

NetSDK_C# (Intelligent AI)






Programming Manual



Foreword

Safety Instructions

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

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

Revision History

Version	Revision Content	Release Time
V1.0.1	Updated the dependent library information.	April 2021
V1.0.0	First release.	August 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 appearing in the Manual to help you understand the function of each module.

Term	Explanation
Face detection	Detect the faces and their feature information (age, gender, and expression) through the intelligent analysis of videos.
Face recognition	Detect whether the faces are in the armed face database through the intelligent analysis of videos, including face detection.
Face database	Detect whether the faces are in the face database in real time by importing some face images into IVSS, NVR, camera IPC, and other devices in advance.
ITC	Intelligent Traffic Camera, which can capture vehicle images and automatically analyze traffic events.
Tripwire detection	Detection of crossing the warning line.
Intrusion detection	Detection of objects intruding into the warning zone, including "Crossing region" and "In the region".
People counting	Number of people in the camera calibration region.

Table of Contents

Foreword	I
Glossary.....	III
1 Overview.....	1
1.1 General	1
1.2 Applicability	2
1.3 Application Scenarios	2
1.3.1 Face Detection/Face Recognition/Human Detection	2
1.3.2 People Counting	3
1.3.3 Intelligent Traffic	4
1.3.4 General Behavior	5
1.3.5 Turnstile	6
2 Function Modules	8
2.1 SDK Initialization.....	8
2.1.1 Introduction	8
2.1.2 Interface Overview.....	8
2.1.3 Process	8
2.1.4 Sample Code	9
2.2 Device Login.....	10
2.2.1 Introduction	10
2.2.2 Interface Overview.....	10
2.2.3 Process	11
2.2.4 Sample Code	12
2.3 Real-time Monitoring	12
2.3.1 Introduction	12
2.3.2 Interface Overview.....	13
2.3.3 Process	13
2.3.4 Sample Code	17
2.4 Subscribing Intelligent Event.....	18
2.4.1 Introduction	18
2.4.2 Interface Overview.....	18
2.4.3 Process	19
2.4.4 Sample Code	20
3 Face Detection	22
3.1 Subscription to Event.....	22
3.1.1 Introduction	22
3.1.2 Process Description.....	22
3.1.3 Enumeration and Structure	22
3.2 Sample Code	22
4 Face Recognition	24
4.1 Subscription to Event.....	24
4.1.1 Introduction	24
4.1.2 Process Description.....	24
4.1.3 Enumeration and Structure	24
4.2 Sample Code	24

5 General Behavior	27
5.1 Subscription to Event.....	27
5.1.1 Introduction	27
5.1.2 Process Description.....	27
5.1.3 Enumeration and Structure	27
5.2 Sample Code	27
6 Human Detection.....	29
6.1 Subscription to Event.....	29
6.1.1 Introduction	29
6.1.2 Process Description.....	29
6.1.3 Enumeration and Structure	29
6.2 Sample Code	29
7 Thermal Imaging Temperature Measurement Event	31
7.1 Subscription to Event.....	31
7.1.1 Introduction	31
7.1.2 Process Description.....	31
7.1.3 Enumeration and Structure	31
7.2 Sample Code	31
8 Access Control Event.....	34
8.1 Subscription to Event.....	34
8.1.1 Introduction	34
8.1.2 Process Description.....	34
8.1.3 Enumeration and Structure	34
8.2 Sample Code	34
9 Intelligent Traffic	36
9.1 Subscription to Events	36
9.1.1 Introduction	36
9.1.2 Process Description.....	36
9.1.3 Enumeration and Structural Body	36
9.2 Sample Code	38
10 Person and ID Card Comparison Event	52
10.1 Subscription to Event	52
10.1.1 Introduction	52
10.1.2 Process Description.....	53
10.1.3 Enumeration and Structure	53
10.2 Sample Code	53
11 People Counting	56
11.1 Subscription to Event	56
11.1.1 Introduction	56
11.1.2 Interface Overview.....	56
11.1.3 Process Description.....	57
11.1.4 Sample Code	58
12 Interface Definition	60
12.1 NetSDK Initialization	60
12.1.1 NetSDK Initialization Init.....	60
12.1.2 NetSDK Cleanup	60
12.1.3 SetAutoReconnect	60

12.1.4 SetNetworkParam	61
12.2 Device Login	61
12.2.1 LoginWithHighLevelSecurity	61
12.2.2 Logout.....	62
12.3 Real-time Monitoring	62
12.3.1 RealPlay	62
12.3.2 StopRealPlay	63
12.3.3 SaveRealData	63
12.3.4 StopSaveRealData	64
12.3.5 SetRealDataCallBack	64
12.4 Subscribing Intelligent Event.....	64
12.4.1 NETClient.RealLoadPicture	64
12.4.2 NETClient.StopLoadPic	65
12.5 People Counting	65
12.5.1 AttachVideoStatSummary	65
12.5.2 DetachVideoStatSummary	66
13 Callback Function Definition	67
13.1 Note	67
13.2 fDisconnect.....	67
13.3 fHaveReConnect	67
13.4 fRealDataCallBack	68
13.5 fAnalyzerDataCallBack	68
13.6 fVideoStatSumCallBack	69
Appendix 1 Cybersecurity Recommendations	70

1 Overview

1.1 General

This document provides reference information of the packaging engineering NetSDKCS for the C# NetSDK library, including main functions, interface functions, and callback.

The main functions include general functions, face detection, face recognition, general behavior event, human detection, thermal imaging temperature measurement, access control event, people counting, intelligent traffic, person and ID card comparison.

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

Table 1-1 Files in the NetSDK library

Library type	Library file name	Library file description
Function library	dhnet sdk.dll	Library file
	avnet sdk.dll	Library file
Configuration library	dhconfig sdk.dll	Library file
Play (encoding/decoding) auxiliary library	dhplay.dll	Play library
	fish eye.dll	Fisheye correction library
Dhnet sdk Auxiliary Library	lvsDrawer.dll	Image display library
	StreamConvertor.dll	Transcoding library

- For files included in the C# packaging engineering, see Table 1-2.

Table 1-2 Files in the NetSDKCS engineering

File name	File Description
NetSDK.cs	For packaging C# interfaces to be called by customers
NetSDKStruct.cs	For storing the used structure enumeration.
OriginalSDK.cs	For introducing C interfaces in the NetSDK library into the C# engineering.



- The function library and configuration library of NetSDK are necessary libraries.
- The function library is the main body of NetSDK, which is used for communication interaction between client and products, remote control, search, configuration, acquisition and processing of stream data.
- NetSDK library is the foundation of NetSDKCS engineering. The reference path of NetSDK library is defined in the OriginalSDK.cs file. In actual use, put the NetSDK library in the corresponding path. Users can customize the reference path.
- Customers can directly reference this packaging engineering in their own engineering, or put the files in the packaging engineering in their own engineering for use, or refer to this packaging engineering for packaging.

1.2 Applicability

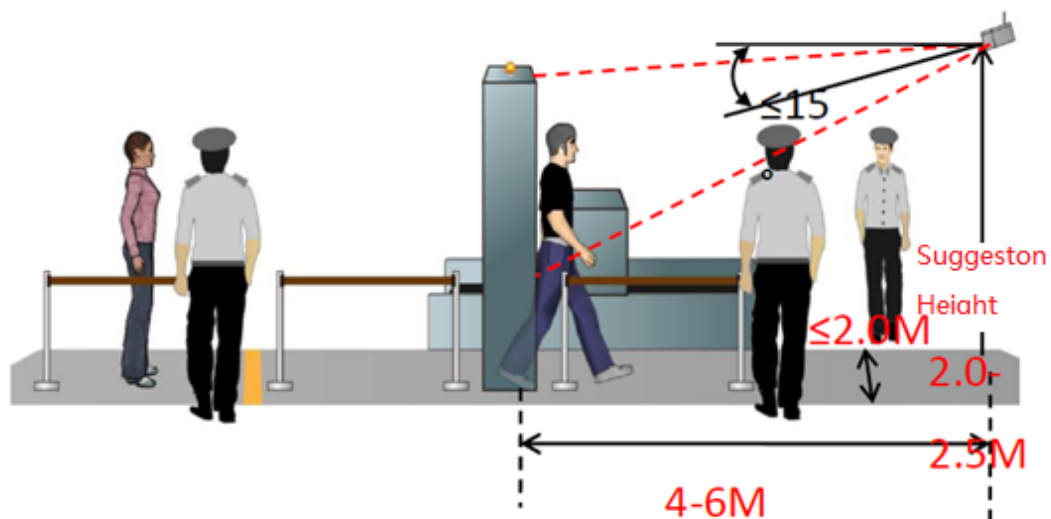
- Recommended memory: No less than 512 M.
- System supported by SDK:
Windows 10/ Windows 8.1/ Windows 7 and Windows Server 2008/ 2003.

1.3 Application Scenarios

1.3.1 Face Detection/Face Recognition/Human Detection

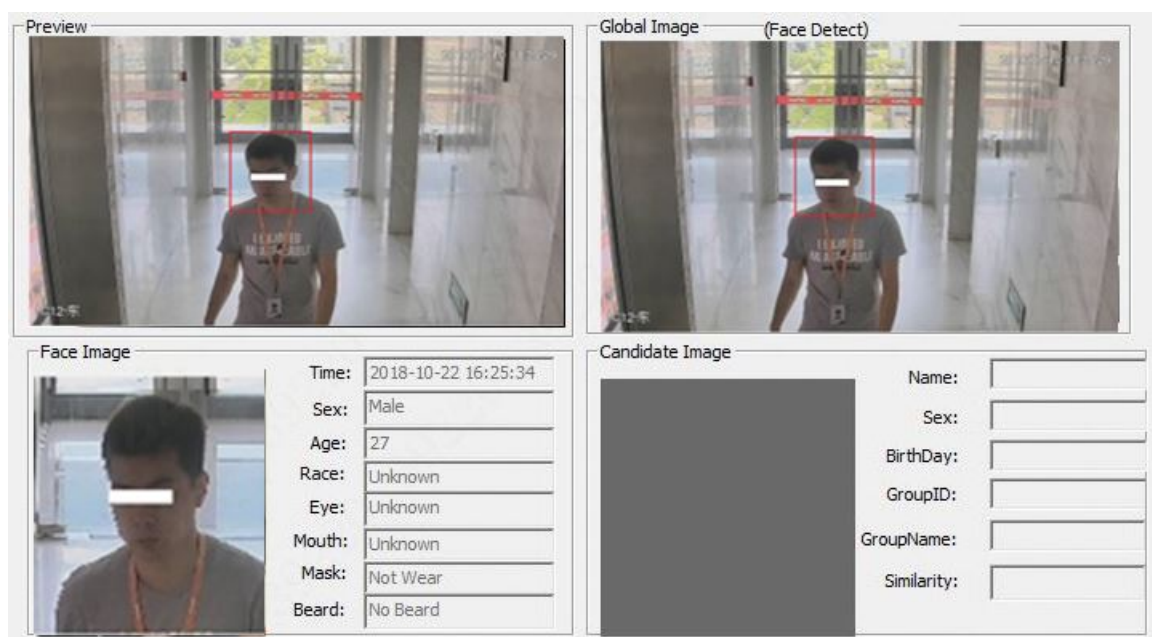
For the application scenarios of face detection, face recognition, and human recognition devices, see Figure 1-1.

Figure 1-1 Face recognition



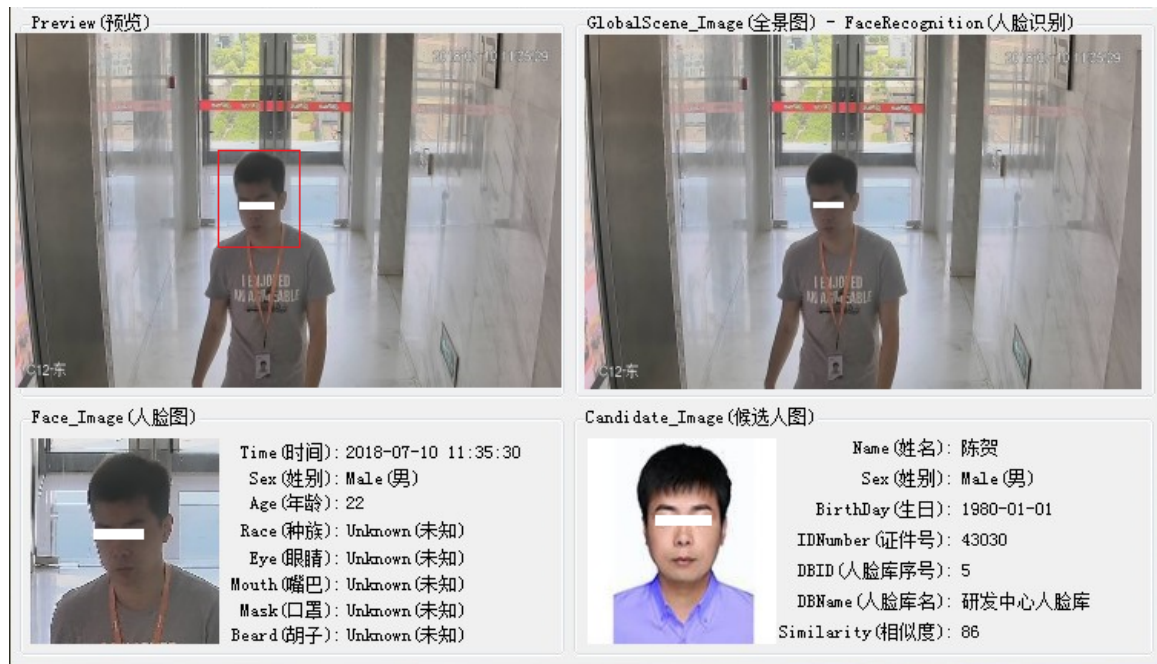
For the face detection scenario, see Figure 1-2.

