NetSDK Programming Manual
(Camera)

V1.0.1

Foreword

Purpose

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

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

The Manual describes the SDK interfaces and processes of the general function modules for IPC, SD and Thermal IP Camera (TPC). For more function modules and data structures, refer to *NetSDK Development Manual*.

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

Reader

- SDK software development engineers
- Project managers
- Product managers

Signals

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

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

Revision History

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

About the Manual

- The Manual is for reference only. If there is inconsistency between the Manual and the actual product, the actual product shall govern.
- All the designs and software are subject to change without prior written notice. The product updates might cause some differences between the actual product and the Manual. Please contact the customer service for the latest program and supplementary documentation.
- There still might be deviation between the actual value of some data and the value provided, if there is any doubt or dispute, please refer to our final explanation.
- Please contact the supplier or customer service if there is any problem occurred when using the device.
- We are not liable for any loss caused by the operations that do not comply with the Manual.
- All trademarks, registered trademarks and the company names in the Manual are the properties of their respective owners.
- Please visit our website or contact your local service engineer for more information. If there is any uncertainty or controversy, please refer to our final explanation.

Glossary

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

Term	Definition
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 using different colors to show the statistical activity of an area during a certain period.

Table of Contents

1 Overview

1.1 General

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

The following are the main functions:

SDK initialization, device login, real-time monitoring, PTZ control, voice talk, video snapshot, heat map (temperature) and heat map (activity).

The development kit might be different dependent on the environment.

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

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

Table 1-1

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

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

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

Table 1-2

NOTE

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

1.2 Applicability

- Recommended memory: No less than 512 M
- System supported by SDK:
  - ◇ Windows
    Windows 10/Windows 8.1/Windows 7 and Windows Server 2008/2003
  - ◇ Linux
    The common Linux systems such as Red Hat/SUSE

2 Function Modules

There are seven function modules in this chapter. Each function module includes SDK initialization, device login, logout, and SDK resource release. The optional processes do not affect the use of other processes.

2.1 SDK Initialization

2.1.1 Introduction

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

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

2.1.2 Interface Overview

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

Table 2-1

2.1.3 Process

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

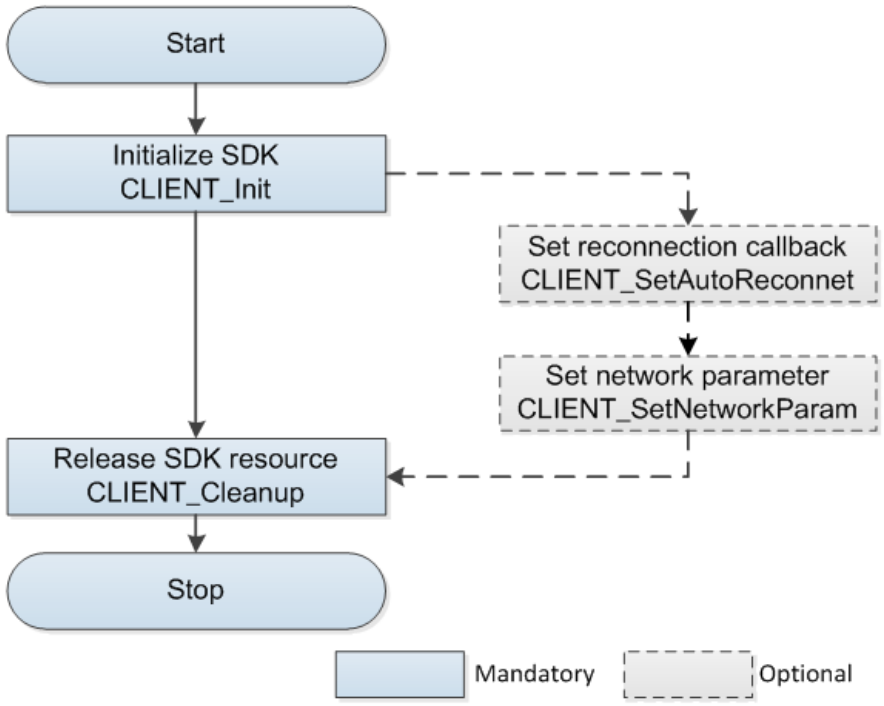

Figure 2-1

Process Description

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

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

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

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

Notes for Process

- Call **CLIENT_Init** and **CLIENT_Cleanup** in pairs. It supports 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 ths callback.
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser)
```

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

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

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

2.2 Device Initialization

2.2.1 Introduction

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

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

2.2.2 Interface Overview

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

Table 2-2

2.2.3 Process

2.2.3.1 Device Initialization

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

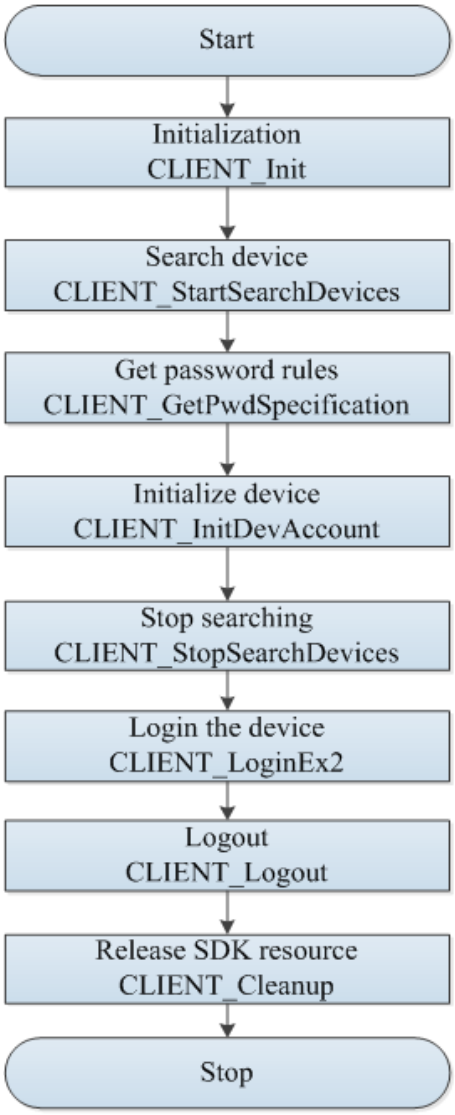

Figure 2-2

Process Description

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

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

NOTE

Multi-thread calling is not supported.

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

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

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

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

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

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

Notes for Process

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

2.2.3.2 Password Reset

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

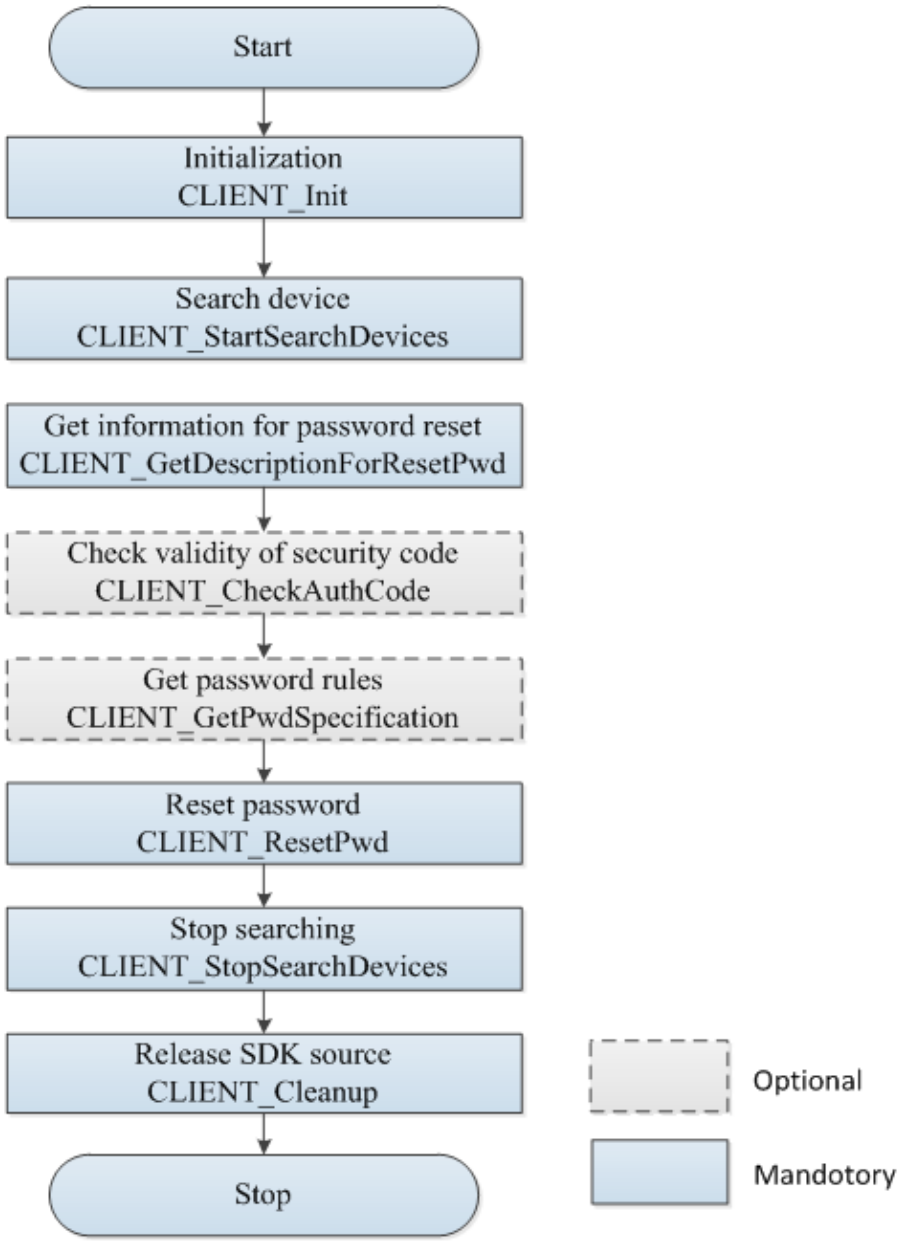

Figure 2-3

Process Description

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

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

NOTE

Multi-thread calling is not supported.

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

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

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

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

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

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

Step 9 After using the function module, call **CLIENT_Logout** to logout the device.

Step 10 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

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

2.2.4 Example Code

2.2.4.1 Device Initialization

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

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

2.2.4.2 Password Reset

