NetSDK Programming Manual
(Intelligent Traffic)

V1.0.1

Foreword

Purpose

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

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

The Manual describes the SDK interfaces and processes of the general function modules for Intelligent Traffic Camera (ITC), Intelligent Traffic System (ITSE), and IPMECK. For more function modules and data structures, refer to *NetSDK Development Manual*.

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

Reader

- SDK software development engineers
- Project managers
- Product managers

Signals

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

Signal Words	Meaning
 TIPS	Provides methods to help you solve a problem or save you time.
 NOTE	Provides additional information as the emphasis and supplement to the text.

Revision History

No.	Version	Revision Content	Release Time
1	V1.0.0	First Release.	December 30, 2017
2	V1.0.1	Delete some library files in "Table 1-1."	January, 2019

About the Manual

- The Manual is for reference only. If there is inconsistency between the Manual and the actual product, the actual product shall govern.
- 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 between the actual value of some data and the value provided, if there is any doubt or dispute, please refer to our final explanation.
- Please contact the supplier or customer service if there is any problem occurred when using the device.
- We are not liable for any loss caused by the operations that do not comply with the Manual.
- All trademarks, registered trademarks and the company names in the Manual are the properties of their respective owners.
- Please visit our website or contact your local service engineer for more information. If there is any uncertainty or controversy, please refer to our final explanation.

Glossary

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

Term	Definition
ITC	Intelligent Traffic Camera, which is featured by capturing pictures of vehicles and automatically analyzing the traffic events.
ITSE	Intelligent Traffic System, also named as intelligent box, is connected with ITC to provide store the pictures and analyzed data.
IPMECK	Controls the opening and closing of barrier.
Login Handle	A kind of handle that connects with ITC, ITSE and IPMECK. If the connection is successful, the handle is not null (32-bit 4 bytes, 64-bit 8 bytes). This handle is used in most of function modules and will not be null till logged out.
Video Channel	The video of ITC or ITSE is expressed by channel ID. The single lens ITC has only one channel, and multi-lens ITC and ITSE have multiple channels.
Query Handle	A kind of handle that sends query request to ITSE. If the request is successful, the handle is not null (32-bit 4 bytes, 64-bit 8 bytes). It is used to query a particular function module and will not be null until logged out.
Media File	The picture captured by ITC and will be identified and analyzed automatically.
Intelligent Capture	The user needs to capture some scenarios manually. The device analyzes the captured pictures and sends the results to the user.
Intelligent Traffic Event	When the vehicle is passing the traffic junction or the capturing range, the ITC will capture and analyze send the pictures, and then send the results to the user.
Traffic Junction	The traffic junction where the device capture each passing vehicle. The device will analyze and identify the captured pictures and send the results to the user.
Open Barrier Gate	On the traffic junction installed with IPMECK and barrier gate, open the barrier gate to let the vehicle go through the control of IPMECK.
Close Barrier Gate	On the traffic junction installed with IPMECK and barrier gate, close the barrier gate to let the vehicle go through the control of IPMECK.

Table of Contents

1 Overview

1.1 General

The Manual introduces SDK interfaces reference information that includes main function modules, interface definition, and callback definition.

The following are the main functions:

SDK initialization, device login, real-time monitoring, download of intelligent images, manual capture, report of intelligent traffic event, vehicle flow statistics, and barrier control.

The development kit might be different dependent on the environment.

- For the files included in Windows development kit, see Table 1-1.

Library type	Library file name	Library file description
Function library	dhnetsdk.h	Header file
	dhnetsdk.lib	Lib file
	dhnetsdk.dll	Library file
	avnetsdk.dll	Library file
Configuration library	avglobal.h	Header file
	dhconfigsdk.h	Configuration Header file
	dhconfigsdk.lib	Lib file
	dhconfigsdk.dll	Library file
Auxiliary library of playing (coding and decoding)	dhplay.dll	Playing library
	fisheye.dll	Fisheye correction library
Dependent library of "avnetsdk.dll"	Infra.dll	Infrastructure library
	json.dll	JSON library
	NetFramework.dll	Network infrastructure library
	Stream.dll	Media transmission structure package library
	StreamSvr.dll	Streaming service
Auxiliary library of "dhnetsdk.dll"	IvsDrawer.dll	Image display library

Table 1-1

- For the files included in Linux development kit, see Table 1-2.

Library type	Library file name	Library file description
Function library	dhnetsdk.h	Header file
	libdhnetsdk.so	Library file
	libavnetsdk.so	Library file
Configuration library	avglobal.h	Header file
	dhconfigsdk.h	Configuration Header file
	libdhconfigsdk.so	Configuration library
Auxiliary library of	libInfra.so	Infrastructure library

Library type	Library file name	Library file description
"libavnetsdk.so"	libNetFramework.so	Network infrastructure library
	libStream.so	Media transmission structure package library
	libStreamSvr.so	Streaming service

Table 1-2

NOTE

- The function library and configuration library are necessary libraries.
- The function library is the main body of SDK, which is used for communication interaction between client and products, remotely controls device, queries device data, configures device data information, as well as gets and handles the streams.
- The configuration library packs and parses the structures of configuration functions.
- It is recommended to use auxiliary library of playing (coding and decoding) to parse and play the streams.
- The auxiliary library decodes the audio and video streams for the functions such as monitoring and voice talk, and collects the local audio.
- If the function library includes avnetsdk.dll or libavnetsdk.so, the corresponding dependent library is necessary.

1.2 Applicability

- Recommended memory: No less than 512 M.
- System supported by SDK:
  - ◇ Windows
    Windows 10/Windows 8.1/Windows 7 and Windows Server 2008/2003
  - ◇ Linux
    The common Linux systems such as Red Hat/SUSE
- Applicable devices include by not limited to the following:
  ITC215-GVRB3A Series, ITC215-PU1A Series, ITC215-PU1B Series, ITC215-PU1C Series, ITC217-PW1B-IRLZ Series, ITC237-PW1A-IRZ Series, ITC217-PW1B-IRLZ10 Series, ITC237 Series, ITSE1604-GN5A-D Series, ITSE0400-GN5A-B Series, ITSE0804-GN5B-D Series, IPMECK-200EB Series, and IPMECK-200OB Series

1.3 Application Scenario

- ITC and ITSET used at the traffic junction to capture the traffic violations and count the vehicle flow. See Figure 1-1.

Figure 1-1

- ITC, ITSE and IPMECK at the gate of parking lot to control the entrance and exit of the vehicles and monitor the availability of parking space. See Figure 1-2.

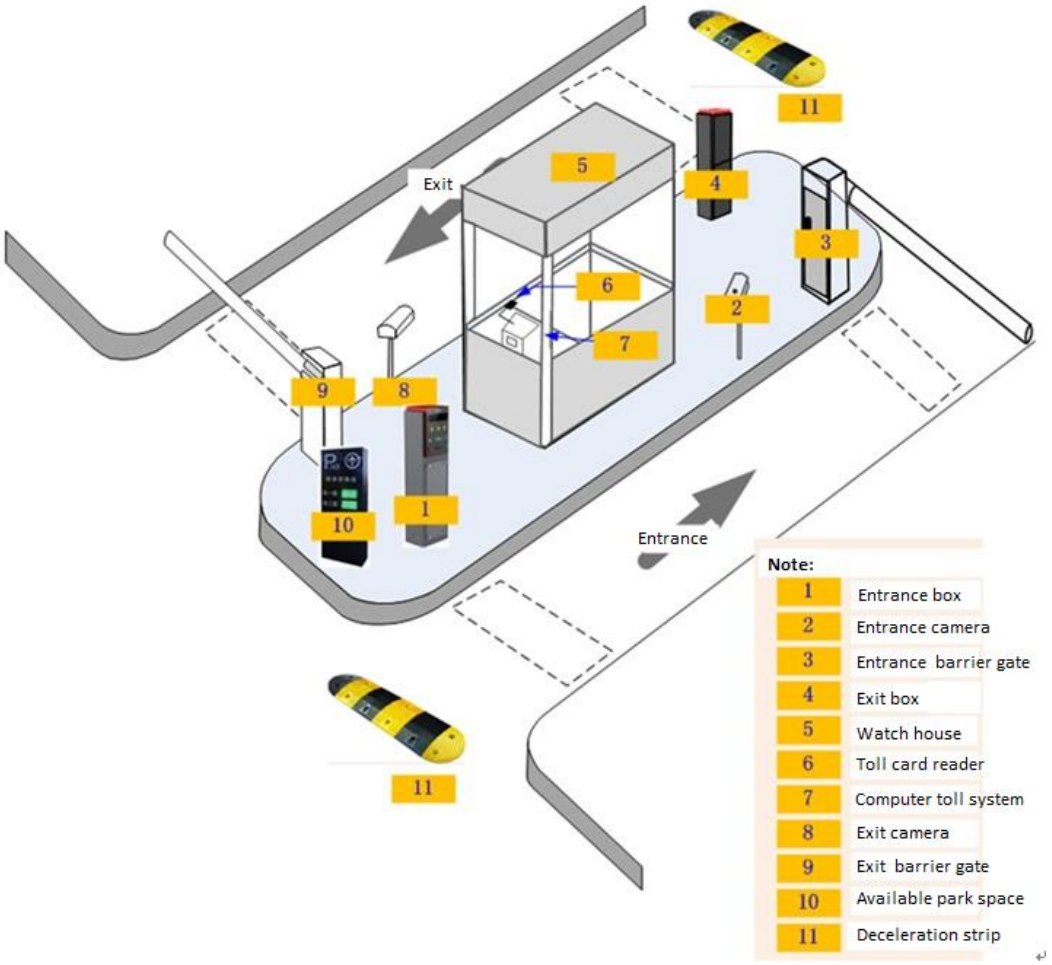
11
Exit
5
4
3
6
2
7
9
8
10
1
Entrance
11
Note:
1 Entrance box
2 Entrance camera
3 Entrance barrier gate
4 Exit box
5 Watch house
6 Toll card reader
7 Computer toll system
8 Exit camera
9 Exit barrier gate
10 Available park space
11 Deceleration strip

Figure 1-2

.

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 set some parameters that affect the SDK overall functions.

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

2.1.2 Interface Overview

Interface	Implication
CLIENT_Init	SDK initialization.
CLIENT_Cleanup	SDK cleaning up.
CLIENT_SetAutoReconnect	Setting of reconnection after disconnection.
CLIENT_SetNetworkParam	Setting of network environment.

Table 2-1

2.1.3 Process

For the process of SDK initialization, see Figure 2-1.

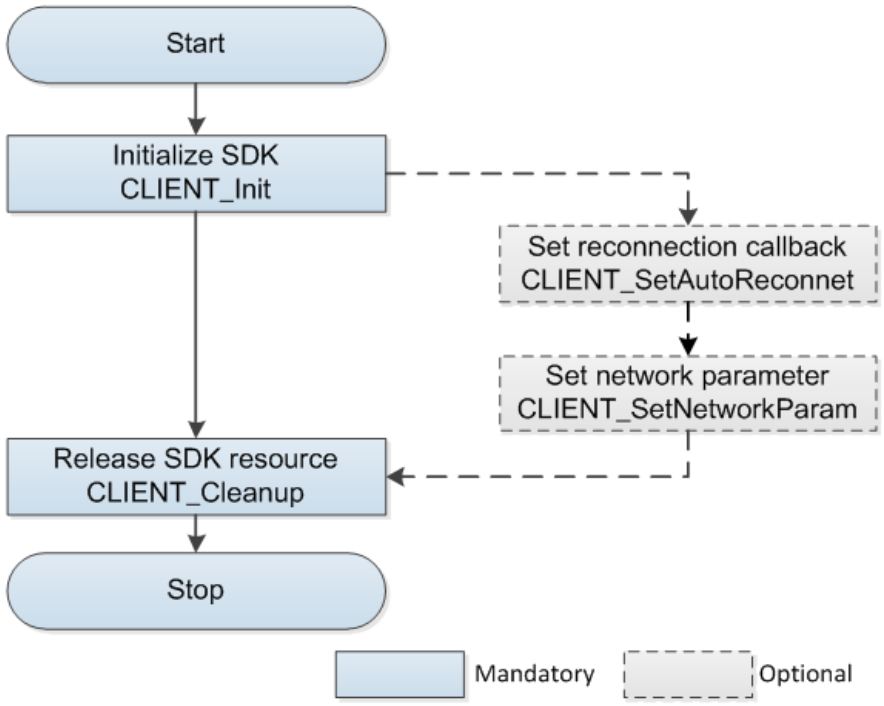

Figure 2-1

Process Description

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

Step 2 (Optional) Call **CLIENT_SetAutoReconnect** to set reconnection callback to allow the auto reconnecting after disconnection.

