

NetSDK (Thermal Camera)

Programming Manual



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 Thermal Camera (TPC). For more function modules and data structures, refer to *NetSDK Development Manual*. For detailed information on basic service processes, including initialization, login, general alarms and intelligent alarms, refer to *NetSDK Programming Guide*.






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

Readers

- SDK software development engineers
- Project managers
- Product managers

Safety Instructions

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

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

Revision History

Version	Revision Content	Release Time
V2.0.0	For detailed information on basic services, refer to <i>NetSDK Programming Guide</i> .	February 2025
V1.0.0	First release.	June 2024

Privacy Protection Notice

As the device user or data controller, you might collect personal data of others such as face, fingerprints, car plate number, email address, phone number, GPS and so on. You need to be in compliance with the local privacy protection laws and regulations to protect the legitimate rights and interests of other people by implementing measures include but not limited to: providing clear and visible identification to inform data subject the existence of surveillance area and providing related contact.

About the Manual

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

Glossary

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

Term	Definition
Main Stream	A type of video stream that usually has better resolution and clarity and provides a better experience if the network resource is not restricted.
Sub Stream	A type of video stream that usually has lower resolution and clarity than the main stream but demands less network resources. The user can choose the stream type according to the particular scenes.
Video Channel	The video is numbered from 0 and each video receives a channel number. Currently, except the TPC, the other types of devices usually have only one channel that is numbered as 0.
Login Handle	The first step to access to the device is login (authentication). The device receives a unique ID that refers to the login handle upon the successful login. This handle will be used by the subsequent procedures and stay valid until logout.
Relative Positioning	A fast positioning method in PTZ control by providing the difference value of the PTZ coordinates (X-axis and Y-axis) to the device which accord to the present PTZ location and the difference value to calculate and transfer to the final location. This method also supports ZOOM control.
Absolute Positioning	A fast positioning method in PTZ control which provides certain horizontal and vertical coordinates (angular coordinate) to the device. The device directly transfers to the user specified location. This method also supports ZOOM control.
PCM	Pulse Code Modulation is one of the coding methods of digital communication and converts the analog signal into digital signal without encoding loss. It is suitable for the user who requires higher data transfer rate and bandwidth.
PTZ	Pan Tilt Zoom is all-round movement and lens zoom control.
Heat Map (Temperature)	Corresponds to the temperature distribution data of a scene and each pixel has a data point.
Heat Map (Activity)	Indicates a picture with using different colors to show the statistical activity of an area during a certain period.

Table of Contents

Foreword	I
Glossary	III
1 Overview	1
1.1 General.....	1
1.2 Applicability	2
2 Function Modules	3
2.1 SDK Initialization	3
2.1.1 Introduction.....	3
2.1.2 Interface Overview	3
2.1.3 Process	3
2.1.4 Example Code	4
2.2 Device Login	5
2.2.1 Introduction.....	5
2.2.2 Interface Overview	5
2.2.3 Process	5
2.2.4 Example Code	6
2.3 Live Viewing	7
2.3.1 Introduction.....	7
2.3.2 Interface Overview	7
2.3.3 Process	8
2.3.4 Example Code	11
2.4 Video Snapshot.....	12
2.4.1 Introduction.....	12
2.4.2 Interface Overview	12
2.4.3 Process	12
2.4.4 Example Code	15
2.5 Alarm Sending.....	15
2.5.1 Introduction.....	15
2.5.2 Interface Overview	15
2.5.3 Process	16
2.5.4 Example Code	17
2.6 AI Event Subscription	18
2.6.1 Introduction.....	18
2.6.2 Interface Overview	18
2.6.3 Process	19
2.6.4 Example Code	20
2.7 Heat Map (Temperature).....	20
2.7.1 Introduction.....	20
2.7.2 Interface Overview	21
2.7.3 Process	21
2.7.4 Example Code	22
2.8 PTZ Control.....	23
2.8.1 Preset.....	23

2.8.2 PTZ Absolute Position	26
2.8.3 Continuous PTZ Control	28
2.9 Fire Alarm	31
2.9.1 Introduction.....	31
2.9.2 Interface Overview	32
2.9.3 Process	32
2.9.4 Example Code	35
2.10 Getting Cold/Hot Spot.....	36
2.10.1 Introduction	36
2.10.2 Interface Overview	36
2.10.3 Process.....	36
2.10.4 Example Code.....	37
2.11 Temperature Information of Random Region.....	38
2.11.1 Introduction	38
2.11.2 Interface Overview	38
2.11.3 Process.....	38
2.11.4 Example Code.....	39
2.12 Temperature Measurement	40
2.12.1 Rule Setting	40
2.12.2 Global Setting	43
2.12.3 Abnormal Temperature Alarm.....	46
2.12.4 Getting Temperature Information.....	49
2.12.5 Initiatively Sending Temperature Information	51
3 Interface Definition	55
3.1 SDK Initialization	55
3.1.1 SDK CLIENT_Init.....	55
3.1.2 CLIENT_Cleanup.....	55
3.1.3 CLIENT_SetAutoReconnect.....	55
3.1.4 CLIENT_SetNetworkParam	56
3.2 Device Login	56
3.2.1 CLIENT_LoginWithHighLevelSecurity	56
3.2.2 CLIENT_Logout	57
3.3 Live Viewing	57
3.3.1 CLIENT_RealPlayEx	57
3.3.2 CLIENT_StopRealPlayEx.....	58
3.3.3 CLIENT_SaveRealData	59
3.3.4 CLIENT_StopSaveRealData.....	59
3.3.5 CLIENT_SetRealDataCallBackEx2	59
3.4 Video Snapshot	60
3.4.1 CLIENT_SnapPictureToFile.....	60
3.4.2 CLIENT_CapturePictureEx.....	60
3.5 Alarm Sending.....	61
3.5.1 CLIENT_SetDVRMessCallBack	61
3.5.2 CLIENT_StartListenEx	61
3.5.3 CLIENT_StopListen	62
3.6 AI Event Subscription	63
3.6.1 CLIENT_RealLoadPictureEx	63

3.6.2 CLIENT_StopLoadPic	64
3.7 Heat Map	64
3.7.1 CLIENT_RadiometryAttach.....	64
3.7.2 CLIENT_RadiometryDetach	65
3.7.3 CLIENT_RadiometryFetch.....	65
3.7.4 CLIENT_RadiometryDataParse.....	65
3.8 PTZ Control.....	66
3.8.1 CLIENT_QueryDevState.....	66
3.8.2 CLIENT_DHPTZControlEx2	67
3.9 Fire Alarm	67
3.9.1 CLIENT_RealLoadPictureEx	67
3.9.2 CLIENT_StopLoadPic	68
3.9.3 CLIENT_SetDVRMessCallBack	68
3.9.4 CLIENT_StartListenEx	69
3.9.5 CLIENT_StopListen	69
3.10 RadiometryGetCurrentHotColdSpotInfo.....	69
3.11 RadiometryGetRandomRegionTemper.....	70
3.12 Temperature Measurement	70
3.12.1 CLIENT_GetNewDevConfig	70
3.12.2 CLIENT_ParseData.....	71
3.12.3 CLIENT_PacketData	72
3.12.4 LIENT_SetNewDevConfig.....	72
3.12.5 CLIENT_SetDVRMessCallBack.....	73
3.12.6 CLIENT_StartListenEx.....	73
3.12.7 CLIENT_StopListen.....	74
3.12.8 CLIENT_RadiometryGetCurTemperAll.....	75
3.12.9 CLIENT_RadiometryAttachTemper	75
3.12.10 CLIENT_RadiometryAttachTemper	76
4 Callback Definition	77
4.1 fDisconnect.....	77
4.2 fHaveReConnect.....	77
4.3 fRealDataCallBackEx2	78
4.4 fMessCallBack.....	79
4.5 fAnalyzerDataCallBack	80
4.6 fRadiometryAttachCB.....	80
Appendix 1 Cybersecurity Recommendations	82

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, PTZ control, voice talk, video snapshot, heat map and heat map.

The development kit might be different dependent on the environment.

Table 1-1 Files of Windows development kit

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
Auxiliary library of "dhnetSDK.dll"	lvsDrawer.dll	Image display library
	StreamConvertor.dll	Transcoding library

Table 1-2 Files of Linux development kit

Library type	Library file name	Library file description
Function library	dhnetSDK.h	Header file
	libdhnetSDK.so	Library file
	libavnetSDK.so	Library file
Configuration library	avglobal.h	Header file
	dhconfigSDK.h	Configuration Header file
	libdhconfigSDK.so	Configuration library
Auxiliary library of "libdhnetSDK.so"	libStreamConvertor.so	Transcoding library



- 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.

- We recommend you 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.

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

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.

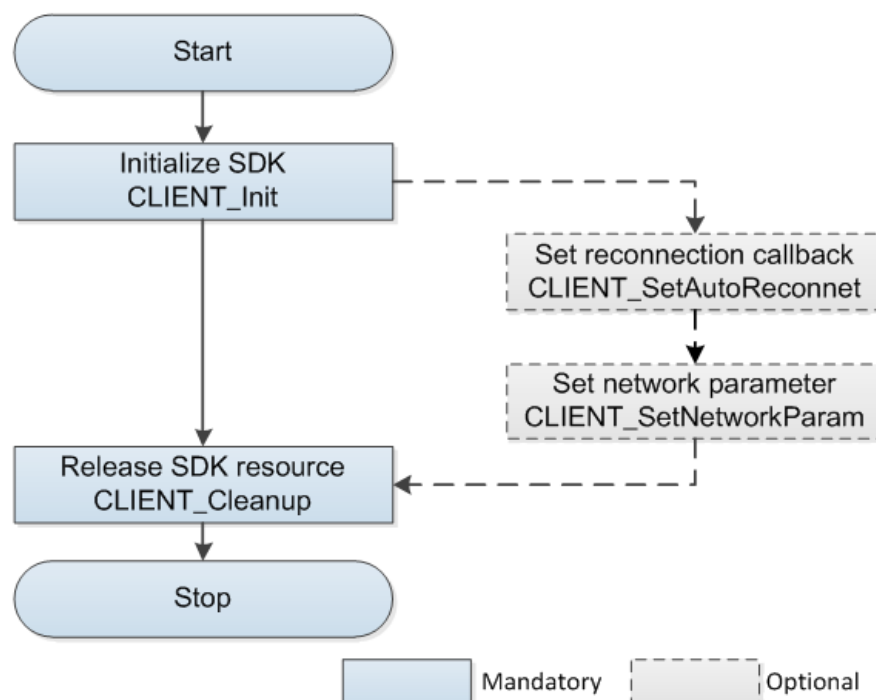
2.1.2 Interface Overview

Table 2-1 Interfaces of SDK initialization

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

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 **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 single thread 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 the device 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 this callback.
```

```
void CALLBACK DisconnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser)
```

```
{  
    printf("Call DisconnectFunc: lLoginID[0x%x]\n", lLoginID);  
}
```

```
// Initialize SDK
```

```
BOOL bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0);
```

```
if (FALSE == bNetSDKInitFlag)
```

```
{  
    printf("Initialize client SDK fail; \n");  
    return -1;  
}
```

```
// Clean up the SDK resource
```

```
if (TRUE == bNetSDKInitFlag)
```

```
{  
    CLIENT_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 get a unique login ID upon login to the device and should call login ID before using other SDK interfaces. The login ID becomes invalid once logged out.

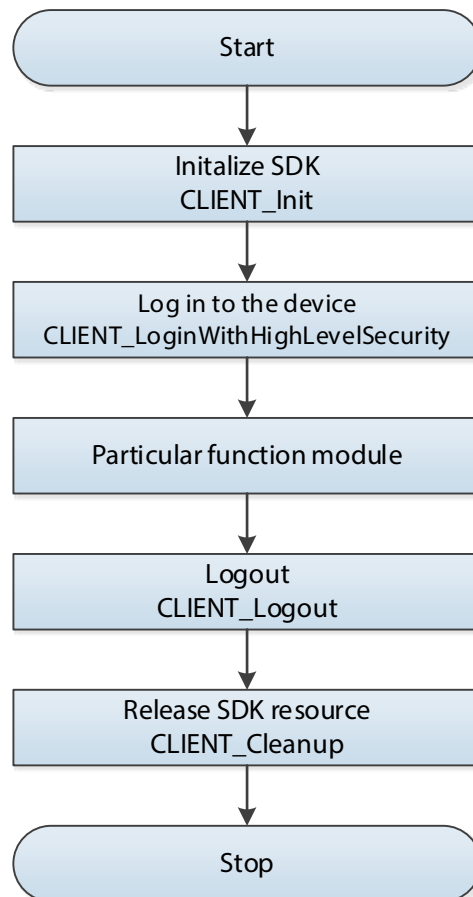
2.2.2 Interface Overview

Table 2-2 Interfaces of device login

Interface	Implication
CLIENT_LoginWithHighLevelSecurity	Log in to the device with high level security. CLIENT_LoginEx2 can still be used, but there are security risks, so it is highly recommended to use the interface CLIENT_LoginWithHighLevelSecurity to log in to the device.
CLIENT_Logout	Logout.

2.2.3 Process

Figure 2-2 Process of device login



Process Description

- Step 1** Call **CLIENT_Init** to initialize SDK.
- Step 2** Call **CLIENT_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 **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 log in 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.

Table 2-3 Common error code

Interface	Implication
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

For more information about error codes, see "CLIENT_LoginWithHighLevelSecurity interface" in *Network SDK Development Manual.chm*. The example code to avoid error code 3 is as follows.

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

2.2.4 Example Code

```

NET_DEVICEINFO_Ex stDevInfo = {0};
int nError = 0;
// Log in to the device
NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);

LLONG ILoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

// Logout the device
if (0 != ILoginHandle)
{
    CLIENT_Logout(ILoginHandle);
}

```

2.3 Live Viewing

2.3.1 Introduction

Real-time viewing gets 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 to 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.

2.3.2 Interface Overview

Table 2-4 Interfaces of real-time monitoring

Interface	Implication
CLIENT_RealPlayEx	Start real-time viewing extension interface.
CLIENT_StopRealPlayEx	Stop real-time viewing extension interface.

Interface	Implication
CLIENT_SaveRealData	Start saving the real-time viewing data to the local path.
CLIENT_StopSaveRealData	Stop saving the real-time viewing data to the local path.
CLIENT_SetRealDataCallBackEx2	Set real-time viewing data callback function extension interface.

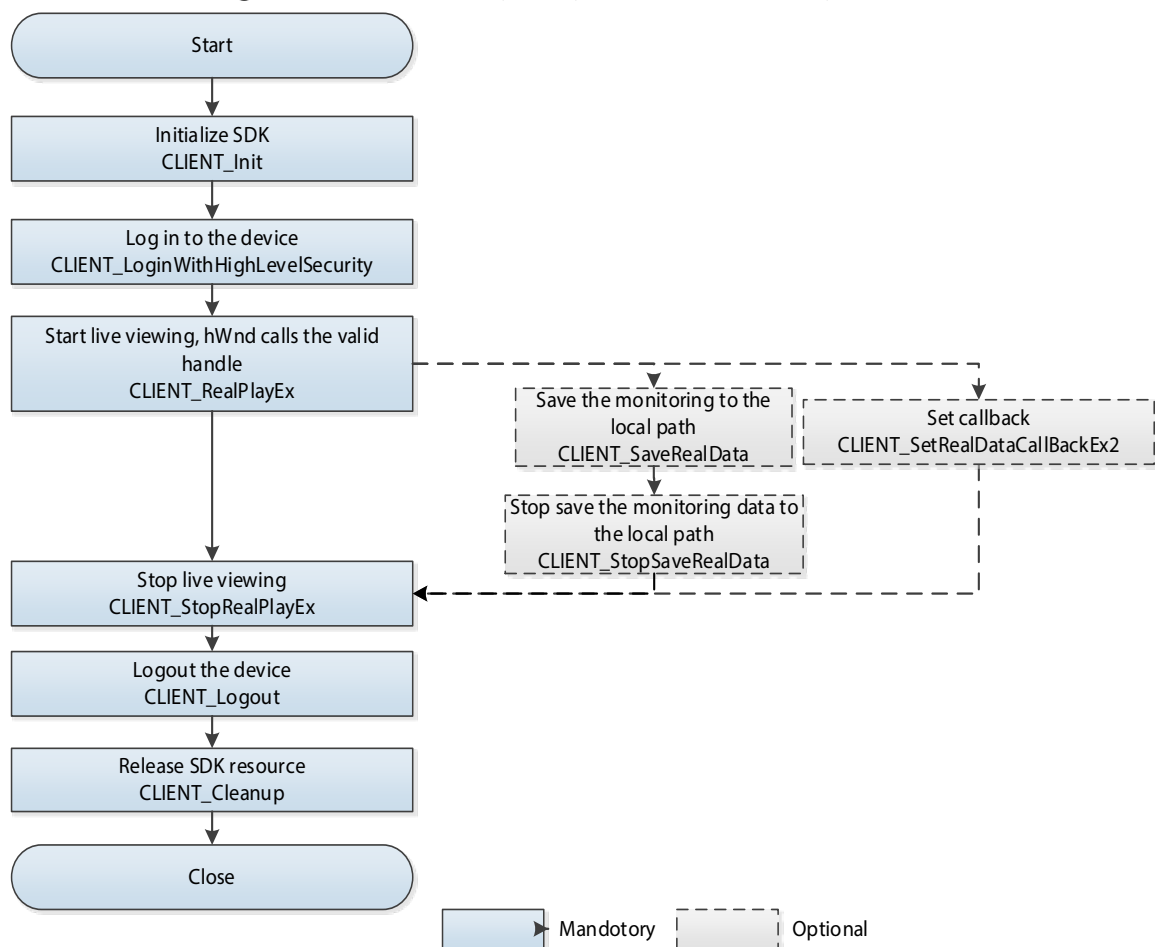
2.3.3 Process

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

2.3.3.1 SDK Decoding Play

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

Figure 2-3 Process of playing by SDK decoding library



Process Description

- Step 1 Call **CLIENT_Init** to initialize SDK.
- Step 2 Call **CLIENT_LoginWithHighLevelSecurity** to log in to 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 saved as 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**.