```
//Firstly, call CLIENT_StartSearchDevices to get the device information.
//Get the information for password reset
NET_IN_DESCRIPTION_FOR_RESET_PWD stIn = {sizeof(stIn)};
strncpy(stIn.szMac, szMac, sizeof(stIn.szMac) - 1); //Set mac value
```

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

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

NET_OUT_DESCRIPTION_FOR_RESET_PWD stOut = {sizeof(stOut)};

char szTemp[360];

stOut.pQrCode = szTemp;

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

//(Optional) Check the security code

NET_IN_CHECK_AUTHCODE stIn1 = {sizeof(stIn1)};

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

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

NET_OUT_CHECK_AUTHCODE stOut1 = {sizeof(stOut1)};

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

//Get password rules

NET_IN_PWD_SPECI stIn2 = {sizeof(stIn2)};

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

NET_OUT_PWD_SPECI stOut2 = {sizeof(stOut2)};

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

//Reset password

NET_IN_RESET_PWD stIn3 = {sizeof(stIn3)};

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

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

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

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

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

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

NET_OUT_RESET_PWD stOut3 = {sizeof(stOut3)};

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

2.3 Device Login

2.3.1 Introduction

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

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

2.3.2 Interface Overview

Interface	Implication
CLIENT_LoginEx2	Login.
CLIENT_Logout	Logout.

Table 2-3

2.3.3 Process

For the process of login, see Figure 2-4.

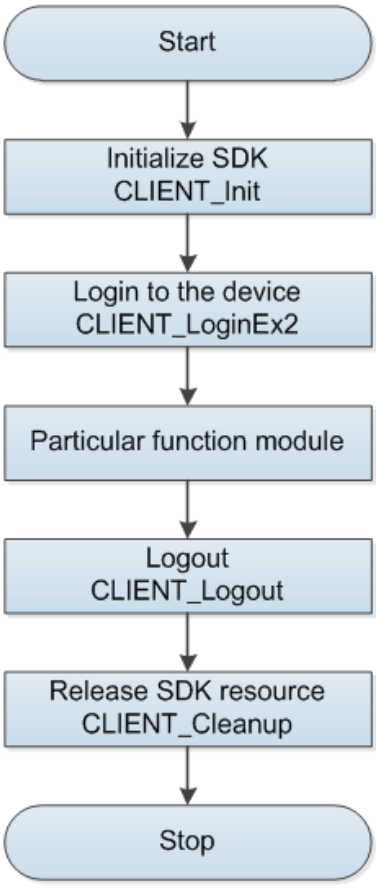

Figure 2-4

Process Description

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

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

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

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

Step 5 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

- Login handle: When the login is successful, the returned value is not 0 (even the handle is smaller than 0, the login is also successful). One device can login multiple times with different handle at each login. If there is not special function module, it is suggested to login only one time. The login handle can be repeatedly used on other function modules.
- Logout: The interface will release the opened functions internally, but it is not suggested to rely on the cleaning up function. For example, if you opened the monitoring function, you should call the interface that stops the monitoring function when it is no longer required.
- Use login and logout in pairs: The login consumes some memory and socket information and release sources once logout.
- Login failure: It is suggested to check the failure through the error parameter of the login interface. For the common error code, see Table 2-4.

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

Table 2-4

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

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

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

2.3.4 Example Code

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

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

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

2.4 Real-time Monitoring

2.4.1 Introduction

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

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

- Supports calling the window handle for SDK to directly decode and play the stream (Windows system only).
- Supports calling the real-time stream 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.

2.4.2 Interface Overview

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

Table 2-5

2.4.3 Process

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

2.4.3.1 SDK Decoding Play

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

For the process of playing by SDK decoding library, see Figure 2-5.

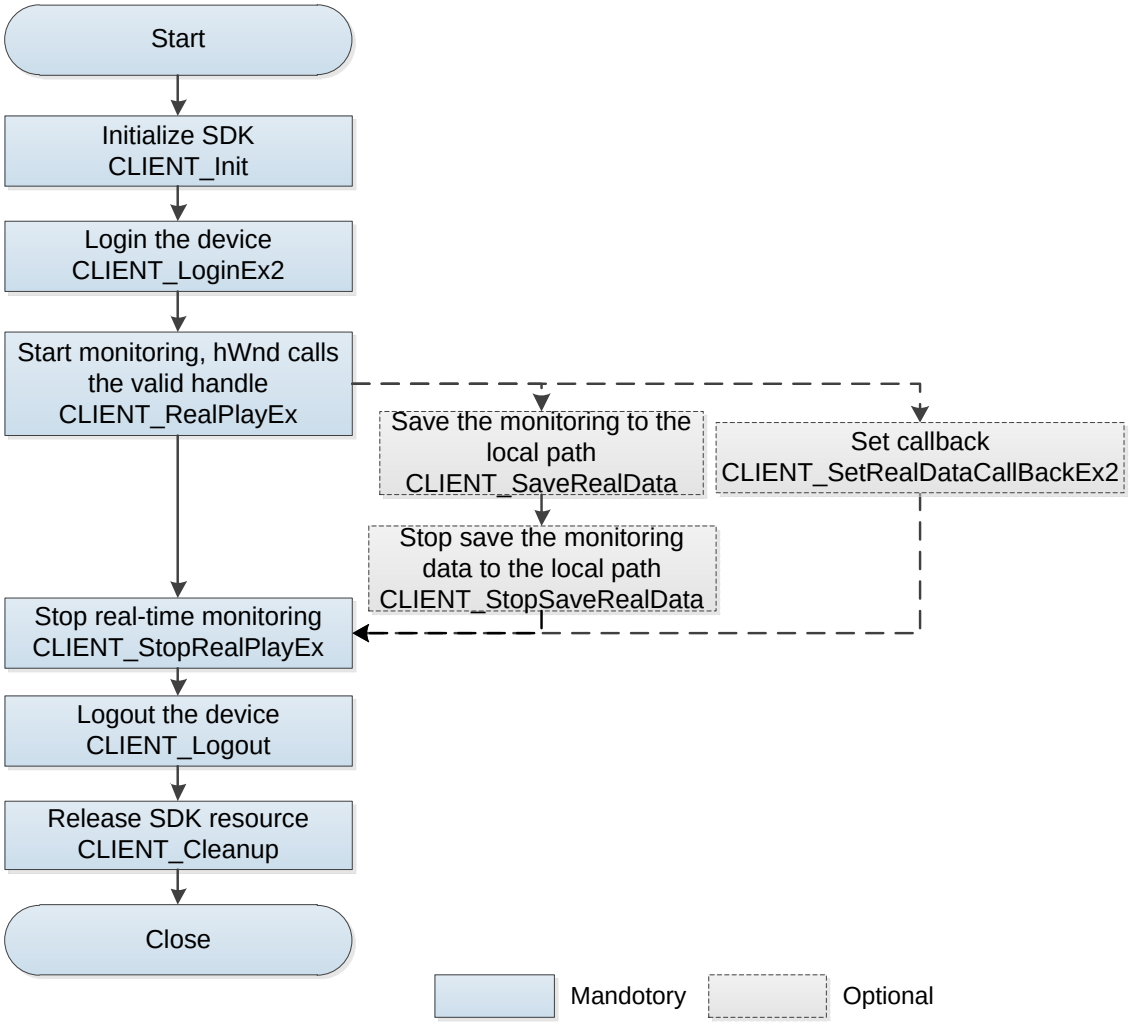

Figure 2-5

Process Description

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

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

Step 3 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 obtain two login handles to deal with the main stream and sub stream respectively.
- Calling succeeded but no image: SDK decoding needs to use dhplay.dll. It is suggested to check if dhplay.dll and its auxiliary library are missing under the running directory. See Table 1-1.
- If the system resource is insufficient, the device might return error instead of stream. You can receive an event DH_REALPLAY_FAILD_EVENT in the alarm callback that is set in CLIENT_SetDVRMessCallBack. This event includes the detailed error codes. See "DEV_PLAY_RESULT Structure" in *Network SDK Development Manual.chm*.