Step 3 (Optional) Call **CLIENT_SetNetworkParam** to set network login parameter that includes connection timeout and connection attempts.

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

Notes for Process

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

2.1.4 Example Code

```
// Set this callback through CLIENT_Init. When the device is disconnected, SDK informs the user through ths callback.
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser)
```

```
{
    printf("Call DisConnectFunc: lLoginID[0x%x]\n", lLoginID);
}
// Initialize SDK
CLIENT_Init(DisConnectFunc, 0);

// .... Call the functional interface to handle the process

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

2.2 Device Initialization

2.2.1 Introduction

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

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

2.2.2 Interface Overview

Interface	Implication
CLIENT_StartSearchDevices	Search in the LAN to find the uninitialized devices.
CLIENT_InitDevAccount	Initialization interface.
CLIENT_GetDescriptionForResetPwd	Get the password reset information: mobile phone number, email address, and QR code.
CLIENT_CheckAuthCode	Check the validity of security code.
CLIENT_ResetPwd	Reset password.
CLIENT_GetPwdSpecification	Get the password rules.
CLIENT_StopSearchDevices	Stop searching.

Figure 2-2

2.2.3 Process

2.2.3.1 Device Initialization

For the process of device initialization, see Figure 2-3.

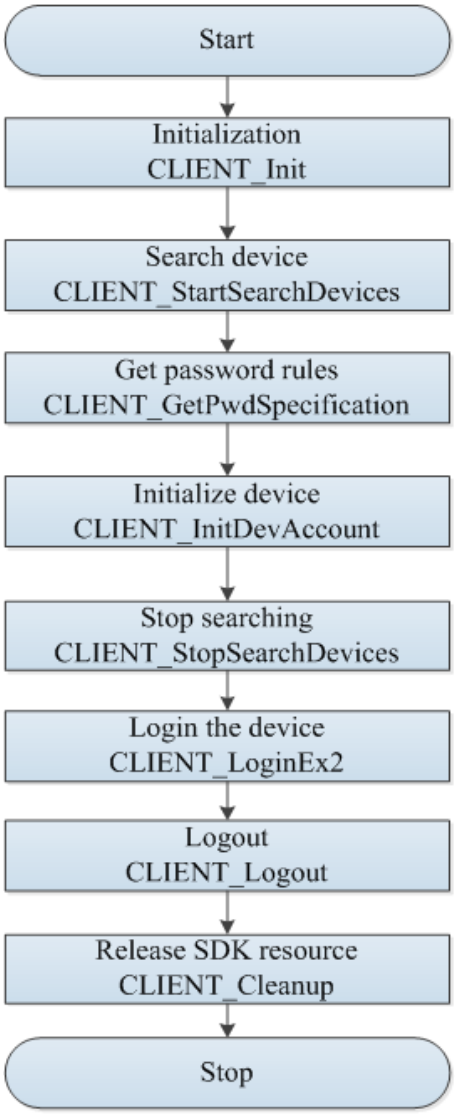

Figure 2-3

Process Description

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

Step 2 Call **CLIENT_StartSearchDevices** to search the devices within the LAN and get the device information.

NOTE

Multi-thread calling is not supported.

Step 3 Call **CLIENT_GetPwdSpecification** to get the password rules.

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

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

Step 6 Call **CLIENT_LoginEx2** and login the admin account with the configured password.

Step 7 After using the function module, call **CLIENT_Logout** to logout the device.

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

Notes for Process

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

2.2.3.2 Password Reset

For the process of device initialization, see Figure 2-4.

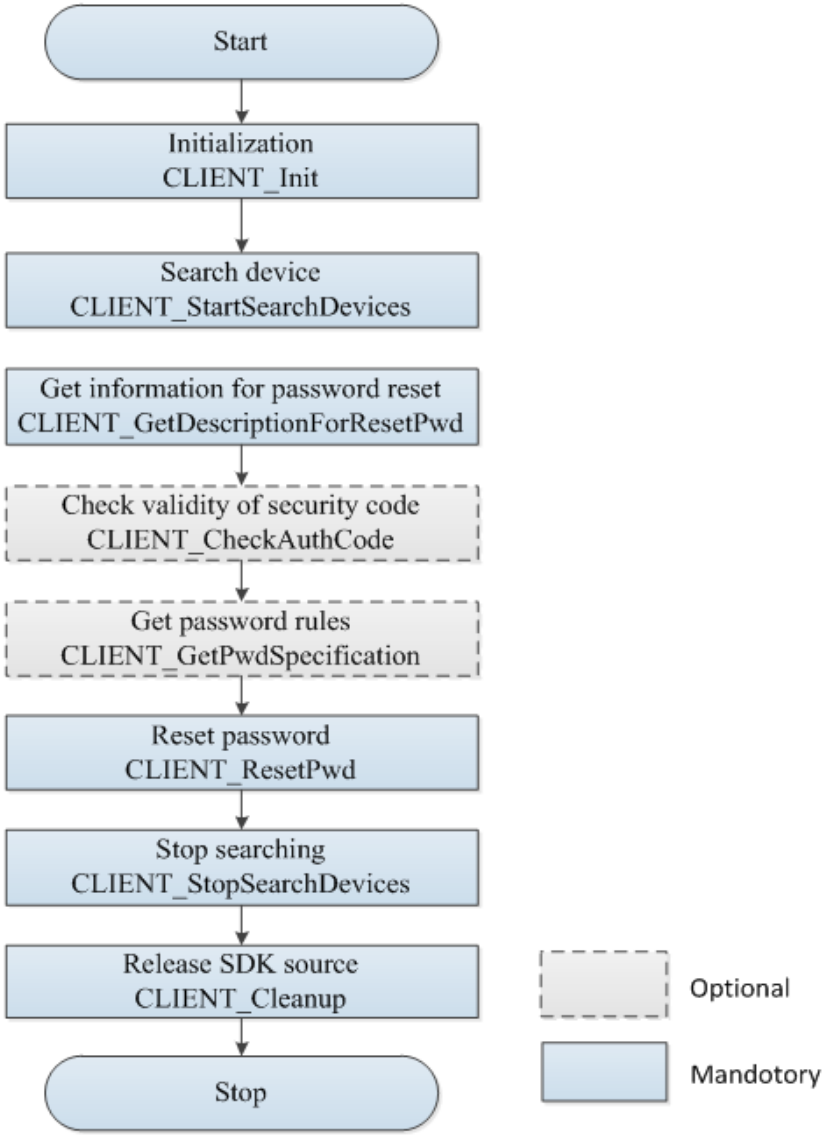

Figure 2-4

Process Description

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

Step 2 Call **CLIENT_StartSearchDevices** to search the devices within the LAN and get the device information.

NOTE

Multi-thread calling is not supported.

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

Step 4 (Optional) Scan the QR code obtained from the previous step to get the security code, and then validate it through **CLIENT_CheckAuthCode**.

Step 5 (Optional) Call **CLIENT_GetPwdSpecification** to get the password rules.

Step 6 Call **CLIENT_ResetPwd** to reset the password.

Step 7 Call **CLIENT_StopSearchDevices** to stop searching.

Step 8 Call **CLIENT_LoginEx2** and login the admin account with the configured password.

Step 9 After using the function module, call **CLIENT_Logout** to logout the device.
Step 10 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

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

2.2.4 Example Code

2.2.4.1 Device Initialization

```
//Firstly, call CLIENT_StartSearchDevices to get the device information.
//Get the password rules
NET_IN_PWD_SPECI stIn = {sizeof(stIn)};
strncpy(stIn.szMac, szMac, sizeof(stIn.szMac) - 1);
NET_OUT_PWD_SPECI stOut = {sizeof(stOut)};
CLIENT_GetPwdSpecification(&stIn, &stOut, 3000, NULL);//In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter. Set the password according to the rules which are used for preventing user from setting the passwords that are not supported by the device.

//Device initialization
NET_IN_INIT_DEVICE_ACCOUNT sInitAccountIn = {sizeof(sInitAccountIn)};
NET_OUT_INIT_DEVICE_ACCOUNT sInitAccountOut = {sizeof(sInitAccountOut)};
sInitAccountIn.byPwdResetWay = 1;//1 stands for password reset by mobile phone number, and 2 stands for password reset by email
strncpy(sInitAccountIn.szMac, szMac, sizeof(sInitAccountIn.szMac) - 1);//Set mac value
strncpy(sInitAccountIn.szUserName, szUserName, sizeof(sInitAccountIn.szUserName) - 1);//Set user name
strncpy(sInitAccountIn.szPwd, szPwd, sizeof(sInitAccountIn.szPwd) - 1);//Set password
strncpy(sInitAccountIn.szCellPhone, szRig, sizeof(sInitAccountIn.szCellPhone) - 1);//If the byPwdResetWay is set as 1, please set szCellPhone field; if the byPwdResetWay is set as 2, please set sInitAccountIn.szMail field.
CLIENT_InitDevAccount(&sInitAccountIn, &sInitAccountOut, 5000, NULL);
```

2.2.4.2 Password Reset