Figure 1-2 Face detection scenario



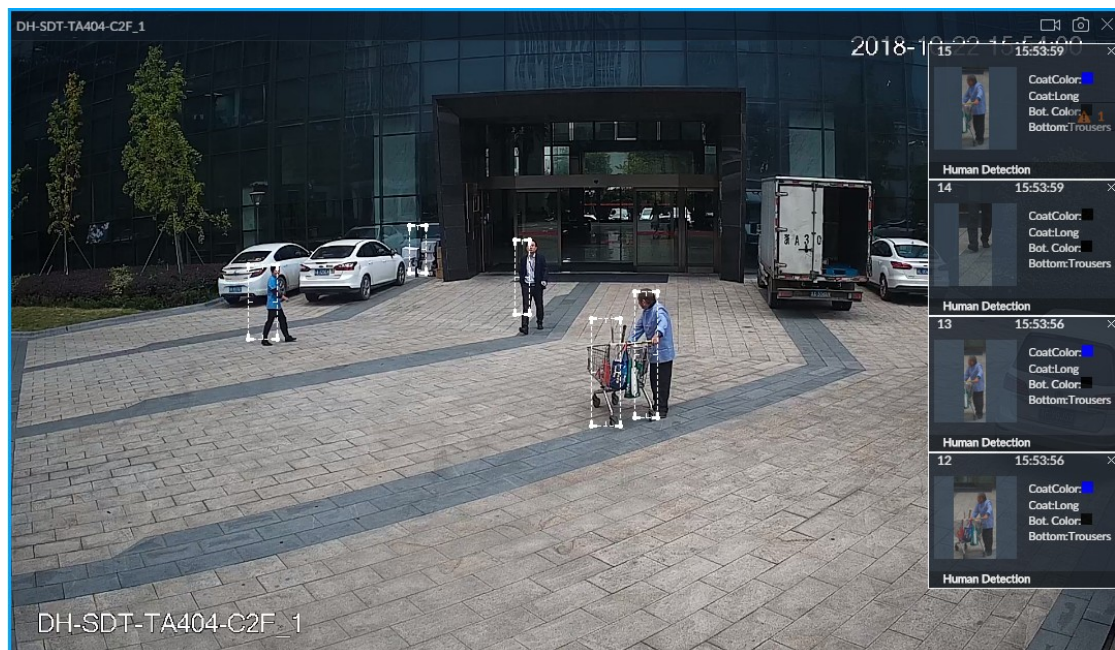
For the face recognition scenario, see Figure 1-3.

Figure 1-3 Face recognition scenario



For the human detection scenario, see Figure 1-4.

Figure 1-4 Human detection scenario



1.3.2 People Counting

For the application of people counting products in the actual scenario, see Figure 1-5.

Figure 1-5 People counting



1.3.3 Intelligent Traffic

ITC at the intersection is used for capturing traffic violations and traffic flow statistics, see Figure 1-6.

Figure 1-6 Applications of ITC at the intersection

Device Login(设备登录)

IP(设备IP): Port(端口): Name(用户): admin Pwd(密码): Logout(登出)

Operate(操作)

channel(通道): 1 StopReal(停止监视) OpenStrobe(打开道闸) UnSubscribeEvent(取消订阅事件) ManualSnap(手动抓拍)

Vehicle Info(车辆信息)

PlateType 车牌类型: Normal

PlateColor 车牌颜色: Blue

VehicleType 车身类型: Unknown

VehicleColor 车身颜色: Green

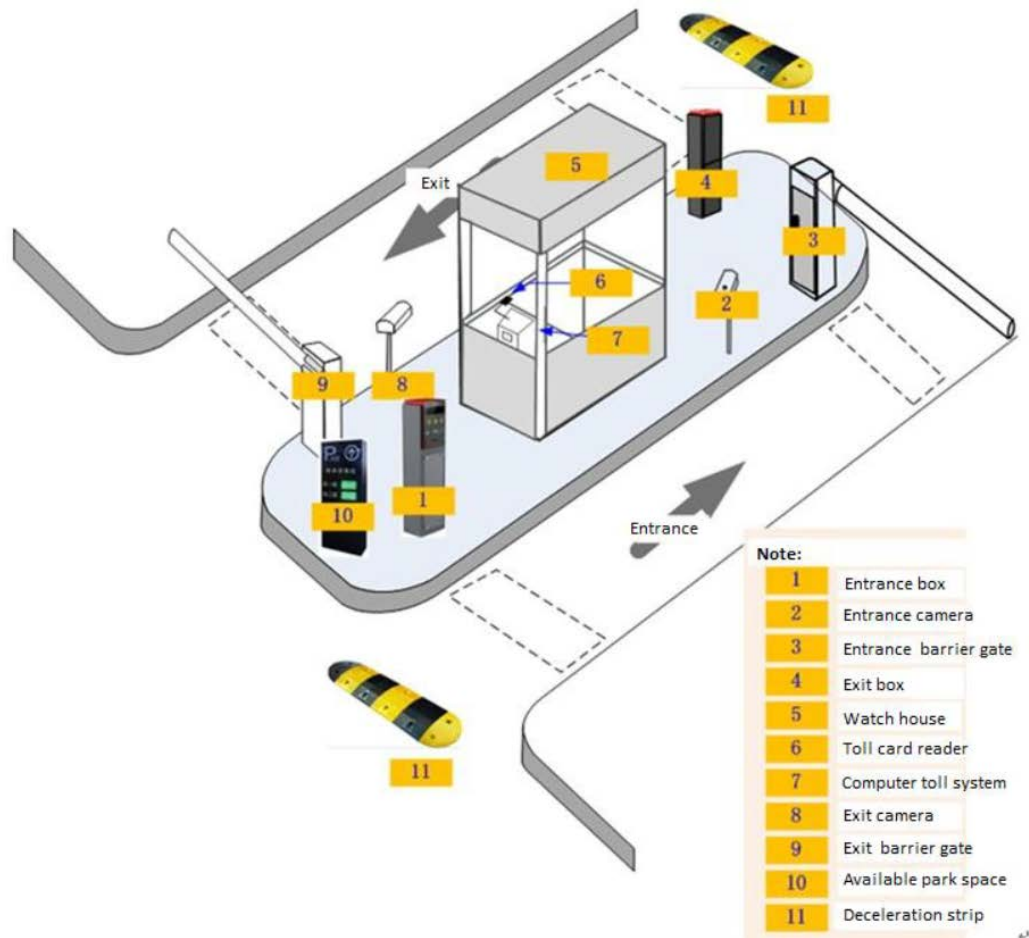
PlateNumber 车牌号:

LaneNumber 车道号: 4

ID(序号)	Time(时间)	Event(事件)	GroupID(组ID)	Index(编号)	Count(总数)	PlateNumber...	PlateType
6	2020-07-04 ...	junction(卡口)		1	1	浙AV 1	Normal
5	2020-07-04 ...	junction(卡口)		1	1	皖A7 3	Normal
4	2020-07-04 ...	junction(卡口)		1	1	浙AL 1	Normal
3	2020-07-04 ...	junction(卡口)		1	1	浙AX 7	Normal
2	2020-07-04 ...	over speed(超速)		1	1	浙A1 N	Normal
1	2020-07-04 ...	over speed(超速)		1	1	浙A5 9	Normal

ITC at the entrance and exit of the parking lot is used for controlling vehicles for entering and exiting the parking lot and monitoring whether there are parking spaces available. See Figure 1-7.

Figure 1-7 Applications of ITC at the entrance and exit of the parking lot



1.3.4 General Behavior

Corresponding alarm event is triggered when a person or vehicle crosses the rule line (tripwire) or intrudes into the warning zone (intrusion), while the target object (person or vehicle) can be distinguished.

For the application scenarios of general behaviors, see Figure 1-8 and Figure 1-9.

Figure 1-8 General behavior scenario—tripwire



Figure 1-9 General behavior scenario—intrusion



1.3.5 Turnstile

The access control turnstile is mainly applied in parks, scenic areas, schools, residence areas, and office buildings. After the collected face images and personnel information is uploaded to the platform, the platform issues the data to the turnstile system.

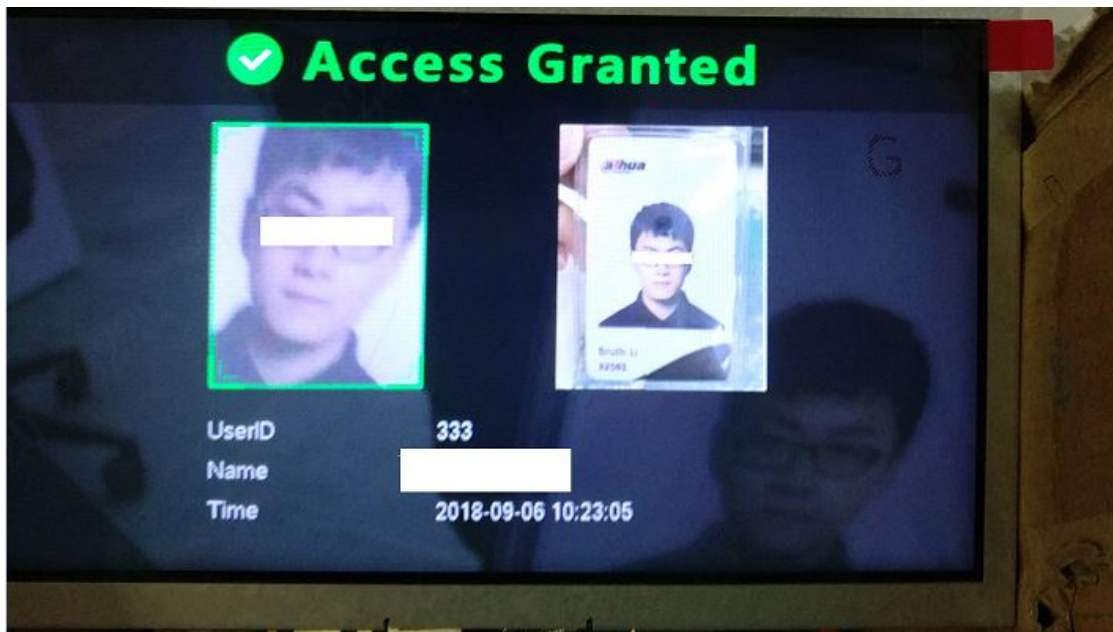
For the appearance of the access control turnstile, see Figure 1-10.

Figure 1-10 Swing turnstile appearance



You can unlock the turnstile by face or swiping card. For unlocking by face, see Figure 1-11.

Figure 1-11 Unlocking by face



2 Function Modules

2.1 SDK Initialization

2.1.1 Introduction

Initialization is the first step of SDK to conduct all the function modules. It does not have the surveillance function but can some parameters configured will affect the SDK overall functions.

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

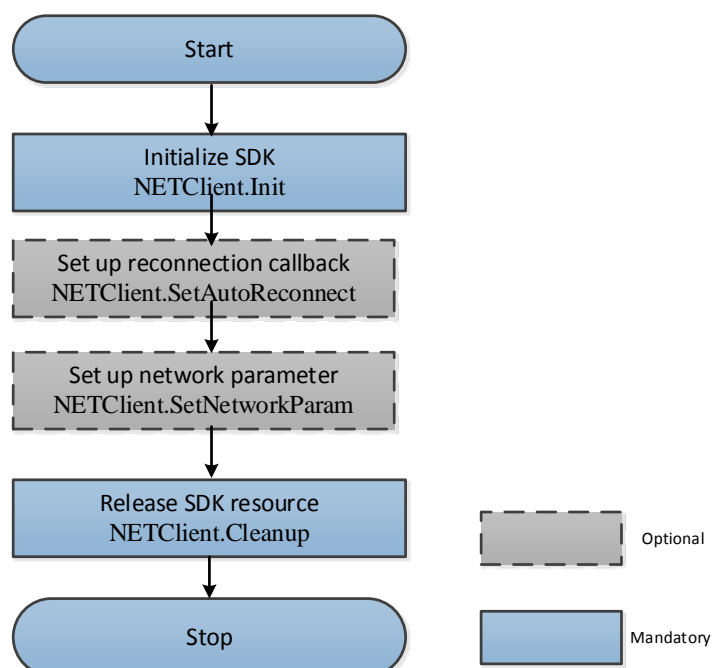
2.1.2 Interface Overview

Table 2-1 Interfaces of SDK initialization

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

2.1.3 Process

Figure 2-1 Process of SDK initialization



Process Description

- Step 1 Call **CLIENT_Init** to initialize SDK.
- 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 SDK functions, call **NETClient.Cleanup** to release SDK 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, alarm and snapshot subscription can be resumed after reconnection is successful.

2.1.4 Sample Code

```
// Announce static callback entrust (for common reentrusting, release before callback might occur)
private static fDisconnectCallBack m_DisConnectCallBack;    //Disconnection callback
private static fHaveReConnectCallBack m_ReConnectCallBack; //Reconnection callback

//Realize entrusting
m_DisConnectCallBack = new fDisconnectCallBack(DisConnectCallBack);
m_ReConnectCallBack = new fHaveReConnectCallBack(ReConnectCallBack);

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

//Configure disconnection callback
NETClient.SetAutoReconnect(m_ReConnectCallBack, IntPtr.Zero);

// Configure network parameter
NET_PARAM param = new NET_PARAM()
{
    nWaittime = 10000, //Waiting timeout duration (ms)
    nConnectTime = 5000, // Connection timeout duration (ms)
}
```



```
};  
NETClient.SetNetworkParam(param);  
  
// Clean up initialization resource  
NETClient.Cleanup();
```

2.2 Device Login

2.2.1 Introduction

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

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

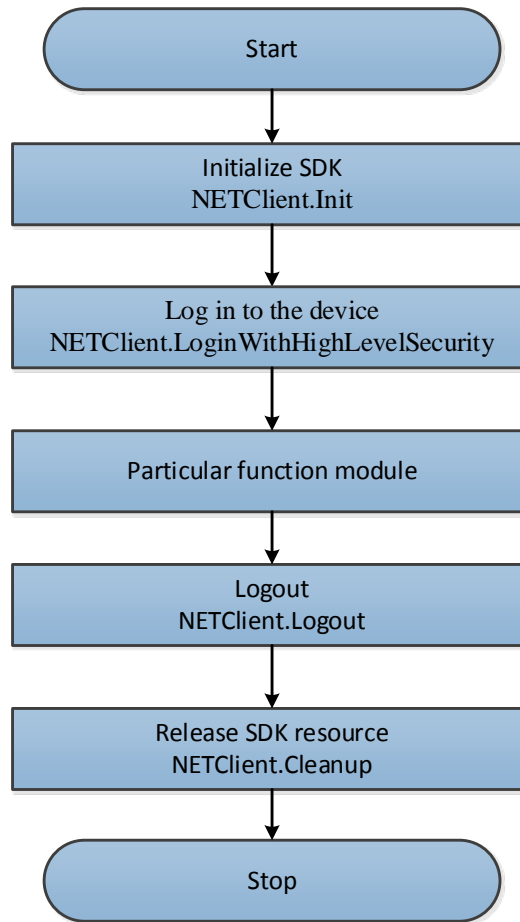
2.2.2 Interface Overview

Table 2-2 Interfaces of device login

Interface	Implication
NETClient.LoginWithHighLevelSecurity	Login interface.
NETClient.Logout	Logout interface.

