Preface

This document aims to introduce how to use the Hope-Wish Video SDK.

This SDK is applicable to all standardized front-end equipment of our company, and some customized equipment does not support some functions in the SDK due to differences.

This document is only based on the description of the 2.1.0.x version of the video SDK. There may be problems with the manual update lag in the future. The functions of the SDK are subject to the header files corresponding to each version.

If you encounter any problems during use, please contact us in time.

1.Over View 3
2. SDK Version Updates 4
3. Call follow 14
3.1 SDK Main Process 14
3.2 SDKAnti-droneCallProcess 15
4. Basic Interface Definition 17
4.1 Sdk main interface 17
4.2 Device interface 20
4.3 Video interfaces 29
4.4Video playback Interface 34
4.5 Record interface 46
4.6 PTZ interface 48
4.7 Alarm interface 58
4.8 Capture interface: 67
4.9 Angle interface 69
4.10 Preset interface 83
4.11 Cruise interface 86
4.12 Config interface 91
4.13 Serial interface 102
4.14 Focus query interface 107
4.15 Power status interface 112
4.17 Voice intercom interface 117
4.18 Exten interface 118
4.19 NVR interface 121
4.20 Handware trace interface 122
5 Infrared thermal imaging IPC device interface 133
5.1 Infrared thermal imaging IPC device interface 133
5.2 Get the movement configure interface 138
5.3 Get PTZ configuration interface 144
5.4 Get device configuration interface 148
5.5 Get smart analysis configuration interface 155
5.6 Get alarm configuration interface 159
5.7 Get trace configuration interface 161
5.8 Get temperature measurement configuration interface 165
5.9 Get the angle management interface 168
5.10 Get preset management interface 171
5.11 Disconnect 173
5.12 Release device pointer 174
6 Precautions 174

1.Over View

The video SDK mainly opens the interface for some of the features of the company's front-end equipment. In addition to the conventional video preview and PTZ control, it also involves the interface processing of azimuth information, focus information, alarm information, and transparent data information.

Devices that support real-time return can receive real-time azimuth and field of view information through the SDK. If the device does not support real-time return, the user can call the interface query method to obtain the azimuth and field of view of the device, and the alarm information The function is only valid for dual-band alarm equipment.

The angle, alarm, focus, and transparent transmission modules will all turn on the transparent transmission mode. Once the transparent transmission mode is enabled, all control commands are transmitted transparently, including ordinary control of the pan/tilt in 8 directions, lens control, etc.

Because our equipment distinguishes between single visible light (thermal imaging) and multi-spectral equipment, if the transparent transmission mode is turned on on visible light, the pan/tilt control cannot be performed on other software or web pages.

Also note that only one transparent transmission mode can be enabled at the same time.

2. SDK Version Updates

SDKImprint

Darw	Version	Increment
2020-10-10	v2.1.0.9	1. Adjust the bottom speed conversion, remove 0 2.The HPWS protocol realizes angle return, pan-tilt and lens control, and preset positions
2020-10-07	v2.1.0.8	1.The C interface adds an interface to query the horizontal angle, pitch and field of view angle
2020-09-11	v2.1.0.7	1.C# tracking module increases the interface for miss-target event registration 2.Adjust the MFC tool layout
2020-09-10	v2.1.0.6	1.Add image stabilization interface 2.Add optical fog and optical image stabilization switch 3.Increase the angular velocity method to control the PTZ and turntable interface 4.c interface adds the interface to start tracking and stop tracking 5.Optimize the hope-wish IPC protocol
2020-08-18	v2.1.0.4	1.Optimize the tracking module code of hope-wish ipc 2.Optimize the code of the tracking sample program
2020-08-14	v2.1.0.3	1.Change the name to VideoTool 2.The tracking module adds the interface of switching tracking scene and tracking algorithm 3.Connect instead ping Commond to Confirm the device is online 4.Fix the phenomenon of increased empty connections caused by special circumstances 5.Separate Hope-Wish protocol control sdk：Hpws.IrdSDK.dll 6.The video sdk integrates the Hope-Wish protocol sdk, and the overall Hope-Wish protocol sdk can also be used separately 7.Hope-Wish protocol SDK adds tracking and temperature measurement related interfaces 8.Video sdk is compatible with the tracking function of Hope-Wish protocol 9.Video sdk removes the concept of address code,