```
//Firstly, call CLIENT_StartSearchDevices to get the device information.
//Get the information for password reset
NET_IN_DESCRIPTION_FOR_RESET_PWD stIn = {sizeof(stIn)};
strncpy(stIn.szMac, szMac, sizeof(stIn.szMac) - 1); //Set mac value
strncpy(stIn.szUserName, szUserName, sizeof(stIn.szUserName) - 1);//Set user name
stIn.byInitStatus = bStstus; //bStstus is the value of return field byInitStatus of device search interface (Callback
```

of CLIENT_SearchDevices and CLIENT_StartSearchDevice, and CLIENT_SearchDevicesByIPs)

NET_OUT_DESCRIPTION_FOR_RESET_PWD stOut = {sizeof(stOut)};

char szTemp[360];

stOut.pQrCode = szTemp;

CLIENT_GetDescriptionForResetPwd(&stIn, &stOut, 3000, NULL);//In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter. After successful connection, stout will output a QR code with address of stOut.pQrCode. Scan this QR code to get the security code for password reset. This security code will be sent to the reserved mobile phone or email box.

//(Optional) Check the security code

NET_IN_CHECK_AUTHCODE stIn1 = {sizeof(stIn1)};

strncpy(stIn1.szMac, szMac, sizeof(stIn1.szMac) - 1); //Set mac value

strncpy(stIn1.szSecurity, szSecu, sizeof(stIn1.szSecurity) - 1); // szSecu is the security code sent to the reserved mobile phone or email box

NET_OUT_CHECK_AUTHCODE stOut1 = {sizeof(stOut1)};

bRet = CLIENT_CheckAuthCode(&stIn1, &stOut1, 3000, NULL); //In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter

//Get password rules

NET_IN_PWD_SPECI stIn2 = {sizeof(stIn2)};

strncpy(stIn2.szMac, szMac, sizeof(stIn2.szMac) - 1); //Set mac value

NET_OUT_PWD_SPECI stOut2 = {sizeof(stOut2)};

CLIENT_GetPwdSpecification(&stIn2, &stOut2, 3000, NULL);// In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter. Set the password according to the rules which are used for preventing user from setting the passwords that are not supported by the device

//Reset password

NET_IN_RESET_PWD stIn3 = {sizeof(stIn3)};

strncpy(stIn3.szMac, szMac, sizeof(stIn3.szMac) - 1); //Set mac value

strncpy(stIn3.szUserName, szUserName, sizeof(stIn3.szUserName) - 1); //Set user name

strncpy(stIn3.szPwd, szPassWd, sizeof(stIn3.szPwd) - 1); //szPassWd is the password reset according to the rules

strncpy(stIn3.szSecurity, szSecu, sizeof(stIn1.szSecurity) - 1); //szSecu is the security code sent to the reserved mobile phone or email box

stIn3.byInitStaus = bStstus; //bStstus is the value of return field byInitStatus of device search interface (Callback of CLIENT_SearchDevices and CLIENT_StartSearchDevice, and CLIENT_SearchDevicesByIPs)

stIn3.byPwdResetWay = bPwdResetWay; // bPwdResetWay is the value of return field byPwdResetWay of device search interface (Callback of CLIENT_SearchDevices and CLIENT_StartSearchDevice, and CLIENT_SearchDevicesByIPs)

NET_OUT_RESET_PWD stOut3 = {sizeof(stOut3)};

CLIENT_ResetPwd(&stIn3, &stOut3, 3000, NULL);//In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter.

2.3 Device Login

2.3.1 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.

2.3.2 Interface Overview

Interface	Implication
CLIENT_LoginEx2	Login.
CLIENT_Logout	Logout.

Table 2-2

2.3.3 Process

For the process of login, see Figure 2-5.

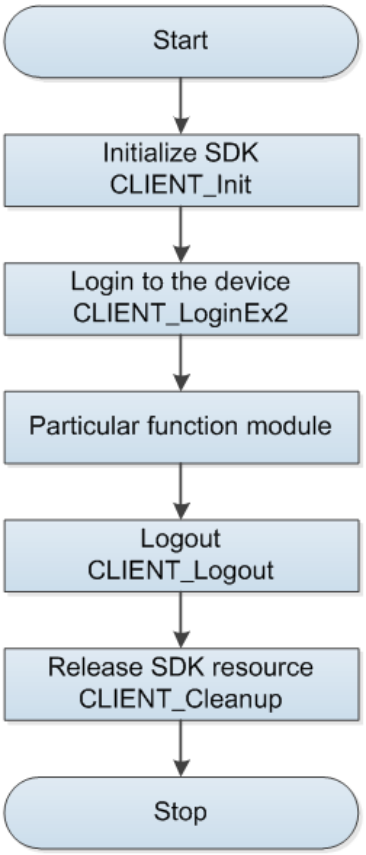

Figure 2-5

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 After successful login, you can realize the required function module.

Step 4 After using the function module, call **CLIENT_Logout** to logout the device.

Step 5 After using all SDK functions, call **CLIENT_Cleanup** to release SDK 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.
- 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. For the common error code, see Table 2-3.

Error code	Meaning
1	Password is wrong.
2	User name does not exist.
3	Login timeout.
4	The account has been logged in.
5	The account has been locked.
6	The account is blacklisted.
7	Out of resources, the system is busy.
8	Sub connection failed.
9	Main connection failed.
10	Exceeded the maximum user connections.
11	Lack of avnetsdk or avnetsdk dependent library.
12	USB flash disk is not inserted into device, or the USB flash disk information error.
13	The client IP is not authorized with login.

Table 2-3

The example code to avoid error code 3 is as follows.

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

For more information about error codes, see "CLIENT_LoginEx2 interface" in *Network SDK Development Manual.chm*.

2.3.4 Example Code

```
NET_DEVICEINFO_Ex stDevInfo = {0};
```

```
int nError = 0;
// Login the device
LLONG lLoginHandle = CLIENT_LoginEx2(szDevIp, nPort, szUserName, szPasswd,
    EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfo, &nError);
// Logout the device
if (0 != lLoginHandle)
{
    CLIENT_Logout (lLoginHandle);
}
```

2.4 Real-time Monitoring

2.4.1 Introduction

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

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

- Supports calling the window handle for SDK 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 SDK interface.

2.4.2 Interface Overview

Interface	Implication
CLIENT_RealPlayEx	Start real-time monitoring.
CLIENT_StopRealPlayEx	Stop real-time monitoring.
CLIENT_SaveRealData	Start saving the real-time monitoring data to the local path.
CLIENT_StopSaveRealData	Stop saving the real-time monitoring data to the local path.
CLIENT_SetRealDataCallBackEx2	Set real-time monitoring data callback.

Table 2-4

2.4.3 Process

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

2.4.3.1 SDK Decoding Play

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

For the process of playing by SDK decoding library, see Figure 2-6.

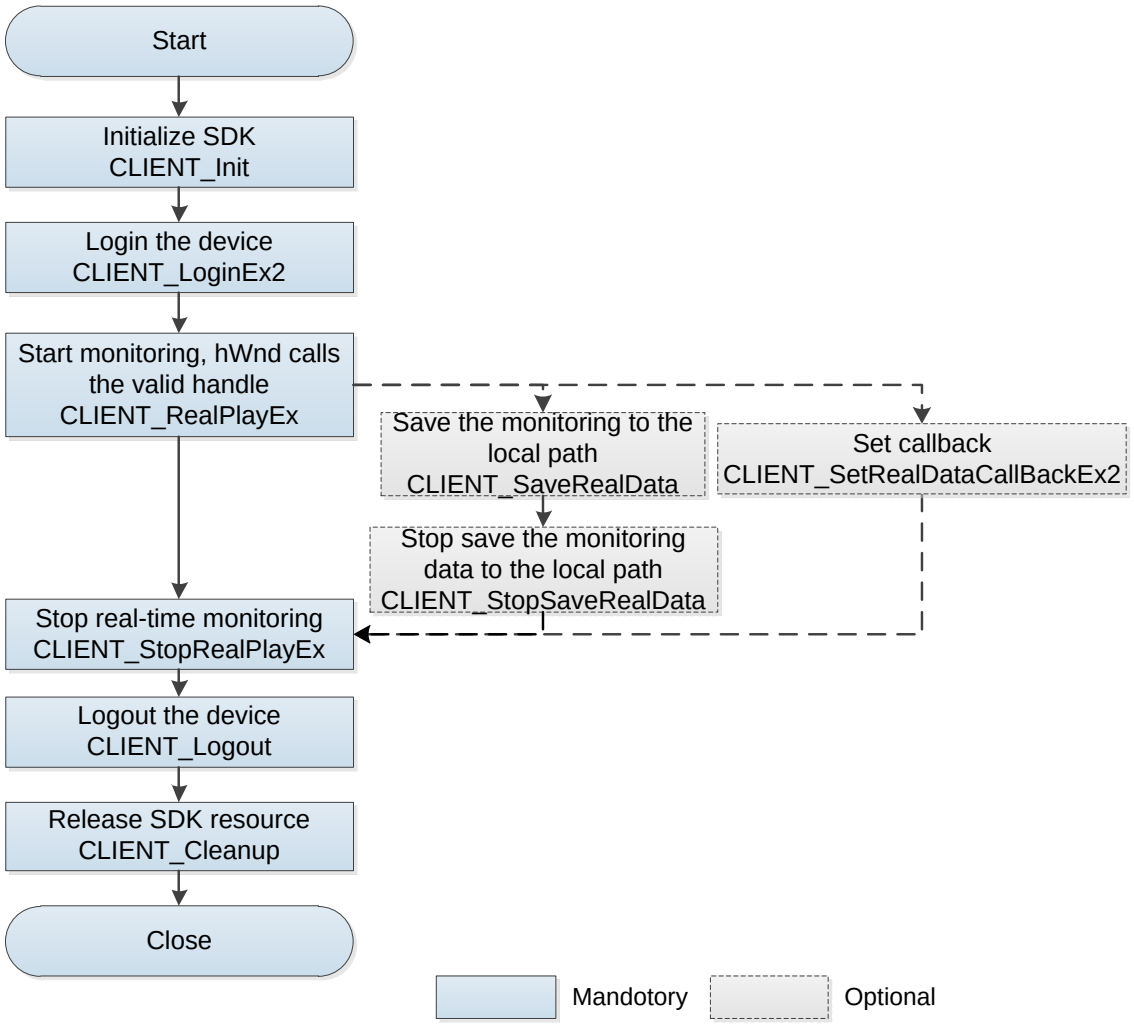

Figure 2-6

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 Call **CLIENT_RealPlayEx** to enable the real-time monitoring. The parameter **hWnd** is a valid window handle.

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

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

Step 6 (Optional) If you call **CLIENT_SetRealDataCallBackEx2**, 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 using the real-time function, call **CLIENT_StopRealPlayEx** to stop real-time monitoring**.**

Step 8 After using the function module, call **CLIENT_Logout** to logout the device.

Step 9 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

- SDK 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 **CLIENT_Init**.

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

- 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: SDK decoding needs to use dhplay.dll. It is suggested to check if dhplay.dll and its auxiliary library are missing under the running directory. See Table 1-1.
- If the system resource is insufficient, the device might return error instead of stream. You can receive an event DH_REALPLAY_FAILD_EVENT in the alarm callback that is set in CLIENT_SetDVRMessCallBack. This event includes the detailed error codes. See "DEV_PLAY_RESULT Structure" in *Network SDK Development Manual.chm*.
- 32 channels limit: The decoding consumes resources especially for the high definition videos. Considering the limited resources at the client, currently the maximum channels are set to be 32. If more than 32, it is suggested to use third party play library. See "2.4.3.2 Call Third Party Library."

2.4.3.2 Call Third Party Library

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

For the process of calling the third party library, see Figure 2-7.

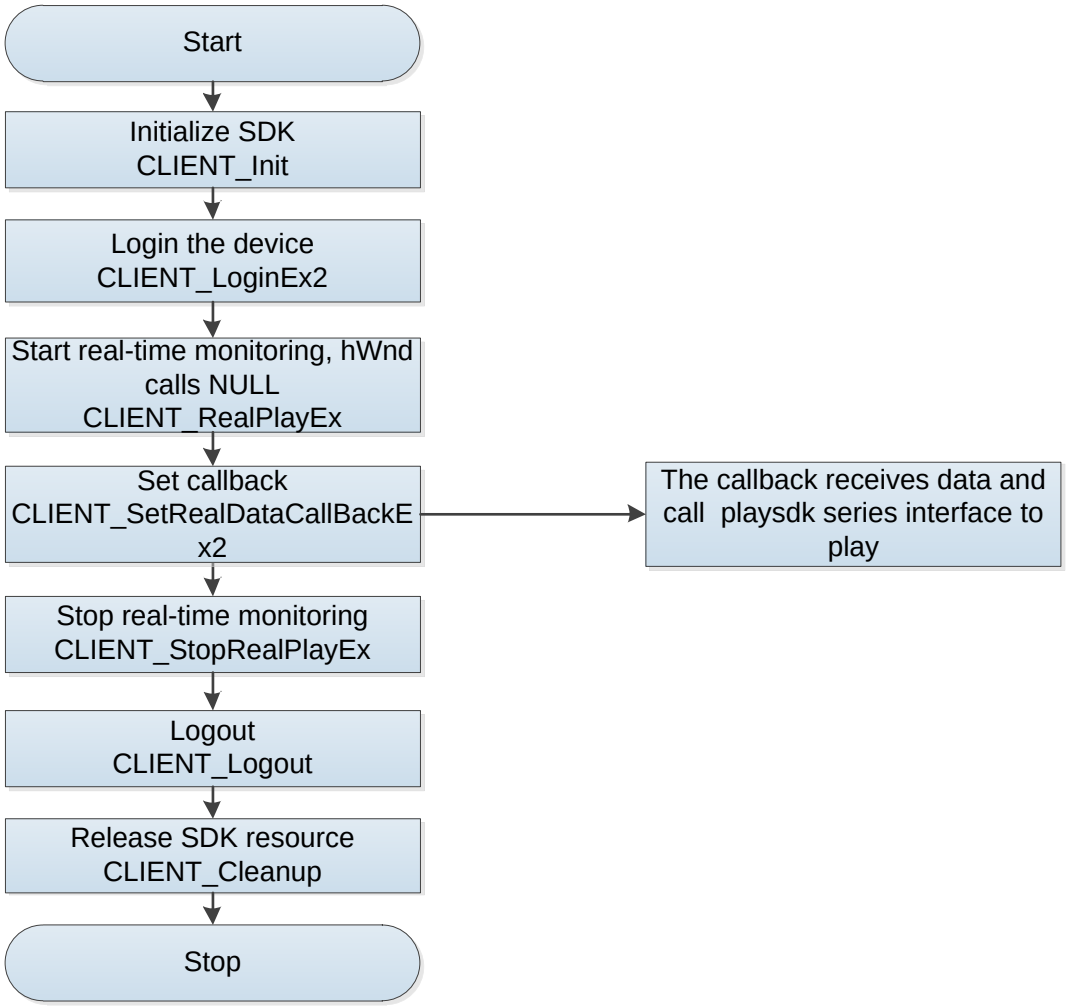

Figure 2-7

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 After successful login, call **CLIENT_RealPlayEx** to enable real-time monitoring. The parameter hWnd is NULL.

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

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

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

Step 7 After using the function module, call **CLIENT_Logout** to logout the device.

Step 8 After using all SDK functions, call **CLIENT_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.

◇ SDK callbacks can only moves into the next process after returning from you. It is not recommended for you to consume time for the unnecessary operations; otherwise the performance could be affected.

2.4.4 Example Code

2.4.4.1 SDK Decoding Play