2.2.3 Process

Figure 2-2 Proces of login



Process Description

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

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

2.2.4 Sample Code

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

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

2.3 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 for you to perform 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.
NETClient.StopRealPlay	Stop real-time monitoring.
NETClient.SaveRealData	Start saving the real-time monitoring data to the local path.
NETClient.StopSaveRealData	Stop saving the real-time monitoring data to the local path.
NETClient.SetRealDataCallBack	Set real-time monitoring data callback.

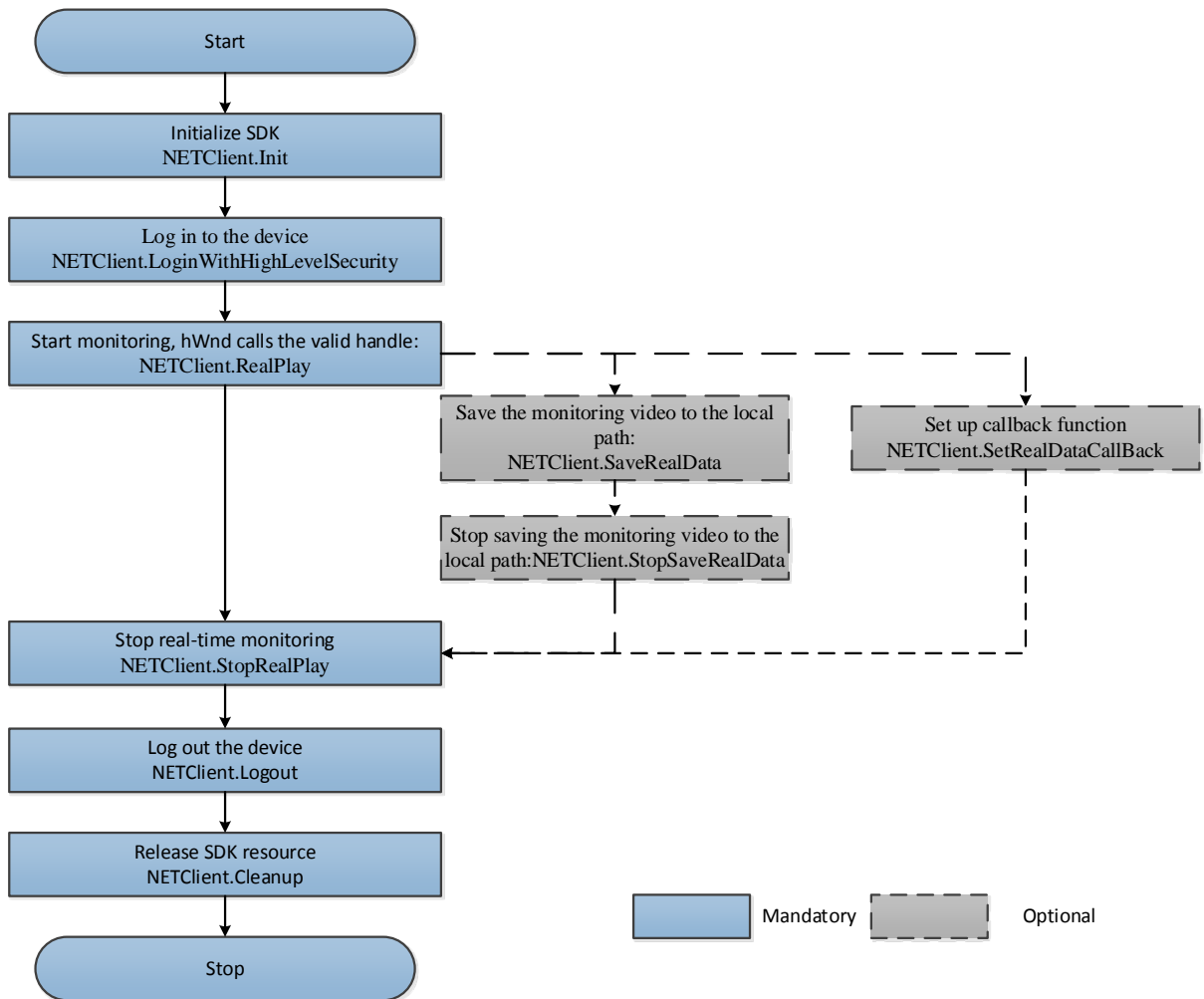
2.3.3 Process

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

2.3.3.1 SDK Decoding Play

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 SDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to login the device.
- Step 3 Call **NETClient.RealPlay** to enable 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 forward the video file. If save the video file, see the step 4 and step 5.
- Step 7 After completing the real-time monitoring, call **NETClient.StopRealPlay** to stop real-time monitoring.
- Step 8 After using the function module, call **NETClient.Logout** to logout the device.
- Step 9 After using all SDK functions, call **NETClient.Cleanup** to release SDK resource.

Notes for Process

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

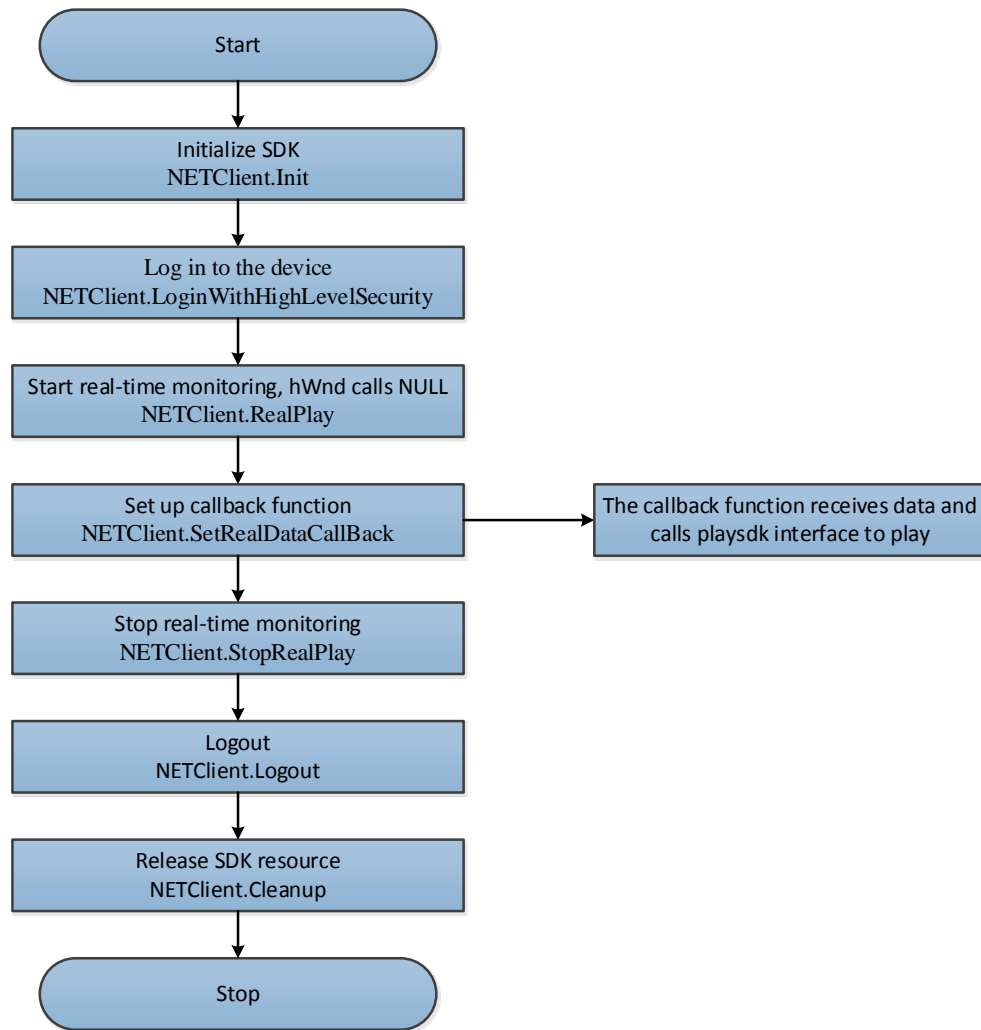
```
NET_PARAM param = new NET_PARAM()
{
    nGetConnInfoTime = 5000, //Get sub connection timeout duration (ms)
};
NETClient.SetNetworkParam(param);
```

- 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 have already opened the main stream video on the channel 1 and still want to open the sub stream video on the same channel, you can close the main stream first and then open the sub stream.
 - ◇ Login twice to obtain two login handles to deal with the main stream and sub stream respectively.
- Calling succeeded but no image: 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 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 library



Process Description

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to login the device.
- Step 3 After successful login, call **NETClient.RealPlay** to enable 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 completing the real-time monitoring, call **NETClient.StopRealPlay** to stop real-time monitoring.
- Step 7 After using the function module, call **NETClient.Logout** to logout the device.
- Step 8 After using all SDK functions, call **NETClient.Cleanup** to release SDK resource.

Notes for Process

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

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

2.3.4 Sample Code

2.3.4.1 NetSDK Decoding Play

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

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

2.3.4.2 Call Third Party Library

```
Take opening the main stream monitoring of channel 1 as an example.
IntPtr m_RealPlayID = NETClient.RealPlay(m_LoginID, nChannel, null, EM_RealPlayType.Realplay);
if (IntPtr.Zero == m_RealPlayID)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

//Configure real-time monitoring callback function.
private static fRealDataCallBackEx2 m_RealDataCallBackEx2;
m_RealDataCallBackEx2 = new fRealDataCallBackEx2(RealDataCallBackEx);
NETClient.SetRealDataCallBack(m_RealPlayID, m_RealDataCallBackEx2, IntPtr.Zero,
EM_REALDATA_FLAG.DATA_WITH_FRAME_INFO | EM_REALDATA_FLAG.PCM_AUDIO_DATA |
EM_REALDATA_FLAG.RAW_DATA | EM_REALDATA_FLAG.YUV_DATA);
private void RealDataCallBackEx(IntPtr lRealHandle, uint dwDataType, IntPtr pBuffer, uint dwBufSize,
IntPtr param, IntPtr dwUser)
{

```



```

EM_REALDATA_FLAG type = (EM_REALDATA_FLAG)dwDataType;
switch (type)
{
    case EM_REALDATA_FLAG.RAW_DATA:
        //Process operations.
        break;
}

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

```

2.4 Subscribing Intelligent Event

2.4.1 Introduction

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

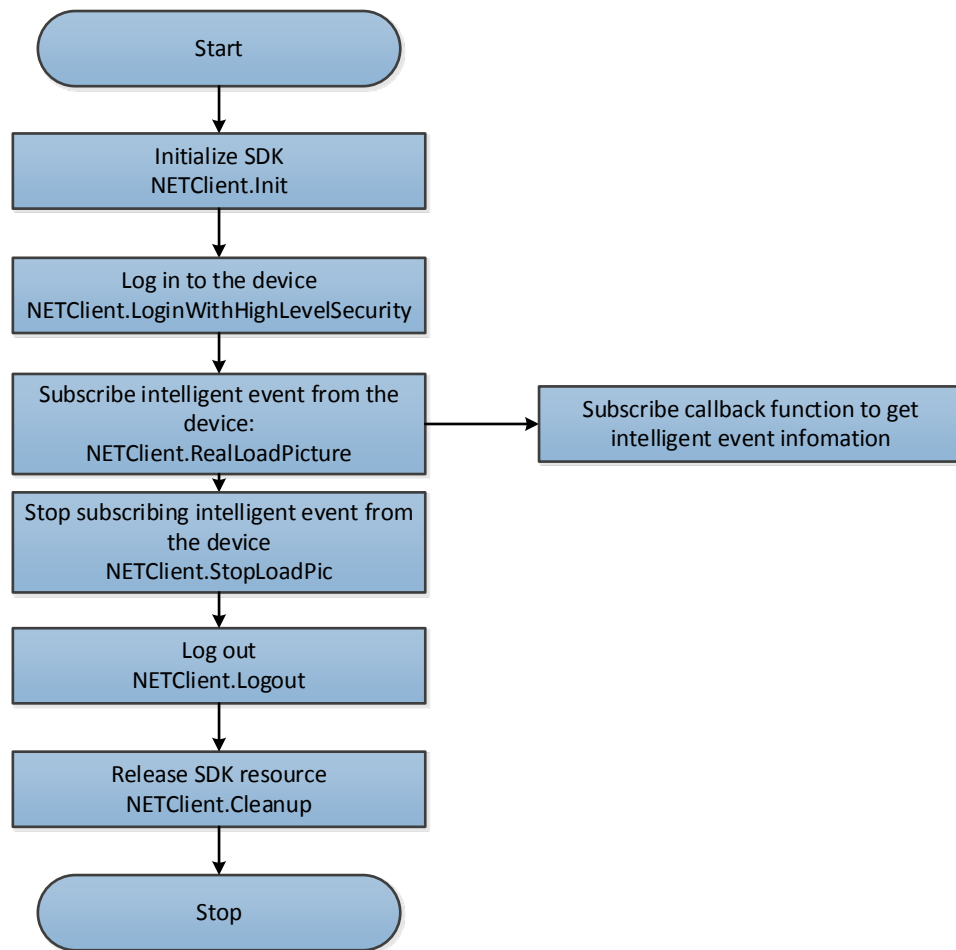
2.4.2 Interface Overview

Table 2-4 Interfaces of subscribing intelligent event

Interface	Implication
NETClient.RealLoadPicture	Subscribe intelligent event.
NETClient.StopLoadPic	Cancel subscribing the intelligent event.

2.4.3 Process

Figure 2-5 Process of uploading face event



Process Description

- Step 1 Call **NETClient.Init** to initialize NetSDK.
- Step 2 Call **NETClient.LoginWithHighLevelSecurity** to login the device.
- Step 3 Call **NETClient.RealLoadPicture** to subscribe intelligent event from the device.
- Step 4 After successful subscribe, call fAnalyzerDataCallBack to upload the intelligent events. Through this function, you can filter out the intelligent events you need.
- Step 5 After using the intelligent event function, call **NETClient.StopLoadPic** to stop subscribing intelligent events.
- Step 6 After using the function module, call **NETClient.Logout** to logout the device.
- Step 7 After using all SDK functions, call **NETClient.Cleanup** to release SDK resource.

Notes for Process

- Support to subscribe single intelligent event and all the intelligent events (EM_EVENT_IVS_TYPE.ALL).
- Set up whether to receive pictures: Network of some devices is 3G or 4G. When NetSDK is connected to the device, if you do not want to receive images, you can set up the value of

bNeedPicFile in the NETClient.RealLoadPicture interface as false. After that, only intelligent traffic event info will be received and images will not be received.