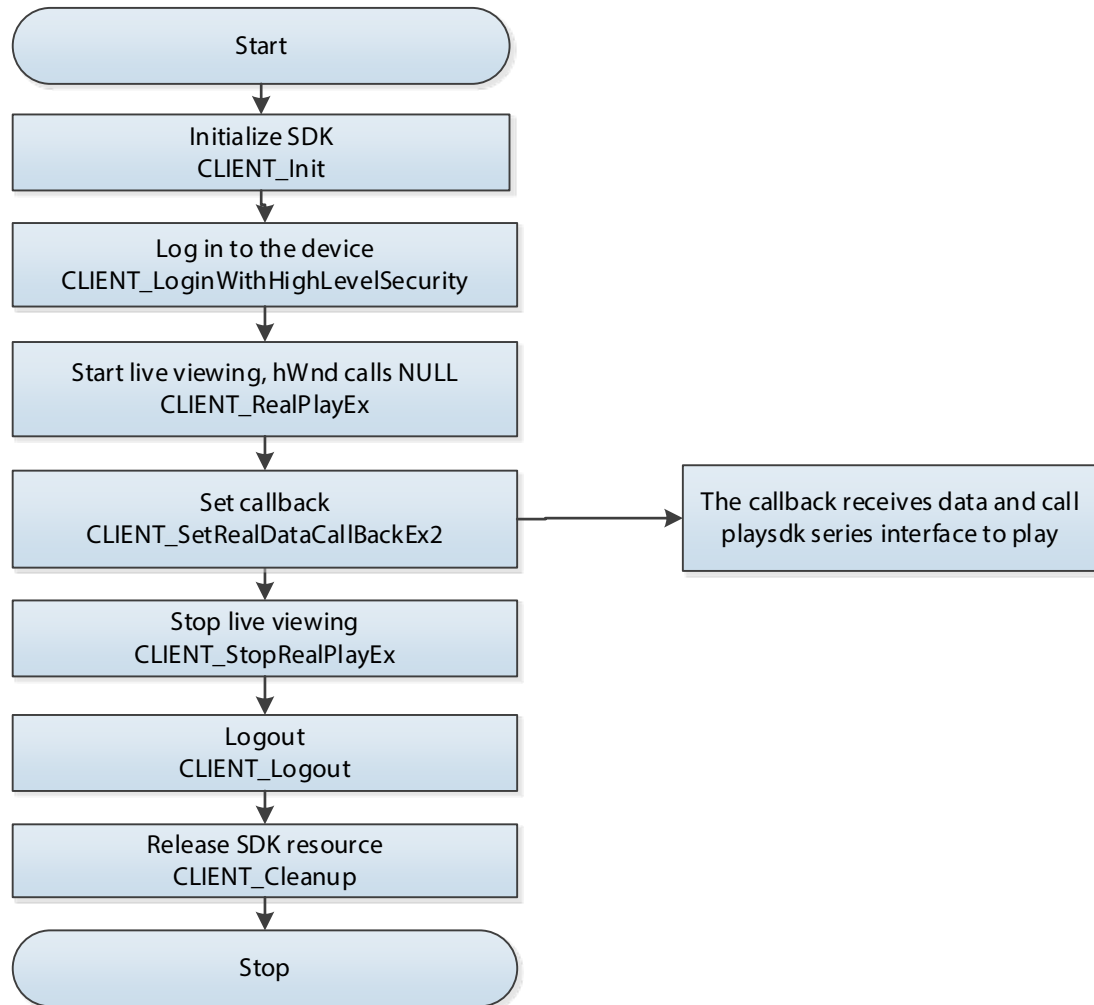
```
NET_PARAM stuNetParam = {0};
stuNetParam.nGetConnInfoTime = 5000; // unit ms
CLIENT_SetNetworkParam (&stuNetParam);
```

- Failed to repeat opening: For some models, the same channel cannot be opened for multiple times during the one entire logged in status. If you are trying to open it repeatedly, you will success in the first try but get failed afterwards. In this case, you can try the following:
 - ◇ Close the opened channel. For example, if you already opened the main stream video on the channel 1 and still want to open the sub stream video on the same channel, you can close the main stream first and then open the sub stream.
 - ◇ Login twice to get 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_FAILED_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 "2.3.3.2 Call Third Party Play Library."

2.3.3.2 Call Third Party Play Library

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

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



Process Description

- Step 1 Call **CLIENT_Init** to initialize SDK.
- Step 2 Call **CLIENT_LoginWithHighLevelSecurity** to log in to 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 live viewing, call **CLIENT_StopRealPlayEx** to stop live viewing.
- 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 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 Example Code

2.3.4.1 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 IRealHandle = CLIENT_RealPlayEx(ILoginHandle, 0, hWnd, DH_RType_Realplay);
if (NULL == IRealHandle)
{
    printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
}
printf("input any key to quit!\n");
getchar();
// Stop live viewing
if (NULL != IRealHandle)
{
    CLIENT_StopRealPlayEx(IRealHandle);
}
```

2.3.4.2 Call Play Library

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

```

printf("input any key to quit!\n");
getchar();
// Stop live viewing
if (0 != IRealHandle)
{
    CLIENT_StopRealPlayEx(IRealHandle);
}

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

```

2.4 Video Snapshot

2.4.1 Introduction

Video snapshot can get the picture data of the playing video. This section introduces the following two snapshot ways:

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

2.4.2 Interface Overview

Table 2-5 Interfaces of video snapshot

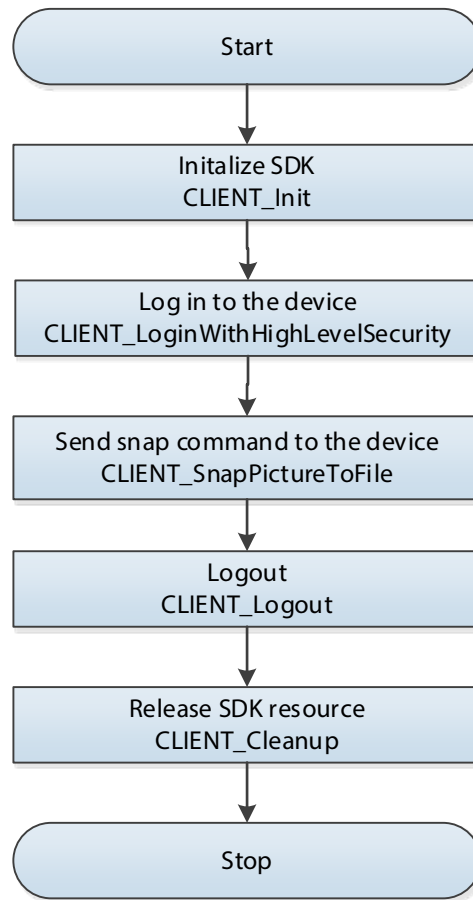
Interface	Implication
CLIENT_SnapPictureToFile	Snap picture and send to the user.
CLIENT_CapturePictureEx	Snap local pictures and the parameters could be live viewing handle or playback handle.

2.4.3 Process

Video snapshot is consisted of network snapshot and local snapshot.

2.4.3.1 Network Snapshot

Figure 2-5 Process of network snapshot



Process Description

- Step 1 Call **CLIENT_Init** to initialize SDK.
- Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.
- Step 3 Call **CLIENT_SnapPictureToFile** to get the picture data.
- Step 4 Call **CLIENT_Logout** to logout the device.
- Step 5 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

- Picture size limit: SDK allocates the fixed memory to receive the picture data returned from the device. If the picture is larger than the fixed memory, SDK will return the truncated data.
- SDK provides the interface to modify the default memory. If the picture (for example, the high definition picture) is truncated, you can modify the value of nPicBufSize bigger. The example code is as follows. After calling **CLIENT_Init**, call the example code just one time.

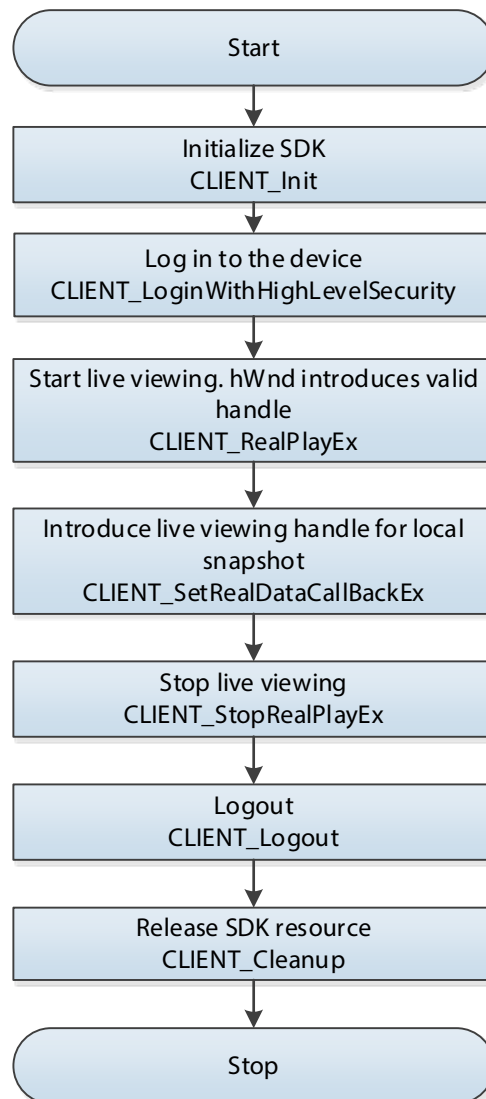
```
NET_PARAM stuNetParam = {0};  
stuNetParam.nPicBufSize = 4*1024*1024; // unit byte  
CLIENT_SetNetworkParam(&stuNetParam);
```

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

- Snapshot configuration: You can configure the items such as quality and definition for the snapshot. However, if the default configurations are satisfactory, do not modify them. For more details of the example code, see the SDK package on the website
- Picture save format: The picture data returns as memory and the interface supports saving it as file (the precondition is that you have set the `szFilePath` field of `NET_IN_SNAP_PIC_TO_FILE_PARAM`).

2.4.3.2 Local Snapshot

Figure 2-6 Process of local snapshot



Process Description

- Step 1 Call **CLIENT_Init** to initialize SDK.
- Step 2 Call **CLIENT_LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **CLIENT_RealPlayEx** to start monitoring and get the live viewing handle.
- Step 4 Call **CLIENT_CapturePictureEx** to introduce the live viewing handle.
- Step 5 Call **CLIENT_StopRealPlayEx** to stop the live viewing.
- Step 6 Call **CLIENT_Logout** to logout the device.
- Step 7 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.4.4 Example Code

```
// Network snapshot example
NET_IN_SNAP_PIC_TO_FILE_PARAM stuInParam = {sizeof(stuInParam)};
NET_OUT_SNAP_PIC_TO_FILE_PARAM stuOutParam = {sizeof(stuOutParam)};
SNAP_PARAMS stuSnapParams = {0};
stuSnapParams.Channel = 0;    // Take the first channel as an example
int nBufferLen = 2*1024*1024;
char* pBuffer = new char[nBufferLen]; // Picture cache
memset(pBuffer, 0, nBufferLen);
stuOutParam.szPicBuf = pBuffer;
stuOutParam.dwPicBufLen = nBufferLen;
if (FALSE == CLIENT_SnapPictureToFile(ILoginHandle, &stuSnapParams))
{
    printf("CLIENT_SnapPictureEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
}
delete[] pBuffer;

// Example of local snapshot. The handle hPlayHandle is gotten from opening live viewing.
if (FALSE == CLIENT_CapturePictureEx(hPlayHandle, "test.jpg", NET_CAPTURE_JPEG))
{
    printf("CLIENT_CapturePictureEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
}
```

2.5 Alarm Sending

2.5.1 Introduction

The implementation method for alarm sending is to log in to the device through the SDK and subscribe to the alarm function. If the device detects an alarm event, it will immediately send it to SDK. The corresponding alarm information can be gotten through the alarm callback function.

Before subscribing to behavior detection events, you need to configure behavior detection rules on the device web page.

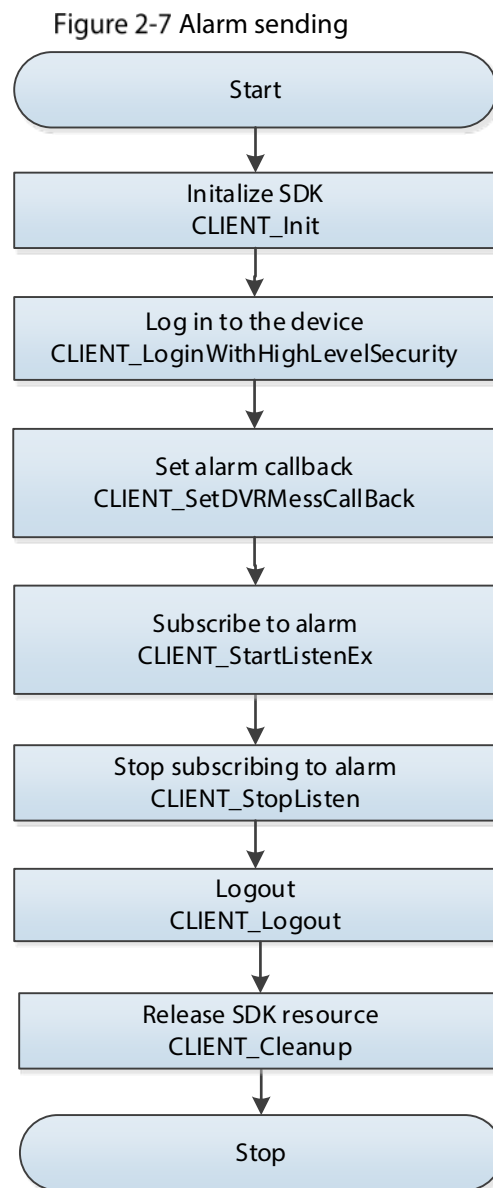
2.5.2 Interface Overview

Table 2-6 Interface of alarm sending

Interface	Implication
CLIENT_SetDVRMessCallBack	Set the alarm callback interface.
CLIENT_StartListenEx	Subscribe to the alarm expansion interface.

Interface	Implication
CLIENT_StopListen	Stop subscribing to alarm.

2.5.3 Process



Process Description

- Step 1 Call **CLIENT_Init** to initialize SDK.
- Step 2 Call **CLIENT_LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **CLIENT_SetDVRMessCallBack** to set the alarm callback. This interface should be called before subscribing alarm.
- Step 4 Call **CLIENT_StartListenEx** to subscribe to alarm from the device. After successful subscription, the device informs the users through the callback of **CLIENT_SetDVRMessCallBack**.
- Step 5 After using the functions of alarm sending, call **CLIENT_StopListen** to stop subscribing alarm.
- Step 6 Call **CLIENT_Logout** to logout the device.

Step 7 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

If the previously sent alarms are not sent, check whether the network was disconnected. If the network was disconnected, the alarm cannot be sent after automatical reconnection. You need to stop the subscription and then subscribe to the alarms again.

2.5.4 Example Code

```
// Alarm callback
int CALLBACK afMessCallBack(LONG ICommand, LLONG ILineID, char *pBuf, DWORD dwBufLen,
char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
    if(ICommand == DH_ALARM_FIREWARNING_INFO)// Thermal camera heat detection event sending
    {
        printf("Thermal camera heat detection event sending");
        ALARM_FIREWARNING_INFO_DETAIL* pstStorageInfo = (ALARM_FIREWARNING_INFO_DETAIL*)pBuf;
        // Then you can get the corresponding alarm information through pstStorageInfo
    }
}

// Set alarm callback
CLIENT_SetDVRMessCallBack(&afMessCallBack,0);
// Subscribe to alarm from the device
BOOL bRet = CLIENT_StartListenEx(ILoginHandle);
if(!bRet)
{
    printf("CLIENT_StartListenEx Failed, Last Error[%x]\n" ,
    CLIENT_GetLastError());
}
else
{
    printf("listen succeed.\n");
}

// Stop subscribing to alarm
BOOL bRet = CLIENT_StopListen(ILoginHandle);
if(!bRet)
{
    printf("CLIENT_StopListen Failed, Last Error[%x]\n" ,
```