```
// Take opening the main stream monitoring of channel 1 as an example. The parameter hWnd is a handle of interface window.
LLONG lRealHandle = CLIENT_RealPlayEx(lLoginHandle, 0, hWnd, DH_RType_Realplay);
if (NULL == lRealHandle)
{
    printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
}
printf("input any key to quit!\n");
getchar();
// Stop preview
if (NULL != lRealHandle)
{
    CLIENT_StopRealPlayEx(lRealHandle);
}
```

2.4.4.2 Call Play Library

```
void CALLBACK RealDataCallBackEx(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LLONG param, LDWORD dwUser);
// Take opening the main stream monitoring of channel 1 as an example.
LLONG lRealHandle = CLIENT_RealPlayEx(lLoginHandle, 0, NULL, DH_RType_Realplay);
if (NULL == lRealHandle)
{
    printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
}
else
{
    DWORD dwFlag = REALDATA_FLAG_RAW_DATA; // Initial data labels
    CLIENT_SetRealDataCallBackEx2(lRealHandle, &RealDataCallBackEx, NULL, dwFlag);
}

printf("input any key to quit!\n");
getchar();
```

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

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

2.5 Download of Media File

2.5.1 Introduction

You can get the decoded pictures from ITSE through SDK and saves into the local path for further use.

To download the media files, the SDK connects to the device firstly. It sends query command per the query condition of media file and sends the download command after getting the query result to the device, and then the device will send the media files and decoded data to you.

2.5.2 Interface Overview

Interface	Implication
CLIENT_FindFileEx	Query per the query condition of file.
CLIENT_GetTotalFileCount	Get the queried quantity.
CLIENT_FindNextFileEx	Query the information of media file.
CLIENT_FindCloseEx	Close the query.
CLIENT_DownloadMediaFile	Download the media file.
CLIENT_StopDownloadMediaFile	Stop the download.

Table 2-5

2.5.3 Process

The process of this function module is consisted of querying and downloading the media file.

2.5.3.1 Query of Media File

For the process of querying the media file information, see Figure 2-8.

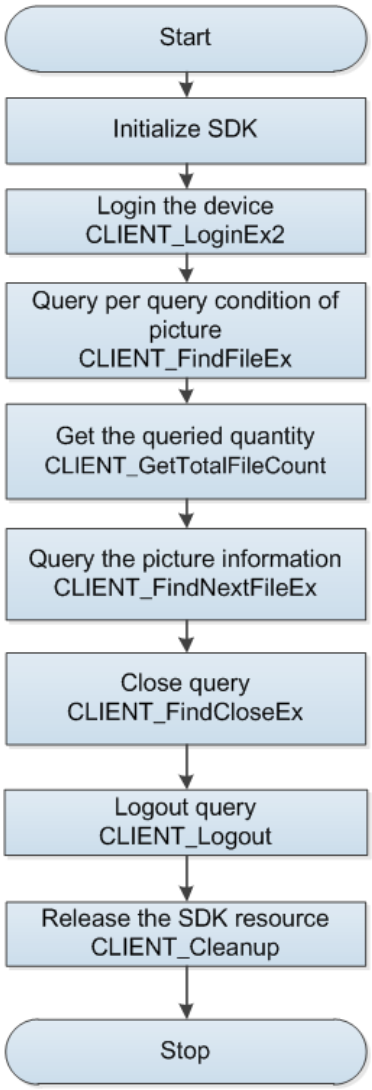

Figure 2-8

Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.
Step 2 Call **CLIENT_LoginEx2** to login the device.
Step 3 Call **CLIENT_FindFileEx** to query per the query condition of media file.
Step 4 Call **CLIENT_GetTotalFileCount** to get the queried total number.
Step 5 Call **CLIENT_FindNextFileExCall** to review information of all the files.
Step 6 Call **CLIENT_FindCloseEx** to close query.
Step 7 After using the function module, call **CLIENT_Logout** to logout the device.
Step 8 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

- Applicable device
  This process applies to ITSE devices. Please be noted that ITC only captures and

identifies pictures and it is not capable of storing the data

- Parameters
  Use DH_FILE_QUERY_TRAFFICCAR_EX for parameter emType in CLIENT_FindFileEx, and the corresponding structure is MEDIA_QUERY_TRAFFICCAR_PARAM_EX. Use the corresponding structure MEDIAFILE_TRAFFICCAR_INFO_EX for interface CLIENT_FindNextFileEx.

2.5.3.2 Download of Media File

For the process of downloading the media file information, see Figure 2-9.

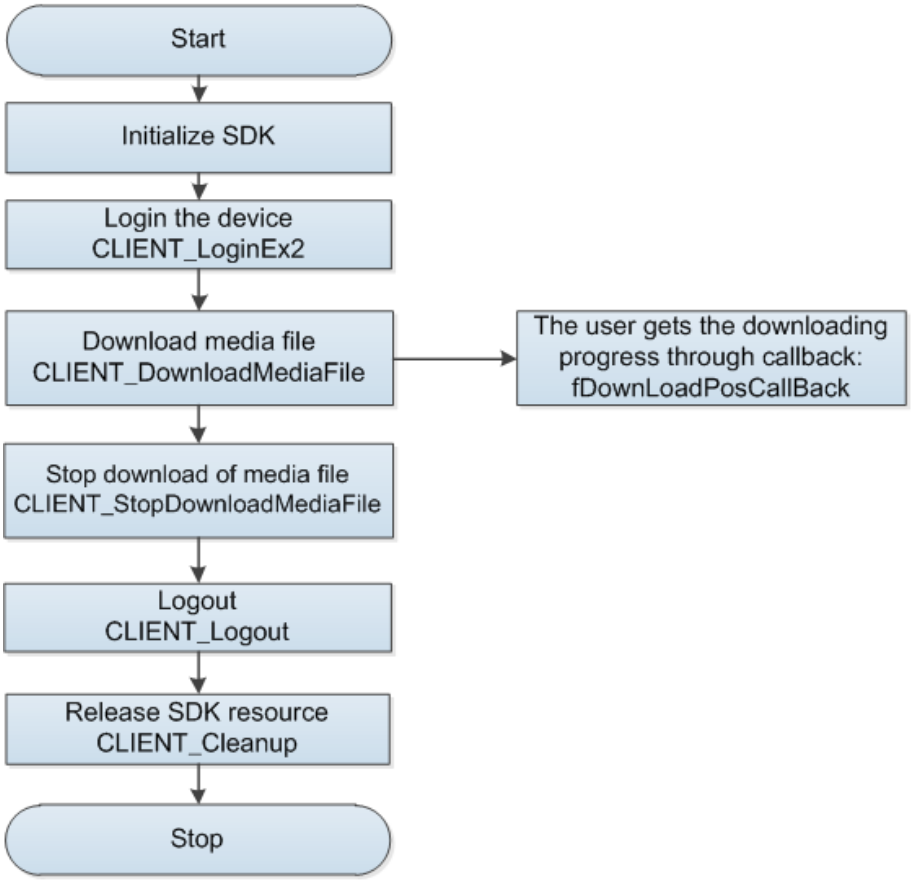

Figure 2-9

Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.
Step 2 Call **CLIENT_LoginEx2** to login the device.
Step 3 Call **CLIENT_DownloadMediaFile** to download the media file.
Step 4 Call **CLIENT_StopDownloadMediaFile** to close the download.
Step 5 After using the function module, call **CLIENT_Logout** to logout the device.
Step 6 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

- Applicable device
  ITSE device. Please be noted that ITC only captures and identifies pictures and it is not

capable of storing the data

- Parameters
  Use only DH_FILE_QUERY_TRAFFICCAR for parameter emType in CLIENT_DownloadMediaFile, and the DH_FILE_QUERY_TRAFFICCAR_EX is not supported. The parameter lpMediaFileInfo is obtained through querying the media file.

2.5.4 Example Code

2.5.4.1 Query of Media File

```
int main()
{
    ……
    //Query condition of media file
    MEDIA_QUERY_TRAFFICCAR_PARAM_EX stuCondition = {0};
    stuCondition.dwSize = sizeof(MEDIA_QUERY_TRAFFICCAR_PARAM_EX);
    stuCondition.stuParam.nMediaType = 1;
    ……
    //Query the media file
    LLONG lFindHandle = CLIENT_FindFileEx(lLoginHandle, DH_FILE_QUERY_TRAFFICCAR_EX, (void*)&stuCondition, NULL);
    if(NULL == lFindHandle)
    {
        printf("CLIENT_FindFileEx: failed! Error code: %x.\n", CLIENT_GetLastError());
        return -1;
    }
    int nCount = 0;
    //Gets the quantity of queried media files
    BOOL bRet = CLIENT_GetTotalFileCount(lFindHandle,&nCount,NULL);
    if(FLASE == bRet)
    {
        printf("CLIENT_GetTotalFileCount: failed! Error code: %x.\n", CLIENT_GetLastError());
        return -2;
    }
    //Review one queried media file per one time
    int nMaxConut = 1;
    do
    {
        MEDIAFILE_TRAFFICCAR_INFO_EX mediaFileInfo = {0};
        mediaFileInfo.dwSize = sizeof(MEDIAFILE_TRAFFICCAR_INFO_EX);
```

```
            //Query a single media file
            bRet = CLIENT_FindNextFileEx(lFindHandle, nMaxConut, (void*)&mediaFileInfo,
sizeof(MEDIAFILE_TRAFFICCAR_INFO_EX), NULL);
            if(FALSE == bRet)
            {
               printf("CLIENT_FindNextFileEx: failed! Error code: %x.\n", CLIENT_GetLastError());
            }
        }
        While ((nCount -= nMaxConut) > 0);
        //Close query
        bRet = CLIENT_FindCloseEx(lFindHandle);
      if(FALSE == bRet)
      {
            printf("CLIENT_FindCloseEx: failed! Error code: %x.\n", CLIENT_GetLastError());
            return -3;
      }
}
```

2.5.4.2 Download of Media File

```
int main()
{
        ……
        //Query the obtained media file
        MEDIAFILE_TRAFFICCAR_INFO info = mediaFileInfo.stuInfo;
        //Download the media file
        LLONG lDownloadHandle = CLIENT_DownloadMediaFile(lLoginHandle,
DH_FILE_QUERY_TRAFFICCAR, (void*)&info, szFileName, DownLoadPosCallBack, NULL, NULL);
        if(NULL == lDownloadHandle)
        {
              printf("CLIENT_DownloadMediaFile: failed! Error code: %x.\n", CLIENT_GetLastError());
        }
        Sleep(5000);
        //Close download
        BOOL bRet = CLIENT_StopDownloadMediaFile(lDownloadHandle);
        if(FALSE == bRet)
        {
            printf("CLIENT_StopDownloadMediaFile: failed! Error code: %x.\n", CLIENT_GetLastError());
        }
}
```