- Call -1 through channel number to subscribe all channels. Some intelligent traffic products do not support all channel subscription. If you failed to call all channels by calling -1, please try to subscribe a single channel.
- When multiple devices subscribe channels at the same time, if you want to differentiate devices, there are two methods. One is to establish mapping relation between device IP, login handle, and subscription handle; if you want to differentiate devices, there are two methods. Method one: establish the mapping relation among device IP, login handle, and subscription handle; the device IP will be located through subscription handle returned by the callback function. Method two: subscribe dwUser called by subscription function will be returned in the callback function.

2.4.4 Sample Code

```
// Intelligent event uploading callback function
private static fAnalyzerDataCallBack m_AnalyzeHandle;
m_AnalyzeHandle = new fAnalyzerDataCallBack(AnalyzerDataCallBack);

//Subscribe intelligent traffic event upload
//The sample subscribed crowd density detection event, and m_LoginID is device login handle.
Subscribe channel 0 will return event images.
IntPtr realLoadPicturHandle = NETClient.RealLoadPicture(m_LoginID, 0,
(uint)EM_EVENT_IVS_TYPE.CROWDDETECTION, true, m_AnalyzeHandle, IntPtr.Zero, IntPtr.Zero);
if (IntPtr.Zero == realLoadPicturHandle)
{
    MessageBox.Show(this, NETClient.GetLastError());
    return;
}

//Cancel intelligent event uploading
NETClient.StopLoadPic(realLoadPicturHandle);

//Intelligent event uploading callback
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)
    {
        // Crowd density detection event
        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 event; ");
            }
        }
    }
}
```

```
Console.WriteLine("Channel No.: " + info.nChannelID + "; ");
Console.WriteLine("Time: " + info.UTC.ToString() + "; ");
/* .....
**Other operations related to the event**
.....*/
});
}
break;
default:
break;
    }
    return 0;
}
```

3 Face Detection

3.1 Subscription to Event

3.1.1 Introduction

When the camera detects the appearance of faces in the specified region, an intelligent event message is generated and reported to NetSDK.

3.1.2 Process Description

This chapter is only about callback of specific events. For event subscription and receiving, see “2.4 Subscribing Intelligent Event”.

3.1.3 Enumeration and Structure

Enumerated value corresponding to the event: EM_EVENT_IVS_TYPE.FACEDETECT

Structure corresponding to the event: NET_DEV_EVENT_FACEDETECT_INFO

Notes

- Complete the face detection event callback in two times, which include panorama and face cutout respectively. Determine whether they are the same event by the stuObject.nRelativeID returned to the structure data.

3.2 Sample Code

```
private int m_GroupID = 0;
    private int AnalyzerDataCallBack(IntPtr IAnalyzerHandle, uint dwEventType, IntPtr
pEventInfo, IntPtr pBuffer, uint dwBufSize, IntPtr dwUser, int nSequence, IntPtr reserved)
    {
        if (m_AnalyzerID == IAnalyzerHandle)
        {
            switch (dwEventType)
            {
                case (uint)EM_EVENT_IVS_TYPE.FACEDETECT:
                {
                    NET_DEV_EVENT_FACEDETECT_INFO info =
(NET_DEV_EVENT_FACEDETECT_INFO)Marshal.PtrToStructure(pEventInfo,
typeof(NET_DEV_EVENT_FACEDETECT_INFO));
                    if (m_GroupID != info.stuObject.nRelativeID)
                    {
                        m_GroupID = info.stuObject.nObjectID;
```

```

        byte[] globalScenePicInfo = new byte[dwBufSize];
        Marshal.Copy(pBuffer, globalScenePicInfo, 0, (int)dwBufSize);
        using (MemoryStream stream = new
MemoryStream(globalScenePicInfo))
        {
            try // add try catch for catch exception when the stream
is not image format,and the stream is from device.
            {
                Image globalScenelImage =
Image.FromStream(stream);
                pictureBox_globalimage.Image =
globalScenelImage;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }
    else
    {
        byte[] personFaceInfo = new byte[dwBufSize];
        Marshal.Copy(pBuffer, personFaceInfo, 0, (int)dwBufSize);
        using (MemoryStream stream = new
MemoryStream(personFaceInfo))
        {
            try // add try catch for catch exception when the stream
is not image format,and the stream is from device.
            {
                Image facelImage = Image.FromStream(stream);
                pictureBox_faceimage.Image = facelImage;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }
    }
    break;
default:
    break;
}
}
return 0;

```

4 Face Recognition

4.1 Subscription to Event

4.1.1 Introduction

When the camera detects the appearance of faces in the specified region, it will search the face database for corresponding faces and data, and an intelligent event message will be generated and reported to NetSDK.

4.1.2 Process Description

This chapter is only about callback of specific events. For event subscription and receiving, see “2.4 Subscribing Intelligent Event”.

4.1.3 Enumeration and Structure

Enumerated value corresponding to the event: EM_EVENT_IVS_TYPE.FACERECOGNITION

Structure corresponding to the event: NET_DEV_EVENT_FACERECOGNITION_INFO

4.2 Sample Code

```
private int AnalyzerDataCallBack(IntPtr IAnalyzerHandle, uint dwEventType, IntPtr
pEventInfo, IntPtr pBuffer, uint dwBufSize, IntPtr dwUser, int nSequence, IntPtr reserved)
{
    if (m_AnalyzerID == IAnalyzerHandle)
    {
        switch (dwEventType)
        {
            case (uint)EM_EVENT_IVS_TYPE.FACERECOGNITION:
            {
                NET_DEV_EVENT_FACERECOGNITION_INFO info =
                (NET_DEV_EVENT_FACERECOGNITION_INFO)Marshal.PtrToStructure(pEventInfo,
                typeof(NET_DEV_EVENT_FACERECOGNITION_INFO));
                if (IntPtr.Zero != pBuffer && dwBufSize > 0)
                {
                    if (info.bGlobalScenePic)
                    {
                        if (info.stuGlobalScenePicInfo.dwFileLenth > 0)
                        {
                            byte[] globalScenePicInfo = new
                            byte[info.stuGlobalScenePicInfo.dwFileLenth];
```

```

        Marshal.Copy(IntPtr.Add(pBuffer,
(int)info.stuGlobalScenePicInfo.dwOffSet), globalScenePicInfo, 0,
(int)info.stuGlobalScenePicInfo.dwFileLenth);

        using (MemoryStream stream = new
MemoryStream(globalScenePicInfo))
        {
            try // add try catch for catch exception when
the stream is not image format,and the stream is from device.
            {
                Image globalScenelImage =
Image.FromStream(stream);

                pictureBox_image.Image =
globalScenelImage;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }
    if (info.stuObject.stPicInfo.dwFileLenth > 0)
    {
        byte[] personFaceInfo = new
byte[info.stuObject.stPicInfo.dwFileLenth];
        Marshal.Copy(IntPtr.Add(pBuffer,
(int)info.stuObject.stPicInfo.dwOffSet), personFaceInfo, 0, (int)info.stuObject.stPicInfo.dwFileLenth);
        using (MemoryStream stream = new
MemoryStream(personFaceInfo))
        {
            try // add try catch for catch exception when the
stream is not image format,and the stream is from device.
            {
                Image facelImage = Image.FromStream(stream);
                pictureBox_faceimage.Image = facelImage;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }
    if (info.nCandidateNum > 0)
    {
        var candidatesInfo =
info.stuCandidates.ToList().OrderByDescending(p => p.bySimilarity).ToArray();

```



```

NET_CANDIDATE_INFO maxSimilarityPersonInfo =
candidatesInfo[0];

        if
(maxSimilarityPersonInfo.stPersonInfo.szFacePicInfo[0].dwFileLenth > 0)
        {
            byte[] candidateInfo = new
byte[maxSimilarityPersonInfo.stPersonInfo.szFacePicInfo[0].dwFileLenth];
            Marshal.Copy(IntPtr.Add(pBuffer,
(int)maxSimilarityPersonInfo.stPersonInfo.szFacePicInfo[0].dwOffSet), candidateInfo, 0,
(int)maxSimilarityPersonInfo.stPersonInfo.szFacePicInfo[0].dwFileLenth);
            using (MemoryStream stream = new
MemoryStream(candidateInfo))
            {
                try // add try catch for catch exception when
the stream is not image format,and the stream is from device.
                {
                    Image candidateImage =
Image.FromStream(stream);
                    pictureBox_candidateimage.Image =
candidateImage;
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex);
                }
            }
        }
        break;
    default:
        break;
    }
}
return 0;
}

```

5 General Behavior

5.1 Subscription to Event

5.1.1 Introduction

General behaviors mainly include intrusion and tripwire. Intrusion indicates that an alarm is triggered when a person who intrudes into or crosses the specified region is detected. Tripwire indicates that an alarm is triggered when a person who crosses the line set by the camera is detected.

5.1.2 Process Description

This chapter is only about callback of specific events. For event subscription and receiving, see “2.4 Subscribing Intelligent Event”.

5.1.3 Enumeration and Structure

Enumerated value corresponding to the intrusion: EM_EVENT_IVS_TYPE.CROSSREGIONDETECTION

Structure corresponding to the intrusion: NET_DEV_EVENT_CROSSREGION_INFO

Enumerated value corresponding to the tripwire: EM_EVENT_IVS_TYPE.CROSSLINEDETECTION

Structure corresponding to the tripwire: NET_DEV_EVENT_CROSSLINE_INFO

5.2 Sample Code

```
private int AnalyzerDataCallBack(IntPtr IAnalyzerHandle, uint dwEventType, IntPtr
pEventInfo, IntPtr pBuffer, uint dwBufSize, IntPtr dwUser, int nSequence, IntPtr reserved)
{
    if (m_AnalyzerID == IAnalyzerHandle)
    {
        switch (dwEventType)
        {
            // cross region event
            case (uint)EM_EVENT_IVS_TYPE.CROSSREGIONDETECTION:
            {
                NET_DEV_EVENT_CROSSREGION_INFO info =
                (NET_DEV_EVENT_CROSSREGION_INFO)Marshal.PtrToStructure(pEventInfo,
                typeof(NET_DEV_EVENT_CROSSREGION_INFO));
                this.BeginInvoke(new Action(() =>
                {
                    var list_item = new ListViewItem();
                    list_item.Text = info.nEventID.ToString();
                    list_item.SubItems.Add(info.nChannelID.ToString());
                    list_item.SubItems.Add(info.UTC.ToString());
```

```

        // Please take out additional information as needed
        realLoad_listView.BeginUpdate();
        realLoad_listView.Items.Add(list_item);
        if (realLoad_listView.Items.Count > ListViewCount)
        {
            realLoad_listView.Items.RemoveAt(ListViewCount);
        }
        realLoad_listView.EndUpdate();

        ShowToPicCtrl(pBuffer, dwBufSize);
    });
}
break;
// cross line event
case (uint)EM_EVENT_IVS_TYPE.CROSSLINEDETECTION:
{
    NET_DEV_EVENT_CROSSLINE_INFO info =
(NET_DEV_EVENT_CROSSLINE_INFO)Marshal.PtrToStructure(pEventInfo,
typeof(NET_DEV_EVENT_CROSSLINE_INFO));
    this.BeginInvoke(new Action(() =>
    {
        var list_item = new ListViewItem();
        list_item.Text = info.nEventID.ToString();
        list_item.SubItems.Add(info.nChannelID.ToString());
        list_item.SubItems.Add(info.UTC.ToString());

        // Please take out additional information as needed
        realLoad_listView.BeginUpdate();
        realLoad_listView.Items.Add(list_item);
        if (realLoad_listView.Items.Count > ListViewCount)
        {
            realLoad_listView.Items.RemoveAt(ListViewCount);
        }
        realLoad_listView.EndUpdate();

        ShowToPicCtrl(pBuffer, dwBufSize);
    }));
}
break;
default:
    break;
}
}
return 0;
}

```

6 Human Detection

6.1 Subscription to Event

6.1.1 Introduction

When the camera detects human features in the specified region, an intelligent event message is generated and reported to NetSDK.

6.1.2 Process Description

This chapter is only about callback of specific events. For event subscription and receiving, see “2.4 Subscribing Intelligent Event”.

6.1.3 Enumeration and Structure

Enumerated value corresponding to the event: EM_EVENT_IVS_TYPE.HUMANTRAIT

Structure corresponding to the event: NET_DEV_EVENT_HUMANTRAIT_INFO

6.2 Sample Code

```
private int AnalyzerDataCallBack(IntPtr IAnalyzerHandle, uint dwEventType, IntPtr
pEventInfo, IntPtr pBuffer, uint dwBufSize, IntPtr dwUser, int nSequence, IntPtr reserved)
{
    if (m_AnalyzerID == IAnalyzerHandle)
    {
        switch (dwEventType)
        {
            //Human feature event
            case (uint)EM_EVENT_IVS_TYPE.HUMANTRAIT:
            {
                NET_DEV_EVENT_HUMANTRAIT_INFO info =
                (NET_DEV_EVENT_HUMANTRAIT_INFO)Marshal.PtrToStructure(pEventInfo,
                typeof(NET_DEV_EVENT_HUMANTRAIT_INFO));
                string pic_name = String.Format("./image/{0}_", info.nGroupID);
                // Save panoramic image
                if (info.stuSceneImage.nLength > 0)
                {
                    byte[] bytes = new byte[info.stuSceneImage.nLength];
                    Marshal.Copy(new IntPtr(pBuffer.ToInt32() +
                    info.stuSceneImage.nOffset), bytes, 0, (int)info.stuSceneImage.nLength);
                    pic_name = pic_name + "Panorama.jpg";
```

```

WriteFile(bytes, pic_name,
(int)info.stuScenelImage.nLength);
}
// Save face image
if (info.stuFacelImage.nLength > 0)
{
    pic_name = pic_name + "Face image.jpg";
    byte[] bytes = new byte[info.stuHumanImage.nLength];
    Marshal.Copy(new IntPtr(pBuffer.ToInt32() +
info.stuHumanImage.nOffSet), bytes, 0, (int)info.stuHumanImage.nLength);
    WriteFile(bytes, pic_name,
(int)info.stuHumanImage.nLength);
}
// Save face panorama
if (info.stuFaceScenelImage.nLength > 0)
{
    pic_name = pic_name + "Face panorama.jpg";
    byte[] bytes = new byte[info.stuFaceScenelImage.nLength];
    Marshal.Copy(new IntPtr(pBuffer.ToInt32() +
info.stuFaceScenelImage.nOffSet), bytes, 0, (int)info.stuFaceScenelImage.nLength);
    WriteFile(bytes, pic_name,
(int)info.stuFaceScenelImage.nLength);
}
//Save human image
if (info.stuHumanImage.nLength > 0)
{
    pic_name = pic_name + "Human image.jpg";
    byte[] bytes = new byte[info.stuHumanImage.nLength];
    Marshal.Copy(new IntPtr(pBuffer.ToInt32() +
info.stuHumanImage.nOffSet), bytes, 0, (int)info.stuHumanImage.nLength);
    WriteFile(bytes, pic_name,
(int)info.stuHumanImage.nLength);
}
}
break;
default:
break;
}
}
return 0;
}

```

7 Thermal Imaging Temperature Measurement

Event

7.1 Subscription to Event

7.1.1 Introduction

When the thermal imaging device TPC detects human in the specified region, it will get and report body temperature through the thermal imaging technology.

