening Voice Talk

Item	Description	
Name	Open voice talk.	
Function	<pre>IntPtr StartTalk(IntPtr ILoginID, fAudioDataCallBack pfcB, IntPtr dwUser);</pre>	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[in] pfcB	Audio data callback.
	[in] dwUser	Parameter of audio data callback.
Return value	<ul style="list-style-type: none"> Success: Returns a non-zero value. Failure: 0. 	
Note	None.	

3.7.2 Stopping Voice Talk

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

3.7.3 Starting Local Recording

Item	Description	
Name	Start local recording.	
Function	bool RecordStart(IntPtr ILoginID);	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
Return value	<ul style="list-style-type: none">• Success: true• Failure: false.	
Note	None.	

3.7.4 Stopping Local Recording

Item	Description	
Name	Stop local recording.	
Function	bool RecordStop(IntPtr ILoginID);	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
Return value	<ul style="list-style-type: none">• Success: true• Failure: false.	
Note	None.	

3.7.5 Talk Data Sending

Item	Description	
Name	Send audio data to device.	
Function	<pre>int TalkSendData(IntPtr ITalkHandle, IntPtr pSendBuf, uint dwBufSize);</pre>	
Parameter	[in] ITalkHandle	Return value of NETClient.StartTalk.
	[in] pSendBuf	Pointer of audio data block that needs sending.
	[in] dwBufSize	Length of audio data block that needs sending. The unit is byte.
Return value	<ul style="list-style-type: none">• Success: Length of audio data block.• Failure: -1.	
Note	None.	

3.7.6 Audio Decoding

Item	Description	
Name	Decode audio data.	
Function	<pre>void AudioDec(IntPtr pAudioDataBuf, uint dwBufSize);</pre>	
Parameter	[in] pAudioDataBuf	Pointer of audio data block that needs decoding.
	[in] dwBufSize	Length of audio data block that needs decoding. The unit is byte.
Return value	None.	
Note	None.	

3.8 Video Snapshot

3.8.1 Capturing Picture to File

Item	Description	
Name	Capture image.	
Function	<pre>bool SnapPictureToFile(IntPtr ILoginID, ref NET_IN_SNAP_PIC_TO_FILE_PARAM inParam, ref NET_OUT_SNAP_PIC_TO_FILE_PARAM outParam, int nWaitTime);</pre>	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[in] inParam	Input parameter.
	[in] outParam	Output parameter.
	[in] nWaitTime	Timeout. The unit is millisecond.
Return value	<ul style="list-style-type: none">• Success: true• Failure: false.	
Note	Synchronous interface. The device captures picture and sends to the user through network. This function is required on some devices.	

3.8.2 Capturing Picture

Item	Description	
Name	Capture image.	
Function	<pre>bool CapturePicture(IntPtr hPlayHandle, string pchPicFileName, EM_NET_CAPTURE_FORMATS eFormat);</pre>	
Parameter	[in] hPlayHandle	Return value of NETClient.RealPlay.
	[in] hPlayHandle	The file path which need to be saved.
	[in] pchPicFileName	Picture format.
Return value	<ul style="list-style-type: none">• Success: true• Failure: false.	
Note	<ul style="list-style-type: none">• Synchronous interface. Write the picture data into file.• The picture is captured from the real-time monitoring data stream sent by the device.	

3.9 Intelligent Event

3.9.1 Subscribing Intelligent Event

Item	Description	
Name	Subscribe intelligent event.	
Function	<pre>IntPtr RealLoadPicture(IntPtr ILoginID, int nChannelID, uint dwAlarmType, bool bNeedPicFile, fAnalyzerDataCallBack cbAnalyzerData, IntPtr dwUser, IntPtr reserved);</pre>	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[in] nChannelID	Channel ID.
	[in] dwAlarmType	Intelligent traffic event types.
	[in] bNeedPicFile	Whether picture file is needed.
	[in] cbAnalyzerData	Intelligent event callback function.
	[in] dwUser	The user customized data.
	[in] reserved	Reserved parameter.
Return value	<ul style="list-style-type: none">• Success: Returns a non-zero value.• Failure: 0.	
Note	<ul style="list-style-type: none">• Manual snapshot of intelligent traffic needs to call this interface in advance to receive snapshots.• Intelligent traffic event reporting needs to call this interface in advance to receive intelligent traffic event information and pictures.	