```
//Download progress callback
void CALLBACK DownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD
dwDownLoadSize, LDWORD dwUser)
{
    if (dwDownLoadSize == -1) //Download finished
    {
        printf("lPlayHandle: %p Download end!\n", lPlayHandle);
     }
}
```

2.6 Manual Capture

2.6.1 Introduction

You can send the command through SDK to ITC or ITSE to capture pictures. The device will automatically analyze the pictures and report to you.

This function mainly applies to analyze the vehicles, detect if the vehicles have broken any regulations, and save the vehicles information.

2.6.2 Interface Overview

Interface	Implication
CLIENT_RealLoadPictureEx	Subscribe intelligent traffic event.
CLIENT_ControlDeviceEx	Manual capture.
CLIENT_StopLoadPic	Stop subscribing intelligent traffic event.

Table 2-6

2.6.3 Process

For the process of manual capture, see Figure 2-10.

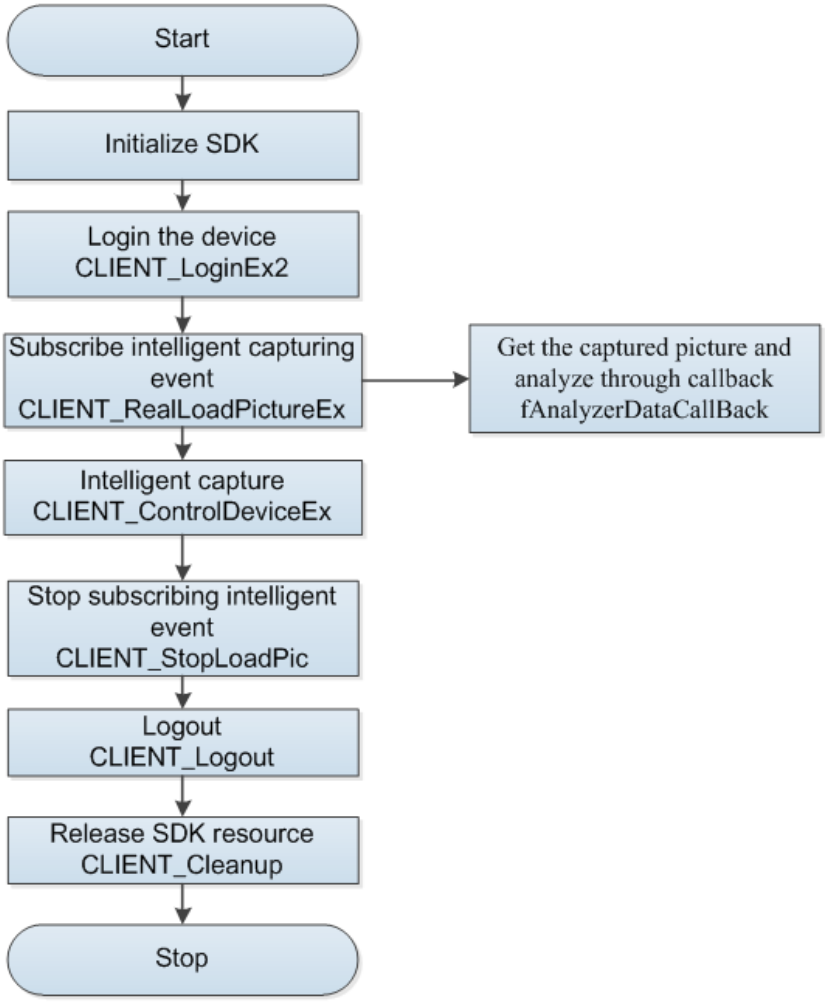

Figure 2-10

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 Call **CLIENT_RealLoadPictureEx** to start subscribing intelligent traffic event.

Step 4 Call **CLIENT_ControlDeviceEx** to trigger intelligent capturing. Set parameter emType as DH_MANUAL_SNAP.

Step 5 Inform you of manual capturing event through the callback fAnalyzerDataCallBack.

Step 6 Call **CLIENT_StopLoadPic** to stop subscribing intelligent event.

Step 7 After using the function module, call **CLIENT_Logout** to logout the device.

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

Notes for Process

Setting of cache for receiving pictures:

Because SDK default cache is 2M, when the data is over 2M, call **CLIENT_SetNetworkParam** to set the receiving cache; otherwise the data pack will be lost.

2.6.4 Example Code

```
int main()
{
    ……
    //Subscribe intelligent capturing event
    LLONG lAnalyerHandle = CLIENT_RealLoadPictureEx(lLoginHandle, 0, (DWORD)EVENT_IVS_ALL, TRUE, AnalyzerDataCallBack, NULL, NULL);
    if(NULL == lAnalyerHandle)
    {
        printf("CLIENT_RealLoadPictureEx: failed! Error code %x.\n", CLIENT_GetLastError());
        return -1;
    }
    MANUAL_SNAP_PARAMETER stuManualSnap = {0};
    stuManualSnap.nChannel = 0;
    sprintf((char*)stuManualSnap.bySequence,"abc");
    //Intelligent capturing
    BOOL bRet = CLIENT_ControlDeviceEx(lLoginHandle,DH_MANUAL_SNAP,&stuManualSnap);
    if(FALSE == bRet)
    {
        printf("CLIENT_ControlDeviceEx: failed! Error code %x.\n", CLIENT_GetLastError());
        return -2;
    }
    Sleep(5000);
    //Stop subscribing intelligent capturing event
    BOOL bRet = CLIENT_StopLoadPic(lAnalyerHandle);
    if(FALSE == bRet)
    {
        printf("CLIENT_StopLoadPic: failed! Error code %x.\n", CLIENT_GetLastError());
        return -3;
    }
    return 0;
}
//Callback of intelligent capturing
int CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
    switch(dwAlarmType)
    {
        case EVENT_IVS_TRAFFIC_MANUALSNAP:
```

```
            {
                DEV_EVENT_TRAFFIC_MANUALSNAP_INFO*  pInfo  =
                (DEV_EVENT_TRAFFIC_MANUALSNAP_INFO*)pAlarmInfo;
                printf("ManualSnapNo: %s", (char*)pInfo-> szManualSnapNo);
                    ……
                break;
            }
        default:
            break;
    }
    return 0;
}
```

2.7 Upload of Intelligent Traffic Event

2.7.1 Introduction

The device decodes the real-time stream and sends the detected intelligent traffic event to you. The event includes the situations such as traffic violation, availability of parking space.

To upload the event, SDK connects to the device and subscribe the intelligent event. The device will send the event to SDK once such event has been detected.

2.7.2 Interface Overview

Interface	Implication
CLIENT_RealLoadPictureEx	Subscribe intelligent traffic event.
CLIENT_StopLoadPic	Stop subscribing intelligent traffic event.

Table 2-7

2.7.3 Process

For the upload process of intelligent traffic event, see Figure 2-11.

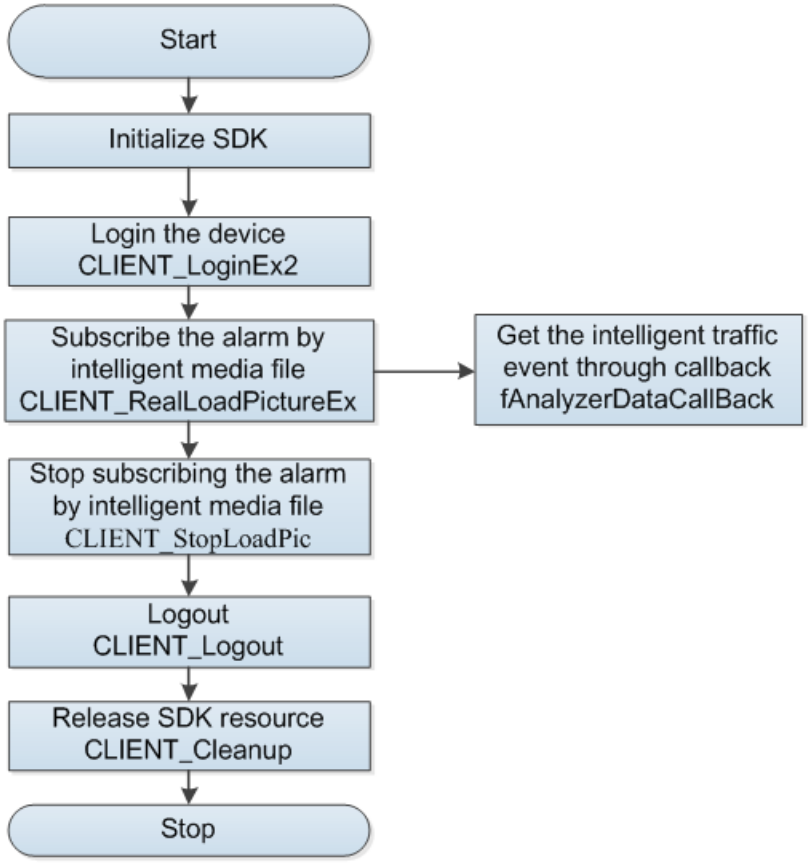

Figure 2-11

Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.
Step 2 Call **CLIENT_LoginEx2** to login the device.
Step 3 Call **CLIENT_ RealLoadPictureEx** to subscribe the intelligent traffic event.
Step 4 Get the uploaded event through callback fAnalyzerDataCallBack and send to you.
Step 5 Call **CLIENT_StopLoadPic** to stop subscribing event.
Step 6 After using the function module, call **CLIENT_Logout** to logout the device.
Step 7 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

- Subscribed event type: Support subscribing all event type (EVENT_IVS_ALL) at the same time or subscribing a single event type.
- Setting of cache for receiving pictures: Because SDK default cache is 2M, when the data is over 2M, call **CLIENT_SetNetworkParam** to set the receiving cache, otherwise the data pack will be lost.
- Setting of whether to receive pictures: Because some devices have 3G or 4G network, when SDK is connecting to the device, if it does not need to receive picture, set the parameter bNeedPicFile as False in interface CLIENT_ RealLoadPictureEx to only receive the intelligent event without picture.

2.7.4 Example Code

```
int main()
{
    ……
    //Subscribe the upload of intelligent traffic event
    LLONG lAnalyerHandle = CLIENT_RealLoadPictureEx(lLoginHandle, 0, (DWORD)EVENT_IVS_ALL, TRUE, AnalyzerDataCallBack, NULL, NULL);
    if(NULL == lAnalyerHandle)
    {
        printf("CLIENT_RealLoadPictureEx: failed! Error code %x.\n", CLIENT_GetLastError());
        return -1;
    }
    Sleep(5000);
    //Stop subscribing the upload of intelligent traffic event
    BOOL bRet = CLIENT_StopLoadPic(lAnalyerHandle);
    if(FALSE == bRet)
    {
        printf("CLIENT_StopLoadPic: failed! Error code %x.\n", CLIENT_GetLastError());
        return -2;
    }
    return 0;
}
//Callback for upload of intelligent traffic event
int CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
    switch(dwAlarmType)
    {
        ……
        case EVENT_IVS_TRAFFIC_RUNREDLIGHT: // Event of running the red light
            {
                DEV_EVENT_TRAFFIC_RUNREDLIGHT_INFO* pInfo = (DEV_EVENT_TRAFFIC_RUNREDLIGHT_INFO*)pAlarmInfo;
                ……
                break;
            }
        ……
        default:
            break;
```