7.1.2 Process Description

This chapter is only about callback of specific events. For event subscriprion and receiving, see “2.4 Subscribing Intelligent Event”.

7.1.3 Enumeration and Structure

Enumerated value corresponding to the event: EM_EVENT_IVS_TYPE.ANATOMY_TEMP_DETECT

Structure corresponding to the event: NET_DEV_EVENT_ANATOMY_TEMP_DETECT_INFO

7.2 Sample Code

```
private int AnalyzerDataCallBack(IntPtr IAnalyzerHandle, uint dwEventType, IntPtr
pEventInfo, IntPtr pBuffer, uint dwBufSize, IntPtr dwUser, int nSequence, IntPtr reserved)
{
    if (m_AnalyzerID == IAnalyzerHandle)
    {
        switch (dwEventType)
        {
            // Intelligent body temperature measurement event
            case (uint)EM_EVENT_IVS_TYPE.ANATOMY_TEMP_DETECT:
            {
                NET_DEV_EVENT_ANATOMY_TEMP_DETECT_INFO info =
                (NET_DEV_EVENT_ANATOMY_TEMP_DETECT_INFO)Marshal.PtrToStructure(pEventInfo,
                typeof(NET_DEV_EVENT_ANATOMY_TEMP_DETECT_INFO));
                this.BeginInvoke(new Action(() =>
                {
                    // Display event information
                    nAnatomyTempEventNum++;
                    var list_item = new ListViewItem();
```

```

list_item.Text = nAnatomyTempEventNum.ToString();
list_item.SubItems.Add(info.nChannelID.ToString());
list_item.SubItems.Add(info.nAction.ToString());
list_item.SubItems.Add(info.szName.ToString());
list_item.SubItems.Add(info.nEventID.ToString());
list_item.SubItems.Add(info.nSequence.ToString());
list_item.SubItems.Add(info.nPresetID.ToString());
list_item.SubItems.Add(info.PTS.ToString());

listView_EventInfo.BeginUpdate();
listView_EventInfo.Items.Add(list_item);
listView_EventInfo.EndUpdate();

// Body temperature information in the region
var list_item2 = new ListViewItem();
list_item2.Text = info.stManTempInfo.nObjectID.ToString();

list_item2.SubItems.Add(info.stManTempInfo.dbHighTemp.ToString());

list_item2.SubItems.Add(info.stManTempInfo.nTempUnit.ToString());

list_item2.SubItems.Add(info.stManTempInfo.bIsOverTemp.ToString());

list_item2.SubItems.Add(info.stManTempInfo.bIsUnderTemp.ToString());

listView_ManTempInfo.BeginUpdate();
listView_ManTempInfo.Items.Add(list_item2);
listView_ManTempInfo.EndUpdate();

// Start position of visible image panorama: pBuffer +
stVisScenImage.nOffset

// Length of visible image panorama:
stVisScenImage.nLength

// Display visible image
IntPtr buffer = IntPtr.Add(pBuffer,
(int)info.stVisScenImage.nOffset);
ShowToPicCtrl(buffer, info.stVisScenImage.nLength,
pb_VisScenImage);

// Start position of thermal imaging panorama: pBuffer +
stVisScenImage.nOffset

// Length of thermal imaging panorama:
stThermalScenImage.nLength

// Display image of thermal imaging
IntPtr buffer2 = IntPtr.Add(pBuffer,
(int)info.stThermalScenImage.nOffset);

```

```
                                ShowToPicCtrl(buffer2, info.stThermalScenelImage.nLength,  
pb_ThermalScenelImage);  
                                }));  
                                }  
                                break;  
                                default:  
                                break;  
                                }  
                                }  
                                return 0;  
                                }
```


8 Access Control Event

8.1 Subscription to Event

8.1.1 Introduction

When the access control device is unlocked, the unlocking related event information is reported, including event, unlocking mode, unlocking personnel, and other corresponding information.

8.1.2 Process Description

This chapter is only about callback of specific events. For event subscription and receiving, see “2.4 Subscribing Intelligent Event”.

8.1.3 Enumeration and Structure

Enumerated value corresponding to the event: EM_EVENT_IVS_TYPE.ACCESS_CTL

Structure corresponding to the event: NET_DEV_EVENT_ACCESS_CTL_INFO

8.2 Sample Code

```
private int AnalyzerDataCallBack(IntPtr IAnalyzerHandle, uint dwEventType, IntPtr
pEventInfo, IntPtr pBuffer, uint dwBufSize, IntPtr dwUser, int nSequence, IntPtr reserved)
{
    EM_EVENT_IVS_TYPE type = (EM_EVENT_IVS_TYPE)dwEventType;
    switch (type)
    {
        case EM_EVENT_IVS_TYPE.ACCESS_CTL:
            NET_DEV_EVENT_ACCESS_CTL_INFO info =
            (NET_DEV_EVENT_ACCESS_CTL_INFO)Marshal.PtrToStructure(pEventInfo,
            typeof(NET_DEV_EVENT_ACCESS_CTL_INFO));
            this.BeginInvoke(new Action(() =>
            {
                _EventID++;
                ListViewItem item = new ListViewItem();
                item.Text = _EventID.ToString();
                item.SubItems.Add(info.szUserID);
                item.SubItems.Add(info.szCardNo);
                item.SubItems.Add(info.UTC.ToShortString());
                switch (info.emOpenMethod)
                {
                    case EM_ACCESS_DOOROPEN_METHOD.CARD:
                        item.SubItems.Add("Card");
```

```

        break;
    case EM_ACCESS_DOOROPEN_METHOD.FINGERPRINT:
        item.SubItems.Add("FingerPrint");
        break;
    default:
        item.SubItems.Add("Unknown");
        break;
    }

    listView_event.BeginUpdate();
    listView_event.Items.Insert(0, item);
    if (listView_event.Items.Count > EventLines)
    {
        listView_event.Items.RemoveAt(EventLines);
    }
    listView_event.EndUpdate();

    });
    break;
default:
    break;
}
return 0;
}

```

9 Intelligent Traffic

9.1 Subscription to Events

9.1.1 Introduction

Intelligent traffic devices analyze real-time video streams. When traffic events (traffic violations, parking spot occupied/empty, and more) are detected, the events will be sent to users.

There are three ways that events can be sent: SDK auto connection and subscription of intelligent events from devices (when devices detected events, the events will be sent to SDK)

9.1.2 Process Description

This chapter is only about callback of specific events. For event subscriprion and receiving, see “2.4 Subscribing Intelligent Event”.

9.1.3 Enumeration and Structural Body

Enumeration value for running red lights: EM_EVENT_IVS_TYPE.TRAFFIC_RUNREDLIGHT

Structural body for running red lights: NET_DEV_EVENT_TRAFFIC_RUNREDLIGHT_INFO

Enumeration value for ANPR events: EM_EVENT_IVS_TYPE.TRAFFICJUNCTION

Structural body for ANPR events: NET_DEV_EVENT_TRAFFICJUNCTION_INFO

Enumeration value for overspeed events: EM_EVENT_IVS_TYPE.TRAFFIC_OVERSPEED

Structural body for overspeed events: NET_DEV_EVENT_TRAFFIC_OVERSPEED_INFO

Enumeration value for low speed events: EM_EVENT_IVS_TYPE.TRAFFIC_UNDERSPEED

Structural body for low speed events: NET_DEV_EVENT_TRAFFIC_UNDERSPEED_INFO

Enumeration value for manual capture: EM_EVENT_IVS_TYPE.TRAFFIC_MANUALSNAP

Structural body for manual capture: NET_DEV_EVENT_TRAFFIC_MANUALSNAP_INFO

Enumeration value for parking spot occupied events:

EM_EVENT_IVS_TYPE.TRAFFIC_PARKINGSPACEPARKING

Structural body for parking spot occupied events:

NET_DEV_EVENT_TRAFFIC_PARKINGSPACEPARKING_INFO

Enumeration value for parking spot empty events:
EM_EVENT_IVS_TYPE.TRAFFIC_PARKINGSPACENOPARKING

Structural body for parking spot empty events:
NET_DEV_EVENT_TRAFFIC_PARKINGSPACENOPARKING_INFO

Enumeration value for crossing lane line events: EM_EVENT_IVS_TYPE.TRAFFIC_OVERLINE

Structural body for crossing lane line events: NET_DEV_EVENT_TRAFFIC_OVERLINE_INFO

Enumeration value for wrong-way driving events: EM_EVENT_IVS_TYPE.TRAFFIC_RETROGRADE

Structural body for wrong-way driving events: NET_DEV_EVENT_TRAFFIC_RETROGRADE_INFO

Enumeration value for driving on the wrong lane events: EM_EVENT_IVS_TYPE.TRAFFIC_WRONGROUTE

Structural body for driving on the wrong lane events: NET_DEV_EVENT_TRAFFIC_WRONGROUTE_INFO

Enumeration value for illegal lane change events: EM_EVENT_IVS_TYPE.TRAFFIC_CROSSLANE

Structural body for illegal lane change events: NET_DEV_EVENT_TRAFFIC_CROSSLANE_INFO

Enumeration value for crossing yellow line events: EM_EVENT_IVS_TYPE.TRAFFIC_OVERYELLOWLINE

Structural body for crossing yellow line events: NET_DEV_EVENT_TRAFFIC_OVERYELLOWLINE_INFO

Enumeration value for yellow-plate vehicle in lane events:
EM_EVENT_IVS_TYPE.TRAFFIC_YELLOWPLATEINLANE

Structural body for yellow-plate vehicle in lane events:
NET_DEV_EVENT_TRAFFIC_YELLOWPLATEINLANE_INFO

Enumeration value for not yield to pedestrian events:
EM_EVENT_IVS_TYPE.TRAFFIC_PEDESTRAINPRIORITY

Structural body for not yield to pedestrian events: NET_DEV_EVENT_TRAFFIC_PEDESTRAINPRIORITY_INFO

Enumeration value for vehicle occupying lane events: EM_EVENT_IVS_TYPE.TRAFFIC_VEHICLEINROUTE

Structural body for vehicle occupying lane events: NET_DEV_EVENT_TRAFFIC_VEHICLEINROUTE_INFO

Enumeration value for occupying bus lane events: EM_EVENT_IVS_TYPE.TRAFFIC_VEHICLEINBUSROUTE

Structural body for occupying bus lane events: NET_DEV_EVENT_TRAFFIC_VEHICLEINBUSROUTE_INFO

Enumeration value for illegal reversing events: EM_EVENT_IVS_TYPE.TRAFFIC_BACKING

Structural body for illegal reversing events: NET_DEV_EVENT_IVS_TRAFFIC_BACKING_INFO

Enumeration value for not wearing seatbelt events: EM_EVENT_IVS_TYPE.TRAFFIC_WITHOUT_SAFEBELT

Structural body for not wearing seatbelt events: NET_DEV_EVENT_TRAFFIC_WITHOUT_SAFEBELT

9.2 Sample Code

```
private int AnalyzerDataCallBack(IntPtr IAnalyzerHandle, uint dwEventType, IntPtr pEventInfo, IntPtr pBuffer,
uint dwBufSize, IntPtr dwUser, int nSequence, IntPtr reserved)
{
    EM_EVENT_IVS_TYPE type = (EM_EVENT_IVS_TYPE)dwEventType;
    switch (type)
    {
        case EM_EVENT_IVS_TYPE.TRAFFIC_RUNREDLIGHT:
            {
                NET_DEV_EVENT_TRAFFIC_RUNREDLIGHT_INFO info =
                (NET_DEV_EVENT_TRAFFIC_RUNREDLIGHT_INFO)Marshal.PtrToStructure(pEventInfo,
                typeof(NET_DEV_EVENT_TRAFFIC_RUNREDLIGHT_INFO));

                EventInfo eventInfo = new EventInfo();
                eventInfo.ID = m_ID.ToString();
                eventInfo.Time = info.UTC.ToString();
                eventInfo.Type = "run red light";
                eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
                eventInfo.Index = info.stuFileInfo.bIndex.ToString();
                eventInfo.Count = info.stuFileInfo.bCount.ToString();
                eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
                eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
                eventInfo.PlateType = info.stTrafficCar.szPlateType;
                eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
                eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
                eventInfo.VehicleType =
                System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
                eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
                eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
                eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
                eventInfo.Offset = info.stuObject.stPicInfo.dwOffSet;
                if (IntPtr.Zero != pBuffer && dwBufSize > 0)
                {
                    eventInfo.Buffer = new byte[dwBufSize];
                    Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
                }
                this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
                m_ID++;
                break;
            }
    }
}
```

```

        case EM_EVENT_IVS_TYPE.TRAFFICJUNCTION:
        {
            NET_DEV_EVENT_TRAFFICJUNCTION_INFO info =
(NET_DEV_EVENT_TRAFFICJUNCTION_INFO)Marshal.PtrToStructure(pEventInfo,
typeof(NET_DEV_EVENT_TRAFFICJUNCTION_INFO));

            EventInfo eventInfo = new EventInfo();
            eventInfo.ID = m_ID.ToString();
            eventInfo.Time = info.UTC.ToString();
            eventInfo.Type = "junction";
            eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
            eventInfo.Index = info.stuFileInfo.bIndex.ToString();
            eventInfo.Count = info.stuFileInfo.bCount.ToString();
            eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
            eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
            eventInfo.PlateType = info.stTrafficCar.szPlateType;
            eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
            eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
            eventInfo.VehicleType =
System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
            eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
            eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
            eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
            eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
            if (IntPtr.Zero != pBuffer && dwBufSize > 0)
            {
                eventInfo.Buffer = new byte[dwBufSize];
                Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
            }
            this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
            m_ID++;
            break;
        }
        case EM_EVENT_IVS_TYPE.TRAFFIC_OVERSPEED:
        {
            NET_DEV_EVENT_TRAFFIC_OVERSPEED_INFO info =
(NET_DEV_EVENT_TRAFFIC_OVERSPEED_INFO)Marshal.PtrToStructure(pEventInfo,
typeof(NET_DEV_EVENT_TRAFFIC_OVERSPEED_INFO));

            EventInfo eventInfo = new EventInfo();
            eventInfo.ID = m_ID.ToString();
            eventInfo.Time = info.UTC.ToString();