3.9.2 Unsubscribing Smart

Item	Description	
Name	Unsubscribe intelligent event.	
Function	<pre>bool StopLoadPic(IntPtr IAnalyzerHandle);</pre>	
Parameter	[in] ILoginID	Return value of NETClient.RealLoadPicture.
Return value	<ul style="list-style-type: none">• Success: true• Failure: false.	
Note	None	

3.10 Alarm Upload

3.10.1 Setting Alarm Callback

Item	Description	
Name	Set alarm callback.	
Function	void SetDVRMessCallBack(fMessCallBackEx cbMessage, IntPtr dwUser);	
Parameter	[in] cbMessage	<ul style="list-style-type: none">• Message callback that can call the device status, such as alarm status.• When setting as 0, the calling is prohibited.
	[in] dwUser	The user customized data.
Return value	None.	
Note	<ul style="list-style-type: none">• Sets device message callback to obtain the device current status. It has nothing to do with the calling sequence. There's no calling by default.• The callback fMessCallBack is valid after calling the alarm message subscription interface StartListen first.	

3.10.2 Subscribing to Alarm

Item	Description	
Name	Subscribe to alarm.	
Function	bool StartListen(IntPtr ILoginID);	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
Return value	<ul style="list-style-type: none">• Success: true• Failure: false.	
Note	The message from the subscribed device is called from the set value of SetDVRMessCallBack.	

3.10.3 Stopping Alarm Subscription

Item	Description	
Name	Stop alarm subscription.	
Function	bool StopListen(IntPtr ILoginID);	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
Return value	<ul style="list-style-type: none">• Success: true• Failure: false.	
Note	None.	

3.11 Device Status and Information

3.11.1 Querying Device State

Item	Description	
Name	Directly get the connection status of remote device.	
Function	<pre>bool QueryDevState(IntPtr ILoginID, int nType, ref object obj, Type typeName, int waittime);</pre>	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[in] nType	Query the information type. When the connection status of remote device is obtained, the nType becomes EM_DEVICE_STATE.VIRTUALCAMERA.
	[out] obj	Receives the data buffer returned from query. The corresponding structure is NET_VIRTUALCAMERA_STATE_INFO.
	[in] typeName	Query structure type.
	[in] waittime	Waiting time for query.
Return value	<ul style="list-style-type: none">• Success: true• Failure: false.	
Note	None.	

3.11.2 Querying Device Information

Item	Description	
Name	Directly get the connection status of remote device.	
Function	<pre>bool QueryDevInfo(IntPtr ILoginID, EM_QUERY_DEV_INFO emQueryType, IntPtr pInBuf, IntPtr pOutBuf, int nWaitTime = 1000);</pre>	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity..
	[in] emQueryType	Query type: When the connection status of device is obtained, nQueryType becomes EM_QUERY_DEV_INFO.GET_CAMERA_STATE.
	[in] pInBuf	Input buffer. When the connection status of device is obtained, the corresponding structure is NET_IN_GET_CAMERA_STATEINFO.
	[out] pOutBuf	Output buffer. When the connection status of device is

Item	Description	
		obtained, the corresponding structure is NET_OUT_GET_CAMERA_STATEINFO.
	[in] nWaitTime	Waiting time for query. The default is 1000ms.
Return value	<ul style="list-style-type: none"> • Success: true • Failure: false. 	
Note	None.	

3.11.3 Subscribing to State of Remote Device

Item	Description	
Name	Subscribe to the remote device status.	
Function	<pre>IntPtr AttachCameraState(IntPtr ILoginID, NET_IN_CAMERASTATE pstInParam, ref NET_OUT_CAMERASTATE pstOutParam, int nWaitTime = 3000);</pre>	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[in] pstInParam	Subscription input parameter.
	[out] pstOutParam	Subscription output parameter.
	[in] nWaitTime	Waiting time for query. The default is 3000ms.
Return value	<ul style="list-style-type: none"> • Success: Returns a non-zero value. • Failure: 0. 	
Note	For the state callback (fCameraStateCallBack) in the input parameter, see "Chapter 4 Callback."	

3.11.4 Stopping Subscribing State of Remote Device

Item	Description	
Name	Stop subscribing the state of remote device.	
Function	<pre>bool DetachCameraState(IntPtr IAttachHandle);</pre>	
Parameter	[in] IAttachHandle	Return value of subscribing remote device state.
Return value	<ul style="list-style-type: none"> • Success: true • Failure: false. 	
Note	None.	

3.11.5 Getting Information of Remote Device

Item	Description
Name	Get the remote device information.