```
    }
    return 0;
}
```

2.8 Vehicle Flow Statistics

2.8.1 Introduction

ITC device counts on all the passing vehicles to analyze the traffic status and directly send the result to you or to ITSE that sends to you.

2.8.2 Interface Overview

Interface	Implication
CLIENT_StartTrafficFluxStat	Subscribe intelligent traffic event
CLIENT_StopTrafficFluxStat	Stop subscribing intelligent traffic event

Table 2-8

2.8.3 Process

For the process of counting vehicles, see Figure 2-12.

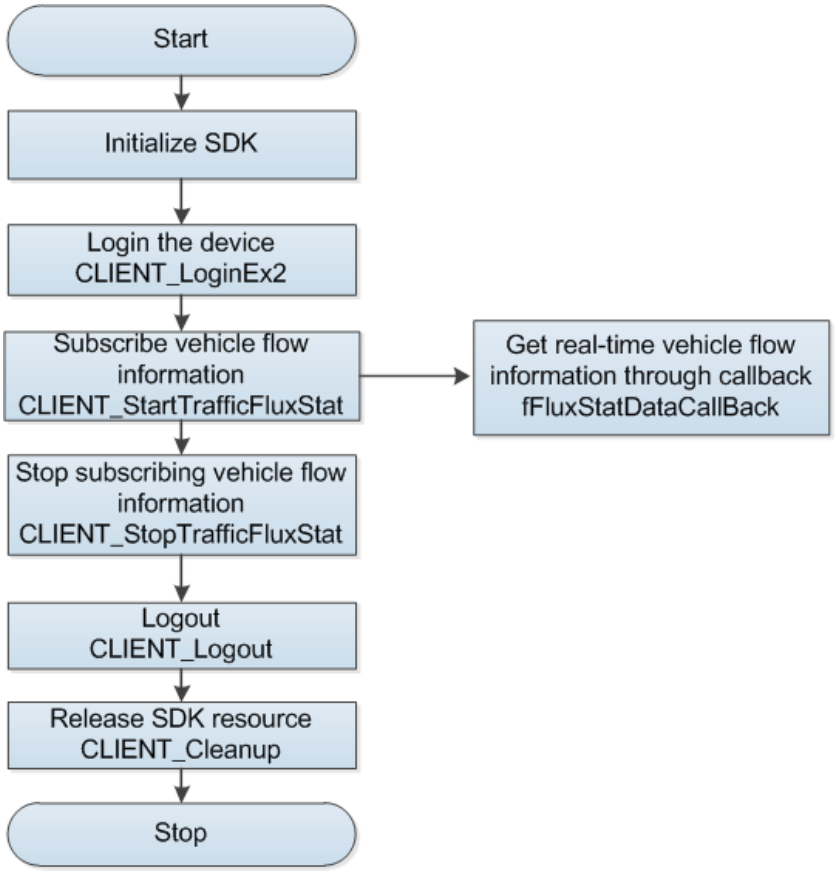

Figure 2-12

Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.
Step 2 Call **CLIENT_LoginEx2** to login the device.
Step 3 Call **CLIENT_StartTrafficFluxStat** to subscribe the vehicle flow information.
Step 4 Get the vehicles information uploaded by ITC or ITSE through callback fFluxStatDataCallBack and inform you.
Step 5 Call **CLIENT_StopTrafficFluxStat** to stop subscribing the vehicle flow information.
Step 6 After using the function module, call **CLIENT_Logout** to logout the device.
Step 7 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

Callback data type: The parameter pEventInfo corresponds to structure of DEV_EVENT_TRAFFIC_FLOWSTAT_INFO.

2.8.4 Example Code

```
int main()
{
    ……
    NET_IN_TRAFFICFLUXSTAT stuIn = {0};
    stuIn.dwSize = sizeof(NET_IN_TRAFFICFLUXSTAT);
    stuIn.cbData = FluxStatDataCallBack;
    NET_OUT_TRAFFICFLUXSTAT stuOut = {0};
    stuOut.dwSize = sizeof(NET_OUT_TRAFFICFLUXSTAT);
    //Subscribe the vehicle flow statistics
    LLONG lFluxStatHandle = CLIENT_StartTrafficFluxStat(lLoginHandle, &stuIn, &stuOut);
    if(NULL == lFluxStatHandle)
    {
            printf("CLIENT_StartTrafficFluxStat: failed! Error code %x.\n", CLIENT_GetLastError());
        return -1;
    }
    Sleep(5000);
    //Stop subscribing the vehicle flow statistics
    BOOL bRet = CLIENT_StopTrafficFluxStat(lFluxStatHandle);
    if(FALSE == bRet)
    {
        printf("CLIENT_StopTrafficFluxStat: failed! Error code %x.\n", CLIENT_GetLastError());
        return -2;
    }
    return 0;
```

```
}
//Callback of vehicle flow statistics
int CALLBACK FluxStatDataCallBack (LLONG lFluxStatHandle, DWORD dwEventType, void* pEventInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
    DEV_EVENT_TRAFFIC_FLOWSTAT_INFO* pInfo = (DEV_EVENT_TRAFFIC_FLOWSTAT_INFO*)pEventInfo;
    ……
    return 0;
}
```

2.9 Barrier Control

2.9.1 Introduction

IPMECK device can control the opening and closing operations of road barrier. You can send the command through SDK to IPMECK for the control of barrier.

The barrier control mainly applies to the places such as parking lot, toll gate, and gate of district.

Applicable device: IPMECK device

2.9.2 Interface Overview

Interface	Implication
CLIENT_ControlDeviceEx	Barrier control.

Table 2-9

2.9.3 Process

For the process of barrier control, see Figure 2-13.

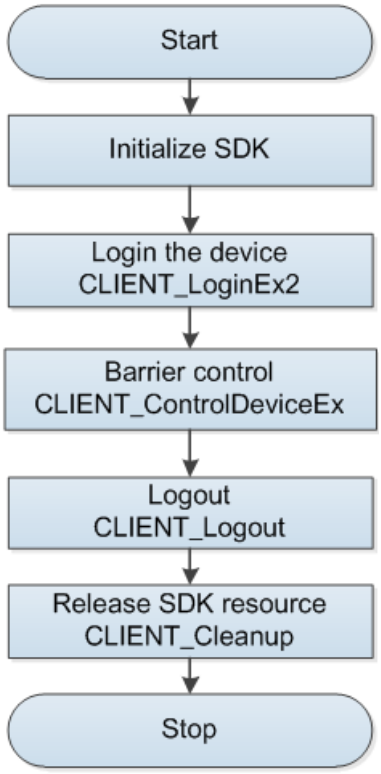

Figure 2-13

Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.
Step 2 Call **CLIENT_LoginEx2** to login the device.
Step 3 Call **CLIENT_ControlDeviceEx** to open or close the barrier.
Step 4 After using the function module, call **CLIENT_Logout** to logout the device.
Step 5 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.9.4 Example Code

```
int main()
{
    ……
    NET_CTRL_OPEN_STROBE stuOpenStrobe = {0};
    stuOpenStrobe.dwSize = sizeof(NET_CTRL_OPEN_STROBE);
    stuOpenStrobe.nChannelId = 0;
    sprintf(stuOpenStrobe.szPlateNumber,"GBBD51SMR");
    //Open barrier
    BOOL bRet = CLIENT_ControlDeviceEx(lLoginHandle,DH_CTRL_OPEN_STROBE,&stuOpenStrobe);
    if(FALSE == bRet)
    {
        printf("CLIENT_ControlDeviceEx: Open strobe failed! Error code %x.\n", CLIENT_GetLastError());
```