```

```

        eventInfo.Type = "over speed";
        eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
        eventInfo.Index = info.stuFileInfo.bIndex.ToString();
        eventInfo.Count = info.stuFileInfo.bCount.ToString();
        eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
        eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
        eventInfo.PlateType = info.stTrafficCar.szPlateType;
        eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
        eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
        eventInfo.VehicleType = info.stTrafficCar.nVehicleType;
        eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
        eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
        eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
        eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
        if (IntPtr.Zero != pBuffer && dwBufSize > 0)
        {
            eventInfo.Buffer = new byte[dwBufSize];
            Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
        }
        this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
        m_ID++;
        break;
    }
    case EM_EVENT_IVS_TYPE.TRAFFIC_UNDERSPEED:
    {
        NET_DEV_EVENT_TRAFFIC_UNDERSPEED_INFO info =
        (NET_DEV_EVENT_TRAFFIC_UNDERSPEED_INFO)Marshal.PtrToStructure(pEventInfo,
        typeof(NET_DEV_EVENT_TRAFFIC_UNDERSPEED_INFO));
        EventInfo eventInfo = new EventInfo();
        eventInfo.ID = m_ID.ToString();
        eventInfo.Time = info.UTC.ToString();
        eventInfo.Type = "under speed";
        eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
        eventInfo.Index = info.stuFileInfo.bIndex.ToString();
        eventInfo.Count = info.stuFileInfo.bCount.ToString();
        eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
        eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
        eventInfo.PlateType = info.stTrafficCar.szPlateType;
        eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
    }
}

```

```

        eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
        eventInfo.VehicleType
System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
        eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
        eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
        eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
        eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
        if (IntPtr.Zero != pBuffer && dwBufSize > 0)
        {
            eventInfo.Buffer = new byte[dwBufSize];
            Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
        }
        this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
        m_ID++;
        break;
    }
    case EM_EVENT_IVS_TYPE.TRAFFIC_MANUALSNAP:
    {
        NET_DEV_EVENT_TRAFFIC_MANUALSNAP_INFO info
        (NET_DEV_EVENT_TRAFFIC_MANUALSNAP_INFO)Marshal.PtrToStructure(pEventInfo,
        typeof(NET_DEV_EVENT_TRAFFIC_MANUALSNAP_INFO));
        EventInfo eventInfo = new EventInfo();
        eventInfo.ID = m_ID.ToString();
        eventInfo.Time = info.UTC.ToString();
        eventInfo.Type = "manual snap";
        eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
        eventInfo.Index = info.stuFileInfo.bIndex.ToString();
        eventInfo.Count = info.stuFileInfo.bCount.ToString();
        eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
        eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
        eventInfo.PlateType = info.stTrafficCar.szPlateType;
        eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
        eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
        eventInfo.VehicleType
System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
        eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
        eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
        eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
        eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
        if (IntPtr.Zero != pBuffer && dwBufSize > 0)

```



```

        {
            eventInfo.Buffer = new byte[dwBufSize];
            Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
        }
        this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
        m_ID++;
        break;
    }
    case EM_EVENT_IVS_TYPE.TRAFFIC_PARKINGSPACEPARKING:
    {
        NET_DEV_EVENT_TRAFFIC_PARKINGSPACEPARKING_INFO info =
        (NET_DEV_EVENT_TRAFFIC_PARKINGSPACEPARKING_INFO)Marshal.PtrToStructure(pEventInfo,
        typeof(NET_DEV_EVENT_TRAFFIC_PARKINGSPACEPARKING_INFO));
        EventInfo eventInfo = new EventInfo();
        eventInfo.ID = m_ID.ToString();
        eventInfo.Time = info.UTC.ToString();
        eventInfo.Type = "parking space(Occupied)";
        eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
        eventInfo.Index = info.stuFileInfo.bIndex.ToString();
        eventInfo.Count = info.stuFileInfo.bCount.ToString();
        eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
        eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
        eventInfo.PlateType = info.stTrafficCar.szPlateType;
        eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
        eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
        eventInfo.VehicleType =
        System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
        eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
        eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
        eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
        eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
        if (IntPtr.Zero != pBuffer && dwBufSize > 0)
        {
            eventInfo.Buffer = new byte[dwBufSize];
            Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
        }
        this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
        m_ID++;
        break;
    }
}

```

```

        case EM_EVENT_IVS_TYPE.TRAFFIC_PARKINGSPACENOPARKING:
        {
            NET_DEV_EVENT_TRAFFIC_PARKINGSPACENOPARKING_INFO info =
(NET_DEV_EVENT_TRAFFIC_PARKINGSPACENOPARKING_INFO)Marshal.PtrToStructure(pEventInfo,
typeof(NET_DEV_EVENT_TRAFFIC_PARKINGSPACENOPARKING_INFO));

            EventInfo eventInfo = new EventInfo();
            eventInfo.ID = m_ID.ToString();
            eventInfo.Time = info.UTC.ToString();
            eventInfo.Type = "no spaces(Empty)";
            eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
            eventInfo.Index = info.stuFileInfo.bIndex.ToString();
            eventInfo.Count = info.stuFileInfo.bCount.ToString();
            eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
            eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
            eventInfo.PlateType = info.stTrafficCar.szPlateType;
            eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
            eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
            eventInfo.VehicleType =
System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
            eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
            eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
            eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
            eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
            if (IntPtr.Zero != pBuffer && dwBufSize > 0)
            {
                eventInfo.Buffer = new byte[dwBufSize];
                Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
            }
            this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
            m_ID++;
            break;
        }
        case EM_EVENT_IVS_TYPE.TRAFFIC_OVERLINE:
        {
            NET_DEV_EVENT_TRAFFIC_OVERLINE_INFO info =
(NET_DEV_EVENT_TRAFFIC_OVERLINE_INFO)Marshal.PtrToStructure(pEventInfo,
typeof(NET_DEV_EVENT_TRAFFIC_OVERLINE_INFO));

            EventInfo eventInfo = new EventInfo();
            eventInfo.ID = m_ID.ToString();
            eventInfo.Time = info.UTC.ToString();

```

```

eventInfo.Type = "over line";
eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
eventInfo.Index = info.stuFileInfo.bIndex.ToString();
eventInfo.Count = info.stuFileInfo.bCount.ToString();
eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
eventInfo.PlateType = info.stTrafficCar.szPlateType;
eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
eventInfo.VehicleType = info.stTrafficCar.szVehicleType;
System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
if (IntPtr.Zero != pBuffer && dwBufSize > 0)
{
    eventInfo.Buffer = new byte[dwBufSize];
    Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
}
this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
m_ID++;
break;
}
case EM_EVENT_IVS_TYPE.TRAFFIC_RETROGRADE:
{
    NET_DEV_EVENT_TRAFFIC_RETROGRADE_INFO info =
(NET_DEV_EVENT_TRAFFIC_RETROGRADE_INFO)Marshal.PtrToStructure(pEventInfo,
typeof(NET_DEV_EVENT_TRAFFIC_RETROGRADE_INFO));
    EventInfo eventInfo = new EventInfo();
    eventInfo.ID = m_ID.ToString();
    eventInfo.Time = info.UTC.ToString();
    eventInfo.Type = "retrograde";
    eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
    eventInfo.Index = info.stuFileInfo.bIndex.ToString();
    eventInfo.Count = info.stuFileInfo.bCount.ToString();
    eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
    eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
    eventInfo.PlateType = info.stTrafficCar.szPlateType;
    eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;

```

```

        eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
        eventInfo.VehicleType
System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
        eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
        eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
        eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
        eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
        if (IntPtr.Zero != pBuffer && dwBufSize > 0)
        {
            eventInfo.Buffer = new byte[dwBufSize];
            Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
        }
        this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
        m_ID++;
        break;
    }
    case EM_EVENT_IVS_TYPE.TRAFFIC_WRONGROUTE:
    {
        NET_DEV_EVENT_TRAFFIC_WRONGROUTE_INFO info
        (NET_DEV_EVENT_TRAFFIC_WRONGROUTE_INFO)Marshal.PtrToStructure(pEventInfo,
        typeof(NET_DEV_EVENT_TRAFFIC_WRONGROUTE_INFO));
        EventInfo eventInfo = new EventInfo();
        eventInfo.ID = m_ID.ToString();
        eventInfo.Time = info.UTC.ToString();
        eventInfo.Type = "wrong route";
        eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
        eventInfo.Index = info.stuFileInfo.bIndex.ToString();
        eventInfo.Count = info.stuFileInfo.bCount.ToString();
        eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
        eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
        eventInfo.PlateType = info.stTrafficCar.szPlateType;
        eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
        eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
        eventInfo.VehicleType
System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
        eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
        eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
        eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
        eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
        if (IntPtr.Zero != pBuffer && dwBufSize > 0)

```

```

        {
            eventInfo.Buffer = new byte[dwBufSize];
            Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
        }
        this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
        m_ID++;
        break;
    }
    case EM_EVENT_IVS_TYPE.TRAFFIC_CROSSLANE:
    {
        NET_DEV_EVENT_TRAFFIC_CROSSLANE_INFO info =
(NET_DEV_EVENT_TRAFFIC_CROSSLANE_INFO)Marshal.PtrToStructure(pEventInfo,
typeof(NET_DEV_EVENT_TRAFFIC_CROSSLANE_INFO));
        EventInfo eventInfo = new EventInfo();
        eventInfo.ID = m_ID.ToString();
        eventInfo.Time = info.UTC.ToString();
        eventInfo.Type = "lane change illegally";
        eventInfo.GroupID = info.stuFileInfo.nGroupId.ToString();
        eventInfo.Index = info.stuFileInfo.bIndex.ToString();
        eventInfo.Count = info.stuFileInfo.bCount.ToString();
        eventInfo.PlateNumber = info.stuTrafficCar.szPlateNumber;
        eventInfo.PlateColor = info.stuTrafficCar.szPlateColor;
        eventInfo.PlateType = info.stuTrafficCar.szPlateType;
        eventInfo.VehicleColor = info.stuTrafficCar.szVehicleColor;
        eventInfo.VehicleSize = GetVehicleSize(info.stuTrafficCar.nVehicleSize);
        eventInfo.VehicleType =
System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
        eventInfo.LaneNumber = info.stuTrafficCar.nLane.ToString();
        eventInfo.Address = Marshal.PtrToStringAnsi(info.stuTrafficCar.szDeviceAddress);
        eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
        eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
        if (IntPtr.Zero != pBuffer && dwBufSize > 0)
        {
            eventInfo.Buffer = new byte[dwBufSize];
            Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
        }
        this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
        m_ID++;
        break;
    }
}

```

```

        case EM_EVENT_IVS_TYPE.TRAFFIC_OVERYELLOWLINE:
        {
            NET_DEV_EVENT_TRAFFIC_OVERYELLOWLINE_INFO info =
(NET_DEV_EVENT_TRAFFIC_OVERYELLOWLINE_INFO)Marshal.PtrToStructure(pEventInfo,
typeof(NET_DEV_EVENT_TRAFFIC_OVERYELLOWLINE_INFO));

            EventInfo eventInfo = new EventInfo();
            eventInfo.ID = m_ID.ToString();
            eventInfo.Time = info.UTC.ToString();
            eventInfo.Type = "over yellow line";
            eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
            eventInfo.Index = info.stuFileInfo.bIndex.ToString();
            eventInfo.Count = info.stuFileInfo.bCount.ToString();
            eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
            eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
            eventInfo.PlateType = info.stTrafficCar.szPlateType;
            eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
            eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
            eventInfo.VehicleType =
System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
            eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
            eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
            eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
            eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
            if (IntPtr.Zero != pBuffer && dwBufSize > 0)
            {
                eventInfo.Buffer = new byte[dwBufSize];
                Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
            }
            this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
            m_ID++;
            break;
        }
        case EM_EVENT_IVS_TYPE.TRAFFIC_YELLOWPLATEINLANE:
        {
            NET_DEV_EVENT_TRAFFIC_YELLOWPLATEINLANE_INFO info =
(NET_DEV_EVENT_TRAFFIC_YELLOWPLATEINLANE_INFO)Marshal.PtrToStructure(pEventInfo,
typeof(NET_DEV_EVENT_TRAFFIC_YELLOWPLATEINLANE_INFO));

            EventInfo eventInfo = new EventInfo();
            eventInfo.ID = m_ID.ToString();
            eventInfo.Time = info.UTC.ToString();

```

```

        eventInfo.Type = "yellow plate in line";
        eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
        eventInfo.Index = info.stuFileInfo.bIndex.ToString();
        eventInfo.Count = info.stuFileInfo.bCount.ToString();
        eventInfo.PlateNumber = info.stuTrafficCar.szPlateNumber;
        eventInfo.PlateColor = info.stuTrafficCar.szPlateColor;
        eventInfo.PlateType = info.stuTrafficCar.szPlateType;
        eventInfo.VehicleColor = info.stuTrafficCar.szVehicleColor;
        eventInfo.VehicleSize = GetVehicleSize(info.stuTrafficCar.nVehicleSize);
        eventInfo.VehicleType = info.stuTrafficCar.szVehicleType;
        eventInfo.LaneNumber = info.stuTrafficCar.nLane.ToString();
        eventInfo.Address = Marshal.PtrToStringAnsi(info.stuTrafficCar.szDeviceAddress);
        eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
        eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
        if (IntPtr.Zero != pBuffer && dwBufSize > 0)
        {
            eventInfo.Buffer = new byte[dwBufSize];
            Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
        }
        this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
        m_ID++;
        break;
    }
    case EM_EVENT_IVS_TYPE.TRAFFIC_PEDESTRAINPRIORITY:
    {
        NET_DEV_EVENT_TRAFFIC_PEDESTRAINPRIORITY_INFO info =
        (NET_DEV_EVENT_TRAFFIC_PEDESTRAINPRIORITY_INFO)Marshal.PtrToStructure(pEventInfo,
        typeof(NET_DEV_EVENT_TRAFFIC_PEDESTRAINPRIORITY_INFO));
        EventInfo eventInfo = new EventInfo();
        eventInfo.ID = m_ID.ToString();
        eventInfo.Time = info.UTC.ToString();
        eventInfo.Type = "not courteous";
        eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
        eventInfo.Index = info.stuFileInfo.bIndex.ToString();
        eventInfo.Count = info.stuFileInfo.bCount.ToString();
        eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
        eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
        eventInfo.PlateType = info.stTrafficCar.szPlateType;
        eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
    }
}