```

    CLIENT_GetLastError());
}
else
{
    printf("stop listen succeed.\n");
}

```

2.6 AI Event Subscription

2.6.1 Introduction

The implementation method for AI event subscription is to log in to the device through the SDK and subscribe to the AI event function. If the device detects an AI event, it will immediately send it to SDK.

Before subscribing to behavior detection events, you need to configure behavior detection rules on the device web page.

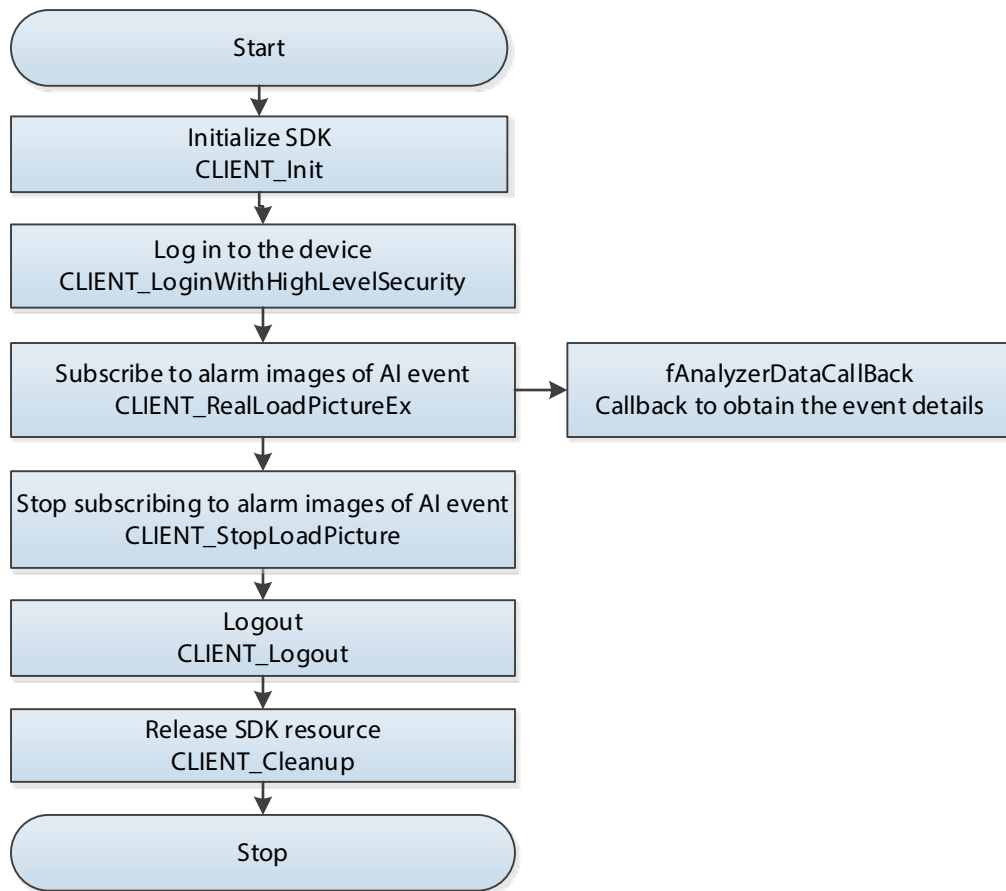
2.6.2 Interface Overview

Table 2-7 Interface of alarm sending

Interface	Implication
CLIENT_RealLoadPictureEx	Subscribe to AI event.
CLIENT_StopLoadPic	Stop subscribing to AI event.

2.6.3 Process

Figure 2-8 Event sending



Process Description

- Step 1 Call **CLIENT_Init** to initialize SDK.
- Step 2 Call **CLIENT_LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **CLIENT_RealLoadPictureEx** to subscribe to AI event.
- Step 4 After successful subscription, callback **fAnalyzerDataCallBack** to send AI event. Through **fAnalyzerDataCallBack**, the device can filter the AI events as needed according to the alarm type.
- Step 5 After using the functions of alarm sending, call **CLIENT_StopLoadPic** to stop subscribing AI event.
- Step 6 Call **CLIENT_Logout** to log out of the device.
- Step 7 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

- Subscription event type: If different AI events need to be sent at the same time, you can subscribe to all AI events (**EVENTIVS-ALL**); and you can also subscribe to a single AI event.
- Set the cache for receiving images: By default, the receiving cache of SDK is 2 M. When the image data retrieved from the callback is greater than 2 M, the user needs to call the **CLIENT_SetNetworkParam** interface to set the cache for receiving images; otherwise, packets larger than 2 M will be discarded.

- Set whether to receive images: Some devices are using 3G or 4G network, they do not want to receive some unnecessary image. When SDK connects to the device, set the parameter **bNeedPicFile** in the **CLIENT-RealLoadPictureEx** interface set to False to avoid receiving some unnecessary images. For example, you can receive face event information without images.

2.6.4 Example Code

```
// Callback of AI event sending
int CALLBACK AnalyzerDataCallBack(LLONG IAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE
*pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
    switch(dwAlarmType)
    {
        // Filter the AI events as needed
        .....
        case EVENT_IVS_FIREDETECTION: //Heat detection event
        .....
        default:
        break;
    }
}

// Subscribe to AI event sending
LLONG IAnalyzerHandle = CLIENT_RealLoadPictureEx(ILoginHandle, 0, (DWORD)EVENT_IVS_ALL, TRUE,
AnalyzerDataCallBack, NULL, NULL);
if(NULL == IAnalyzerHandle)
{
    printf("CLIENT_RealLoadPictureEx: failed! Error code %x.\n", CLIENT_GetLastError());
    return -1;
}

// Stop subscribing to AI event sending
CLIENT_StopLoadPic(IAnalyzerHandle);
```

2.7 Heat Map (Temperature)

2.7.1 Introduction

Heat map (temperature) function gets the temperature distribution data, and the gray map and temperature map, that is the grayscale and temperature of each pixel in the image.



Only the devices equipped with temperature measurement support this function.

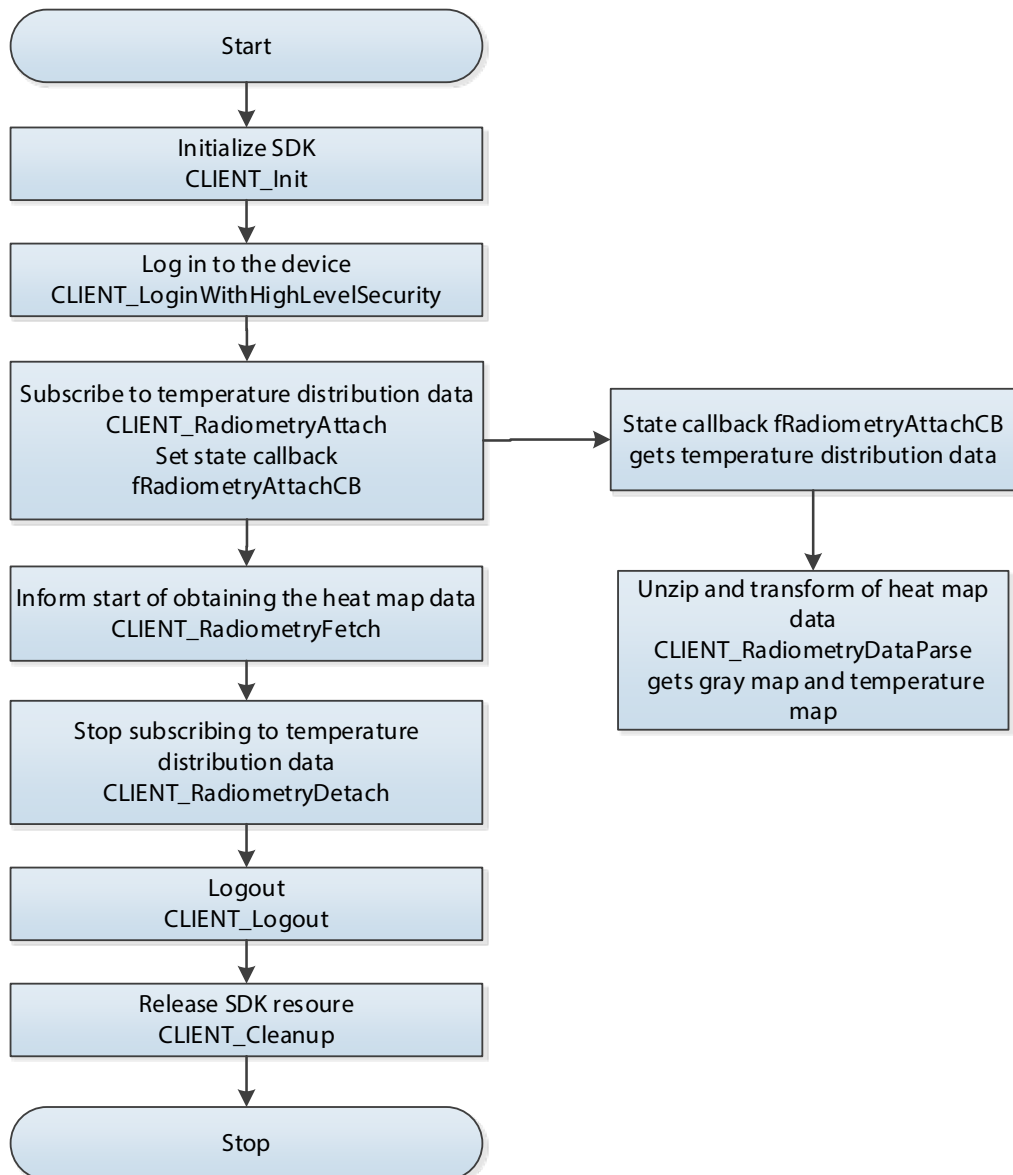
2.7.2 Interface Overview

Table 2-8 Interfaces of heat map (temperature)

Interface	Implication
CLIENT_RadiometryAttach	Subscribe to the temperature distribution data (heat map).
CLIENT_RadiometryDetach	Stop subscribing the temperature distribution data.
CLIENT_RadiometryFetch	Inform start of getting the heat map data.
CLIENT_RadiometryDataParse	Unzip the heat map data and convert it to the gray data and temperature data in the unit of pixel.

2.7.3 Process

Figure 2-9 Process of heat map (temperature)



Process Description

- Step 1 Call **CLIENT_Init** to initialize SDK.
- Step 2 Call **CLIENT_LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **CLIENT_RadiometryAttach** to subscribe to the temperature distribution data and register status callback. After the device has sent the temperature status, the callback **fRadiometryAttachCB** will inform you.
- Step 4 Call **CLIENT_RadiometryFetch** to inform the device to start getting the heat map data. Call this interface whenever you need the heat map data.
- Step 5 After receiving the temperature status information, call **CLIENT_RadiometryDataParse** to get the gray data and temperature data of each pixel.
- Step 6 Call **CLIENT_RadiometryDetach** to stop subscribing the temperature distribution data after finishing use.
- Step 7 After using the function module, call **CLIENT_Logout** to log out of the device.
- Step 8 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

- No uploaded heat map data: You will get the heat map data only after calling **CLIENT_RadiometryFetch**. If you call **CLIENT_RadiometryDetach** before receiving the heat map data, you will not receive the heat map data.
- Subscribing to the channel number: If the camera has one channel, the subscription channel number is 0; if the camera has two channels, the subscription channel is 0 or 1. For the TPC with two channels, the second channel is TPC.

2.7.4 Example Code

```
// Temperature state callback
void CALLBACK cbRadiometryAttachCB(LLONG lAttachHandle, NET_RADIOMETRY_DATA* pBuf, int nBufLen,
LDWORD dwUser)
{
    int nPixel = pBuf->stMetaData.nWidth*pBuf->stMetaData.nHeight;
    unsigned short *pGray = new unsigned short[nPixel];
    memset(pGray,0,nPixel);

    float *pTemp = new float[nPixel];
    memset(pTemp,0,nPixel);

    CLIENT_RadiometryDataParse(pBuf,pGray,pTemp);
    delete[] pGray;
    delete[] pTemp;
}
```

```

// Subscribe to the heat map
NET_IN_RADIOMETRY_ATTACH stIn = {sizeof(stIn), 1, cbRadiometryAttachCB};
NET_OUT_RADIOMETRY_ATTACH stOut = {sizeof(stOut)};

LLONG attachHandle = CLIENT_RadiometryAttach(loginId, &stIn, &stOut, 3000);
if (NULL == attachHandle)
{
// Subscription failed
}

// Inform the device to start collecting the data.
NET_IN_RADIOMETRY_FETCH stInFetch = {sizeof(stInFetch), 1};
NET_OUT_RADIOMETRY_FETCH stOutFetch = {sizeof(stOutFetch)};

CLIENT_RadiometryFetch(m_LoginID, &stInFetch, &stOutFetch, 3000);

// Stop subscription after finishing using this function
CLIENT_RadiometryDetach(attachHandle);

```

2.8 PTZ Control

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

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

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

2.8.1 Preset

2.8.1.1 Introduction

Get, add/change, and delete presets

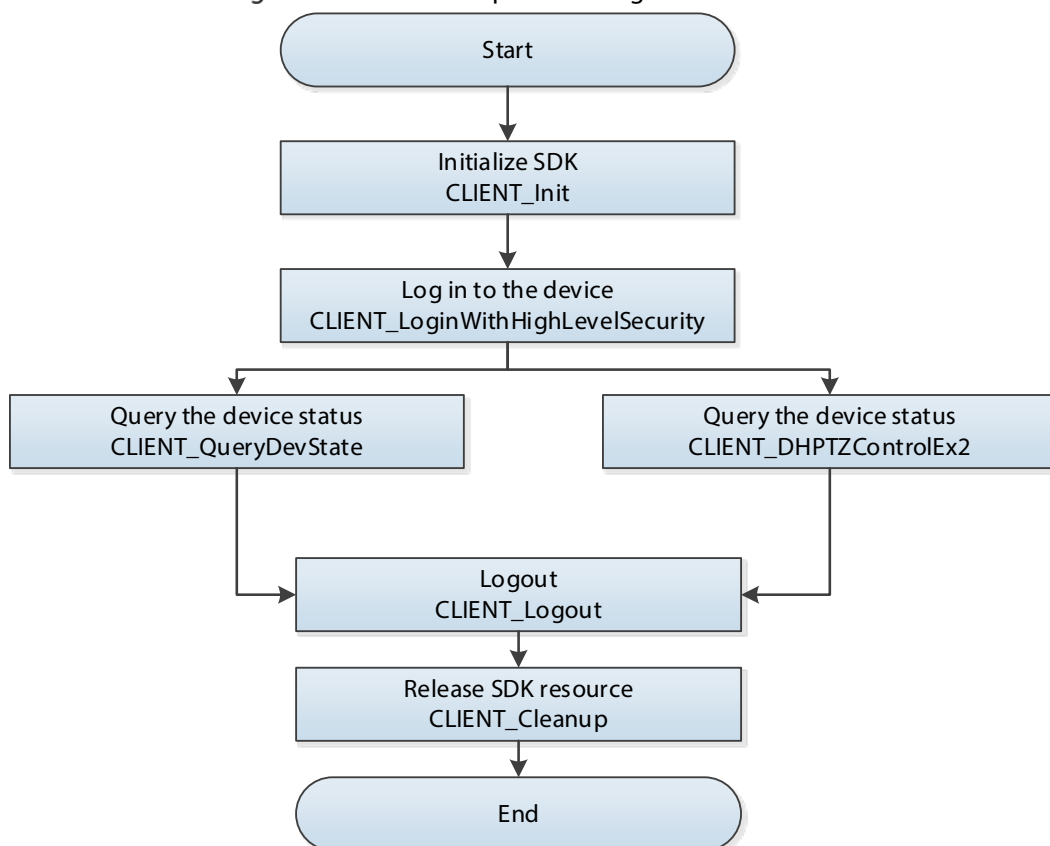
2.8.1.2 Interface Overview

Table 2-9 Interfaces of preset configuration

Interface	Implication
CLIENT_QueryDevState	Get the preset list. Parameter nType is DH_DEVSTATE_PTZ_PRESET_LIST. Take the nIndex and szName fields from the NET-PTZ-PRESET structure, and ignore other fields.
CLIENT_DHPTZControlEx2	Add, change preset. Parameter dwPTZCommand is DH_PTZ_POINT_SET_CONTROL.
CLIENT_DHPTZControlEx2	Delete preset Parameter dwPTZCommand is DH_PTZ_POINT_DEL_CONTROL.

2.8.1.3 Process

Figure 2-10 Process of preset configuration



Process Description

- Step 1 Call **CLIENT_Init** to initialize SDK.
- Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.
- Step 3 Call **CLIENT_QueryDevState** to query the device status.
- 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.

2.8.1.4 Example Code

```
//Get the preset list of the PTZ
void QueryPtzPresetList()
{
    int RetLen = 0;
    NET_PTZ_PRESET_LIST Info;
    Info.dwSize = sizeof(NET_PTZ_PRESET_LIST); //Required
    Info.dwMaxPresetNum = 20; //Required
    Info.pstuPtzPorsetList = new NET_PTZ_PRESET[Info.dwMaxPresetNum]; //Required

    BOOL QueryDevStateReturn = CLIENT_QueryDevState(IILoginHandle, DH_DEVSTATE_PTZ_PRESET_LIST,
(char*)&Info, sizeof(Info), &RetLen, 5000);
    if(QueryDevStateReturn)
    {
        printf("CLIENT_QueryDevState Success preset amount=%d;\n", Info.dwRetPresetNum);
    }
    else
    {
        printf("CLIENT_QueryDevState Failed ErrorCode: 0x%x\n", CLIENT_GetLastError());
    }
}

//Add, change preset
void PTZControlPointSet()
{
    bool DHPTZControlEx2Return = CLIENT_DHPTZControlEx2(IILoginHandle, 0,
DH_PTZ_POINT_SET_CONTROL, NULL, 8, NULL, FALSE, "preset 8");
    if(DHPTZControlEx2Return)
    {
        printf("CLIENT_DHPTZControlEx2 Success\n");
    }
    else
    {
        printf("CLIENT_DHPTZControlEx2 Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
    }
}
```