```
        return -1;
    }
    NET_CTRL_CLOSE_STROBE stuCloseStrobe = {0};
    stuCloseStrobe.dwSize = sizeof(NET_CTRL_CLOSE_STROBE);
    stuCloseStrobe.nChannelId = 0;
    //Close barrier
    bRet = CLIENT_ControlDeviceEx(lLoginHandle,DH_CTRL_CLOSE_STROBE,&stuCloseStrobe);
    if(FALSE == bRet)
    {
        printf("CLIENT_ControlDeviceEx: Close strobe failed! Error code %x.\n",
CLIENT_GetLastError());
        return -2;
    }
    return 0;
}
```

3 Interface Definition

3.1 SDK Initialization

3.1.1 SDK CLIENT_Init

Item	Description	
Name	Initialize SDK.	
Function	BOOL CLIENT_Init(fDisConnect cbDisConnect, LDWORD dwUser);	
Parameter	[in]cbDisConnect	Disconnection callback.
	[in]dwUser	User parameter of disconnection callback.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	● The precondition for calling other function modules of SDK. ● The callback will not send to the user after the device is disconnected if the callback is set as NULL.	

3.1.2 CLIENT_Cleanup

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

3.1.3 CLIENT_SetAutoReconnect

Item	Description	
Name	Set auto reconnection callback.	
Function	void CLIENT_SetAutoReconnect(fHaveReConnect cbAutoConnect, LDWORD dwUser);	
Parameter	[in]cbAutoConnect	Reconnection callback.
	[in]dwUser	User parameter of disconnection callback.

Item	Description
Return value	None.
Note	Set the reconnection callback interface. If the callback is set as NULL, it will not connect automatically.

3.1.4 CLIENT_SetNetworkParam

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

3.2 Device Initialization

3.2.1 CLIENT_StartSearchDevices

Item	Description	
Name	Search the device.	
Function	LLONG CLIENT_StartSearchDevices (fSearchDevicesCB cbSearchDevices, void* pUserData, char* szLocalIp=NULL);	
Parameter	[in]cbSearchDevices	Device information callback.
	[out]pUserData	User data.
	[in]szLocalIp	● In case of single network card, enter NULL, which means using the host PC IP. ● In case of multiple network card, enter the IP of the specified network card.
Return value	Searching handle.	
Note	Multi-thread calling is not supported.	

3.2.2 CLIENT_InitDevAccount

Item	Description
Name	Initialize the device.

Item	Description	
Function	BOOL CLIENT_InitDevAccount(const NET_IN_INIT_DEVICE_ACCOUNT *pInitAccountIn, NET_OUT_INIT_DEVICE_ACCOUNT *pInitAccountOut, DWORD dwWaitTime, char *szLocalIp);	
Parameter	[in]pInitAccountIn	Corresponds to structure of NET_IN_INIT_DEVICE_ACCOUNT.
	[out]pInitAccountOut	Corresponds to structure of NET_OUT_INIT_DEVICE_ACCOUNT.
	[in]dwWaitTime	Timeout.
	[in]szLocalIp	● In case of single network card, the last parameter is not required to be filled. ● In case of multiple network card, enter the IP of the host PC for the last parameter.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.2.3 CLIENT_GetDescriptionForResetPwd

Name	Description	
Name	Get information for password reset.	
Function	BOOL CLIENT_GetDescriptionForResetPwd(const NET_IN_DESCRIPTION_FOR_RESET_PWD *pDescriptionIn, NET_OUT_DESCRIPTION_FOR_RESET_PWD *pDescriptionOut, DWORD dwWaitTime, char *szLocalIp);	
Parameter	[in]pDescriptionIn	Corresponds to structure of NET_IN_DESCRIPTION_FOR_RESET_PWD.
	[out]pDescriptionOut	Corresponds to structure of NET_OUT_DESCRIPTION_FOR_RESET_PWD.
	[in]dwWaitTime	Timeout.
	[in]szLocalIp	● In case of single network card, the last parameter is not required to be filled. ● In case of multiple network card, enter the IP of the host PC for the last parameter.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.2.4 CLIENT_CheckAuthCode

Item	Description	
Name	Check the validity of security code.	
Function	BOOL CLIENT_CheckAuthCode(const NET_IN_CHECK_AUTHCODE *pCheckAuthCodeIn, NET_OUT_CHECK_AUTHCODE *pCheckAuthCodeOut, DWORD dwWaitTime, char *szLocalIp);	
Parameter	[in]pCheckAuthCodeIn	Corresponds to structure of NET_IN_CHECK_AUTHCODE.
	[out]pCheckAuthCodeOut	Corresponds to structure of NET_OUT_CHECK_AUTHCODE.
	[in]dwWaitTime	Timeout.
	[in]szLocalIp	● In case of single network card, the last parameter is not required to be filled. ● In case of multiple network card, enter the IP of the host PC for the last parameter.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.2.5 CLIENT_ResetPwd

Item	Description	
Name	Reset the password.	
Function	BOOL CLIENT_ResetPwd(const NET_IN_RESET_PWD *pResetPwdIn, NET_OUT_RESET_PWD *pResetPwdOut, DWORD dwWaitTime, char *szLocalIp);	
Parameter	[in]pResetPwdIn	Corresponds to structure of NET_IN_RESET_PWD.
	[out]pResetPwdOut	Corresponds to structure of NET_OUT_RESET_PWD.
	[in]dwWaitTime	Timeout.
	[in]szLocalIp	● In case of single network card, the last parameter is not required to be filled. ● In case of multiple network card, enter the IP of the host PC for the last parameter.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.2.6 CLIENT_GetPwdSpecification

Item	Description	
Name	Get password rules.	
Function	BOOL CLIENT_GetPwdSpecification(const NET_IN_PWD_SPECI *pPwdSpeciIn, NET_OUT_PWD_SPECI *pPwdSpeciOut, DWORD dwWaitTime, char *szLocalIp);	
Parameter	[in]pPwdSpeciIn	Corresponds to structure of NET_IN_PWD_SPECI.
	[out]pPwdSpeciOut	Corresponds to structure of NET_OUT_PWD_SPECI.
	[in]dwWaitTime	Timeout.
	[in]szLocalIp	● In case of single network card, the last parameter is not required to be filled. ● In case of multiple network card, enter the IP of the host PC for the last parameter.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.2.7 CLIENT_StopSearchDevices

Item	Description	
Name	Stop searching.	
Function	BOOL CLIENT_StopSearchDevices (LLONG lSearchHandle);	
Parameter	[in] lSearchHandle	Searching handle.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	Multi-thread calling is not supported.	

3.3 Device Login

3.3.1 CLIENT_LoginEx2

Item	Description
Name	Login the device.

Item	Description	
Function	LLONG CLIENT_LoginEx2(const char *pchDVRIP, WORD wDVRPort, const char *pchUserName, const char *pchPassword, EM_LOGIN_SPAC_CAP_TYPE emSpecCap, void* pCapParam, LPNET_DEVICEINFO_Ex lpDeviceInfo, int *error);	
Parameter	[in]pchDVRIP	Device IP.
	[in]wDVRPort	Device port.
	[in]pchUserName	User name.
	[in]pchPassword	Password.
	[in]emSpecCap	Login category.
	[in]pCapParam	Login category parameter.
	[out]lpDeviceInfo	Device information.
	[out]error	Error for login failure.
Return value	● Success: Not 0. ● Failure: 0.	
Note	None.	

The following table shows information about error code:

Code of error	Meaning
1	Wrong password.
2	The user name does not exist.
3	Login timeout.
4	The account has logged in.
5	The account has been locked.
6	The account has been blacklisted.
7	The device resource is insufficient and the system is busy.
8	Sub connection failed.
9	Main connection failed.
10	Exceeds the maximum allowed number of user connections.
11	Lacks the dependent libraries such as avnetsdk or avnetsdk.
12	USB flash disk is not inserted or the USB flash disk information is wrong.
13	The IP at client is not authorized for login.

Table 3-1

3.3.2 CLIENT_Logout

Item	Description
Name	Logout the device.

Item	Description	
Function	BOOL CLIENT_Logout(LLONG lLoginID);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginEx2.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.4 Real-time Monitoring

3.4.1 CLIENT_RealPlayEx

Item	Description	
Name	Open the real-time monitoring.	
Function	LLONG CLIENT_RealPlayEx(LLONG lLoginID, int nChannelID, HWND hWnd, DH_RealPlayType rType);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginEx2.
	[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.	

The following table shows information about preview type:

Preview type	Meaning
DH_RType_Realplay	Real-time preview.
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.

Preview type	Meaning
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.

Table 3-2

3.4.2 CLIENT_StopRealPlayEx

Item	Description	
Name	Stop the real-time monitoring.	
Function	BOOL CLIENT_StopRealPlayEx(LLONG lRealHandle);	
Parameter	[in]lRealHandle	Return value of CLIENT_RealPlayEx.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.4.3 CLIENT_SaveRealData

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

3.4.4 CLIENT_StopSaveRealData

Item	Description	
Name	Stop saving the real-time monitoring data as file.	
Function	BOOL CLIENT_StopSaveRealData(LLONG lRealHandle);	
Parameter	[in] lRealHandle	Return value of CLIENT_RealPlayEx.
Return value	● Success: TRUE. ● Failure: FALSE.	

Item	Description
Note	None.

3.4.5 CLIENT_SetRealDataCallBackEx2

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

The following table shows information about parameter dwFlag:

dwFlag	Meaning
REALDATA_FLAG_RAW_DATA	Initial data labels.
REALDATA_FLAG_DATA_WITH_FRAME_INFO	Data labels with frame information.
REALDATA_FLAG_YUV_DATA	YUV data labels.
REALDATA_FLAG_PCM_AUDIO_DATA	PCM audio data labels.

Table 3-3

3.5 Download of Medial File

3.5.1 CLIENT_FindFileEx

Item	Description
Name	Query the media file per query condition.
Function	LLONG CLIENT_FindFileEx(LLONG lLoginID, EM_FILE_QUERY_TYPE emType, void* pQueryCondition, void* reserved, int waittime);

Item	Description	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] emType	Query information type of media file, see Table 3-4.
	[in] pQueryCondition	Query condition.
	[in]reserved	Reserved parameter, not valid.
	[in] waittime	Timeout.
Return value	● Success: Not 0. ● Failure: 0.	
Note	When querying the media file, use DH_FILE_QUERY_TRAFFICCAR_EX for parameter emType. The parameter pQueryCondition corresponds to structure MEDIA_QUERY_TRAFFICCAR_PARAM_EX.	

The following table shows information about parameter emType:

emType enumeration definition	Meaning	Corresponding structure of pQueryCondition
DH_FILE_QUERY_TRAFFICCAR	Traffic vehicles information	MEDIA_QUERY_TRAFFICCAR_PARAM
DH_FILE_QUERY_FACE	Face information	MEDIAFILE_FACERECOGNITION_PARAM
DH_FILE_QUERY_FILE	File information	NET_IN_MEDIA_QUERY_FILE
DH_FILE_QUERY_TRAFFICCAR_EX	Traffic vehicles information (extension)	MEDIA_QUERY_TRAFFICCAR_PARAM_EX
DH_FILE_QUERY_FACE_DETECTION	Face detection information	MEDIAFILE_FACE_DETECTION_PARAM

Table 3-4

3.5.2 CLIENT_GetTotalFileCount

Item	Description	
Name	Get the total number of queried files.	
Function	BOOL CLIENT_GetTotalFileCount(LLONG lFindHandle, int* pTotalCount, void* reserved, int waittime);	
Parameter	[in] lFindHandle	Return value of CLIENT_FindFileEx.
	[out] pTotalCount	The total number of queried information.
	[in]reserved	Reserved parameter, not valid.
	[in] waittime	Timeout.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.5.3 CLIENT_FindNextFileEx

Item	Description	
Name	Query the media file.	
Function	int CLIENT_FindNextFileEx(LLONG lFindHandle, int nFilecount, void* pMediaFileInfo, int maxlen, void* reserved, int waittime);	
Parameter	[in] lFindHandle	Return value of CLIENT_FindFileEx.
	[in] nFilecount	Query number.
	[out] pMediaFileInfo	Output cache of media file information.
	[in] maxlen	Value of maximum cache area.
	[in]reserved	Reserved parameter, not valid.
	[in] waittime	Timeout.
Return value	Returns the total number of queried media files. The query is called finished if the return value is smaller than the query number.	
Note	None.	

3.5.4 CLIENT_FindCloseEx

Item	Description	
Name	Stop querying the media file.	
Function	BOOL CLIENT_FindCloseEx(LLONG lFindHandle);	
Parameter	[in] lFindHandle	Return value of CLIENT_FindFileEx.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.5.5 CLIENT_DownloadMediaFile

Item	Description
Name	Download the media file.

Item	Description	
Function	LLONG CLIENT_DownloadMediaFile(LLONG lLoginID, EM_FILE_QUERY_TYPE emType, void* lpMediaFileInfo, char* sSavedFileName, fDownLoadPosCallBack cbDownLoadPos, LDWORD dwUserData, void* reserved);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] emType	Media file type, see Table 3-4.
	[in] lpMediaFileInfo	Media file information.
	[in] sSavedFileName	Save path.
	[in] cbDownLoadPos	Callback of download progress: fDownLoadPosCallBack.
	[in] dwUserData	Corresponding user number of callback.
	[in]reserved	Reserved parameter, not valid.
Return value	● Success: Not 0. ● Failure: 0.	