```

```

        eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
        eventInfo.VehicleType
System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
        eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
        eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
        eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
        eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
        if (IntPtr.Zero != pBuffer && dwBufSize > 0)
        {
            eventInfo.Buffer = new byte[dwBufSize];
            Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
        }
        this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
        m_ID++;
        break;
    }
    case EM_EVENT_IVS_TYPE.TRAFFIC_VEHICLEINROUTE:
    {
        NET_DEV_EVENT_TRAFFIC_VEHICLEINROUTE_INFO info
(NET_DEV_EVENT_TRAFFIC_VEHICLEINROUTE_INFO)Marshal.PtrToStructure(pEventInfo,
typeof(NET_DEV_EVENT_TRAFFIC_VEHICLEINROUTE_INFO));
        EventInfo eventInfo = new EventInfo();
        eventInfo.ID = m_ID.ToString();
        eventInfo.Time = info.UTC.ToString();
        eventInfo.Type = "vehicle in route";
        eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
        eventInfo.Index = info.stuFileInfo.bIndex.ToString();
        eventInfo.Count = info.stuFileInfo.bCount.ToString();
        eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
        eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
        eventInfo.PlateType = info.stTrafficCar.szPlateType;
        eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
        eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
        eventInfo.VehicleType
System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
        eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
        eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
        eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
        eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
        if (IntPtr.Zero != pBuffer && dwBufSize > 0)

```



```

        {
            eventInfo.Buffer = new byte[dwBufSize];
            Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
        }
        this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
        m_ID++;
        break;
    }
    case EM_EVENT_IVS_TYPE.TRAFFIC_VEHICLEINBUSROUTE:
    {
        NET_DEV_EVENT_TRAFFIC_VEHICLEINBUSROUTE_INFO info =
        (NET_DEV_EVENT_TRAFFIC_VEHICLEINBUSROUTE_INFO)Marshal.PtrToStructure(pEventInfo,
        typeof(NET_DEV_EVENT_TRAFFIC_VEHICLEINBUSROUTE_INFO));
        EventInfo eventInfo = new EventInfo();
        eventInfo.ID = m_ID.ToString();
        eventInfo.Time = info.UTC.ToString();
        eventInfo.Type = "vehicle in bus route";
        eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
        eventInfo.Index = info.stuFileInfo.bIndex.ToString();
        eventInfo.Count = info.stuFileInfo.bCount.ToString();
        eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
        eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
        eventInfo.PlateType = info.stTrafficCar.szPlateType;
        eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
        eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
        eventInfo.VehicleType =
        System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
        eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
        eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
        eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
        eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
        if (IntPtr.Zero != pBuffer && dwBufSize > 0)
        {
            eventInfo.Buffer = new byte[dwBufSize];
            Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
        }
        this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
        m_ID++;
        break;
    }
}

```

```

        case EM_EVENT_IVS_TYPE.TRAFFIC_BACKING:
        {
            NET_DEV_EVENT_IVS_TRAFFIC_BACKING_INFO info =
(NET_DEV_EVENT_IVS_TRAFFIC_BACKING_INFO)Marshal.PtrToStructure(pEventInfo,
typeof(NET_DEV_EVENT_IVS_TRAFFIC_BACKING_INFO));

            EventInfo eventInfo = new EventInfo();
            eventInfo.ID = m_ID.ToString();
            eventInfo.Time = info.UTC.ToString();
            eventInfo.Type = "backing";
            eventInfo.GroupID = info.stuFileInfo.nGroupID.ToString();
            eventInfo.Index = info.stuFileInfo.bIndex.ToString();
            eventInfo.Count = info.stuFileInfo.bCount.ToString();
            eventInfo.PlateNumber = info.stTrafficCar.szPlateNumber;
            eventInfo.PlateColor = info.stTrafficCar.szPlateColor;
            eventInfo.PlateType = info.stTrafficCar.szPlateType;
            eventInfo.VehicleColor = info.stTrafficCar.szVehicleColor;
            eventInfo.VehicleSize = GetVehicleSize(info.stTrafficCar.nVehicleSize);
            eventInfo.VehicleType =
System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
            eventInfo.LaneNumber = info.stTrafficCar.nLane.ToString();
            eventInfo.Address = Marshal.PtrToStringAnsi(info.stTrafficCar.szDeviceAddress);
            eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
            eventInfo.OffSet = info.stuObject.stPicInfo.dwOffSet;
            if (IntPtr.Zero != pBuffer && dwBufSize > 0)
            {
                eventInfo.Buffer = new byte[dwBufSize];
                Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
            }
            this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
            m_ID++;
            break;
        }
        case EM_EVENT_IVS_TYPE.TRAFFIC_WITHOUT_SAFEBELT:
        {
            NET_DEV_EVENT_TRAFFIC_WITHOUT_SAFEBELT info =
(NET_DEV_EVENT_TRAFFIC_WITHOUT_SAFEBELT)Marshal.PtrToStructure(pEventInfo,
typeof(NET_DEV_EVENT_TRAFFIC_WITHOUT_SAFEBELT));

            EventInfo eventInfo = new EventInfo();
            eventInfo.ID = m_ID.ToString();
            eventInfo.Time = info.UTC.ToString();

```

```

        eventInfo.Type = "without safe belt";
        eventInfo.GroupID = info.stuFileInfo.nGroupId.ToString();
        eventInfo.Index = info.stuFileInfo.bIndex.ToString();
        eventInfo.Count = info.stuFileInfo.bCount.ToString();
        eventInfo.PlateNumber = info.stuTrafficCar.szPlateNumber;
        eventInfo.PlateColor = info.stuTrafficCar.szPlateColor;
        eventInfo.PlateType = info.stuTrafficCar.szPlateType;
        eventInfo.VehicleColor = info.stuTrafficCar.szVehicleColor;
        eventInfo.VehicleSize = GetVehicleSize(info.stuTrafficCar.nVehicleSize);
        eventInfo.VehicleType = info.stuTrafficCar.szVehicleType;
    }
    System.Text.Encoding.Default.GetString(info.stuVehicle.szObjectSubType);
    eventInfo.LaneNumber = info.stuTrafficCar.nLane.ToString();
    eventInfo.Address = Marshal.PtrToStringAnsi(info.stuTrafficCar.szDeviceAddress);
    eventInfo.FileLenth = info.stuObject.stPicInfo.dwFileLenth;
    eventInfo.Offset = info.stuObject.stPicInfo.dwOffset;
    if (IntPtr.Zero != pBuffer && dwBufSize > 0)
    {
        eventInfo.Buffer = new byte[dwBufSize];
        Marshal.Copy(pBuffer, eventInfo.Buffer, 0, (int)dwBufSize);
    }
    this.BeginInvoke((Action<EventInfo>)UpdateUI, eventInfo);
    m_ID++;
    break;
}
default:
    Console.WriteLine(dwEventType.ToString("X"));
    break;
}
return 0;
}

```

10 Person and ID Card Comparison Event

10.1 Subscription to Event

10.1.1 Introduction

Detect whether the person matches by comparing the detected person information with the ID Card information.

10.1.2 Process Description

This chapter is only about callback of specific events. For event subscriprion and receiving, see “2.4 Subscribing Intelligent Event”.

10.1.3 Enumeration and Structure

- Enumerated value corresponding to the event: EM_EVENT_IVS_TYPE.CITIZEN_PICTURE_COMPARE.
- Structure corresponding to the event: NET_DEV_EVENT_CITIZEN_PICTURE_COMPARE_INFO.

10.2 Sample Code

```
private int AnalyzerDataCallBack(IntPtr IAnalyzerHandle, uint dwEventType, IntPtr pEventInfo,
IntPtr pBuffer, uint dwBufSize, IntPtr dwUser, int nSequence, IntPtr reserved)
{
    switch ((EM_EVENT_IVS_TYPE)dwEventType)
    {
        case EM_EVENT_IVS_TYPE.CITIZEN_PICTURE_COMPARE:
        {
            NET_DEV_EVENT_CITIZEN_PICTURE_COMPARE_INFO info =
            (NET_DEV_EVENT_CITIZEN_PICTURE_COMPARE_INFO)Marshal.PtrToStructure(pEventInfo,
            typeof(NET_DEV_EVENT_CITIZEN_PICTURE_COMPARE_INFO));

            Name = "Name: " + info.szCitizen;
            string strSexType;
            if (info.emSex == EM_CITIZENIDCARD_SEX_TYPE.MALE)
            {
                strSexType = "Male";
            }
            else if (info.emSex == EM_CITIZENIDCARD_SEX_TYPE.FEMALE)
            {
                strSexType = "Female";
            }
            else
            {
                strSexType = "Unknown";
            }
            Gender = "Gender: " + strSexType;

            Birth = "Date of birth: " + info.stuBirth.ToDateTime().ToString("yyyy-MM-dd");
            Address = "Address: " + info.szAddress;
            ID_Num = "ID card: " + info.szNumber;
```

```

        Authority = "Issuing authority: " + info.szAuthority;
        Similarity = "Matching degree: " + info.nSimilarity.ToString();
        Result = "Matching result: " + (info.bCompareResult ? "Successful" : "Failed");

        // Save images collected on site
        if (info.stulmageInfo[0].dwFileLenth != 0)
        {
            byte[] imageinfo = new byte[info.stulmageInfo[0].dwFileLenth];
            Marshal.Copy(pBuffer, imageinfo, 0,
(int)info.stulmageInfo[0].dwFileLenth);
            Dispatcher.BeginInvoke(new Action(() =>
            {
                FacemapImage = ByteToBitmapImage(imageinfo);
            }));
        }

        // Save ID card images
        if (info.stulmageInfo[1].dwFileLenth != 0)
        {
            byte[] imageinfo = new byte[info.stulmageInfo[1].dwFileLenth];
            Marshal.Copy(new IntPtr(pBuffer.ToInt32() +
info.stulmageInfo[1].dwOffSet), imageinfo, 0, (int)info.stulmageInfo[1].dwFileLenth);
            Dispatcher.BeginInvoke(new Action(() =>
            {
                IDPhoto = ByteToBitmapImage(imageinfo);
            }));
        }

        // Display one local face database image and one scenario image
        for (int i = 0; i < 6; i++)
        {
            uint length = info.stulmageInfoEx[i].dwFileLenth;
            if ((NET_EM_CITIZEN_PICTURE_COMPARE_TYPE.LOCAL ==
info.stulmageInfoEx[i].emType)
            {
                if (info.stulmageInfoEx[i].dwFileLenth != 0)
                {
                    byte[] localinfo = new
byte[info.stulmageInfoEx[i].dwFileLenth];

```

```

        Marshal.Copy(new IntPtr(pBuffer.ToInt32() +
info.stulmageInfoEx[i].dwOffSet), localinfo, 0, (int)info.stulmageInfoEx[i].dwFileLenth);

        Dispatcher.BeginInvoke(new Action(() =>
        {
            LocalImage = ByteToBitmapImage(localinfo);
        }));
    }
}
else if (NET_EM_CITIZEN_PICTURE_COMPARE_TYPE.FACEMAP ==
info.stulmageInfoEx[i].emType)
{
    //ShowPhoto(pBuffer, length, FullImage);
    if (info.stulmageInfoEx[i].dwFileLenth != 0)
    {
        byte[] imageinfo = new
byte[info.stulmageInfoEx[i].dwFileLenth];
        Marshal.Copy(new IntPtr(pBuffer.ToInt32() +
info.stulmageInfoEx[i].dwOffSet), imageinfo, 0, (int)info.stulmageInfoEx[i].dwFileLenth);
        Dispatcher.BeginInvoke(new Action(() =>
        {
            FullImage = ByteToBitmapImage(imageinfo);
        }));
    }
}
}
break;
}

return 0;
}

```

11 People Counting

11.1 Subscription to Event

11.1.1 Introduction

A camera is installed in the business region, and the intelligent analysis server accurately counts the number of people entering and exiting each entrance

in real time according to the video data collected by the camera. Such products are widely used in large-scale business, tourism industry, public safety, cultural industry expo, chain and other industries.

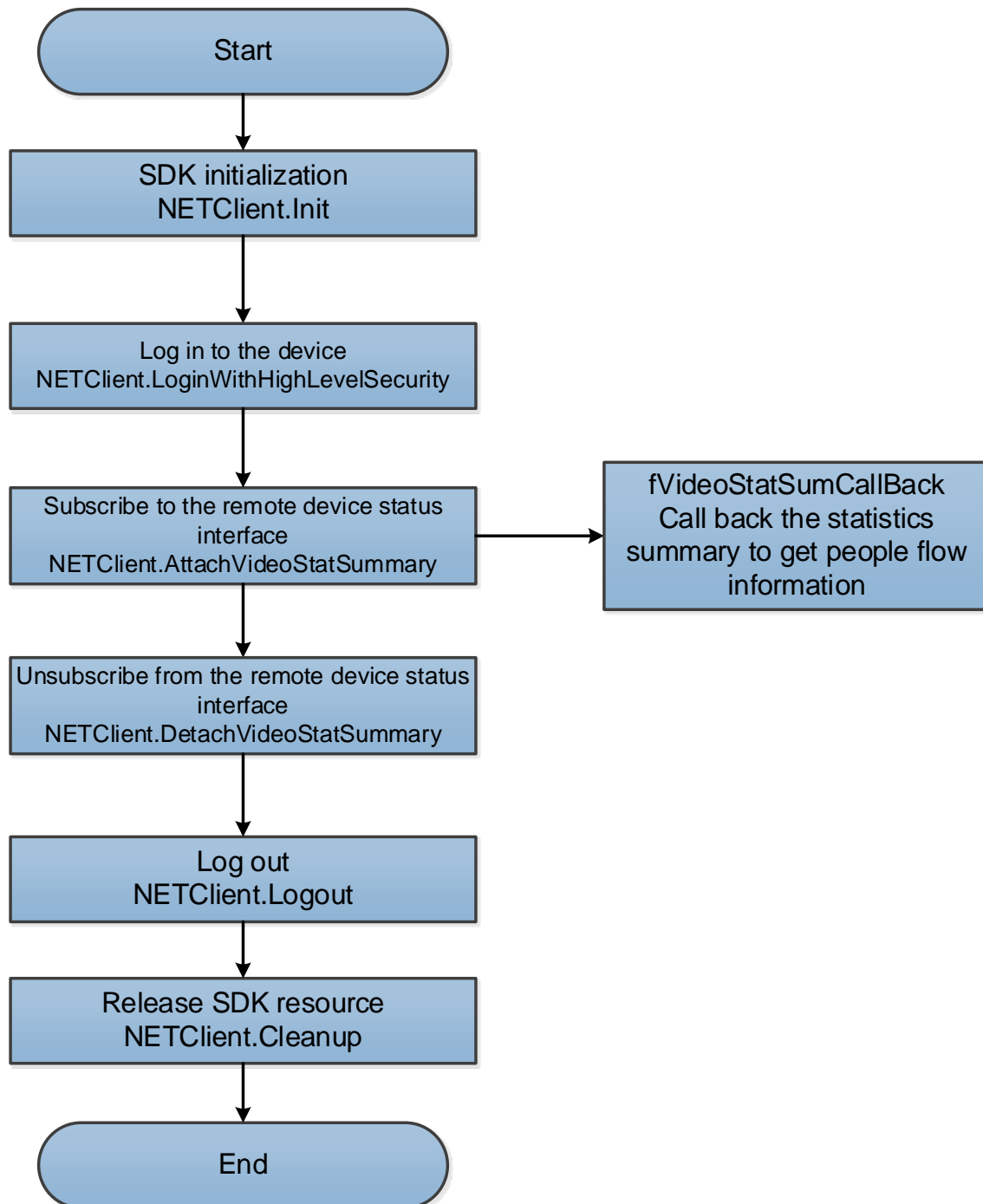
Through real-time subscription to people counting data, you can get the total number of people entered and exited in real time.