```
//Delete preset
void PTZControlPointDel()
{
    bool    DHPTZControlEx2Return    =    CLIENT_DHPTZControlEx2(ILLoginHandle,    0,
DH_PTZ_POINT_DEL_CONTROL, NULL, 8, NULL, FALSE, NULL);
    if(DHPTZControlEx2Return)
    {
        printf("CLIENT_DHPTZControlEx2 Success\n");
    }
    else
    {
        printf("CLIENT_DHPTZControlEx2 Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
    }
}
}
```

2.8.2 PTZ Absolute Position

2.8.2.1 Introduction

Get and set the PTZ absolute position

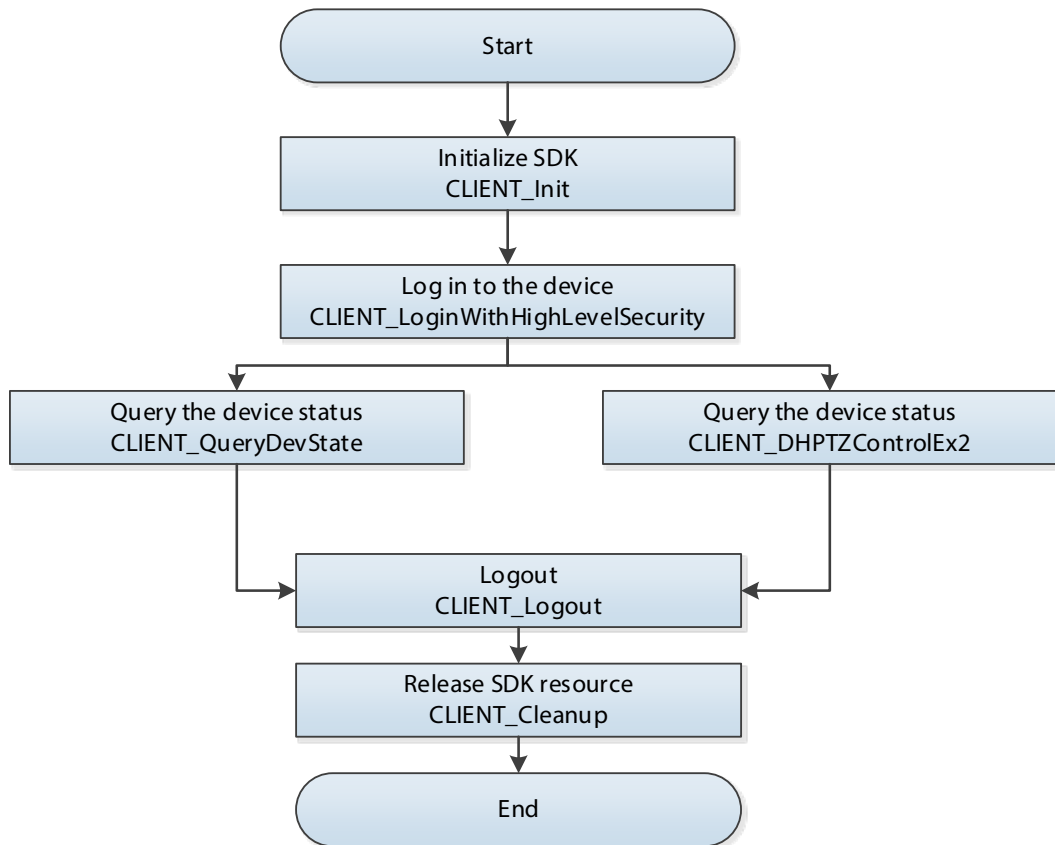
2.8.2.2 Interface Overview

Table 2-10 Interfaces of PTZ absolute position configuration

Interface	Implication
CLIENT_QueryDevState	Get the PTZ absolute position. Parameter nType is DH_DEVSTATE_PTZ_LOCATION. Take the fields nPTZPan[0,3600], nPTZTilt[-1800,1800], nPTZZoom[0,128] from the DH_PTZ_LOCATION_INFO structure, and ignore other fields.
CLIENT_DHPTZControlEx2	Set the PTZ absolute position. Parameter dwPTZCommand is DH_EXTPTZ_MOVE_ABSOLUTELY.

2.8.2.3 Process

Figure 2-11 Process of preset configuration



2.8.2.4 Example Code

```
//Get the PTZ position
void QueryPtzLocationPTZ()
{
    int RetLen = 0;
    DH_PTZ_LOCATION_INFO Info;
    Info.nChannelID = 0; //Required

    BOOL QueryDevStateReturn = CLIENT_QueryDevState(ILoginHandle, DH_DEVSTATE_PTZ_LOCATION,
(char*)&Info, sizeof(Info), &RetLen, 5000);
    if(QueryDevStateReturn)
    {
        printf("CLIENT_QueryDevState Success; P value=%d, T value=%d, zoom value=%d\n",Info.nPTZPan,Info.nPTZTilt,Info.nPTZZoom);
    }
    else
    {

```

```

        printf("CLIENT_QueryDevState Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
    }
}

//Set the PTZ position
void PTZControlMoveAbsolutely()
{
    PTZ_CONTROL_ABSOLUTELY info;
    info.stuPosition.nPositionX = 500; //P value, range [0,3600]
    info.stuPosition.nPositionY = 500; //T value, range [-1800,1800]
    info.stuPosition.nZoom = 2;          //Zoom value, range [0,128]
    info.stuSpeed.fPositionX = 0.9;
    info.stuSpeed.fPositionY = 0.9;
    info.stuSpeed.fZoom = 0.9;

    bool    DHPTZControlEx2Return    =    CLIENT_DHPTZControlEx2(ILLoginHandle,    0,
DH_EXTPTZ_MOVE_ABSOLUTELY, NULL, NULL, NULL, FALSE, &info);
    if(DHPTZControlEx2Return)
    {
        printf("CLIENT_DHPTZControlEx2 Success\n");
    }
    else
    {
        printf("CLIENT_DHPTZControlEx2 Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
    }
}

```

2.8.3 Continuous PTZ Control

2.8.3.1 Introduction

8 directions of PTZ, focus, zoom, and iris control.

2.8.3.2 Interface Overview

Table 2-11 Interface of PTZ control

Interface	Implication
CLIENT_DHPTZControlEx2	<p>Parameter 1: Login handle.</p> <p>Parameter 2: Channel number. Enter the parameter according to actual situation.</p> <p>Parameter 3:</p> <p>Starting dwPTZCommand is DH_EXTPTZ_START</p> <p>Stopping dwPTZCommand is DH_EXTPTZ_STOP</p> <p>Parameter 4: NULL</p> <p>Parameter 5: NULL</p> <p>Parameter 6: NULL</p> <p>Parameter 7: FALSE</p> <p>Parameter 8:</p> <p>Starting param 4 is NET_IN_PTZ_START_INFO</p> <p>Stopping param 4 is NET_IN_PTZ_STOP_INFO</p>

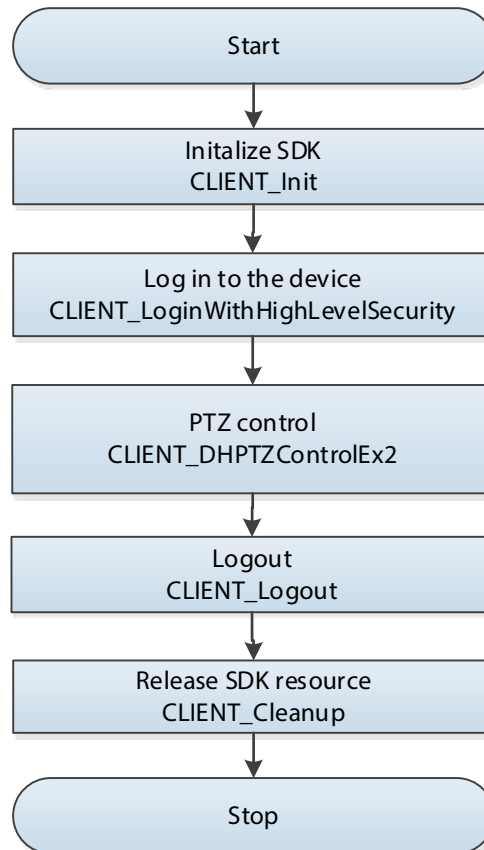
Table 2-12 Description of NET_IN_PTZ_START_INF and NET_IN_PTZ_STOP_INFO

szCode	Description	nArg1	nArg2	nArg3	nArg4	Remark
"Up"	Up	Speed (1-8)				
"Down"	Down	Speed (1-8)				
"Left"	Left	Speed (1-8)				
"Right"	Right	Speed (1-8)				
"ZoomWide"	Zoom (Wide)	Speed (1-8)				
"ZoomTele"	Zoom (Tele)	Speed (1-8)				
"FocusNear"	Focus (Near)	Speed (1-8)				
"FocusFar"	Focus (Near)	Speed (1-8)				
"IrisLarge"	Iris (Large)	Speed (1-8)				
"IrisSmall"	Iris (Small)	Speed (1-8)				
"GotoPreset"	Go to the preset	Preset value				
"SetPreset"	Set the preset	Preset value				
"ClearPreset"	Delete the preset	Preset value				
"Wiper"	Wiper	On 1, off 0				
"StartTour"	Start tour	Tour route				
"AutoTour"	Auto tour					
"StopTour"	Stop tour	Tour route				
"LeftUp"	upper left	Speed (1-8)	Speed (1-8)			
"RightUp"	upper right	Speed (1-8)	Speed (1-8)			
"LeftDown"	lower left	Speed (1-8)	Speed (1-8)			
"RightDown"	lower right	Speed (1-8)	Speed (1-8)			
"AddTour"	Add a preset to the tour	Tour route	Preset value			
"DelTour"	Delete a preset to the tour	Tour route	Preset value			
"ClearTour"	Delete the tour	Tour route				

"AutoPanOn"	Start pan					
"AutoPanOff"	Stop pan					
"SetLeftLimit"	Set the left limit					
"SetRightLimit"	Set the right limit					
"AutoScanOn"	Start scan					
"AutoScanOff"	Stop scan					

2.8.3.3 Process

Figure 2-12 Process of PTZ control (continuous)



2.8.3.4 Example Code

```

//Start controlling PTZ
void PTZControlStart()
{
    NET_IN_PTZ_START_INFO Info;
    Info.dwSize = sizeof(NET_IN_PTZ_START_INFO); //Required
    std::string szCode = "Left";
    std::strcpy(Info.szCode, szCode.c_str());
    Info.nArg1 = 2; //Required, step (1-8)
    int Channel = 0; //Required, enter the channel number according to the actual situation
}

```

```

    bool DHPTZControlEx2Return = CLIENT_DHPTZControlEx2(ILLoginHandle, Channel, DH_EXTPTZ_START,
NULL, NULL, NULL, FALSE, &Info);
    if(DHPTZControlEx2Return)
    {
        printf("CLIENT_DHPTZControlEx2 Success\n");
    }
    else
    {
        printf("CLIENT_DHPTZControlEx2 Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
    }
}

//Stop controlling PTZ
void PTZControlStart()
{
    NET_IN_PTZ_STOP_INFO Info;
    Info.dwSize = sizeof(NET_IN_PTZ_STOP_INFO); //Required
    std::string szCode = "Left";
    std::strcpy(Info.szCode, szCode.c_str());
    Info.nArg1 = 2; //Required, step (1-8)
    int Channel = 0; //Required, enter the channel number according to the actual situation

    bool DHPTZControlEx2Return = CLIENT_DHPTZControlEx2(ILLoginHandle, Channel, DH_EXTPTZ_STOP,
NULL, NULL, NULL, FALSE, &Info);
    if(DHPTZControlEx2Return)
    {
        printf("CLIENT_DHPTZControlEx2 Success\n");
    }
    else
    {
        printf("CLIENT_DHPTZControlEx2 Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
    }
}

```

2.9 Fire Alarm

2.9.1 Introduction

2.9.2 Interface Overview

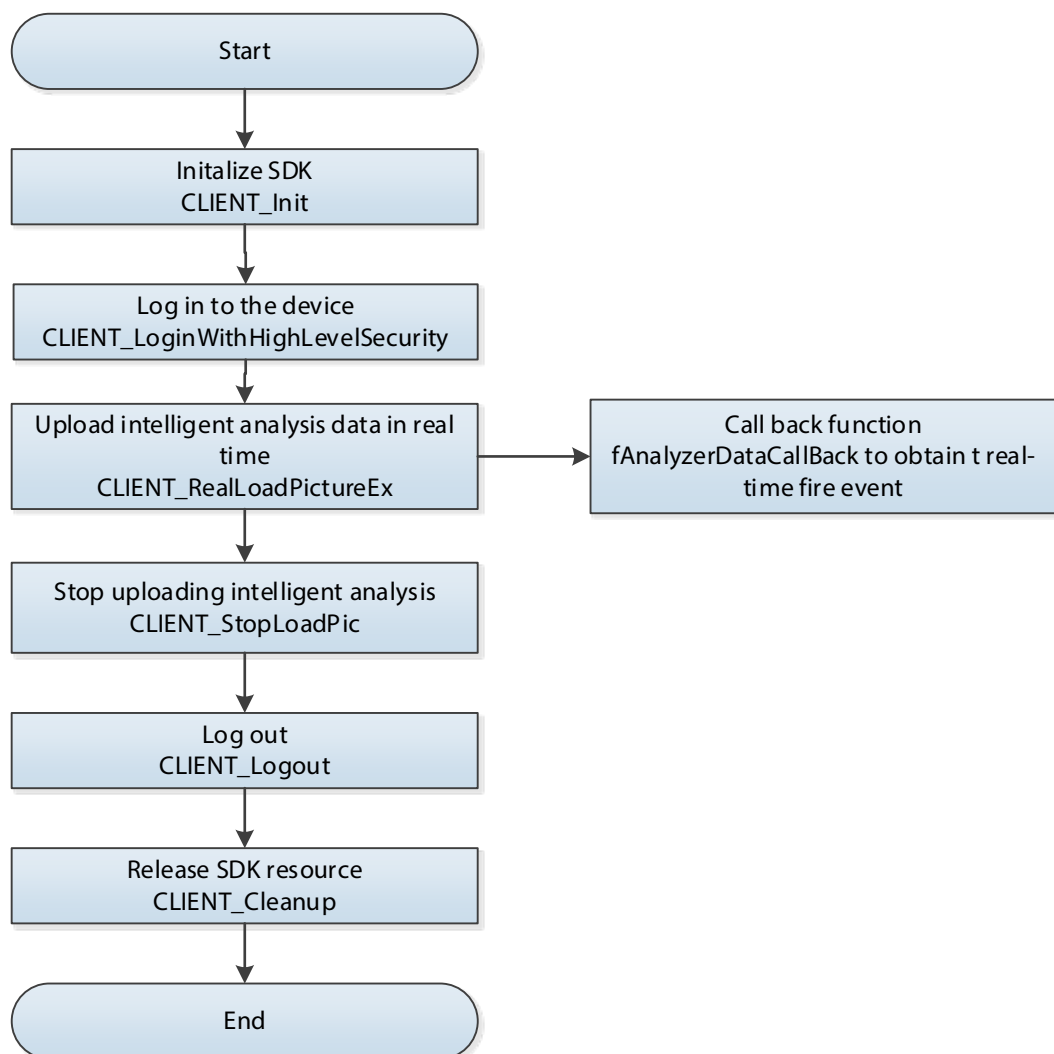
Table 2-13 Interfaces of firepoint

Interface	Implication
CLIENT_RealLoadPictureEx	Subscribe to firepoint event.
CLIENT_StopLoadPic	Stop subscribing to firepoint event.
CLIENT_SetDVRMessCallBack	Set alarm callback function
CLIENT_StartListenEx	Subscribe to firepoint alarm.
CLIENT_StopListen	Stop subscribing to firepoint alarm.

2.9.3 Process

2.9.3.1 Subscribing to Firepoint Event

Figure 2-13 Process of subscribing firepoint event,



Process Description

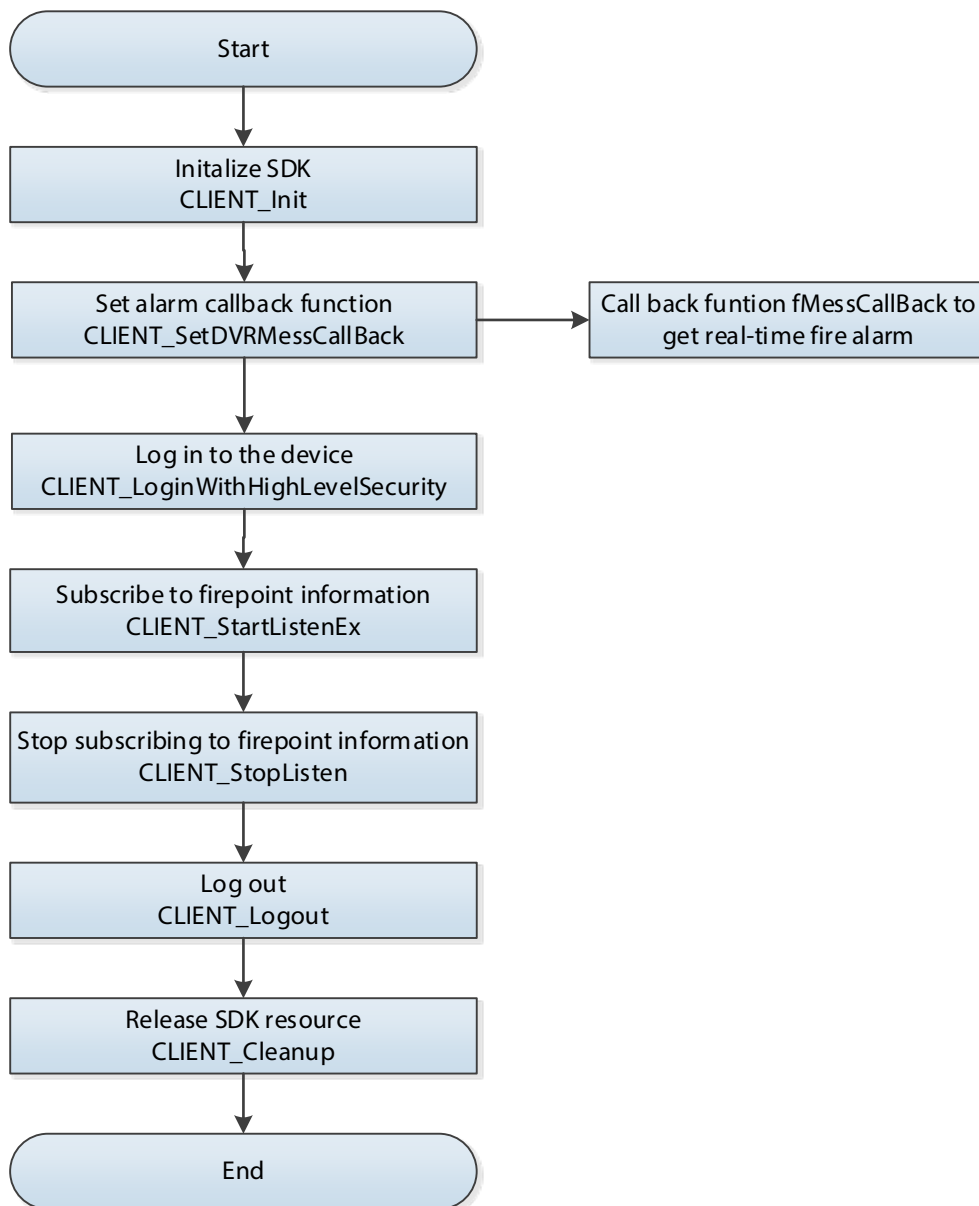
- Step 1 Call **CLIENT_Init** to initialize SDK.
- Step 2 Call **CLIENT_LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **CLIENT_RealLoadPictureEx** to subscribe to event, and when event triggers, **fAnalyzerDataCallBack** will remind you.
- Step 4 Call **CLIENT_StopLoadPic** to stop subscribing to firepoint event.
- Step 5 After using the function module, call **CLIENT_Logout** to log out of the device.
- Step 6 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

Subscribing firepoint information corresponds to the **FireWarning** event on device. When the event is triggered, it will do snapshot operation. When firepoint appears, it sends the word **start**; when firepoint disappears, it sends the word **stop**.

2.9.3.2 Subscribing to Firepoint Information

Figure 2-14 Process of subscribing firepoint information



Process Description

- Step 1 Call **CLIENT_Init** to initialize SDK.
- Step 2 Call **CLIENT_SetDVRMessCallBack** set alarm callback function, and when event triggers, fMessCallBack will remind you.
- Step 3 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.
- Step 4 Call **CLIENT_StartListenEx** to subscribe to firepoint information.
- Step 5 Call **CLIENT_StopListen** to stop subscribing to firepoint information.
- Step 6 After using the function module, call **CLIENT_Logout** to log out of the device.
- Step 7 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

- Subscribing to firepoint information corresponds to the FireWarningInfo event on device. After FireWarning is triggered, the event with firepoint details will be reported, including firepoint number, firepoint positioning calculation, PTZ value of the device, and GPS. The reports are made periodically for timely update of the firepoint.
- FireWarningInfo is state update event, and does not have common linkages, including snapshot and record and so on.

2.9.4 Example Code