- 32 channels limit: The decoding consumes resources especially for the high definition videos. Considering the limited resources at the client, currently the maximum channels are set to be 32. If more than 32, it is suggested to use third party play library. See "2.4.3.2 Call Third Party Play Library."

2.4.3.2 Call Third Party Play Library

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

For the process of calling the third party play library, see Figure 2-6.

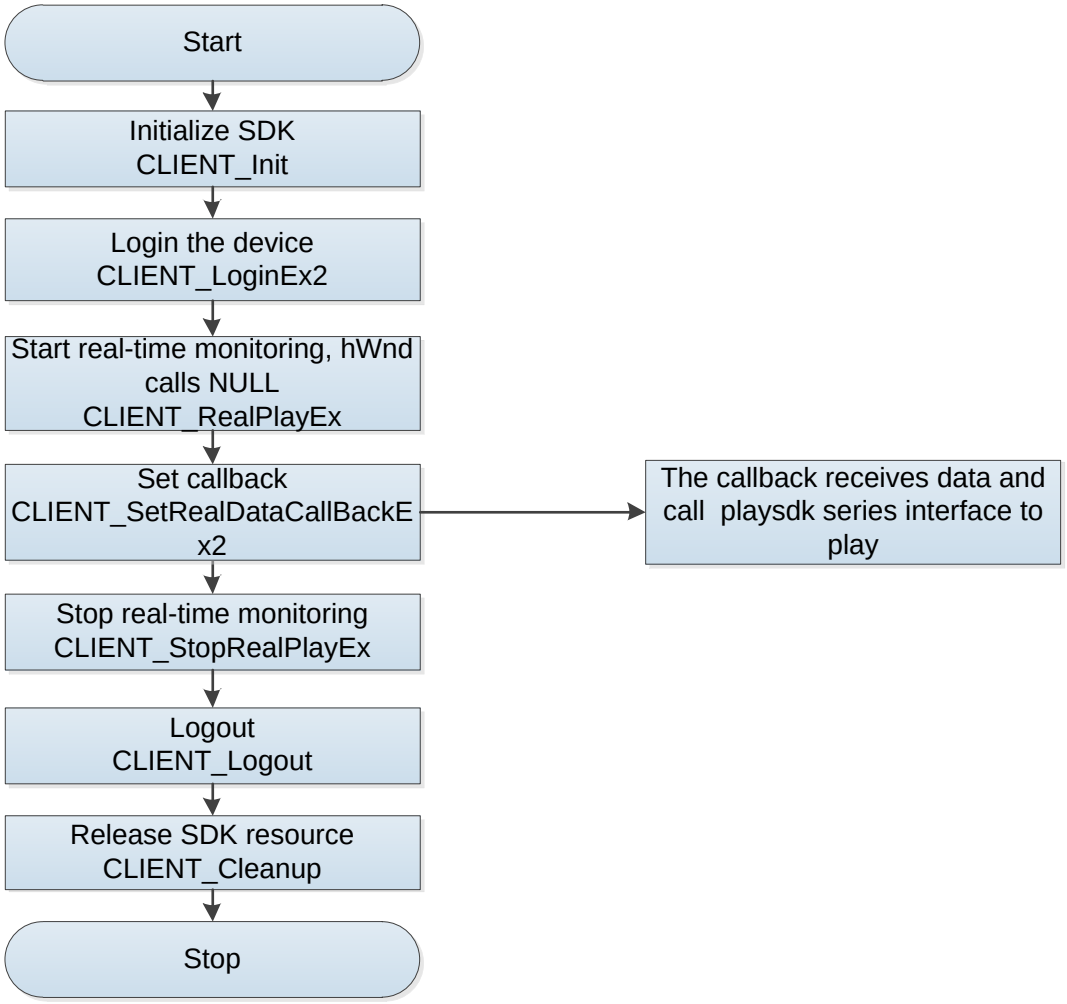

Figure 2-6

Process Description

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

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

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

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

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

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

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

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

Notes for Process

- Stream format: It is recommended to use PlaySDK for decoding.
- Lag image
  - ◇ When using PlaySDK for decoding, there is a default channel cache size (the PLAY_OpenStream interface in playsdk) for decoding. If the stream resolution value is big, it is recommended to modify the parameter value smaller such as 3 M.

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

2.4.4 Example Code

2.4.4.1 SDK Decoding Play

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

2.4.4.2 Call Play Library

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

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

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

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

2.5 Video Snapshot

2.5.1 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.

2.5.2 Interface Overview

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

Table 2-6

2.5.3 Process

Video snapshot is consisted of network snapshot and local snapshot.

2.5.3.1 Network Snapshot

For the process of network snapshot, see Figure 2-7.

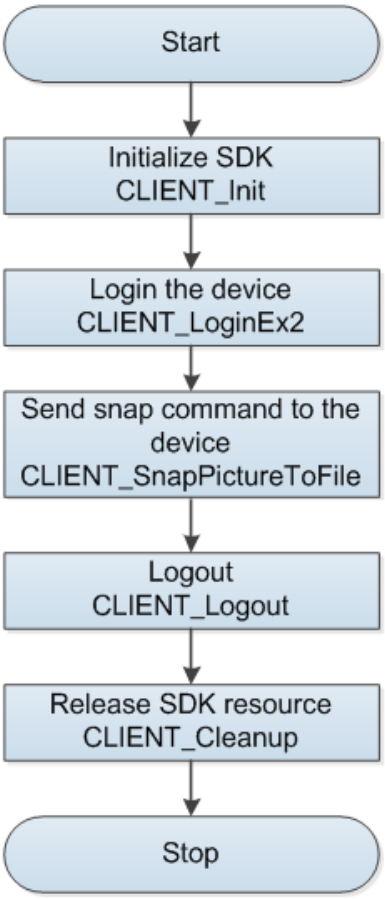

Figure 2-7

Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.
Step 2 Call **CLIENT_LoginEx2** to login the device.
Step 3 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).

2.5.3.2 Local Snapshot

For the process of local snapshot, see Figure 2-8.

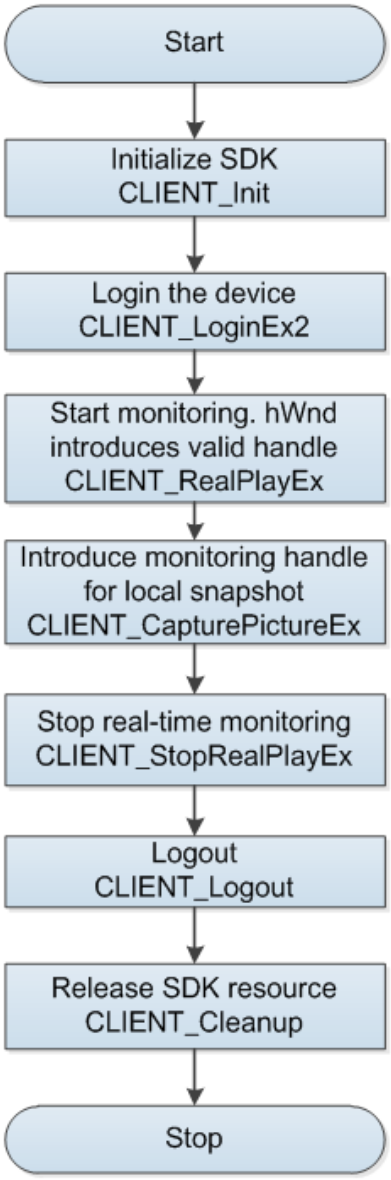

Figure 2-8

Process Description

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

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

Step 3 Call **CLIENT_RealPlayEx** to start monitoring and obtain the monitoring handle.

Step 4 Call **CLIENT_CapturePictureEx** to introduce the monitoring handle.

Step 5 Call **CLIENT_StopRealPlayEx** to stop the real-time monitoring.

Step 6 Call **CLIENT_Logout** to logout the device.

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

2.5.4 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(lLoginHandle, &stuSnapParams))
{
      printf("CLIENT_SnapPictureEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
}
delete[] pBuffer;

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

2.6 PTZ Control

2.6.1 Introduction

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

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

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

2.6.2 Interface Overview

Interface	Implication
CLIENT_DHPTZControlEx2	PTZ control extension interface

Table 2-7

2.6.3 Process

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

For the process of PTZ control, see Figure 2-9.

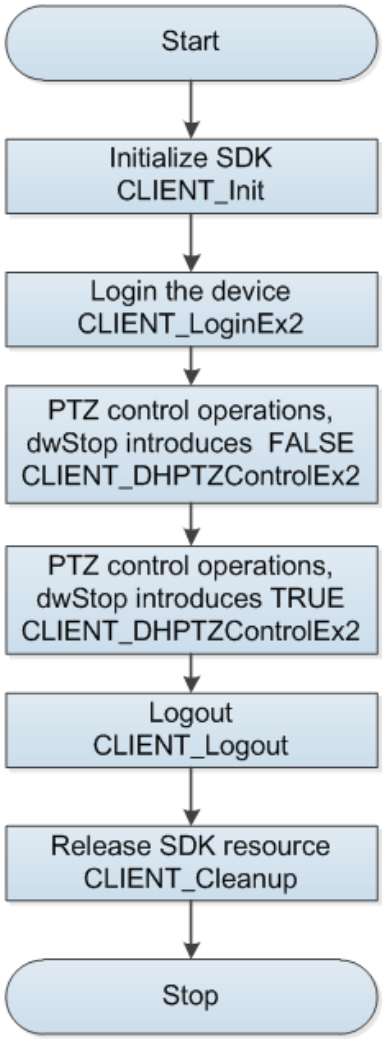

Figure 2-9

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

For the process of one-time operation of PTZ control, see Figure 2-10.

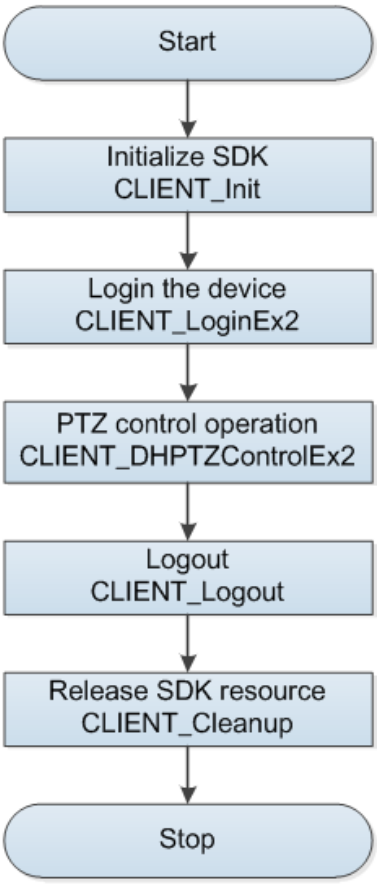

Figure 2-10

Process Description

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

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

Step 3 Call **CLIENT_DHPTZControlEx2** to operate the PTZ according to the situation. Different PTZ command might need different parameters, and part of commands need to call the corresponding stop command, such as moving left and moving right. For details, see "2.6.4 Example Code."

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

Step 5 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

- Fast positioning: For the SD, take the current monitoring image center as origin, and the valid range of horizontal and vertical coordinates is [–8191, 8191]. For example, if the horizontal coordinate is 2000 and the vertical is 2000, the SD moves toward upper right and gets a new origin, which means the coordinate specified every time is only relative to the current location.
- 3-dimensional positioning: For the SD, there is an initial position first. The horizontal coordinate is [0, 3600] and the vertical is [–1800, 1800]. The coordinate specified each time is the absolute coordinate and is irrelevant to the location of the SD image last time.
- For more example code see the SDK package on the website.

2.6.4 Example Code