Note	When downloading vehicles pictures, the parameter emType only supports DH_FILE_QUERY_TRAFFICCAR.	

3.5.6 CLIENT_StopDownloadMediaFile

Item	Description	
Name	Stop downloading the media file.	
Function	BOOL CLIENT_StopDownloadMediaFile(LLONG lFileHandle);	
Parameter	[in] lFindHandle	Return value of CLIENT_DownloadMediaFile.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.6 Manual Capture

3.6.1 CLIENT_RealLoadPictureEx

Item	Description
Name	Subscribe intelligent event.

Item	Description	
Function	LLONG CLIENT_RealLoadPictureEx(LLONG lLoginID, int nChannelID, DWORD dwAlarmType, BOOL bNeedPicFile, fAnalyzerDataCallBack cbAnalyzerData, LDWORD dwUser, void* Reserved);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] nChannelID	Device channel number.
	[in] dwAlarmType	Type of intelligent traffic event, see Table 3-4 and Table 3-5.
	[in] bNeedPicFile	Whether picture is needed.
	[in] cbAnalyzerData	Callback of intelligent event: fAnalyzerDataCallBack.
	[in] dwUser	Corresponding user data of callback.
	[in]Reserved	Reserved parameter, not valid.
Return value	● Success: Not 0. ● Failure: 0.	
Note	● Call this interface in advance for manual capturing to receive the captured pictures. ● Call this interface in advance for event upload to receive the event information and pictures.	

The following table shows information about parameter dwAlarmType:

dwAlarmType macro definition	Value of macro definition	Meaning	Call the corresponding structure of pAlarmInfo
EVENT_IVS_TRAFFIC_MANUALSNAP	0x00000118	Intelligent capturing event	DEV_EVENT_TRAFFIC_MANUALSNAP_INFO

Table 3-5

3.6.2 CLIENT_ControlDeviceEx

Item	Description	
Name	Control device.	
Function	BOOL CLIENT_ControlDeviceEx(LLONG lLoginID, CtrlType emType, void* pInBuf, void* pOutBuf, int nWaitTime);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] emType	Control type, see Table 3-5 and Table 3-6.
	[in] pInBuf	Control input cache, see Table 3-5 and Table 3-6.

Item	Description	
	[in] pOutBuf	Controls output cache.
	[in] nWaitTime	Timeout.
Return value	● Success: TRUE ● Failure: FALSE	
Note	Manually trigger the capturing and receive pictures through subscribing the callback of interface.	

The following table shows information about parameter emType:

emType enumeration definition	Meaning	The corresponding structure of pInBuf
DH_MANUAL_SNAP	Manual capture	MANUAL_SNAP_PARAMETER

Table 3-6

3.6.3 CLIENT_StopLoadPic

Item	Description	
Name	Cancel subscription of intelligent event.	
Function	BOOL CLIENT_StopLoadPic(LLONG lAnalyzerHandle);	
Parameter	[in] lAnalyzerHandle	Return value of CLIENT_RealLoadPictureEx.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	After calling this interface, you will not receive the pictures even if continue to trigger manual capturing.	

3.7 Upload of Intelligent Traffic Event

3.7.1 CLIENT_RealLoadPictureEx

For the interface function, see "3.6.1 CLIENT_RealLoadPictureEx."

For the type of intelligent traffic event, see Table 3-7.

dwAlarmType macro definition	Value of macro definition	Meaning	Corresponding structure of pAlarmInfo
EVENT_IVS_ALL	0x00000001	All events	No
EVENT_IVS_TRAFFICCONTROL	0x00000015	Event of traffic control	DEV_EVENT_TRAFFICCONTROL_INFO
EVENT_IVS_TRAFFICACCIDENT	0x00000016	Event of traffic accident	DEV_EVENT_TRAFFICACCIDENT_INFO
EVENT_IVS_TRAFFICJUNCTION	0x00000017	Event of traffic conjunction	DEV_EVENT_TRAFFICJUNCTION_INFO
EVENT_IVS_TRAFFICGATE	0x00000018	Event of traffic gate	DEV_EVENT_TRAFFICGATE_INFO

dwAlarmType macro definition	Value of macro definition	Meaning	Corresponding structure of pAlarmInfo
EVENT_IVS_TRAFFIC_RUNREDLIGHT	0x00000100	Event of running the red light	DEV_EVENT_TRAFFIC_RUNREDLIGHT_INFO
EVENT_IVS_TRAFFIC_OVERLINE	0x00000101	Event of running over line	DEV_EVENT_TRAFFIC_OVERLINE_INFO
EVENT_IVS_TRAFFIC_RETROGRADE	0x00000102	Event of retrograde	DEV_EVENT_TRAFFIC_RETROGRADE_INFO
EVENT_IVS_TRAFFIC_TURNLEFT	0x00000103	Event of violating regulations by left turn	DEV_EVENT_TRAFFIC_TURNLEFT_INFO
EVENT_IVS_TRAFFIC_TURNRIGHT	0x00000104	Event of violating regulations by right turn	DEV_EVENT_TRAFFIC_TURNRIGHT_INFO
EVENT_IVS_TRAFFIC_UTURN	0x00000105	Event of violating regulations by turning around	DEV_EVENT_TRAFFIC_UTURN_INFO
EVENT_IVS_TRAFFIC_OVERSPEED	0x00000106	Event of running over speed	DEV_EVENT_TRAFFIC_OVERSPEED_INFO
EVENT_IVS_TRAFFIC_UNDERSPEED	0x00000107	Event of running under speed	DEV_EVENT_TRAFFIC_UNDERSPEED_INFO
EVENT_IVS_TRAFFIC_PARKING	0x00000108	Event of illegal parking	DEV_EVENT_TRAFFIC_PARKING_INFO
EVENT_IVS_TRAFFIC_WRONGROUTE	0x00000109	Event of running along the wrong route	DEV_EVENT_TRAFFIC_WRONGROUTE_INFO
EVENT_IVS_TRAFFIC_CROSSLANE	0x0000010A	Event of violating regulations by crossing lanes	DEV_EVENT_TRAFFIC_CROSSLANE_INFO
EVENT_IVS_TRAFFIC_OVERYELLOWLINE	0x0000010B	Event of running on the yellow line	DEV_EVENT_TRAFFIC_OVERYELLOWLINE_INFO
EVENT_IVS_TRAFFIC_DRIVINGONSHOULDER	0x0000010C	Event of running on the road shoulder	DEV_EVENT_TRAFFIC_DRIVINGONSHOULDER_INFO
EVENT_IVS_TRAFFIC_YELLOWPLATEINLANE	0x0000010E	Event of yellow plate occupying the lanes	DEV_EVENT_TRAFFIC_YELLOWPLATEINLANE_INFO

dwAlarmType macro definition	Value of macro definition	Meaning	Corresponding structure of pAlarmInfo
EVENT_IVS_TRAFFIC_PEDESTRAINPRIORITY	0x0000010F	Event of pedestrian priority at zebra crossing	DEV_EVENT_TRAFFIC_PEDESTRAINPRIORITY_INFO
EVENT_IVS_TRAFFIC_PARKINGONYELLOWBOX	0x0000012A	Event of capturing the cars parking at the yellow box	DEV_EVENT_TRAFFIC_PARKINGONYELLOWBOX_INFO
EVENT_IVS_TRAFFIC_PARKINGSPACEPARKING	0x0000012B	Event of parking space taken by cars	DEV_EVENT_TRAFFIC_PARKINGSPACEPARKING_INFO
EVENT_IVS_TRAFFIC_PARKINGSPACENOPARKING	0x0000012C	Event of parking space taken by no cars	DEV_EVENT_TRAFFIC_PARKINGSPACENOPARKING_INFO
EVENT_IVS_TRAFFIC_PEDESTRAIN	0x0000012D	Event about pedestrian	DEV_EVENT_TRAFFIC_PEDESTRAIN_INFO
EVENT_IVS_TRAFFIC_THROW	0x0000012E	Event of throwing objects	DEV_EVENT_TRAFFIC_THROW_INFO
EVENT_IVS_TRAFFIC_IDLE	0x0000012F	Idle event	DEV_EVENT_TRAFFIC_IDLE_INFO
EVENT_IVS_TRAFFIC_RESTRICTED_PLATE	0X00000136	Event of restricted plate	DEV_EVENT_TRAFFIC_RESTRICTED_PLATE
EVENT_IVS_TRAFFIC_OVERSTOPLINE	0X00000137	Event of pressing on the stop line	DEV_EVENT_TRAFFIC_OVERSTOPLINE
EVENT_IVS_TRAFFIC_WITHOUT_SAFEBELT	0x00000138	Event of safety belt unfastened	DEV_EVENT_TRAFFIC_WITHOUT_SAFEBELT
EVENT_IVS_TRAFFIC_DRIVER_SMOKING	0x00000139	Event of driver smoking	DEV_EVENT_TRAFFIC_DRIVER_SMOKING
EVENT_IVS_TRAFFIC_DRIVER_CALLING	0x0000013A	Event of driver calling	DEV_EVENT_TRAFFIC_DRIVER_CALLING
EVENT_IVS_TRAFFIC_PEDESTRAINRUNREDLIGHT	0x0000013B	Event of pedestrian running the red light	DEV_EVENT_TRAFFIC_PEDESTRAINRUNREDLIGHT_INFO
EVENT_IVS_TRAFFIC_PASSNOTINORDER	0x0000013C	Event of passing without order	DEV_EVENT_TRAFFIC_PASSNOTINORDER_INFO

Table 3-7

3.7.2 CLIENT_StopLoadPic

For the interface function, see "3.6.3 CLIENT_StopLoadPic."

3.8 Vehicle Flow Statistics

3.8.1 CLIENT_StartTrafficFluxStat

Item	Description	
Name	Subscribe the statistics of vehicle flow.	
Function	LLONG CLIENT_StartTrafficFluxStat(LLONG lLoginID, NET_IN_TRAFFICFLUXSTAT* pstInParam, NET_OUT_TRAFFICFLUXSTAT* pstOutParam);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2
	[in] pstInParam	Input parameter. Vehicle flow statistics callback: fFluxStatDataCallBack.
	[out] pstOutParam	Output parameter.
Return value	● Success: Not 0. ● Failure: 0.	
Note	None.	

3.8.2 CLIENT_StopTrafficFluxStat

Item	Description	
Name	Stop subscribing the statistics of vehicle flow	
Function	BOOL CLIENT_StopTrafficFluxStat(LLONG lFluxStatHandle);	
Parameter	[in] lFluxStatHandle	Return value of CLIENT_StartTrafficFluxStat
Return value	● Success: TRUE ● Failure: FALSE	
Note	None	

3.9 Barrier Control

For the interface function, see "3.6.2 CLIENT_ControlDeviceEx."

For the barrier control type, see Table 3-8.

emType enumeration definition	Meaning	Corresponding structure of pInBuf
DH_CTRL_OPEN_STROBE	Open barrier	NET_CTRL_OPEN_STROBE

emType enumeration definition	Meaning	Corresponding structure of pInBuf
DH_CTRL_CLOSE_STROBE	Close barrier	NET_CTRL_CLOSE_STROBE

Table 3-8

4 Callback Definition

4.1 fSearchDevicesCB

Item	Description	
Name	Callback of searching devices.	
Function	typedef void(CALLBACK *fSearchDevicesCB)(DEVICE_NET_INFO_EX * pDevNetInfo, void* pUserData);	
Parameter	[out]pDevNetInfo	The searched device information.
	[out]pUserData	User data.
Return value	None.	
Note	None.	

4.2 fDisConnect

Item	Description	
Name	Disconnection callback.	
Function	typedef void (CALLBACK *fDisConnect)(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
Parameter	[out] lLoginID	Return value of CLIENT_LoginEx2.
	[out] pchDVRIP	IP of the disconnected device.
	[out] nDVRPort	Port of the disconnected device.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.3 fHaveReConnect

Item	Description
Name	Reconnection callback.

Item	Description	
Function	typedef void (CALLBACK *fHaveReConnect)(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
Parameter	[out] lLoginID	Return value of CLIENT_LoginEx2.
	[out] pchDVRIP	IP of the reconnected device.
	[out] nDVRPort	Port of the reconnected device.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.4 fRealDataCallBackEx2

Item	Description	
Name	Callback of real-time monitoring data.	
Function	typedef void (CALLBACK * fRealDataCallBackEx2)(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LLONG param, LDWORD dwUser);	
Parameter	[out] lRealHandle	Return value of CLIENT_RealPlayEx.
	[out] dwDataType	Data type: ● 0: Initial data. ● 1: Data with frame information. ● 2: YUV data. ● 3: PCM audio data.
	[out] pBuffer	Address of monitoring data block.
	[out] dwBufSize	Length (unit: byte) of the monitoring data block
	[out] param	Callback parameter structure. Different dwDataType value corresponds to different type. ● The param is blank pointer when dwDataType is 0. ● The param is the pointer of tagVideoFrameParam structure when dwDataType is 1. ● The param is the pointer of tagCBYUVDataParam structure when dwDataType is 2. ● The param is the pointer of

Item	Description	
		tagCBPCMDataParam structure when dwDataType is 3.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.5 fDownLoadPosCallBack

Item	Description	
Name	Callback of media file download process.	
Function	typedef void (CALLBACK *fDownLoadPosCallBack)(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, LDWORD dwUser);	
Parameter	[out]lPlayHandle	Return value of CLIENT_DownloadMediaFile.
	[out]dwTotalSize	Total size.
	[out]dwDownLoadSize	The downloaded data size. ● -1: Download finish. ● -2: Data write error during downloading.
	[out]dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.6 fAnalyzerDataCallBack

Item	Description	
Name	Callback of intelligent event information.	
Function	typedef int (CALLBACK *fAnalyzerDataCallBack)(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE* pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void* reserved);	
Parameter	[out]lAnalyzerHandle	Return value of CLIENT_RealLoadPictureEx.
	[out]dwAlarmType	Type of intelligent event, see Table 3-5.

Item	Description	
	[out]pAlarmInfo	Cache of event information, see Table 3-5.
	[out]pBuffer	Pictures cache.
	[out]dwBufSize	Cache size of pictures.
	[out]dwUser	User parameter of the callback.
	[out]reserved	Reserved.
Return value	None.	
Note	None.	

4.7 fFluxStatDataCallBack

Item	Description	
Name	Callback of intelligent event information.	
Function	typedef int (CALLBACK *fFluxStatDataCallBack)(LLONG lFluxStatHandle, DWORD dwEventType, void* pEventInfo, BYTE* pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void* reserved);	
Parameter	[out]lFluxStatHandle	Return value of CLIENT_StartTrafficFluxStat.
	[out]dwEventType	Type of intelligent event information.
	[out]pEventInfo	Vehicle flow event information.
	[out]pBuffer	Data cache.
	[out]dwBufSize	Data size.
	[out]dwUser	User parameter of the callback.
	[out]nSequence	Sequence.
	[out]reserved	Reserved.
Return value	None.	
Note	The pEventInfo corresponds to DEV_EVENT_TRAFFIC_FLOWSTAT_INFO structure.	