```
//Subscribe to firepoint event
LDWORD dwUser = 0;
g_IRealLoadHandle = CLIENT_RealLoadPictureEx(gILoginHandle, 0, EVENT_IVS_FIREWARNING, TRUE,
AnalyzerDataCallBack, dwUser, NULL);
if (0 == g_IRealLoadHandle)
{
    //Fail to subscribe
    return;
}
//Stop subscribing to firepoint event
if (0 != g_IRealLoadHandle)
{
    if (FALSE == CLIENT_StopLoadPic(g_IRealLoadHandle))
    {
        // Fail to stop subscription
    }
    else
    {
        g_IRealLoadHandle = 0;
    }
}
//Subscribe to firepoint information
CLIENT_SetDVRMessCallBack(MessCallBack, NULL);
if( CLIENT_StartListenEx(gILoginHandle))
{
    g_bStartListenFlag = TRUE;
    printf("Listen Success!\nJust Wait Event....\n");
}
else
{

```

```

        //Fail to subscribe to firepoint information
    }
    //Stop subscribing to firepoint information
    if (g_bStartListenFlag)
    {
        if (!CLIENT_StopListen(g_LoginHandle))
        {
            //Fail to stop subscribing to firepoint information
        }
        else
        {
            g_bStartListenFlag = FALSE;
        }
    }
}

```

2.10 Getting Cold/Hot Spot

2.10.1 Introduction

Only the devices equipped with temperature measurement support this function.

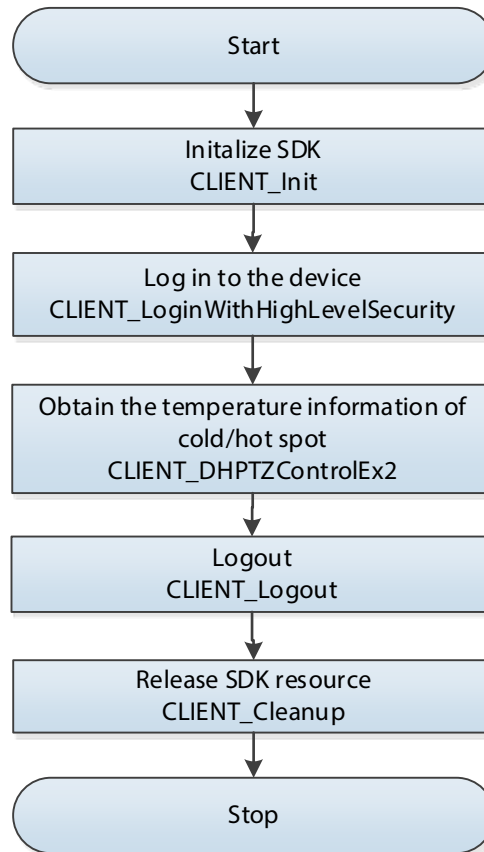
2.10.2 Interface Overview

Table 2-14 Interfaces of heat map (temperature)

Interface	Implication
CLIENT_RadiometryGetCurrentHotColdSpotInfo	<p>Get the information of the cold spot (the spot with the lowest temperature) and hot spot (the spot with the highest temperature).</p> <ol style="list-style-type: none"> 1. The devices equipped with temperature measurement. 2. Enable the cold/hot spot function on the camera web page.

2.10.3 Process

Figure 2-15 Process of getting cold/hot spot



2.10.4 Example Code

//Get the information of the cold spot (the spot with the lowest temperature) and hot spot (the spot with the highest temperature).

```
void GetCurrentHotColdSpotInfo()
```

```
{
```

```
    NET_IN_RADIOMETRY_CURRENTHOTCOLDSPOT_INFO inInfo;
```

```
    inInfo.dwSize = sizeof(NET_IN_RADIOMETRY_CURRENTHOTCOLDSPOT_INFO);    //Required
```

```
    inInfo.nChannel = 1;    //Required, enter the channel number
```

```
    NET_OUT_RADIOMETRY_CURRENTHOTCOLDSPOT_INFO outInfo;
```

```
    outInfo.dwSize = sizeof(NET_OUT_RADIOMETRY_CURRENTHOTCOLDSPOT_INFO); // Required
```

```
    BOOL Return = CLIENT_RadiometryGetCurrentHotColdSpotInfo(ILLoginHandle, &inInfo, &outInfo, 5000);
```

```
    if (Return)
```

```
    {
```

```
        printf("GetCurrentHotColdSpotInfo    Success;    The    highest    temperature    =%f\n",
```

```
outInfo.stuCurrentHotColdSpotInfo.fHotSpotValue);
```

```
    }
```

```
    else
```

```

{
    printf("GetCurrentHotColdSpotInfo Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
}
}

```

2.11 Temperature Information of Random Region

2.11.1 Introduction

Only the devices equipped with temperature measurement support this function.

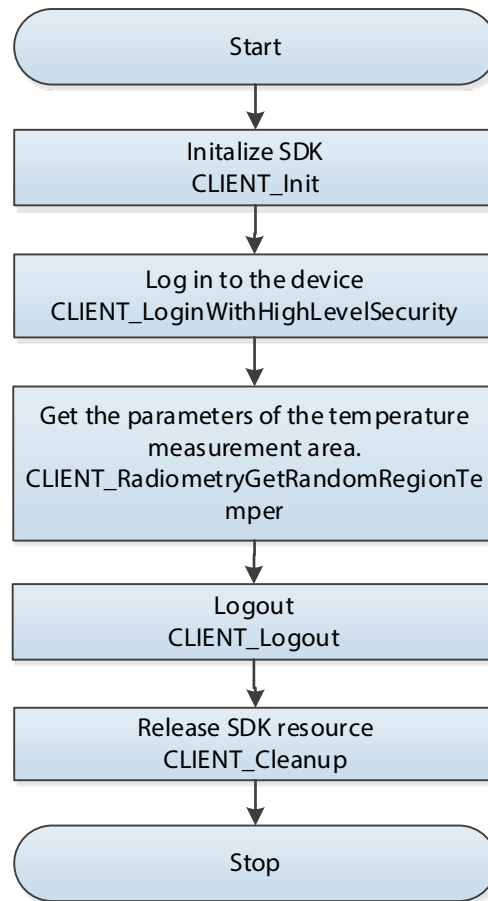
2.11.2 Interface Overview

Table 2-15 Interfaces of temperature information of random region

Interface	Implication
CLIENT_RadiometryGetRandomRegionTemper	<p>Get the parameters of the temperature measurement area.</p> <p>Only the devices equipped with temperature measurement support this interface.</p>

2.11.3 Process

Figure 2-16 Process of getting cold/hot spot



2.11.4 Example Code

```

// Get the temperature information of random region
void GetRandomRegionTemper()
{
    NET_IN_RADIOMETRY_RANDOM_REGION_TEMPER inInfo;
    inInfo.dwSize = sizeof(NET_IN_RADIOMETRY_RANDOM_REGION_TEMPER); //Required
    inInfo.nChannel = 0; //Required, enter the thermal channel number
    inInfo.nPointNum = 4; //Required, vertex number of the graph
    //The following graph is a rectangle
    inInfo.stuPolygon[0].nx = 10; //Required
    inInfo.stuPolygon[0].ny = 10; //Required
    inInfo.stuPolygon[1].nx = 8000; //Required
    inInfo.stuPolygon[1].ny = 10; //Required
    inInfo.stuPolygon[2].nx = 8000; //Required
    inInfo.stuPolygon[2].ny = 8000; //Required
    inInfo.stuPolygon[3].nx = 10; //Required
    inInfo.stuPolygon[3].ny = 8000; //Required
}
  
```

```

NET_OUT_RADIOMETRY_RANDOM_REGION_TEMPER outInfo;
outInfo.dwSize = sizeof(NET_OUT_RADIOMETRY_RANDOM_REGION_TEMPER); // Required

bool Return = CLIENT_RadiometryGetRandomRegionTemper(ILoginHandle, &inInfo, &outInfo, 15000);
if (Return)
{
    printf("GetRandomRegionTemper Success the highest temperature of random region= %d\n",
outInfo.stuRegionTempInfo.nTemperMax);
}
else
{
    printf("GetRandomRegionTemper Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
}
}

```

2.12 Temperature Measurement

2.12.1 Rule Setting

Get, add, and delete temperature measurement rules.

2.12.1.1 Introduction

Only the devices equipped with temperature measurement support this function.

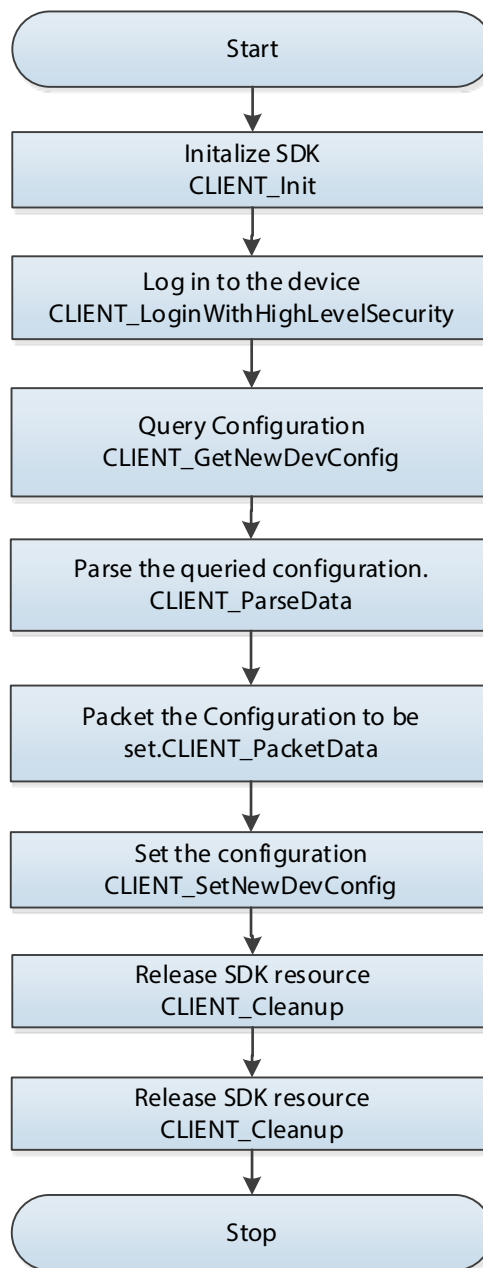
2.12.1.2 Interface Overview

Table 2-16 Interfaces of rule setting

Interface	Implication
CLIENT_GetNewDevConfig	Query the temperature measurement rules.
CLIENT_ParseData	Parse the queried temperature measurement rules.
CLIENT_PacketData	Packet the temperature measurement rules to be set.
CLIENT_SetNewDevConfig	Set the temperature measurement rules.

2.12.1.3 Process

Figure 2-17 Process of rule setting



2.12.1.4 Example Code

```
void ThermometryRule()
{
    //Query the temperature measurement rules.
    int Channel = -1;           //Required, enter the channel number according to the actual situation
    char szJsonBuf[40*512] = {0};
    int nBufferLen = 40*512;
    int nerror = 0;
```



```

    bool    GetNewDevConfigReturn    =    CLIENT_GetNewDevConfig(ILLoginHandle,
CFG_CMD_THERMOMETRY_RULE, Channel, szJsonBuf, nBufferLen, &nerror, 5000);
    if(GetNewDevConfigReturn)
    {
        printf("CLIENT_GetNewDevConfig Success\n");

        //Parse the queried temperature measurement rules.
        CFG_RADIOMETRY_RULE_INFO* RuleInfo = new CFG_RADIOMETRY_RULE_INFO;
        memset(RuleInfo, 0, sizeof(CFG_RADIOMETRY_RULE_INFO));
        bool ParseDataReturn = CLIENT_ParseData(CFG_CMD_THERMOMETRY_RULE, szJsonBuf, RuleInfo,
sizeof(CFG_RADIOMETRY_RULE_INFO), NULL);
        if(ParseDataReturn)
        {
            printf("CLIENT_ParseData Success; rule number=%d, Temperature measurement enabled=%d,
preset number=%d\n",RuleInfo->nCount, RuleInfo->stRule[0].bEnable, RuleInfo->stRule[0].nPresetId);

            // Packet the temperature measurement rules to be set.
            RuleInfo->stRule[0].bEnable = 0;
            char szJsonBuf2[40*512] = {0};
            int nBufferLen2 = 40*512;
            bool PacketDataReturn = CLIENT_PacketData(CFG_CMD_THERMOMETRY_RULE, RuleInfo,
sizeof(CFG_RADIOMETRY_RULE_INFO), szJsonBuf2, nBufferLen2);
            if(PacketDataReturn)
            {
                printf("CLIENT_PacketData Success\n");

                int Restart = 0; //Do you want to restart the device? 1 means Yes, and 0 means No.
                bool    SetNewDevConfigReturn    =    CLIENT_SetNewDevConfig(ILLoginHandle,
CFG_CMD_THERMOMETRY_RULE, Channel, szJsonBuf2, nBufferLen2, &nerror, &Restart, 5000);
                if(SetNewDevConfigReturn)
                {
                    printf("CLIENT_SetNewDevConfig Success \n");
                }
                else
                {
                    printf("CLIENT_SetNewDevConfig    Fail    ErrorCode:    0x%x;    Error=%d\n",
CLIENT_GetLastError(), nerror);
                }
            }
        }
        else
        {

```