```
LONG lParam1 = 0; // Rotating speed in horizontal direction.
LONG lParam2 = 4; // Rotating speed in vertical direction.
LONG lParam3 = 0;
// Continuous operation: take moving upward as example.
// Start moving upward.
BOOL bRet = CLIENT_DHPTZControlEx2(lLoginHandle, nChannelId, DH_PTZ_UP_CONTROL, lParam1,
                                   lParam2, lParam3, FALSE, NULL);
// Stop moving upward.
bRet = CLIENT_DHPTZControlEx2(lLoginHandle, nChannelId, DH_PTZ_UP_CONTROL, lParam1, lParam2,
                                   lParam3, TRUE, NULL);

// One-time operation movement: take fast positioning as an example.
lParam1 = 2000;    // Horizontal coordinate, valid range[–8191,8191]
lParam2 = 2000;    // Vertical coordinate, valid range [–8191,8191]
lParam3 = 1;       // Zoom, valid range (-16–16),1 indicates rotating without zooming
bRet = CLIENT_DHPTZControlEx2(lLoginHandle, nChannelId, DH_EXTPTZ_FASTGOTO, lParam1, lParam2,
                                   lParam3, FALSE, NULL);
```

2.7 Voice Talk

2.7.1 Introduction

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

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

2.7.2 Interface Overview

Interface	Implication
CLIENT_StartTalkEx	Start voice talk
CLIENT_StopTalkEx	Stop voice talk
CLIENT_RecordStartEx	Start client record (valid only in Windows system)
CLIENT_RecordStopEx	Stop client record (valid only in Windows system)
CLIENT_TalkSendData	Send voice data to the device
CLIENT_AudioDecEx	Decode audio data (valid only in Windows system)

Table 2-8

2.7.3 Process

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

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

This process is valid only in Windows system. For the process of voice talk, see Figure 2-11.

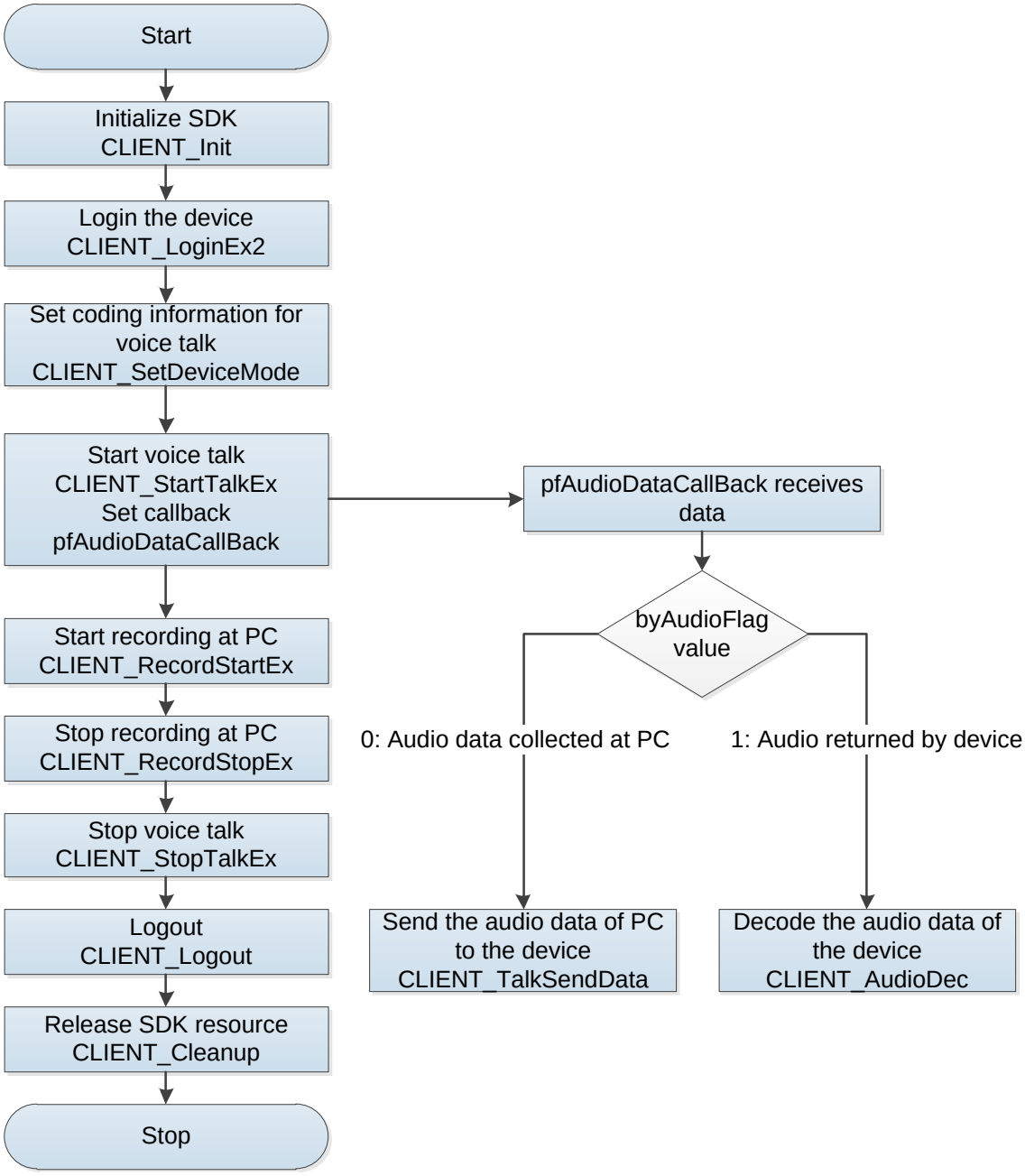

Figure 2-11

Process Description

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

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

Step 3 Call **CLIENT_SetDeviceMode** to set decoding information of voice talk. Set parameter emType as DH_TALK_ENCODE_TYPE.

Step 4 Call **CLIENT_StartTalkEx** to set callback and start voice talk. In the callback, call **CLIENT_AudioDec** to decode the audio data that is sent from the decoding device, and call **CLIENT_TalkSendData** to send the audio data of the PC end to the device.

Step 5 Call **CLIENT_RecordStartEx** to start recording at PC. After this interface is called, the voice talk callback in CLIENT_StartTalkEx will receive the local audio data.

Step 6 After using the voice talk function, call **CLIENT_RecordStopEx** to stop recording.

Step 7 Call **CLIENT_StopTalkEx** to stop voice talk.

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

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

2.7.4 Example Code