		and adds an interface set to single device mode in the ISerial interface file (TC/RC device needs to be called) 10.Optimize the VideoTool.exe tool, remove invalid parts, support 3 channels of video for recording and screenshots 11.Adjust the focus interface module and open the auto focus function of the c interface 12.VideoTool Increase the function of tracking stability test [TRACK] TEST_MISS=1 TEST_MISS_NUM=200
2020-07-22	v2.0.13.1	1.v2 alarm callback alarm frame structure adds id 2.Adjust Demo, add photoelectric tracking operation page 3.Fix the crash caused by quoting demo and keyboard library 4.The sample program name is changed to VideoTool.exe
2020-07-15	v2.0.13.0	1. SDK compilation adjusted to vs2019 2. After opening transparent transmission, delay 10s to set various data return 3.Implement Linux64-bit version support
2020-06-10	v2.0.12.19	1.Hope-Wish protocol supports hmac-sha256 new login verification method 2.Fix the problem that the auto focus interface cannot be completely closed 3.Change the BOOL light parameter to： int video_type HP_light =1 Represents visible light，Corresponds to the previous TRUE HP_ird =0 Represents thermal imaging，Corresponds to the previous FALSE
2020-05-29	v2.0.12.17	1.Add an interface for setting the hotspot threshold 2.Realize the password modification function of Hope-Wish device protocol 3.Fix the bug that the amount of missed targets cannot be analyzed and tracked 4.Increase the electronic compass query-feedback function 5.Fix the positioning playback problem in NVR playback

		6.Gathering protocol adds drawing callback function 7.Univision protocol adds angle positioning interface export 8.Add Haikang to enable electronic anti-shake interface 9.Optimize the tracking module, realize the unification of the guiding tracking and guiding interface, realize the error goal of automatically canceling the tracking, the guiding interface realizes the function of performing zooming every 50m, and the guiding tracking interface adopts internal thread processing, no longer blocking the calling thread, increase Interface for setting tracking threshold 10.c#demo adds interface translation for decoding callback
2020-05-8	v2.0.12.15	1.The c interface adds an interface for setting the original resolution and registering a unified alarm event 2.Support hardware click centering 3.The capture function of the c interface is compatible with the working mode with or without video preview 4.Unified alarm callback adds the callback format of the string 5.No. 200 and 203 preset positions are no longer set by default when angle monitoring is turned on 6.The c interface adds an interface for opening and closing intelligent analysis monitoring (used by Haikang equipment) 7.Optimize the alarm message of Hepu protocol, merge all alarm events into the unified alarm interface
2020-04-29	v2.0.12.10	1.Repair the abnormal phenomenon caused by disconnecting a certain channel after the Univision device is repeatedly connected 2.Increase the interface for obtaining network and image parameters 3.Added guided search function (zoom according to distance when searching) 4.Optimize search function, support level, pitch, snake search 5.The "HPVideoDemo_Radar.exe" program is added

		to the lib directory, which can be used to realize the linkage function with the radars of 38 Institutes and Southern Institute of Technology Radar linkage：1）Thermal imaging channel connected to photoelectric； 2）Click "Radar Linkage" to display the radar linkage interface； 3）Select the radar protocol and enter the latitude and longitude of the radar； 4）Click "Enable Monitoring"
2020-04-22	v2.0.12.8	1.x64 Support Dahua equipment
2020-04-17	v2.0.12.6	1.Adjust the general alarm protocol of thermal image ipc to solve the problem that the device alarm cannot be resolved 2.The pitch angle return resolution range is expanded to -100 degrees, supporting the angle display of overhead devices 3.c interface adds the function of setting the positioning speed
2020-04-09	v2.0.12.5	1.The configuration module adds functions such as system information acquisition, image parameter adjustment, network, time information setting, etc. 2.C language interface to increase the tracking function interface
2020-03-30	v2.0.12.4	1.Increase the feedback function of tracking target information 2.Adjust the tracking video discrimination of the tracking module, and adopt the status of the equipment feedback internally
2020-03-30	v2.0.12.1	1.Add ITrack tracking interface to realize frame selection tracking, trigger tracking, guide tracking and other functions
2020-03-19	v2.0.11.6	1.Increase the voice intercom module of Hepu, Haikang and Dahua agreement 2.Add Haikang NVR channel information query module 3.Increase the interface function of exception log and alarm log 4.Fix the problem that Dahua drawing callback is invalid 5.The device module adds an interface for setting the original video size 6.Develop the alarm function of V2 version, support