```

        printf("CLIENT_PacketData Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
    }
}
else
{
    printf("CLIENT_ParseData Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
}
}
else
{
    printf("CLIENT_GetNewDevConfig Fail ErrorCode: 0x%x; Error=%d\n", CLIENT_GetLastError(), nerror);
}
}

```

2.12.2 Global Setting

Get, add, and delete global rules.

2.12.2.1 Introduction

Only the devices equipped with temperature measurement support this function.

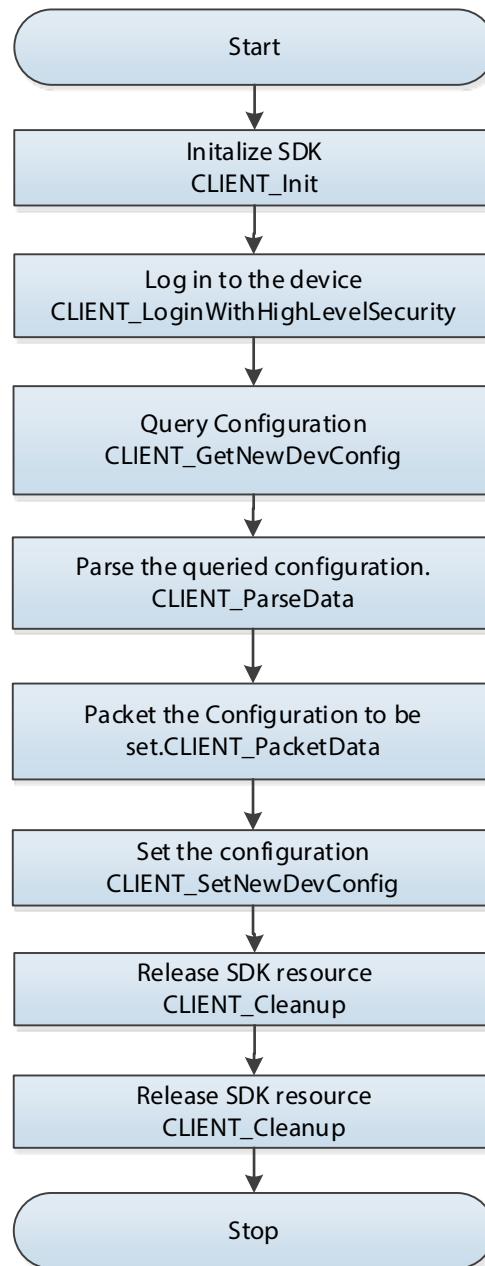
2.12.2.2 Interface Overview

Table 2-17 Interfaces of global setting

Interface	Implication
CLIENT_GetNewDevConfig	Query the global rules.
CLIENT_ParseData	Parse the queried global rules.
CLIENT_PacketData	Packet the global rules to be set.
CLIENT_SetNewDevConfig	Set the global rules.

2.12.2.3 Process

Figure 2-18 Process of global setting



2.12.2.4 Example Code

```
void Thermometry()
{
    //Query the global rules
    int Channel = -1;           // Required, enter the channel number according to the actual situation
    char szJsonBuf[40*512] = {0};
    int nBufferLen = 40*512;
    int nerror = 0;
```

```

    bool GetNewDevConfigReturn = CLIENT_GetNewDevConfig(ILLoginHandle, CFG_CMD_THERMOMETRY,
Channel, szJsonBuf, nBufferLen, &nerror, 5000);
    if(GetNewDevConfigReturn)
    {
        printf("CLIENT_GetNewDevConfig Success\n");

        //Parse the queried global rules
        CFG_THERMOMETRY_INFO* ThermometryInfo = new CFG_THERMOMETRY_INFO;
        memset(ThermometryInfo, 0, sizeof(CFG_THERMOMETRY_INFO));
        bool ParseDataReturn = CLIENT_ParseData(CFG_CMD_THERMOMETRY, szJsonBuf, ThermometryInfo,
sizeof(CFG_THERMOMETRY_INFO), NULL);
        if(ParseDataReturn)
        {
            printf("CLIENT_ParseData      Success;          relative      humidity      =%d\n",ThermometryInfo->nRelativeHumidity);

            //Packet the global rules to be set
            ThermometryInfo->nRelativeHumidity = 88;
            char szJsonBuf2[40*512] = {0};
            int nBufferLen2 = 40*512;
            bool PacketDataReturn = CLIENT_PacketData(CFG_CMD_THERMOMETRY, ThermometryInfo,
sizeof(CFG_THERMOMETRY_INFO), szJsonBuf2, nBufferLen2);
            if(PacketDataReturn)
            {
                printf("CLIENT_PacketData Success\n");

                // Set the global rules.
                int Restart = 0; //Do you want to restart the device? 1 means Yes, and 0 means No.
                bool SetNewDevConfigReturn = CLIENT_SetNewDevConfig(ILLoginHandle,
CFG_CMD_THERMOMETRY, Channel, szJsonBuf2, nBufferLen2, &nerror, &Restart, 5000);
                if(SetNewDevConfigReturn)
                {
                    printf("CLIENT_SetNewDevConfig Success \n");
                }
                else
                {
                    printf("CLIENT_SetNewDevConfig      Fail      ErrorCode:      0x%x;      Error=%d\n",
CLIENT_GetLastError(), nerror);
                }
            }
        }
    }
    else

```

```

        {
            printf("CLIENT_PacketData Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
        }
    }
    else
    {
        printf("CLIENT_ParseData Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
    }
}
else
{
    printf("CLIENT_GetNewDevConfig Fail ErrorCode: 0x%x; Error=%d\n", CLIENT_GetLastError(), nerror);
}
}
}

```

2.12.3 Abnormal Temperature Alarm

After setting the temperature measurement rules, an abnormal temperature alarm will be sent according to the configured rules.

2.12.3.1 Introduction

Only the devices equipped with temperature measurement support this function.

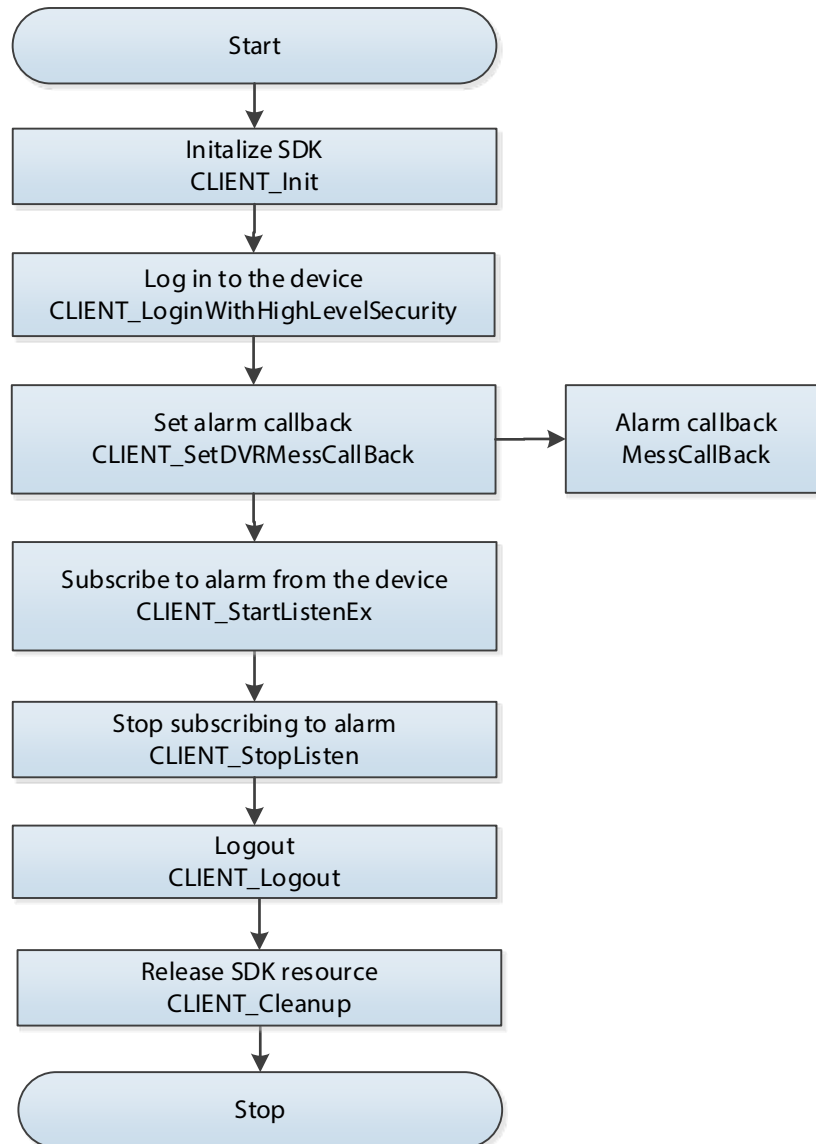
2.12.3.2 Interface Overview

Table 2-18 Interfaces of abnormal temperature alarm

Interface	Implication
CLIENT_SetDVRMessCallBack	Set the alarm callback. Command of parameter fMessCallBack is DH_ALARM_HEATING_TEMPER. pBuf of parameter fMessCallBack is ALARM_HEATING_TEMPER_INFO.
CLIENT_StartListenEx	Subscribe to alarm from the device.
CLIENT_StopListen	Stp subscribing to alarm.

2.12.3.3 Process

Figure 2-19 Process of abnormal temperature alarm



Process Description

- Step 1 Call **CLIENT_Init** to initialize SDK.
- Step 2 Call **CLIENT_LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **CLIENT_SetDVRMessCallBack** to set the alarm callback. This interface should be called before subscribing alarm.
- Step 4 Call **CLIENT_StartListenEx** to subscribe to alarm. After successful subscription, the device informs the users through the callback of **CLIENT_SetDVRMessCallBack**.
- Step 5 After using the functions of alarm sending, call **CLIENT_StopListen** to stop subscribing alarm.
- Step 6 Call **CLIENT_Logout** to logout the device.
- Step 7 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

If the previously sent alarms are not sent, check whether the network was disconnected. If the network was disconnected, the alarm cannot be sent after automatical reconnection. You need to stop the subscription and then subscribe to the alarms again.

2.12.3.4 Example Code

```
//Message callback processing function
BOOL CALLBACK MessCallBack(LONG ICommand, LLONG ILoginID, char *pBuf, DWORD dwBufLen, char
*pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
    //Filter abnormal temperature alarm of temperature measurement point (DH_ALARM_HEATING_TEMPER
    == ICommand)
    {
        ALARM_HEATING_TEMPER_INFO* pInfo = (ALARM_HEATING_TEMPER_INFO*)pBuf;

        printf("abnormal temperature alarm of temperature measurement point ALARM; Name of point with
        abnormal temperature =%s, alarm number =%d, alarm result (value) =%d、 alarm conditions =%d \n",
            pInfo->szName, pInfo->nAlarmId, pInfo->nResult, pInfo->nAlarmContion);
    }
    return TRUE;
}

//Start
void StartListenEx()
{
    //Set the alarm callback
    CLIENT_SetDVRMessCallBack(MessCallBack, NULL);

    //Subscribe to alarm from the device
    BOOL StartListenExReturn = CLIENT_StartListenEx(ILoginHandle);
    if (StartListenExReturn)
    {
        printf("CLIENT_StartListenEx Success\n");
    }
    else
    {
        printf("CLIENT_StartListenEx Fail ErrorCode: 0x%x \n", CLIENT_GetLastError());
    }
}
```

```

printf("Print any key to stop listen\n");
getchar();

//Stop subscribing to alarm
BOOL StopListenReturn = CLIENT_StopListen(ILoginHandle);
if(StopListenReturn)
{
    printf("CLIENT_StopListen Success\n");
}
else
{
    printf("CLIENT_StopListen Fail ErrorCode: 0x%x \n", CLIENT_GetLastError());
}
}

```

2.12.4 Getting Temperature Information

2.12.4.1 Introduction

Only the devices equipped with temperature measurement support this function.

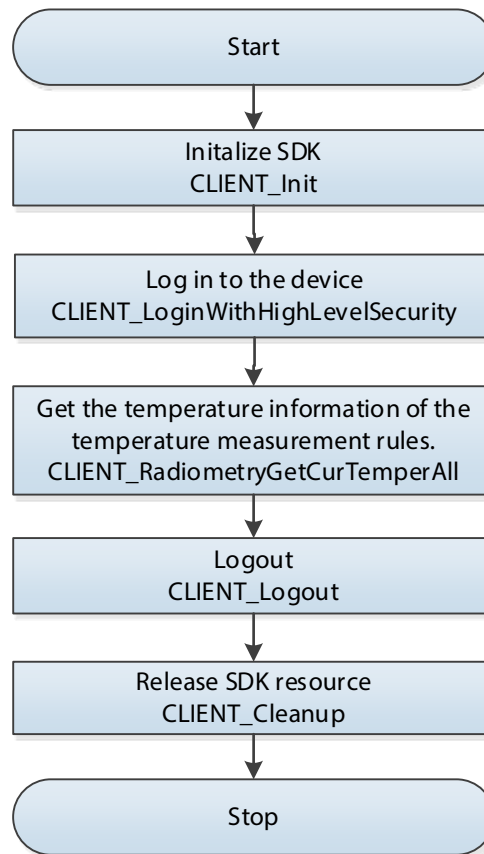
2.12.4.2 Interface Overview

Table 2-19 Interfaces of getting temperature information

Interface	Implication
CLIENT_RadiometryGetCurTemperAll	Get the temperature information of the temperature measurement rules.

2.12.4.3 Process

Figure 2-20 Process of getting temperature information



2.12.4.4 Example Code

```
// Get the temperature information of the temperature measurement rules.
void GetCurTemperAll()
{
    NET_IN_RADIOMETRY_GET_CUR_TEMPER_ALL_INFO inInfo;
    inInfo.dwSize = sizeof(NET_IN_RADIOMETRY_GET_CUR_TEMPER_ALL_INFO);    //Required
    inInfo.nChannel = 1;                                                    //Required, enter the thermal channel
number

    NET_OUT_RADIOMETRY_GET_CUR_TEMPER_ALL_INFO outInfo;
    outInfo.dwSize = sizeof(NET_OUT_RADIOMETRY_GET_CUR_TEMPER_ALL_INFO);    //Required
    outInfo.nMaxTempInfoNum = 20;                                           //Required
    outInfo.pstTempInfo = new NET_RADIOMETRY_TEMP_INFO[outInfo.nMaxTempInfoNum]; //Required
    memset(outInfo.pstTempInfo, 0x00, sizeof(NET_RADIOMETRY_TEMP_INFO)*outInfo.nMaxTempInfoNum);

    BOOL Return = CLIENT_RadiometryGetCurTemperAll(ILLoginHandle, &inInfo, &outInfo, 5000);
    if (Return)
```

```

{
    printf("GetCurTemperAll Success; the actual number of returned rule temperature information
= %d\n", outInfo.nRetTempInfoNum);
}
else
{
    printf("GetCurTemperAll Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
}
}

```

2.12.5 Initiatively Sending Temperature Information

2.12.5.1 Introduction

Only the devices equipped with temperature measurement support this function.

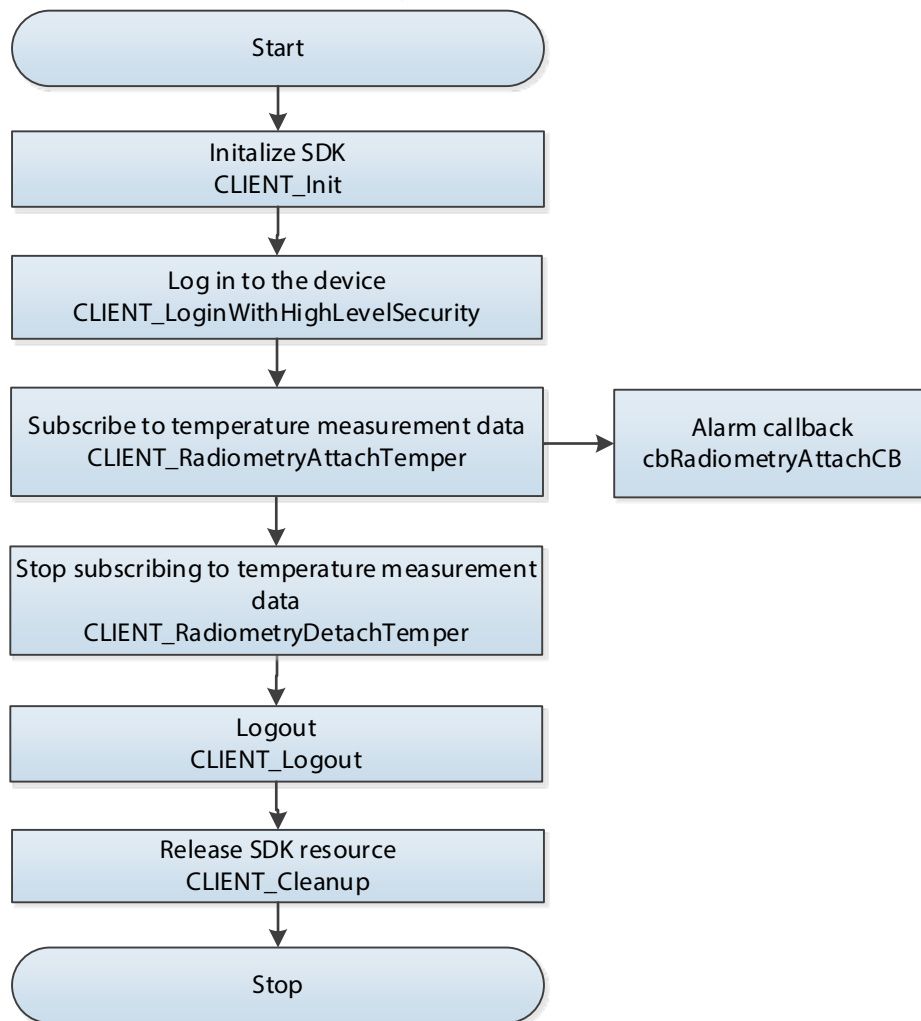
2.12.5.2 Interface Overview

Table 2-20 Interfaces of initiatively sending temperature information

Interface	Implication
CLIENT_RadiometryAttachTemper	Subscribe to temperature measurement data.
CLIENT_RadiometryDetachTemper	Stop subscribing to temperature measurement data.

2.12.5.3 Process

Figure 2-21 Process of initiatively sending temperature information



2.12.5.4 Example Code