```
// Set the voice talk encoding data. Take PCM as an example.
DHDEV_TALKDECODE_INFO curTalkMode;
curTalkMode.encodeType = DH_TALK_PCM;
curTalkMode.nAudioBit = 16;
curTalkMode.dwSampleRate = 8000;
curTalkMode.nPacketPeriod = 25;
CLIENT_SetDeviceMode(lLoginHandle, DH_TALK_ENCODE_TYPE, &curTalkMode);
// Start voice talk
lTalkHandle = CLIENT_StartTalkEx(lLoginHandle, AudioDataCallBack, (LDWORD)NULL);
if(0 != lTalkHandle)
{
    BOOL bSuccess = CLIENT_RecordStartEx(lLoginHandle);
}

// Stop local recording
if (!CLIENT_RecordStopEx(lLoginHandle))
{
    printf("CLIENT_RecordStop Failed!Last Error[%x]\n", CLIENT_GetLastError());
}
```

```
// Stop voice talk
if (0 != lTalkHandle)
{
    CLIENT_StopTalkEx(lTalkHandle);
}
void CALLBACK AudioDataCallBack(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE byAudioFlag, LDWORD dwUser)
{
    if(0 == byAudioFlag)
    {
        // Send the sound data detected by the PC to the device
        LONG lSendLen = CLIENT_TalkSendData(lTalkHandle, pDataBuf, dwBufSize);
        if(lSendLen != (LONG)dwBufSize)
        {
            printf("CLIENT_TalkSendData Failed!Last Error[%x]\n" , CLIENT_GetLastError());
        }
    }
    else if(1 == byAudioFlag)
    {
        // Send the voice data from the device to SDK for encoding and play.
        CLIENT_AudioDec(pDataBuf, dwBufSize);
    }
}
```

2.8 Heat Map (Temperature)

2.8.1 Introduction

Heat map (temperature) function obtains the temperature distribution data, and the gray map and temperature map.

NOTE

Only the devices equipped with temperature measurement support this function.

2.8.2 Interface Overview

Interface	Implication
CLIENT_RadiometryAttach	Subscribe the temperature distribution data (heat map).
CLIENT_RadiometryDetach	Cancel subscribing the temperature distribution data.
CLIENT_RadiometryFetch	Inform start of obtaining the heat map data.

Interface	Implication
CLIENT_RadiometryDataParse	Unzip the heat map data and convert it to the gray data and temperature data in the unit of pixel.

Table 2-9

2.8.3 Process

For the process of heat map (temperature), see Figure 2-12.

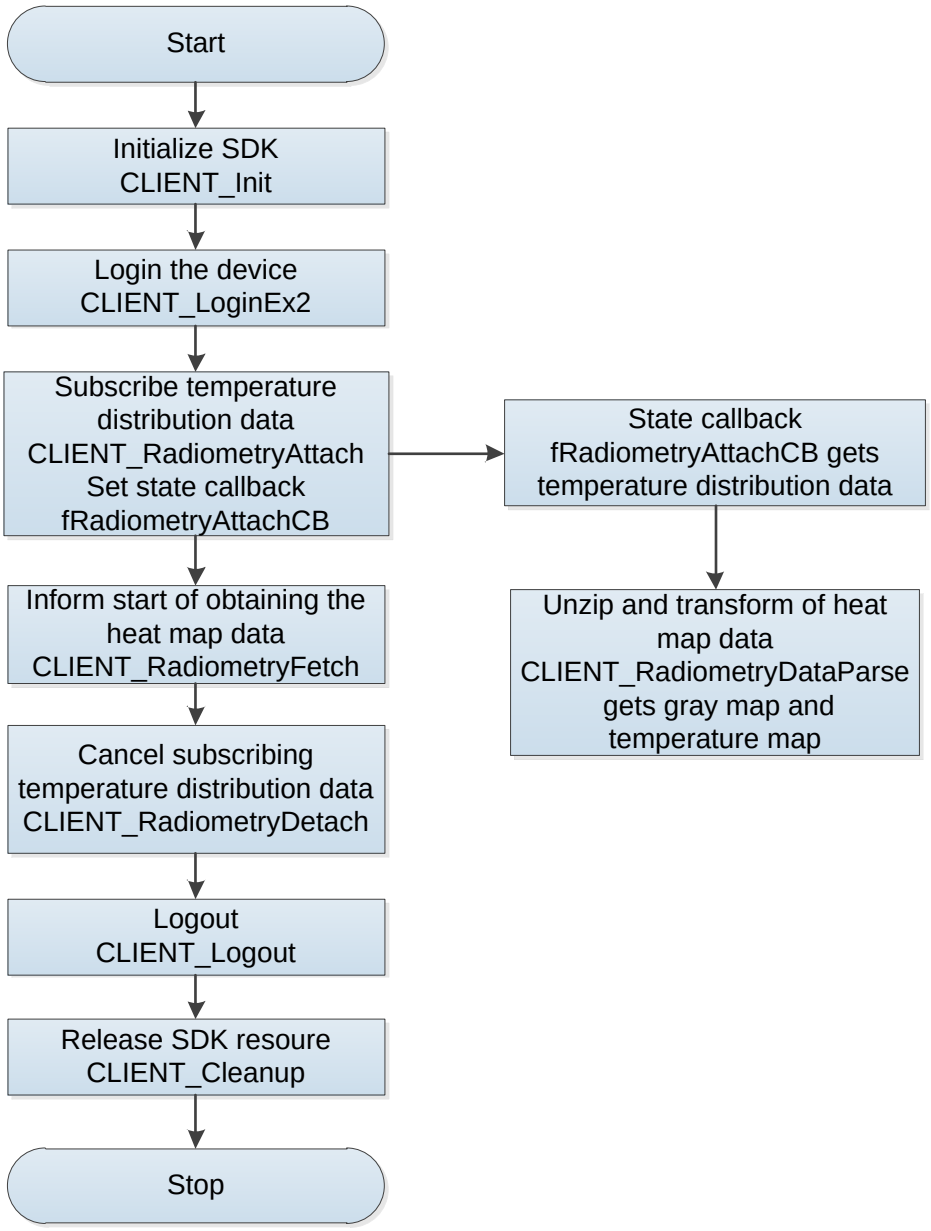

Figure 2-12

Process Description

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

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

Step 3 Call **CLIENT_RadiometryAttach** to subscribe the temperature distribution data and register status callback. After the device has reported the temperature status, the callback **fRadiometryAttachCB** will inform you.

Step 4 Call **CLIENT_RadiometryFetch** to inform the device to start obtaining 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 cancel subscribing the temperature distribution data after finishing use.

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

- 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 the channel number: If the camera has only one sensor, the subscription channel number is 0; if the camera has double sensors, the subscription channel is 0 or 1. For the TPC with double sensors, the second channel is TPC.
- The heat map data returned by the interface is not picture and needs to be converted to pictures based on the rules provided by the device.

2.8.4 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 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_lLoginID, &stInFetch, &stOutFetch, 3000);

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

2.9 Heat Map (Activity)

2.9.1 Introduction

Heat map (activity) function is a picture using different colors to show the statistical activity of an area during a certain period.

NOTE

The Heat map (activity) here is different from that of temperature in "2.8 Heat Map (Temperature)".

2.9.2 Interface Overview

Interface	Implication
CLIENT_QueryDevState	Get the statistical activity information.

Table 2-10

2.9.3 Process

For the process of heat map (activity), see Figure 2-13.

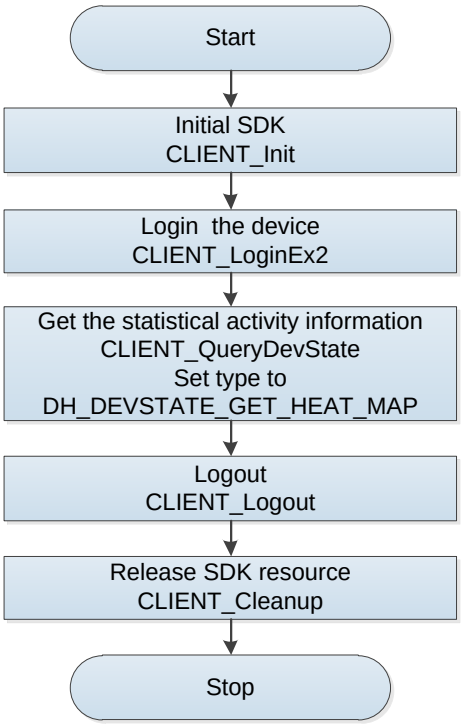

Figure 2-13

Process Description

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

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

Step 3 Call **CLIENT_QueryDevState** to get the statistical activity information, and set the **type** parameter to **DH_DEVSTATE_GET_HEAT_MAP**.

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

The heat map data returned by the interface is not picture data and needs to be converted to pictures based on the rules provided by the device.

2.9.4 Example Code

```
int nRetLen = 0;
NET_QUERY_HEAT_MAP stHeatMap = {sizeof(stHeatMap)};
stHeatMap.stuIn.nChannel = 0;
NET_TIME_EX stBegin = {2017,10,1,0,0,0,0};
NET_TIME_EX stEnd = {2017,10,1,1,1,1,0};
stHeatMap.stuIn.stuBegin = stBegin;
stHeatMap.stuIn.stuEnd = stEnd;
stHeatMap.stuIn.nPlanID = 1;
stHeatMap.stuIn.emDataType = EM_HEAT_PIC_DATA_TYPE_GRAYDATA;// Support gray data by
```