Item	Description	
Function	<pre>bool MatrixGetCameras(IntPtr ILoginID, out NET_MATRIX_CAMERA_INFO[] stuCameras, int nMaxCameraCount, int nWaitTime);</pre>	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[out] stuCameras	Structure array of information of query cameras.
	[in] nMaxCameraCount	Number of query cameras.
	[in] nWaitTime	Waiting time for query.
Return value	<ul style="list-style-type: none"> • Success: true • Failure: false. 	
Note	None.	

3.11.6 Getting Channel Name

Item	Description	
Name	Get the channel name.	
Function	<pre>bool GetNewDevConfig(IntPtr ILoginID, Int32 IChannel, string strCommand, ref object obj, Type typeName, int waittime);</pre>	
Parameter	[in] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[in] IChannel	Device channel number starting from 0.
	[in] strCommand	Command parameter. Query channel name szCommand is NETCLIENT_GetNewDevConfig.
	[out] obj	Query information array, and the corresponding structure is AV_CFG_ChannelName.
	[in] typeName	Query structure type.
	[in] waittime	Waiting timeout for query. The default is 1000ms.
Return value	<ul style="list-style-type: none"> • Success: true • Failure: false. 	
Note	None.	

4 Callback Function

4.1 fDisConnectCallBack

Item	Description	
Name	Disconnection callback.	
Function	<pre>public delegate void fDisConnectCallBack(IntPtr ILoginID, IntPtr pchDVRIP, int nDVRPort, IntPtr dwUser);</pre>	
Parameter	[out] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[out] pchDVRIP	IP of the disconnected device.
	[out] nDVRPort	Port of the disconnected device.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.2 fHaveReConnectCallBack

Item	Description	
Name	Reconnection callback.	
Function	<pre>public delegate void fHaveReConnectCallBack(IntPtr ILoginID, IntPtr pchDVRIP, int nDVRPort, IntPtr dwUser);</pre>	
Parameter	[out] ILoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[out] pchDVRIP	IP of the reconnected device.
	[out] nDVRPort	Port of the reconnected device.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.3 fRealDataCallBackEx

Item	Description
Name	Callback of real-time monitoring data.
Function	<pre>public delegate void fRealDataCallBackEx(</pre>

Item	Description	
	IntPtr IRealHandle, uint dwDataType, IntPtr pBuffer, uint dwBufSize, int param, IntPtr dwUser);	
Parameter	[out] IRealHandle	Return value of NETClient.RealPlay.
	[out] dwDataType	Data type: <ul style="list-style-type: none"> 0: Initial data. 2: YUV data.
	[out] pBuffer	Address of monitoring data block.
	[out] dwBufSize	Length of the monitoring data block. The unit is byte.
	[out] param	Callback parameter structure. Different dwDataType value corresponds to different type. <ul style="list-style-type: none"> The param is blank pointer when dwDataType is 0. The param is the pointer of tagCBYUVDataParam structure when dwDataType is 2.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.4 fAudioDataCallBack

Item	Description	
Name	Callback of audio data of voice talk.	
Function	public delegate void fAudioDataCallBack(IntPtr ITalkHandle, IntPtr pDataBuf, uint dwBufSize, byte byAudioFlag, IntPtr dwUser);	
Parameter	[out] ITalkHandle	Return value of NETClient.StartTalk.
	[out] pDataBuf	Address of audio data block.
	[out] dwBufSize	Length of the audio data block. The unit is byte.
	[out] byAudioFlag	Data type: <ul style="list-style-type: none"> 0: Local collecting. 1: Device sending.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.5 fDownloadPosCallBack

Item	Description	
Name	Progress callback of playback and download by file.	
Function	<pre>public delegate void fDownloadPosCallBack(IntPtr IPlayHandle, uint dwTotalSize, uint dwDownloadSize, IntPtr dwUser);</pre>	
Parameter	[out]IPlayHandle	Return value of playback or download.
	[out]dwTotalSize	Total size. The unit is KB.
	[out]dwDownloadSize	The downloaded size. The unit is KB <ul style="list-style-type: none"> • -1: Current playback stopped. • -2: Failed to write file.
	[out]dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.6 fDataCallBack

Item	Description	
Name	Callback of playback and download data.	
Function	<pre>public delegate int fDataCallBack(IntPtr IRealHandle, uint dwDataType, IntPtr pBuffer, uint dwBufSize, IntPtr dwUser);</pre>	
Parameter	[out]IPlayHandle	Return value of playback or download interface.
	[out] dwDataType	0 (original data).
	[out] pBuffer	Data buffer.
	[out] dwBufSize	Buffer length. The unit is byte.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.7 fTimeDownloadPosCallBack

Item	Description
Name	Callback of download by time.
Function	public delegate void fTimeDownloadPosCallBack(