```
LLONG m_IAttachhandle = 0;

void CALL_METHOD cbRadiometryAttachCB(LLONG IAttachTemperHandle,
NET_RADIOMETRY_TEMPER_DATA* pBuf, int nBufLen, LDWORD dwUser)
{
    for(int i = 0; i < pBuf->nRadiometryTemperNum; i++)
    {
        printf("%d_%d_%d_%d_%d_%d_fTemperAve:%f\n",
            pBuf->stuRadiometryTemperInfo[i].stuTime.dwYear,
            pBuf->stuRadiometryTemperInfo[i].stuTime.dwMonth,
            pBuf->stuRadiometryTemperInfo[i].stuTime.dwDay,
            pBuf->stuRadiometryTemperInfo[i].stuTime.dwHour,
            pBuf->stuRadiometryTemperInfo[i].stuTime.dwMinute,
```

```

        pBuf->stuRadiometryTemperInfo[i].stuTime.dwSecond,
        pBuf->stuRadiometryTemperInfo[i].stuQueryTemperInfo.fTemperAve);
    }
}

void AttachTemper()
{
    NET_IN_RADIOMETRY_ATTACH_TEMPER inInfo;
    inInfo.dwSize = sizeof(NET_IN_RADIOMETRY_ATTACH_TEMPER);

    std::cin >> inInfo.nChannel;
    inInfo.nChannel = 1;

    inInfo.cbNotify = cbRadiometryAttachCB;

    NET_OUT_RADIOMETRY_ATTACH_TEMPER outInfo;
    outInfo.dwSize = sizeof(NET_OUT_RADIOMETRY_ATTACH_TEMPER);

    bool m_IAttachhandle = CLIENT_RadiometryAttachTemper(ILoginHandle, &inInfo, &outInfo, 5000);
    if (m_IAttachhandle > 0)
    {
        printf("CLIENT_RadiometryAttachTemper Success\n");
    }
    else
    {
        printf("CLIENT_RadiometryAttachTemper Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());
    }
}

void DetachTemper()
{
    BOOL Return = CLIENT_RadiometryDetachTemper(m_IAttachhandle);
    if (Return)
    {
        printf("CLIENT_RadiometryDetachTemper Success\n");
    }
    else
    {

```