```
default
stHeatMap.stuOut.nBufLen = 10*1024*1024;
stHeatMap.stuOut.pBufData = new char[stHeatMap.stuOut.nBufLen]; // Apply for memory according to stHeatMap.stuOut.nBufLen
memset(stHeatMap.stuOut.pBufData, 0, stHeatMap.stuOut.nBufLen);
CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_GET_HEAT_MAP,(char *)&stHeatMap,stHeatMap.dwSize, &nRetLen ,3000);
```

3 Interface Definition

3.1 SDK Initialization

3.1.1 SDK CLIENT_Init

Item	Description	
Name	Initialize SDK.	
Function	BOOL CLIENT_Init(fDisConnect cbDisConnect, LDWORD dwUser);	
Parameter	[in]cbDisConnect	Disconnection callback.
	[in]dwUser	User parameter of disconnection callback.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	● The precondition for calling other function modules. ● 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

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

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

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

3.1.4 CLIENT_SetNetworkParam

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

3.2 Device Initialization

3.2.1 CLIENT_StartSearchDevices

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

3.2.2 CLIENT_InitDevAccount

Item	Description
Name	Initialize the device.

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

3.2.3 CLIENT_GetDescriptionForResetPwd

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

3.2.4 CLIENT_CheckAuthCode

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

3.2.5 CLIENT_ResetPwd

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

3.2.6 CLIENT_GetPwdSpecification

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

3.2.7 CLIENT_StopSearchDevices

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

3.3 Device Login

3.3.1 CLIENT_LoginEx2

Item	Description
Name	Login the device.

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

The following table shows information about error code:

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

Table 3-1

3.3.2 CLIENT_Logout

Item	Description
Name	User logout the device.

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

3.4 Real-time Monitoring

3.4.1 CLIENT_RealPlayEx

Item	Description	
Name	Open the real-time monitoring.	
Function	LLONG CLIENT_RealPlayEx(LLONG lLoginID, int nChannelID, HWND hWnd, DH_RealPlayType rType);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginEx2.
	[in]nChannelID	Video channel number is a round number starting from 0.
	[in]hWnd	Window handle valid only under Windows system.
	[in]rType	Preview type.
Return value	● Success: not 0 ● Failure: 0	
Note	Windows system: ● When hWnd is valid, the corresponding window displays picture. ● When hWnd is NULL, get the video data through setting a callback and send to user for handle.	

The following table shows information about preview type:

Preview type	Meaning
DH_RType_Realplay	Real-time preview
DH_RType_Multiplay	Multi-picture preview
DH_RType_Realplay_0	Real-time monitoring—main stream, equivalent to DH_RType_Realplay
DH_RType_Realplay_1	Real-time monitoring—sub stream 1
DH_RType_Realplay_2	Real-time monitoring—sub stream 2
DH_RType_Realplay_3	Real-time monitoring—sub stream 3
DH_RType_Multiplay_1	Multi-picture preview—1 picture
DH_RType_Multiplay_4	Multi-picture preview—4 pictures
DH_RType_Multiplay_8	Multi-picture preview—8 pictures

Preview type	Meaning
DH_RType_Multiplay_9	Multi-picture preview—9 pictures
DH_RType_Multiplay_16	Multi-picture preview—16 pictures
DH_RType_Multiplay_6	Multi-picture preview—6 pictures
DH_RType_Multiplay_12	Multi-picture preview—12 pictures
DH_RType_Multiplay_25	Multi-picture preview—25 pictures
DH_RType_Multiplay_36	Multi-picture preview—36 pictures

Table 3-2

3.4.2 CLIENT_StopRealPlayEx

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

3.4.3 CLIENT_SaveRealData

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

3.4.4 CLIENT_StopSaveRealData

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

Item	Description
Note	None.

3.4.5 CLIENT_SetRealDataCallBackEx2

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

The following table shows information about parameter dwFlag:

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

Table 3-3

3.5 Video Snapshot

3.5.1 CLIENT_SnapPictureToFile

Item	Description	
Name	Snapshot.	
Function	BOOL CLIENT_SnapPictureToFile(LLONG lLoginID, const NET_IN_SNAP_PIC_TO_FILE_PARAM* pInParam, NET_OUT_SNAP_PIC_TO_FILE_PARAM* pOutParam, int nWaitTime);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] pInParam	Input parameter.

Item	Description	
	[in] pOutParam	Output parameter.
	[in] nWaitTime	Timeout. The unit is millisecond.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	● Synchronous interface. The device captures snapshot and sends to the user through internet. ● The device is required to support this function.	

3.5.2 CLIENT_CapturePictureEx

Item	Description	
Name	Snapshot.	
Function	BOOL CLIENT_CapturePictureEx(LLONG hPlayHandle, const char *pchPicFileName, NET_CAPTURE_FORMATS eFormat);	
Parameter	[in] hPlayHandle	Return value of CLIENT_RealPlayEx.
	[in] pchPicFileName	Save path.
	[in] eFormat	Picture format.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	● Synchronous interface. Directly write the picture data as file. ● Capture the pictures from the real-time monitoring data stream from device.	

3.6 PTZ Control

3.6.1 CLIENT_DHPTZControlEx2

Item	Description
Name	PTZ control.
Function	BOOL CLIENT_DHPTZControlEx2(LLONG lLoginID, int nChannelID, DWORD dwPTZCommand, LONG lParam1, LONG lParam2, LONG lParam3, BOOL dwStop , void* param4);