Item	Description	
	IntPtr IPlayHandle, uint dwTotalSize, uint dwDownloadSize, int index, NET_RECORDFILE_INFO recordfileinfo, IntPtr dwUser);	
Parameter	[out] IPlayHandle	Return value of download interface.
	[out] dwTotalSize	Total size of playback. The unit is KB.
	[out] dwDownloadSize	The size that has been played. The unit is KB. <ul style="list-style-type: none"> • -1: Current download finished. • -2: Write file failed.
	[out] index	Index.
	[out] recordfileinfo	Record file information.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.8 fMessCallBackEx

Item	Description	
Name	Alarm callback.	
Function	<pre> public delegate bool fMessCallBackEx(int ICommand, IntPtr ILoginID, IntPtr pBuf, uint dwBufLen, IntPtr pchDVRIP, int nDVRPort, bool bAlarmAckFlag, int nEventID, IntPtr dwUser); </pre>	
Parameter	[out] ICommand	Alarm type. For details, see Table 4-1.
	[out] ILoginID	Return value of login interface.
	[out] pBuf	Receives the buffer of alarm data. The entered data is different dependent on the listen data and value of ICommand.
	[out] dwBufLen	Length of pBuf. The unit is byte.
	[out] pchDVRIP	Device IP.
	[out] nDVRPort	Device port.
	[out] bAlarmAckFlag	The event can be confirmed when the parameter is TRUE, and the event cannot be confirmed when the parameter is FALSE.

Item	Description	
	[out] nEventID	Assign values for CLIENT_AlarmAck interface. When bAlarmAckFlag is TRUE, the data is valid.
	[out] dwUser	User parameter of the callback.
Return value	<ul style="list-style-type: none"> Success: true. Failure: false. 	
Note	In general, set the callback when initializing. Provide the different treatment to callback dependent on the device ID and command value.	

For the information about alarm type, see Table 4-1.

Table 4-1 Alarm type

Alarm Type	Description	pBuf
MOTION_ALARM_EX	Motion detect alarm	<p>The number of data byte is the same with video channel number. Each byte represents the alarm state by dynamic detection of a video channel.</p> <ul style="list-style-type: none"> 1: Alarm. 0: No alarm.
ALARM_STORAGE_FAILURE	Alarm by hard disk damage	ALARM_STORAGE_FAILURE data group.
VIDEOLOST_ALARM_EX	Video loss alarm	<p>The number of data byte is the same with video channel number. Each byte represents the alarm state by video loss of a video channel.</p> <ul style="list-style-type: none"> 1: Alarm. 0: No alarm.
ALARM_FRONTDISCONNECT	Alarm by IPC disconnection	ALARM_FRONTDISCONNECT_INFO.
ALARM_ALARM_EX	External alarm	<p>The number of data byte is the same with video channel number. Each byte represents the alarm state of alarm channel.</p> <ul style="list-style-type: none"> 1: Alarm. 0: No alarm.

4.9 fCameraStateCallback

Item	Description
Name	Callback of remote device state.
Function	<pre>public delegate void fCameraStateCallBack(IntPtr ILoginID, IntPtr IAttachHandle, IntPtr pBuf, int nBufLen, IntPtr dwUser</pre>

Item	Description	
);	
Parameter	[out] ILoginID	Alarm type.
	[out] IAttachHandle	Return value of subscription.
	[out] pBuf	State of front-end device.
	[out] nBufLen	The length of returned data.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	After subscribing the remote device state, if the state of front-end device changes, the information of changed device will be reported.	

4.10 fAnalyzerDataCallback

Item	Description	
Name	Callback of remote device state.	
Function	<pre>public delegate int fAnalyzerDataCallback(IntPtr IAnalyzerHandle, uint dwEventType, IntPtr pEventInfo, IntPtr pBuffer, uint dwBufSize, IntPtr dwUser, int nSequence, IntPtr reserved);</pre>	
Parameter	[out] IAnalyzerHandle	Return value of NETClient.RealLoadPicture.
	[out] dwEventType	Intelligent event type.
	[out] pEventInfo	Event information buffer.
	[out] pBuffer	Image buffer.
	[out] dwBufSize	Image buffer size.
	[out] dwUser	User data.
	[out] nSequence	Serial number.
	[out] reserved	Reserved.
Return value	None.	
Note	After subscribing the remote device state, if the state of front-end device changes, the information of changed device will be reported.	

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 device 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 device (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 device 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 device network security:

1. Physical Protection

We suggest that you perform physical protection to device, especially storage devices. For example, place the device 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 device (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 device 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 device, 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 device 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 device, 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 device 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.