```
        printf("CLIENT_RadiometryDetachTemper Fail ErrorCode: 0x%x\n", CLIENT_GetLastError());  
    }  
}
```

3 Interface Definition

3.1 SDK Initialization

3.1.1 SDK CLIENT_Init

Table 3-1 Initialize SDK

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	<ul style="list-style-type: none">• Success: TRUE.• Failure: FALSE.	
Note	<ul style="list-style-type: none">• The precondition for calling other function modules.• If the callback is set as NULL, the callback will not be sent to the user after the device is disconnected.	

3.1.2 CLIENT_Cleanup

Table 3-2 Clean up SDK

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

3.1.3 CLIENT_SetAutoReconnect

Table 3-3 Set reconnection callback

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

Table 3-4 Set network parameter

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 Login

3.2.1 CLIENT_LoginWithHighLevelSecurity

Table 3-5 Log in with high level security

Item	Description		
Name	Log in to the device with high level security.		
Function	LLONG CLIENT_LoginWithHighLevelSecurity (NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY* pstInParam, NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY* pstOutParam);		
Parameter	[in] pstInParam	[in] dwSize	Structure size.
		[in] szIP	Device IP.
		[in] nPort	Device port.
		[in] szUserName	User name.
		[in] szPassword	Password.
		[in] emSpecCap	Login type.
		[in] pCapParam	Login type parameter.
	[out] pstOutParam	[in]dwSize	Structure size.
		[out] stuDeviceInfo	Device information.
		[out] nError	Error code.
Return value	<ul style="list-style-type: none">● Success: Not 0.● Failure: 0.		

Item	Description
Note	Login the device with high level security. CLIENT_LoginEx2 can still be used, but there are security risks, so it is highly recommended to use the latest interface CLIENT_LoginWithHighLevelSecurity to log in to the device.

Table 3-6 Error code and meaning

Error code	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 avnet sdk or avnet sdk.
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.

3.2.2 CLIENT_Logout

Table 3-7 Log out

Item	Description
Name	User logout the device.
Function	BOOL CLIENT_Logout(LLONG ILoginID);
Parameter	[in] ILoginID Return value of CLIENT_LoginWithHighLevelSecurity.
Return value	<ul style="list-style-type: none"> Success: TRUE. Failure: FALSE.
Note	None.

3.3 Live Viewing

3.3.1 CLIENT_RealPlayEx

Table 3-8 Start the real-time monitoring

Item	Description
Name	Open the live viewing.

Item	Description	
Function	<pre>LLONG CLIENT_RealPlayEx(LLONG ILoginID, int nChannelID, HWND hWnd, DH_RealPlayType rType);</pre>	
Parameter	[in]ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in]nChannelID	Video channel number is a round number starting from 0.
	[in]hWnd	Window handle valid only under Windows system.
	[in]rType	Preview type.
Return value	<ul style="list-style-type: none"> • Success: not 0 • Failure: 0 	
Note	<p>Windows system:</p> <ul style="list-style-type: none"> • When hWnd is valid, the corresponding window displays picture. • When hWnd is NULL, get the video data through setting a callback and send to user for handle. 	

Table 3-9 Live view type and meaning

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

3.3.2 CLIENT_StopRealPlayEx

Table 3-10 Stop the real-time monitoring

Item	Description
Name	Stop live viewing.
Function	<pre>BOOL CLIENT_StopRealPlayEx(LLONG IRealHandle);</pre>

Item	Description	
Parameter	[in] IRealHandle	Return value of CLIENT_RealPlayEx.
Return value	<ul style="list-style-type: none"> • Success: TRUE. • Failure: FALSE. 	
Note	None.	

3.3.3 CLIENT_SaveRealData

Table 3-11 Save the real-time monitoring data as file

Item	Description	
Name	Save the live viewing data as file.	
Function	BOOL CLIENT_SaveRealData(LLONG IRealHandle, const char *pchFileName);	
Parameter	[in] IRealHandle	Return value of CLIENT_RealPlayEx.
	[in] pchFileName	Save path.
Return value	<ul style="list-style-type: none"> • Success: TRUE. • Failure: FALSE. 	
Note	None.	

3.3.4 CLIENT_StopSaveRealData

Table 3-12 Stop saving the real-time monitoring data as file

Item	Description	
Name	Stop saving the live viewing data as file.	
Function	BOOL CLIENT_StopSaveRealData(LLONG IRealHandle);	
Parameter	[in] IRealHandle	Return value of CLIENT_RealPlayEx.
Return value	<ul style="list-style-type: none"> • Success: TRUE. • Failure: FALSE. 	
Note	None.	

3.3.5 CLIENT_SetRealDataCallbackEx2

Table 3-13 Set the callback of real-time monitoring data

Item	Description	
Name	Set the callback of live viewing data.	
Function	BOOL CLIENT_SetRealDataCallbackEx(LLONG IRealHandle, fRealDataCallbackEx2 cbRealData, LDWORD dwUser, DWORD dwFlag);	

Item	Description	
Parameter	[in] IRealHandle	Return value of CLIENT_RealPlayEx.
	[in] cbRealData	Callback of live viewing data flow.
	[in] dwUser	Parameter of callback for live viewing data flow.
	[in] dwFlag	Type of live viewing data in callback. The type is EM_REALDATA_FLAG and supports OR operation.
Return value	<ul style="list-style-type: none"> Success: TRUE. Failure: FALSE. 	
Note	None.	

Table 3-14 dwFlag type and parameter

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.

3.4 Video Snapshot

3.4.1 CLIENT_SnapPictureToFile

Table 3-15 Device snapshot

Item	Description	
Name	Device snapshot.	
Function	<pre> BOOL CLIENT_SnapPictureToFile(LONG ILoginID, const NET_IN_SNAP_PIC_TO_FILE_PARAM* pInParam, NET_OUT_SNAP_PIC_TO_FILE_PARAM* pOutParam, int nWaitTime); </pre>	
Parameter	[in] ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] pInParam	Input parameter.
	[in] pOutParam	Output parameter.
	[in] nWaitTime	Timeout. The unit is millisecond.
Return value	<ul style="list-style-type: none"> Success: TRUE. Failure: FALSE. 	
Note	<ul style="list-style-type: none"> Synchronous interface. The device captures snapshot and sends to the user through internet. The device is required to support this function. 	

3.4.2 CLIENT_CapturePictureEx

Table 3-16 Snapshot

Item	Description
Name	Snapshot.

Item	Description	
Parameter	[in] ILoginID	Return value of CLIENT_LoginEx2.
Return value	<ul style="list-style-type: none"> Success: TRUE. Failure: FALSE. 	
Note	Subscribe to device messages, and the received messages are called back from the settings of CLIENT_SetDVRMessCallBack.	

For details of alarm event types, please see Table 3-19.

Table 3-19 Alarm type

dwAlarmType Macro Definition	Value	Description	Call back the corresponding structure of pAlarmInfo
DH_ALARM_ALARM_EX	0x2101	External alarm	DWORD
DH_MOTION_ALARM_EX	0x2102	Motion detection	DWORD
DH_ALARM_ALARM_EX2	0x2175	Local alarm	ALARM_ALARM_INFO_EX2
DH_ALARM_TEMPERATURE	0x2135	High temperature alarm	ALARM_TEMPERATURE_INFO
DH_ALARM_STORAGE_FAILURE_EX	0x2163	Storage error alarm	ALARM_STORAGE_FAILURE_EX
DH_ALARM_STORAGE_NOT_EXIST	0x3167	Storage group does not exist	ALARM_STORAGE_NOT_EXIST_INFO
DH_ALARM_ALARM_EX2	0x2175	Local alarm	ALARM_ALARM_INFO_EX2
DH_ALARM_HEATING_TEMPERATURE	0x31aa	abnormal temperature alarm of temperature measurement point	ALARM_HEATING_TEMPERATURE_INFO
DH_ALARM_FIREWARNING	0x31b5	Firepoint	ALARM_FIREWARNING_INFO
DH_ALARM_SMOKE_DETECTION	0x31d5	Smoke alarm	ALARM_SMOKE_DETECTION_INFO
DH_ALARM_BETWEENRULE_TEMPERATURE_DIFF	0x31d6	Temperature difference alarm	ALARM_BETWEENRULE_DIFF_TEMPERATURE_INFO
DH_ALARM_HOTSPOT_WARNING	0x31d8	Hot spot abnormal alarm	ALARM_HOTSPOT_WARNING_INFO
DH_ALARM_COLDSPOT_WARNING	0x31d9	Cold point abnormal alarm	ALARM_COLDSPOT_WARNING_INFO
DH_ALARM_FIREWARNING_INFO	0x31da	Fire information setting	ALARM_FIREWARNING_INFO_DETAIL
DH_ALARM_FACE_OVERHEATING	0x31db	High temperature (face)	ALARM_FACE_OVERHEATING_INFO

3.5.3 CLIENT_StopListen

Table 3-20 Stop subscribing to alarm

Item	Description
Name	Stop subscribing to alarm.

Item	Description	
Function	BOOL CLIENT_StopListen(LLONG ILoginID);	
Parameter	[in] ILoginID	Return value of CLIENT_LoginEx2.
Return value	<ul style="list-style-type: none"> • Success: TRUE. • Failure: FALSE. 	
Note	None.	

3.6 AI Event Subscription

3.6.1 CLIENT_RealLoadPictureEx

Table 3-21 Start subscribing to AI events

Item	Description	
Name	Start subscribing to AI events.	
Function	LLONG CALL_METHOD CLIENT_RealLoadPictureEx(LLONG ILoginID, int nChannelID, DWORD dwAlarmType, BOOL bNeedPicFile, fAnalyzer DataCallBack cbAnalyzerData, LDWORD dwUser, void* Reserved);	
Parameter	[in] ILoginID	Login handle.
	[in] nChannelID	Channel number.
	[in] dwAlarmType	Alarm event type.
	[in] bNeedPicFile	Whether subscribe to image files
	[in] cbAnalyzerData	Callback function of AI event.
	[in] dwUser	Custom data
	[in] Reserved	Reserved field
Return value	<ul style="list-style-type: none"> • Success: subscription handle of LLONG type. • Failure: 0. 	
Note	If the interface returns 0, please use CLIENT_GetLastError to get the error code.	

For details of AI event types, please see Table 3-22.

Table 3-22 AI event type

dwAlarmType Definition	Macro	Value	Description	Call back the corresponding structure of pAlarmInfo
EVENT_IVS_ALL		0x00000001	All types	None
EVENT_IVS_CROSSLINEDETECTION		0x00000002	Tripwire	DEV_EVENT_CROSSLINE_INFO
EVENT_IVS_CROSSREGIONDETECTION		0x00000003	Intrusion	DEV_EVENT_CROSSREGION_INFO

3.7.2 CLIENT_RadiometryDetach

Table 3-25 Stop subscribing heat map data

Item	Description
Name	Stop subscribing heat map data.
Function	<pre> BOOL CLIENT_RadiometryDetach(LLONG IAttachHandle); </pre>
Parameter	<div>[in] IAttachHandle</div> <div>Return value of CLIENT_ RadiometryAttach.</div>
Return value	<ul style="list-style-type: none"> ● Success: TRUE. ● Failure: FALSE.
Note	None.

3.7.3 CLIENT_RadiometryFetch

Table 3-26 Capture heat map data

Item	Description
Name	Capture heat map data.
Function	<pre> BOOL CLIENT_RadiometryFetch(LLONG ILoginID, const NET_IN_RADIOOMETRY_FETCH* pInParam, NET_OUT_RADIOOMETRY_FETCH* pOutParam, int nWaitTime); </pre>
Parameter	[in] ILoginID Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] pInParam Input parameter.
	[in] pOutParam Output parameter.
	[in] nWaitTime Timeout. The unit is millisecond.
Return value	<ul style="list-style-type: none"> ● Success: TRUE. ● Failure: FALSE.
Note	None.

3.7.4 CLIENT_RadiometryDataParse

Table 3-27 Parse heat map data

Item	Description
Name	Parse heat map data.
Function	<pre> BOOL CLIENT_RadiometryDataParse(const NET_RADIOMETRY_DATA* pBuf, unsigned short* pImg, float* pTemp); </pre>
Parameter	<div>[in] pBuf</div> <div>Heat map data.</div>

Item	Description	
	[in out] plmg	Unzipped data is a gray map. Introducing null pointer indicates this data is not needed.
	[in out] pTemp	Temperature data of each pixel. Introducing null pointer indicates this data is not needed.
Return value	<ul style="list-style-type: none"> ● Success: TRUE. ● Failure: FALSE. 	
Note	None.	

3.8 PTZ Control

3.8.1 CLIENT_QueryDevState

Table 3-28 Control PTZ

Item	Description	
Name	Query the device status.	
Function	<pre> BOOL CALL_METHOD CLIENT_QueryDevState(LLONG lLoginID, int nType, char *pBuf, int nBufLen, int *pRetLen, int waittime=1000); </pre>	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in]nType	Query the information type.
	[out] pBuf	Used to receive the cache for returned data of query. The data structure of the returned data varies depending on the query type.
	[in]nBufLen	Cache length. The unit is byte.
	[out]pRetLen	The actual length of the returned data. The unit is byte.
	[in]waittime	Waiting time for querying status, and it is 1,000 ms by default, which can be set as needs.
Return value	<ul style="list-style-type: none"> ● Success: TRUE. ● Failure: FALSE. 	
Note	If the interface returns FALSE, please use CLIENT_GetLastError to get the error code.	

3.8.2 CLIENT_DHPTZControlEx2

Table 3-29 Control PTZ

Item	Description	
Name	PTZ control.	
Function	<pre> BOOL CLIENT_DHPTZControlEx2(LLONG ILoginID, int nChannelID, DWORD dwPTZCommand, LONG IParam1, LONG IParam2, LONG IParam3, BOOL dwStop , void* param4); </pre>	
Parameter	[in] ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] nChannelID	Video channel number that is a round number starting from 0.
	[in] dwPTZCommand	Control command type.
	[in] IParam1	Cache length. The unit is byte.
	[in] IParam2	The actual length of the returned data. The unit is byte.
	[in] IParam3	Waiting time for querying status, and it is 1,000 ms by default, which can be set as needs.
	[in] dwStop	Stop mark, which is valid for operations of eight directions. When performing other operations, enter FALSE for this parameter.
	[in] param4	Suppor extended control command parameters
Return value	<ul style="list-style-type: none"> Success: TRUE. Failure: FALSE. 	
Note	If the interface returns FALSE, please use CLIENT_GetLastError to get the error code.	

3.9 Fire Alarm

3.9.1 CLIENT_RealLoadPictureEx

Table 3-30 Subscribe to firepoint alarm event

Item	Description
Name	Subscribe to firepoint alarm event.
Function	<pre> LLONG CLIENT_RealLoadPictureEx(LLONG ILoginID, int nChannelID, DWORD dwAlarmType, </pre>

Item	Description	
	BOOL bNeedPicFile, fAnalyzerDataCallBack cbAnalyzerData, LDWORD dwUser, void* Reserved);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] nChannelID	Channel of subscribing firepoint alarm event.
	[in] dwAlarmType	Expected alarm type Firepoint alarm event: EVENT_IVS_FIREWARNING.
	[in] bNeedPicFile	Whether to subscribe to pictures and files.
	[in] cbAnalyzerData	Callback function of intelligent picture alarm.
	[in] dwUser	Custom data, SDK returns the data to user by calling back intelligent picture alarm function fAnalyzerDataCallBack.
	[out]Reserved	Reserve parameters.
Return value	<ul style="list-style-type: none"> ● Success: TRUE. ● Failure: FALSE. 	
Note	None.	

3.9.2 CLIENT_StopLoadPic

Table 3-31 Stop subscribing firepoint alarm event

Item	Description	
Name	Stop subscribing firepoint alarm event.	
Function	BOOL CLIENT_StopLoadPic(LLONG lAnalyzerHandle);	
Parameter	[in] lAnalyzerHandle	Subscribing ID of firepoint alarm event.
Return value	<ul style="list-style-type: none"> ● Success: TRUE. ● Failure: FALSE. 	
Note	None.	

3.9.3 CLIENT_SetDVRMessCallBack

Table 3-32 Set firepoint information callback function interface

Item	Description	
Name	Set firepoint information callback function interface.	
Function	void CLIENT_SetDVRMessCallBack(fMessCallBack cbMessage, LDWORD dwUser);	
Parameter	[in] cbMessage	Alarm callback function.
	[in] dwUser	Custom data.
Return value	None	

Item	Description
Note	Call CLIENT_SetDVRMessCallBack interface before subscribing alarm, and the received callback function does not include events with pictures.

3.9.4 CLIENT_StartListenEx

Table 3-33 Subscribe to firepoint information interface

Item	Description		
Name	Subscribe to firepoint information interface.		
Function	<pre> BOOL CLIENT_StartListenEx(LLONG ILoginID); </pre>		
Parameter	<table border="1"> <tr> <td>[in] ILoginID</td> <td>Return value of CLIENT_LoginWithHighLevelSecurity.</td> </tr> </table>	[in] ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
[in] ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity.		
Return value	<ul style="list-style-type: none"> • Success: TRUE. • Failure: FALSE. 		
Note	Alarm events of all device give feedback to user by the callback function set by CLIENT_SetDVRMessCallBack interface.		

3.9.5 CLIENT_StopListen

Table 3-34 Stop subscribing to alarm

Item	Description		
Name	Stop subscribing to alarm interface.		
Function	<pre> BOOL CLIENT_StopListen(LLONG ILoginID); </pre>		
Parameter	<table border="1"> <tr> <td>[in] ILoginID</td> <td>Return value of CLIENT_LoginWithHighLevelSecurity.</td> </tr> </table>	[in] ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
[in] ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity.		
Return value	<ul style="list-style-type: none"> • Success: TRUE. • Failure: FALSE. 		
Note	None.		

3.10 RadiometryGetCurrentHotColdSpotInfo

Table 3-35 Get the information of the cold spot and hot spot

Item	Description		
Name	Get the information of the cold spot (the spot with the lowest temperature) and hot spot (the spot with the highest temperature).		
Function	<pre> BOOL CALL_METHOD CLIENT_RadiometryGetCurrentHotColdSpotInfo(LLONG ILoginID, const NET_IN_RADIOMETRY_CURRENTHOTCOLDSPOT_INFO* pInParam, NET_OUT_RADIOMETRY_CURRENTHOTCOLDSPOT_INFO* pOutParam, int nWaitTime); </pre>		
Parameter	<table border="1"> <tr> <td>[in] ILoginID</td> <td>Return value of CLIENT_LoginWithHighLevelSecurity.</td> </tr> </table>	[in] ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
[in] ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity.		

Item	Description	
	[in] pInParam	Interface input parameters, and the memory resource requested and released by users.
	[out] pOutParam	Interface output parameters, and the memory resource requested and released by users.
	[in]waittime	Waiting time for querying status, and it is 1,000 ms by default, which can be set as needs.
Return value	<ul style="list-style-type: none"> Success: TRUE. Failure: FALSE. 	
Note	If the interface returns FALSE, please use CLIENT_GetLastError to get the error code.	

3.11 RadiometryGetRandomRegionTemper

Table 3-36 Get the parameters of the temperature measurement area

Item	Description	
Name	Get the parameters of the temperature measurement area.	
Function	<pre> BOOL CALL_METHOD CLIENT_RadiometryGetRandomRegionTemper(LLONG ILoginID, const NET_IN_RADIOMETRY_RANDOM_REGION_TEMPER* pInParam, NET_OUT_RADIOMETRY_RANDOM_REGION_TEMPER* pOutParam, int nWaitTime); </pre>	
Parameter	[in] ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] pInParam	Interface input parameters, amd the memory resource requested and released by users.
	[out] pOutParam	Interface output parameters, amd the memory resource requested and released by users..
	[in]waittime	Waiting time for querying status, and it is 1,000 ms by default, which can be set as needs.
Return value	<ul style="list-style-type: none"> Success: TRUE. Failure: FALSE. 	
Note	If the interface returns FALSE, please use CLIENT_GetLastError to get the error code.	

3.12 Temperature Measurement

3.12.1 CLIENT_GetNewDevConfig

Table 3-37 Get the channel name

Item	Description
Name	Get the channel name.

Item	Description	
Function	BOOL CLIENT_GetNewDevConfig(LLONG ILoginID, char* szCommand, int nChannelID, char* szOutBuffer, DWORD dwOutBufferSize, Int *error, int waittime=500);	
Parameter	[in] ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] szCommand	Command parameters
	[in] nChannelID	Channel number, which is from 0.
	[out] szOutBuffer	Output buffer.
	[out] dwOutBufferSize	Output buffer size.
	[out] error	Error codes 0: Success 1: Failure 2: Illegal data 3: Cannot handle atpresent 4: No permission
	[in] nWaitTime	Timeout. It is 500 ms.
Return value	<ul style="list-style-type: none"> ● Success: TRUE. ● Failure: FALSE. 	
Note	None.	

3.12.2 CLIENT_ParseData

Table 3-38 Parse configuration data

Item	Description	
Name	Parse the queried configuration.	
Function	BOOL CLIENT_ParseData(char* szCommand, char* szInBuffer, LPVOID lpOutBuffer, DWORD dwOutBufferSize, void* pReserved);	
Parameter	[in] szCommand	Command parameters.
	[in] szInBuffer	input buffer.
	[out] lpOutBuffer	Output buffer.
	[in] dwOutBufferSize	Output buffer size.
	[in] pReserved	Reserved
Return value	<ul style="list-style-type: none"> ● Success: TRUE. ● Failure: FALSE. 	
Note	None.	

3.12.3 CLIENT_PacketData

Table 3-39 Packet data of device configuration

Item	Description	
Name	Packet the configuration to be set.	
Function	<pre> BOOL CLIENT_PacketData(char* szCommand, LPVOID lpInBuffer, DWORD dwInBufferSize, char* szOutBuffer, DWORD dwOutBufferSize); </pre>	
Parameter	[in] szCommand	Command Parameters.
	[in] lpInBuffer	Input buffer.
	[in] dwInBufferSize	Input buffer size.
	[out]szOutBuffer	Output buffer.
	[in] dwOutBufferSize	Output buffer size.
Return value	<ul style="list-style-type: none"> • Success: TRUE. • Failure: FALSE. 	
Note	None.	

3.12.4 LIENT_SetNewDevConfig

Table 3-40 Set device configuration

Item	Description	
Name	Set configuration.	
Function	<pre> BOOL CLIENT_SetNewDevConfig(LLONG lLoginID, char* szCommand, int nChannelID, char* szInBuffer, DWORD dwInBufferSize, int *error, int *restart, int waittime=500); </pre>	
Parameter	[in]lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] szCommand	Command Parameters
	[in] nChannelID	Channel number, which is from 0.
	[in] szInBuffer	Input buffer, json information composed of user storage, which is used for setting configuration.
	[in] dwInBufferSize	Buffer address size.
	[out] error	Error codes 0: Success 1: Failure

Item	Description	
		2: Illegal data 3: Cannot handle atpresent 4: No permission.
	[in] restart	Restart mark address.
	[in] waittime	Timeout. It is 500 ms..
Return value	<ul style="list-style-type: none"> • Success: TRUE. • Failure: FALSE. 	
Note	None.	

3.12.5 CLIENT_SetDVRMessCallBack

Table 3-41 Set firepoint information callback function interface

Item	Description	
Name	Set alarm callback function.	
Function	<pre>void CLIENT_SetDVRMessCallBack(fMessCallBack cbMessage, LDWORD dwUser);</pre>	
Parameter	[in] cbMessage	<ul style="list-style-type: none"> • Message callback function, which can call back the status of devices, such as alarm status. • 0 means callback is prohibited.
	[in] dwUser	Custom data.
Return value	None	
Note	<ul style="list-style-type: none"> • Set the device message callback function to get the current status of the device, unrelated with the call order. SDK does not callback by default. • For this callback function, fMessCallBack must first call the alarm message subscription interface CLIENT_StartListenEx. 	

3.12.6 CLIENT_StartListenEx

Table 3-42 Subscribe to alarm

Item	Description	
Name	Subscribe to alarm.	
Function	<pre>BOOL CLIENT_StartListenEx(LLONG ILoginID);</pre>	
Parameter	[in] ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
Return value	<ul style="list-style-type: none"> • Success: TRUE. • Failure: FALSE. 	
Note	Subscribe to device messages, and the received messages are retrieved from the settings of CLIENT_SetDVRMessCallBack.	

For details of alarm event types, please see Table 3-50.

Table 3-43 Alarm type

dwAlarmType Macro Definition	Value	Description	Call back the corresponding structure of pAlarmInfo
DH_ALARM_ALARM_EX	0x2101	External alarm	DWORD
DH_MOTION_ALARM_EX	0x2102	Motion detection	DWORD
DH_ALARM_ALARM_EX2	0x2175	Local alarm	ALARM_ALARM_INFO_EX2
DH_ALARM_TEMPERATURE	0x2135	High temperature alarm	ALARM_TEMPERATURE_INFO
DH_ALARM_STORAGE_FAILURE_EX	0x2163	Storage error alarm	ALARM_STORAGE_FAILURE_EX
DH_ALARM_STORAGE_NOT_EXIST	0x3167	Storage group does not exist	ALARM_STORAGE_NOT_EXIST_INFO
DH_ALARM_ALARM_EX2	0x2175	High temperature alarm	ALARM_ALARM_INFO_EX2
DH_ALARM_HEATING_TEMPERATURE	0x31aa	abnormal temperature alarm of temperature measurement point	ALARM_HEATING_TEMPERATURE_INFO
DH_ALARM_FIREWARNING	0x31b5	Firepoint	ALARM_FIREWARNING_INFO
DH_ALARM_SMOKE_DETECTION	0x31d5	Smoke alarm	ALARM_SMOKE_DETECTION_INFO
DH_ALARM_BETWEENRULE_TEMPERATURE_DIFF	0x31d6	Temperature difference alarm	ALARM_BETWEENRULE_DIFF_TEMPERATURE_INFO
DH_ALARM_HOTSPOT_WARNING	0x31d8	Hot spot abnormal alarm	ALARM_HOTSPOT_WARNING_INFO
DH_ALARM_COLDSPOT_WARNING	0x31d9	Cold point abnormal alarm	ALARM_COLDSPOT_WARNING_INFO
DH_ALARM_FIREWARNING_INFO	0x31da	Fire information setting	ALARM_FIREWARNING_INFO_DETAIL
DH_ALARM_FACE_OVERHEATING	0x31db	High temperature (face)	ALARM_FACE_OVERHEATING_INFO

3.12.7 CLIENT_StopListen

Table 3-44 Stop subscribing firepoint information interface

Item	Description	
Name	Stop subscribing firepoint information interface.	
Function	BOOL CLIENT_StopListen(LLONG ILoginID);	
Parameter	[in] ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
Return value	<ul style="list-style-type: none"> Success: TRUE. 	

Item	Description
	<ul style="list-style-type: none"> Failure: FALSE.
Note	None.

3.12.8 CLIENT_RadiometryGetCurTemperAll

Table 3-45 Get the temperature information of the rules

Item	Description
Name	Get the temperature information of the temperature measurement rules.
Function	<pre> BOOL CALL_METHOD CLIENT_RadiometryGetCurTemperAll(LLONG ILoginID, const NET_IN_RADIOMETRY_GET_CUR_TEMPER_ALL_INFO* pInParam, NET_OUT_RADIOMETRY_GET_CUR_TEMPER_ALL_INFO* pOutParam, int nWaitTime); </pre>
Parameter	[in] ILoginID Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] pInParam Interface input parameters, and the memory resource requested and released by users.
	[out] pOutParam Interface output parameters, and the memory resource requested and released by users.
	[in] nWaitTime Waiting time for querying status, and it is 1,000 ms by default, which can be set as needs.
Return value	<ul style="list-style-type: none"> Success: TRUE. Failure: FALSE.
Note	If the interface returns FALSE, please use CLIENT_GetLastError to get the error code.

3.12.9 CLIENT_RadiometryAttachTemper

Table 3-46 Subscribe to temperature measurement data

Item	Description
Name	Subscribe to temperature measurement data.
Function	<pre> LLONG CALL_METHOD CLIENT_RadiometryAttachTemper(LLONG ILoginID, const NET_IN_RADIOMETRY_ATTACH_TEMPER* pInParam, NET_OUT_RADIOMETRY_ATTACH_TEMPER* pOutParam, int nWaitTime); </pre>
Parameter	[in] ILoginID Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] pInParam Interface input parameters, and the memory resource requested and released by users.
	[out] pOutParam Interface output parameters, and the memory resource requested and released by users.
	[in] nWaitTime Waiting time for querying status, and it is 1,000 ms by default, which can be set as needs.
Return value	<ul style="list-style-type: none"> Success: TRUE.

Item	Description
	<ul style="list-style-type: none"> Failure: FALSE.
Note	If the interface returns FALSE, please use CLIENT_GetLastError to get the error code.

3.12.10 CLIENT_RadiometryAttachTemper

Table 3-47 Stop subscribing firepoint information interface

Item	Description		
Name	Stop subscribing firepoint information interface.		
Function	<pre> BOOL CALL_METHOD CLIENT_RadiometryDetachTemper(LLONG IAttachTemperHandle); </pre>		
Parameter	<table> <tr> <td>[in] IAttachTemperHandle</td><td>Processing handle.</td></tr> </table>	[in] IAttachTemperHandle	Processing handle.
[in] IAttachTemperHandle	Processing handle.		
Return value	<ul style="list-style-type: none"> Success: TRUE. Failure: FALSE. 		
Note	None.		

4 Callback Definition

4.1 fDisConnect

Table 4-1 Disconnection callback

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

4.2 fHaveReConnect

Table 4-2 Reconnection callback

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

4.3 fRealDataCallBackEx2

Table 4-3 Callback of real-time monitoring data

Item	Description	
Name	Callback of real-time monitoring data.	
Function	<pre>typedef void (CALLBACK * fRealDataCallBackEx2)(LONG IRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LONG param, LDWORD dwUser);</pre>	
Parameter	[out] IRealHandle	Return value of CLIENT_RealPlayEx.
	[out] dwDataType	Data type: <ul style="list-style-type: none"> ● 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. <ul style="list-style-type: none"> ● 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 tagCBPCMDDataParam structure when dwDataType is 3.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.4 fMessCallBack

Table 4-4 Alarm callback

Item	Description	
Name	Alarm callback.	
Function	<pre> BOOL (CALLBACK *fMessCallBack)(LONG ICommand, LLONG ILoginID, char *pBuf, DWORD dwBufLen, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser); </pre>	
Parameter	[out] ICommand	Alarm type. For details, see Table 4-5.
	[out] ILoginID	The return value of login interface.
	[out] pBuf	The cache of received alarm data. The data varies depending on the called listening interface and ICommand value.
	[out] dwBufLen	Length (unit: byte) of pBuf.
	[out] pchDVRIP	Device IP.
	[out] nDVRPort	Port.
	[out] dwUser	Customized data.
Return value	<ul style="list-style-type: none"> Success: TRUE. Failure: FALSE. 	
Note	Setting callback is often called during application initialization. During the callback, different processing methods are adopted based on different device IDs and command values.	

Table 4-5 Alarm type

Alarm Type	Name	pBuf
DH_MOTION_ALARM_EX	Motion detecion	The number of data bytes is the same as the number of video channels. Each byte represents the motion detection alarm status of a video channel. 1 indicates an alarm and 0 indicates no alarm.
DH_ALARM_STORAGE_FAILURE	Hard disk failur	ALARM_STORAGE_FAILURE array
EVENT_ALARM_VIDEOLOSS	Video loss	None
DH_ALARM_FRONTDISCONNECT	TPC Network disconnection	ALARM_FRONTDISCONNECT

Alarm Type	Name	pBuf
DH_ALARM_ALARM_EX	External alarm	The number of data bytes is the same as the number of alarm channels. Each byte represents the alarm status of a video channel. 1 indicates an alarm and 0 indicates no alarm.

4.5 fAnalyzerDataCallback

Table 4-6 Alarm callback

Item	Description	
Name	AI event callback.	
Function	<pre>typedef int (CALLBACK *fAnalyzerDataCallback)(LONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved);</pre>	
Parameter	[out]lAnalyzerHandle	The return value of CLIENT_RealLoadPictureEx.
	[out]dwAlarmType	AI event type.
	[out]pAlarmInfo	Cache of event details.
	[out]pBuffer	Image cache.
	[out]dwBufSize	Size of image cache.
	[out]dwUser	User's data.
	[out]reserved	Reserved.
Return value	None.	
Note	None.	

4.6 fRadiometryAttachCB

Table 4-7 Callback of temperature distribution data

Item	Description
Name	Callback of temperature distribution data.
Function	<pre>typedef void (CALLBACK *fRadiometryAttachCB)(LONG lAttachHandle, NET_RADIOMETRY_DATA* pBuf, int nBufLen, LDWORD dwUser);</pre>

Parameter	[out] IAttachHandle	Return value of CLIENT_RadiometryAttach.
	[out] pBuf	Address of data block.
	[out] nBufLen	Length of the data block. The unit is byte.
	[out] dwUser	User parameter of the callback.
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.