Item	Description	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] nChannelID	Video channel number that is a round number starting from 0.
	[in] dwPTZCommand	Control command type.
	[in] lParam1	Parameter 1.
	[in] lParam2	Parameter 2.
	[in] lParam3	Parameter 3.
	[in] dwStop	Stop mark, which is valid for operations of eight directions. When performing other operations, enter **FALSE** for this parameter.
	[in] param4	Support the following extension command: DH_EXTPTZ_MOVE_ABSOLUTELY DH_EXTPTZ_MOVE_CONTINUOUSLY DH_EXTPTZ_GOTOPRESET DH_EXTPTZ_SET_VIEW_RANGE DH_EXTPTZ_FOCUS_ABSOLUTELY DH_EXTPTZ_HORSECTORSCAN DH_EXTPTZ_VERSECTORSCAN DH_EXTPTZ_SET_FISHEYE_EPTZ
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	For the relationship between dwPTZCommand and Param1, Param2 and Param3, see Table 3-4.	

For the relationship between dwPTZCommand and Param1, Param2 and Param3, see Table 3-4.

dwPTZCommand macro definition	Function	param1	param2	param3
DH_PTZ_UP_CONTROL	Up	None	Vertical speed (1–8)	None
DH_PTZ_DOWN_CONTROL	Down	None	Vertical speed (1–8)	None
DH_PTZ_LEFT_CONTROL	Left	None	Horizontal speed (1–8)	None
DH_PTZ_RIGHT_CONTROL	Right	None	Horizontal speed (1–8)	None
DH_PTZ_ZOOM_ADD_CONTROL	Zoom+	None	Multi-speed	None
DH_PTZ_ZOOM_DEC_CONTROL	Zoom-	None	Multi-speed	None
DH_PTZ_FOCUS_ADD_CONTROL	Focus+	None	Multi-speed	None
DH_PTZ_FOCUS_DEC_CONTROL	Focus-	None	Multi-speed	None
DH_PTZ_APERTURE_ADD_CONTROL	Aperture+	None	Multi-speed	None

dwPTZCommand macro definition	Function	param1	param2	param3
DH_PTZ_APERTURE_DEC_CONTROL	Aperture-	None	Multi-speed	None
DH_PTZ_POINT_MOVE_CONTROL	Move to preset point	None	Value of preset point	None
DH_PTZ_POINT_SET_CONTROL	Set	None	Value of preset point	None
DH_PTZ_POINT_DEL_CONTROL	Delete	None	Value of preset point	None
DH_PTZ_POINT_LOOP_CONTROL	Cruise among points	Cruise route	None	76: Start 99: Automatic 96: Stop
DH_PTZ_LAMP_CONTROL	Lamp wiper	0x01: Start x00: Stop	None	None
DH_EXTPTZ_LEFTTOP	Left top	Vertical speed (1–8)	Horizontal speed (1–8)	None
DH_EXTPTZ_RIGHTTOP	Right top	Vertical speed (1–8)	Horizontal speed (1–8)	None
DH_EXTPTZ_LEFTDOWN	Left bottom	Vertical speed (1–8)	Horizontal speed (1–8)	None
DH_EXTPTZ_RIGHTDOWN	Right bottom	Vertical speed (1–8)	Horizontal speed (1–8)	None
DH_EXTPTZ_ADDTOLOOP	Add preset point to tour	Tour route	Value of preset point	None
DH_EXTPTZ_DELFROMLOOP	Delete preset point in cruise	Cruise route	Value of preset point	None
DH_EXTPTZ_CLOSELOOP	Delete cruise	Cruise route	None	None
DH_EXTPTZ_STARTPANCRUISE	Start horizontal rotation	None	None	None
DH_EXTPTZ_STOPPANCRUISE	Stop horizontal rotation	None	None	None
DH_EXTPTZ_SETLEFTBORDER	Set left border	None	None	None
DH_EXTPTZ_RIGHTBORDER	Set right border	None	None	None
DH_EXTPTZ_STARTLINESCAN	Start line scan	None	None	None
DH_EXTPTZ_CLOSELINESCAN	Stop line scan	None	None	None
DH_EXTPTZ_SETMODESTART	Set mode start	Mode route	None	None
DH_EXTPTZ_SETMODESTOP	Set mode stop	Mode route	None	None

dwPTZCommand macro definition	Function	param1	param2	param3
DH_EXTPTZ_RUNMODE	Running mode	Mode route	None	None
DH_EXTPTZ_STOPMODE	Stop mode	Mode route	None	None
DH_EXTPTZ_DELETEMODE	Delete mode	Mode route	None	None
DH_EXTPTZ_REVERSECOMM	Reverse command	None	None	None
DH_EXTPTZ_FASTGOTO	Fast positioning	Horizontal coordinate (0–8192)	Vertical coordinate (0–8192)	Zoom (4)
DH_EXTPTZ_AUXIOPEN	Open auxiliary switch	Auxiliary point	None	None
DH_EXTPTZ_AUXICLOSE	Close auxiliary switch	Auxiliary point	None	None
DH_EXTPTZ_OPENMENU	Open SD menu	None	None	None
DH_EXTPTZ_CLOSEMENU	Close menu	None	None	None
DH_EXTPTZ_MENUOK	Menu confirm	None	None	None
DH_EXTPTZ_MENUCANCEL	Menu cancel	None	None	None
DH_EXTPTZ_MENUUP	Menu up	None	None	None
DH_EXTPTZ_MENUDOWN	Menu down	None	None	None
DH_EXTPTZ_MENULEFT	Menu left	None	None	None
DH_EXTPTZ_MENURIGHT	Menu right	None	None	None
DH_EXTPTZ_ALARMHANDLE	Alarm action with PTZ	Alarm input channel	Alarm action type: ● Preset point ● Line scan ● Cruise	Linkage value, such as preset point number
DH_EXTPTZ_MATRIXSWITCH	Matrix switch	Monitor device number (video output number)	Video input number	Matrix number
DH_EXTPTZ_LIGHTCONTROL	Light controller	Refer to DH_PTZ_LAMP_CONTROL	None	None

dwPTZCommand macro definition	Function	param1	param2	param3
DH_EXTPTZ_EXACTGOTO	3D positioning	Horizontal angle (0–3600)	Vertical coordinate (0–900)	Zoom (1–128)
DH_EXTPTZ_RESETZERO	Reset to zero	None	None	None
DH_EXTPTZ_UP_TELE	UP +TELE	Speed (1–8)	None	None
DH_EXTPTZ_DOWN_TELE	DOWN +TELE	Speed (1–8)	None	None
DH_EXTPTZ_LEFT_TELE	LEFT +TELE	Speed (1–8)	None	None
DH_EXTPTZ_RIGHT_TELE	RIGHT +TELE	Speed (1–8)	None	None
DH_EXTPTZ_LEFTUP_TELE	LEFTUP +TELE	Speed (1–8)	None	None
DH_EXTPTZ_LEFTDOWN_TELE	LEFTDOWN +TELE	Speed (1–8)	None	None
DH_EXTPTZ_TIGHTUP_TELE	TIGHTUP +TELE	Speed (1–8)	None	None
DH_EXTPTZ_RIGHTDOWN_TELE	RIGHTDOWN +TELE	Speed (1–8)	None	None
DH_EXTPTZ_UP_WIDE	UP +WIDE	Speed (1–8)	None	None
DH_EXTPTZ_DOWN_WIDE	DOWN +WIDE	Speed (1–8)	None	None
DH_EXTPTZ_LEFT_WIDE	LEFT +WIDE	Speed (1–8)	None	None
DH_EXTPTZ_RIGHT_WIDE	RIGHT +WIDE	Speed (1–8)	None	None
DH_EXTPTZ_LEFTUP_WIDE	LEFTUP +WIDE	Speed (1–8)	None	None
DH_EXTPTZ_LEFTDOWN_WIDE	LEFTDOWN +WIDE	Speed (1–8)	None	None
DH_EXTPTZ_RIGHTUP_WIDE	RIGHTUP +WIDE	Speed (1–8)	None	None
DH_EXTPTZ_RIGHTDOWN_WIDE	RIGHTDOWN +WIDE	Speed (1–8)	None	None

Table 3-4

3.7 Voice Talk

3.7.1 CLIENT_StartTalkEx

Item	Description

Item	Description	
Name	Start voice talk.	
Function	LLONG CLIENT_StartTalkEx(LLONG lLoginID, pfAudioDataCallBack pfcb, LDWORD dwUser);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] pfcb	Audio data callback.
	[in] dwUser	Parameter of audio data callback.
Return value	● Success: Not 0. ● Failure: 0.	
Note	None.	

3.7.2 CLIENT_StopTalkEx

Item	Description	