11.1.2 Interface Overview

Interface	Description
NETClient.AttachVideoStatSummary	Subscribe to the people counting event.
NETClient.DetachVideoStatSummary	Unsubscribe from the people counting event.

11.1.3 Process Description

Figure 11-1 Process of subscribing to people counting



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.AttachVideoStatSummary** to subscribe to the people counting event from the device.
- Step 4 After successful subscription, use the callback set up by fVideoStatSumCallBack to inform the user of the counting data reported by the device.

- Step 5 After using the reporting function of the people counting event, call **NETClient.DetachVideoStatSummary** to stop subscribing to the people counting event.
- Step 6 After using the function, call **NETClient.Logout** to log out of the device.
- Step 7 After using NetSDK, call **NETClient.Cleanup** to release NetSDK resource.

11.1.4 Sample Code

```
//Callback
private static fVideoStatSumCallBack m_VideoStatSumCallBack;
m_VideoStatSumCallBack = new fVideoStatSumCallBack(VideoStatSumCallBack);

private void VideoStatSumCallBack(IntPtr lAttachHandle, IntPtr pBuf, uint dwBufLen, IntPtr dwUser)
{
    if (lAttachHandle == m_AttactID)
    {
        NET_VIDEOSTAT_SUMMARY info =
(NET_VIDEOSTAT_SUMMARY)Marshal.PtrToStructure(pBuf, typeof(NET_VIDEOSTAT_SUMMARY));

        m_ListID++;
        this.BeginInvoke(new Action(() =>
        {
            var list_item = new ListViewItem();
            list_item.Text = m_ListID.ToString();
            list_item.SubItems.Add(info.nChannelID.ToString());
            list_item.SubItems.Add(info.nInsidePeopleNum.ToString());
            list_item.SubItems.Add(info.nCurrentDayInsidePeopleNum.ToString());

            list_item.SubItems.Add(info.stuEnteredSubtotal.nOSD.ToString());
            list_item.SubItems.Add(info.stuExitedSubtotal.nOSD.ToString());

            list_item.SubItems.Add(info.stuEnteredSubtotal.nToday.ToString());
            list_item.SubItems.Add(info.stuExitedSubtotal.nToday.ToString());
            list_item.SubItems.Add(info.stuEnteredSubtotal.nTotal.ToString());
            list_item.SubItems.Add(info.stuExitedSubtotal.nTotal.ToString());

            list_Record.BeginUpdate();
            list_Record.Items.Add(list_item);
            list_Record.EndUpdate();

        }));
    }
}
```

```

//Subscribe to the people flow statistics
        NET_IN_ATTACH_VIDEOSTAT_SUM        inParam        =        new
NET_IN_ATTACH_VIDEOSTAT_SUM();
        inParam.dwSize = (uint)Marshal.SizeOf(typeof(NET_IN_ATTACH_VIDEOSTAT_SUM));
        inParam.nChannel = 0;
        inParam.cbVideoStatSum = m_VideoStatSumCallBack;
        NET_OUT_ATTACH_VIDEOSTAT_SUM        outParam        =        new
NET_OUT_ATTACH_VIDEOSTAT_SUM();
        outParam.dwSize                                =
(uint)Marshal.SizeOf(typeof(NET_OUT_ATTACH_VIDEOSTAT_SUM));
        m_AttactID = NETClient.AttachVideoStatSummary(m_LoginID, ref inParam, ref
outParam, 5000);

        if (IntPtr.Zero == m_AttactID)
        {
            MessageBox.Show(NETClient.GetLastError());
            return;
        }

//Unsubscribe from the people flow statistics
        NETClient.DetachVideoStatSummary(m_AttactID);
        m_AttactID = IntPtr.Zero;

```

12 Interface Definition

12.1 NetSDK Initialization

12.1.1 NetSDK Initialization Init

Table 12-1 Initialize SDK

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 initialization parameter
Return value	Success: true. Failure: false.	
Note	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. dwUser parameters called by Init will be returned when callback the same fields in the callback function cbDisconnect. This will help users locate the parameters. It is the same eay fot other functions.	

12.1.2 NetSDK Cleanup

Table 12-2 NetSDK Cleanup

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

12.1.3 SetAutoReconnect

Table 12-3 Set reconnection callback

Item	Description
Name	Set auto reconnection callback.

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

12.1.4 SetNetworkParam

Table 12-4 Set network parameter

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 cache size.
Return value	None.	
Note	Adjust the parameters according to the actual network environment.	

12.2 Device Login

12.2.1 LoginWithHighLevelSecurity

Table 12-5 Log in with high level security

Item	Description	
Name	Login the device with high level security.	
Function	IntPtr LoginWithHighLevelSecurity(string pchDVRIP, ushort wDVRPort, string pchUserName, string pchPassword, EM_LOGIN_SPAC_CAP_TYPE emSpecCap, IntPtr pCapParam, ref NET_DEVICEINFO_Ex deviceInfo);	
Parameter	[in]pchDVRIP	Device IP
	[in]wDVRPort	Device Port
	[in]pchUserName	Username
	[in]pchPassword	Password
	[in]emSpecCap	Login type
	[in]pCapParam	Login type parameter
	[out]deviceInfo	Device info

Item	Description
Return value	<ul style="list-style-type: none"> ● Success: Not 0. ● Failure: 0.
Note	None

12.2.2 Logout

Table 12-6 Log out

Item	Description
Name	Logout the device.
Function	bool Logout(IntPtr ILoginID);
Parameter	[in]ILoginID The value returned by NETClient.LoginWithHighLevelSecurity.
Return value	Success: TRUE. Failure: FALSE.
Note	None.

12.3 Real-time Monitoring

12.3.1 RealPlay

Table 12-7 Start the real-time monitoring

Item	Description
Name	Open the real-time monitoring.
Function	IntPtr RealPlay(IntPtr ILoginID, int nChannelID, IntPtr hWnd, EM_RealPlayType rType = EM_RealPlayType.Realplay);
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	Success: Not 0. Failure: 0.
Note	Windows system: When hWnd is valid, the corresponding window displays picture. When hWnd is NULL, get the video data through setting a callback and send to user for treatment.

Table 12-8 Live view type and meaning

Preview type.	Meaning
DH_RType_Realplay	Real-time preview.

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

12.3.2 StopRealPlay

Table 12-9 Stop the real-time monitoring

Item	Description	
Name	Stop the real-time monitoring.	
Function	bool StopRealPlay(IntPtr IRealHandle);	
Parameter	[in]IRealHandle	Return value of NETClient.RealPlay.
Return value	Success: TRUE. Failure: FALSE.	
Note	None.	

12.3.3 SaveRealData

Table 12-10 Save the real-time monitoring data as file

Item	Description	
Name	Save the real-time monitoring data as file.	
Function	bool SaveRealData(IntPtr IRealHandle, string pchFileName);	
Parameter	[in]IRealHandle	Return value of NETClient.RealPlay.
	[in]pchFileName	Save path.
Return value	Success: TRUE. Failure: FALSE.	
Note	None.	

12.3.4 StopSaveRealData

Table 12-11 Stop saving the real-time monitoring data as file

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

12.3.5 SetRealDataCallBack

Table 12-12 Set the callback of real-time monitoring data

Item	Description	
Name	Set the callback of real-time monitoring data.	
Function	v 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	Success: TRUE. Failure: FALSE.	
Note	None.	

Table 12-13 dwFlag type and parameter

dwFlag	Meaning
0x00000001	The original data of device.
0x00000004	YUV data.

12.4 Subscribing Intelligent Event

12.4.1 NETClient.RealLoadPicture

Table 12-14 Subscribe intelligent event interface

Item	Description
Name	Intelligent image alarm subscription interface.

Item	Description	
Function	<pre> IntPtr RealLoadPicture(IntPtr ILoginID, int nChannelID, uint dwAlarmType, bool bNeedPicFile, fAnalyzerDataCallBack cbAnalyzerData, IntPtr dwUser, IntPtr reserved); </pre>	
Parameter	[in]ILoginID	The value returned by NETClient.LoginWithHighLevelSecurity.
	[in]nChannelID	Device channel number.
	[in]dwAlarmType	Intelligent traffic event type.
	[in]bNeedPicFile	Whether to subscribe picture file
	[in]cbAnalyzerData	Intelligent data analysis callback function.
	[in]dwUser	The user parameters.
	[in]reserved	Reserve parameter.
Return value	Success: non 0. Failure: 0.	
Note	<p>The subscribed interface corresponds to one channel and one type of event.</p> <ul style="list-style-type: none"> If you want to subscribe all event types on the channel, configure the value of dwAlarmType as EVENT_IVS_ALL. If you want to subscribe two types of events on one channel, please call RealLoadPicture twice and import different event types. 	

12.4.2 NETClient.StopLoadPic

Table 12-15 Stop subscribing intelligent event.

Item	Description	
Name	Stop subscribing intelligent event.	
Function	<pre> bool StopLoadPic(IntPtr IAnalyzerHandle); </pre>	
Parameter	[in] IAnalyzerHandle	Value returned by NETClient.RealLoadPicture.
Return value	Success: TRUE. Failure: FALSE.	
Note	None.	

12.5 People Counting

12.5.1 AttachVideoStatSummary

Table 12-16 Subscribe people counting event

Item	Description
Name	Subscribe people counting event.

Item	Description	
Function	IntPtr AttachVideoStatSummary(IntPtr ILoginID, ref NET_IN_ATTACH_VIDEOSTAT_SUM pInParam, ref NET_OUT_ATTACH_VIDEOSTAT_SUM pOutParam, int nWaitTime)	
Parameter	[in] ILoginID	Value returned by NETClient.LoginWithHighLevelSecurity.
	[in] pInParam	Subscribe input parameter of people counting.
	[out] pOutParam	Subscribe output parameter of people counting.
	[in] nWaitTime	Timeout.
Return value	Success: TRUE. Failure: FALSE.	
Note	For statement of the callback function fVideoStatSumCallBack in NET_IN_PLAY_BACK_BY_TIME_INFO, see "13.6 fVideoStatSumCallBack."	

12.5.2 DetachVideoStatSummary

Table 12-17 Cancel subscribing people counting event

Item	Description	
Name	Cancel subscribing people counting event.	
Function	bool DetachVideoStatSummary(IntPtr IAttachHandle)	
Parameter	[in] IAttachHandle	Value returned by NETClient.AttachVideoStatSummary.
Return value	Success: TRUE. Failure: FALSE.	
Note	None.	

13 Callback Function Definition

13.1 Note

It is recommended that the callback function is written as static single instance mode; otherwise the memory will make the program crash.

13.2 fDisConnect

Table 13-1 Disconnection callback

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

13.3 fHaveReConnect

Table 13-2 Reconnection callback

Item	Description	
Name	Reconnection callback.	
Function	public delegate void fHaveReConnectCallBack(IntPtr lLoginID, IntPtr pchDVRIP, int nDVRPort, IntPtr dwUser);	
Parameter	[out]lLoginID	Return value of NETClient.LoginWithHighLevelSecurity.
	[out]pchDVRIP	IP of reconnected device.

Item	Description	
	[out]nDVRPort	Port of reconnected device.
	[out]dwUser	User parameter of callback.
Return value	None.	
Note	None.	

13.4 fRealDataCallback

Table 13-3 Callback of real-time monitoring data

Item	Description	
Name	The callback of real-time monitoring data.	
Function	<pre>public delegate void fRealDataCallBackEx(IntPtr IRealHandle, uint dwDataType, IntPtr pBuffer, uint dwBufSize, int param, IntPtr dwUser);</pre>	
Parameter	[out] IRealHandle	Return value of NETClient.RealPlay.
	[out] dwDataType	Data type, 0-original data, 2-YUV data.
	[out] pBuffer	Monitor block data address.
	[out] dwBufSize	Monitor the length of block data, unit: byte.
	[out] param	Callback data parameter structure, the types are different according to different value of dwDataType. <ul style="list-style-type: none"> When dwDataType is 0, param is the empty pointer. When dwDataType is 2, param is the tagCBYUVDataParam structure pointer.
	[out] dwUser	User parameter of callback.
Return value	None.	
Note	None.	

13.5 fAnalyzerDataCallback

Table 13-4 Intelligent event callback

Item	Description
Name	Intelligent event callback.

Item	Description	
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 CLIENT_RealLoadPictureEx.
	[out]dwEventType	Intelligent event type.
	[out]pEventInfo	Event information cache.
	[out]pBuffer	Picture cache.
	[out]dwBufSize	Picture cache size.
	[out]dwUser	User data.
	[out]nSequence	SN.
	[out] reserved	Reserve.
Return value	None.	
Note	After subscribing intelligent events of remote devices, if cameras also have the intelligent events, information of these events will be sent to management center.	

13.6 fVideoStatSumCallBack

Table 13-5 fVideoStatSumCallBack

Item	Description	
Name	Video statistics summery um callback function	
Function	<pre>public delegate void fVideoStatSumCallBack(IntPtr IAttachHandle, IntPtr pBuf, uint dwBufLen, IntPtr dwUser);</pre>	
Parameter	[out] IAttachHandle	Return values of NETClient. AttachVideoStatSummary.
	[out] pBuf	Cache
	[out] dwBufLen	Downloaded size, unit: KB.
	[out]dwUser	User data.
Return value	None.	
Note	None.	

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.