Name	Stop voice talk.	
Function	BOOL CLIENT_StopTalkEx(LLONG lTalkHandle);	
Parameter	[in] lTalkHandle	Return value of CLIENT_StartTalkEx.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.7.3 CLIENT_RecordStartEx

Item	Description	
Name	Start local recording.	
Function	BOOL CLIENT_RecordStartEx(LLONG lLoginID);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	Valid only in Windows system.	

3.7.4 CLIENT_RecordStopEx

Item	Description
Name	Stop local recording.
Function	BOOL CLIENT_RecordStopEx(LLONG lLoginID);

Item	Description	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	Valid only in Windows system.	

3.7.5 CLIENT_TalkSendData

Item	Description	
Name	Send audio data to device.	
Function	LONG CLIENT_TalkSendData(LLONG lTalkHandle, char *pSendBuf, DWORD dwBufSize);	
Parameter	[in] lTalkHandle	Return value of CLIENT_StartTalkEx.
	[in] pSendBuf	Pointer of audio data block that needs to be sent.
	[in] dwBufSize	Length of audio data black that needs to be sent. The unit is byte.
Return value	● Success: Length of audio data block. ● Failure: -1.	
Note	None.	

3.7.6 CLIENT_AudioDecEx

Item	Description	
Name	Decode audio data.	
Function	BOOL CLIENT_AudioDecEx(LLONG lTalkHandle, char *pAudioDataBuf, DWORD dwBufSize);	
Parameter	[in] lTalkHandle	Return value of CLIENT_StartTalkEx.
	[in] pAudioDataBuf	Pointer of audio data block that needs decoding.
	[in] dwBufSize	Length of audio data black that needs decoding. The unit is byte.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.8 Heat Map (Temperature)

3.8.1 CLIENT_RadiometryAttach

Item	Description	
Name	Start subscribing heat map data.	
Function	LLONG CLIENT_RadiometryAttach(LLONG lLoginID, const NET_IN_RADIOMETRY_ATTACH* pInParam, NET_OUT_RADIOMETRY_ATTACH* pOutParam, int nWaitTime);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] pInParam	Input parameter.
	[in] pOutParam	Output parameter.
	[in] nWaitTime	Timeout. The unit is millisecond.
Return value	● Success: Not 0. ● Failure: 0.	
Note	None.	

3.8.2 CLIENT_RadiometryDetach

Item	Description	
Name	Stop subscribing heat map data.	
Function	BOOL CLIENT_RadiometryDetach(LLONG lAttachHandle);	
Parameter	[in] lAttachHandle	Return value of CLIENT_ RadiometryAttach.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.8.3 CLIENT_RadiometryFetch

Item	Description	
Name	Capture heat map data.	
Function	BOOL CLIENT_RadiometryFetch(LLONG lLoginID, const NET_IN_RADIOMETRY_FETCH* pInParam, NET_OUT_RADIOMETRY_FETCH* pOutParam, int nWaitTime);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] pInParam	Input parameter.

Item	Description	
	[in] pOutParam	Output parameter.
	[in] nWaitTime	Timeout. The unit is millisecond.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.8.4 CLIENT_RadiometryDataParse

Item	Description	
Name	Parse heat map data.	
Function	BOOL CLIENT_RadiometryDataParse(const NET_RADIOMETRY_DATA* pBuf, unsigned short* pImg, float* pTemp);	
Parameter	[in] pBuf	Heat map data.
	[in out] pImg	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	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.9 Heat Map (Activity)

3.9.1 CLIENT_QueryDevState

Item	Description	
Name	Get the connection state of remote device.	
Function	BOOL CLIENT_QueryDevState(LLONG lLoginID, int nType, char *pBuf, int nBufLen, int *pRetLen, int waittime=1000);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginEx2
	[in]nType	Query information type.

Item	Description	
	[out] pBuf	Buffer used to receive the data returned by the query. The data structure of returned data is different according to the type of query.
	[in]nBufLen	Buffer length. The unit is byte.
	[out]pRetLen	The data length returned actually. The unit is byte.
	[in]waittime	Waiting time for query state. The default waiting time is 1000ms. It can be set according to the needs.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

Query Name	nType	pBuf
Get the statistical activity information	DH_DEVSTATE_GET_HEAT_MAP	NET_QUERY_HEAT_MAP

4 Callback Definition

4.1 fSearchDevicesCB

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

4.2 fDisConnect

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

4.3 fHaveReConnect

Item	Description
Name	Reconnection callback.

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

4.4 fRealDataCallBackEx2

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

Item	Description	
	[out] param	Callback parameter structure. Different dwDataType value corresponds to different type. ● The param is blank pointer when dwDataType is 0. ● The param is the pointer of tagVideoFrameParam structure when dwDataType is 1. ● The param is the pointer of tagCBYUVDataParam structure when dwDataType is 2. ● The param is the pointer of tagCBPCMDataParam structure when dwDataType is 3.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.5 pfAudioDataCallBack

Item	Description	
Name	Audio data callback of voice talk.	
Function	typedef void (CALLBACK *pfAudioDataCallBack)(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE byAudioFlag, LDWORD dwUser) ;	
Parameter	[out] lTalkHandle	Return value of CLIENT_StartTalkEx.
	[out] pDataBuf	Address of audio data block.
	[out] dwBufSize	Length of the audio data block. The unit is byte.
	[out] byAudioFlag	Data type: 0: Local collecting. 1: Sending from device.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.6 fRadiometryAttachCB

Item	Description

Name	Callback of temperature distribution data.	
Function	typedef void (CALLBACK *fRadiometryAttachCB)(LLONG lAttachHandle, NET_RADIOMETRY_DATA* pBuf, int nBufLen, LDWORD dwUser) ;	
Parameter	[out] lAttachHandle	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.	