		thermal alarm, temperature measurement alarm, intelligent analysis alarm, unified alarm type 7.Realize the docking of Haikang intelligent analysis alarm 8.Increase the function of hardware frame selection tracking
2020-02-17	v2.0.11.4	1.The Hope-Wish configuration protocol coordinate system is all specified in the equipment protocol, which may involve 704x576, 1280*720, 640*512 2.Adjust the bottom layer to 10sping unreachable equipment for disconnection processing 3.Compatible with NVR preview and playback
2020-02-10	v2.0.11.3	1.Add Hope-Wish intelligent analysis configuration interface 2.Add Hope-Wish intelligent analysis alarm interface 3.Adjust the parameters of the Hope-Wish thermal imaging core configuration interface 4.IAlarm alarm module adds temperature alarm interface 5.The capture interface is compatible with preview and non-preview modes 6.c# demo to increase the temperature alarm sample program
2020-01-09	v2.0.11.1	1.Adjust the c interface and standardize the exported interface name 2.Remove redundant library files
2019-12-31	v2.0.10.12	1.Recompile the x64 library
2019-12-18	v2.0.10.11	1.Fix the problem that Uniview SDK fails to capture pictures 2.Hope-Wish thermal image increased by time playback and download 3.The independent processing of the Vision equipment, no longer mixed with the Haikang protocol module 4.Add the function of capturing pictures without preview 5.Increase the video playback buffer frame number interface, currently only valid for Hikvision, Hepu, and Shihui devices, and buffer 2 frames by default 6.Automatic search interface after adding angle positioning

		7.Increase the peripheral interface module to realize the ranging function
2019-10-14	v2.0.10.2	1.Update Uniview 2.0.2.2 version sdk 2.Remove the query mode of angle, focus, power status and other modules, and devices that do not support active feedback no longer use the timing query solution
2019-08-21	v2.0.10.0	1.Added support for Haikang local file playback, need to import Haikang play library 2.Increase the voice intercom module, currently only realizes the opening and closing of the voice intercom of the Univision device 3.Get the azimuth angle of the target in the screen, add a new interface: support the incoming lens azimuth and field of view 4.Increase the support of Hepu Thermal Image IPC 5.Increase the switch value control of the alarm output of Univision
2019-06-27	v2.0.9.27	1.C interface increased by distance zoom interface
2019-06-27	v2.0.9.26	1.Repair 64-bit preview abnormal problem
2019-06-27	v2.0.9.25	1.Repair cache data capture interface
2019-06-26	v2.0.9.24	1.Capture pictures without previewing the video 2.Increase the interface for status data query and return
2019-06-14	v2.0.9.22	1.Remove itrack interface file, fix the problem of auxiliary switch call in ptz 2.Fix the problem that calling the auxiliary switch to turn off is invalid 3.c interface video preview adds stream selection 4.Update c#demo 5.Mfc demo adjusts the status query, and analyzes the explanation in the transparent transmission callback
2019-06-08	v2.0.9.21	1.Fix the problem that garbled characters are displayed when Haikang dvs is set to OSD 2.c interface adds stream selection function 3.Cancel printing of current bit rate information
2019-05-31	v2.0.9.20	1.Preview interface to increase the choice of stream 2.Add an interface for querying the current bit rate
2019-05-30	v2.0.9.19	1.Replace the sdk of Uniview 2.0.2.1 version 2.Fire alarm detection does not receive the jump,

		only prints the information, but the alarm content is fed back
2019-05-06	v2.0.9.17	1.Increase the configuration function of Haikang osd
2019-04-30	v2.0.9.16	1.Print the specific exception number when the Haikang SDK reports an exception 2.When the jump is not received, no longer feedback alarm information 3.Optimize the alarm analysis process and enhance the analysis strength of the alarm
2019-04-18	v2.0.9.10	1.Improve the function of calling the baud rate setting in Uniview SDK 2.Fix the problem that the azimuth angle may not be returned when the angle query is turned on 3.The angle module adds automatic detection and non-active return mode. Once the detection is successful, it will automatically jump from the query mode to the active return mode 4.Optimize the demo, use PostMesssage processing after callbacks such as transparent transmission, angle, alarm, drawing, etc., to fix the conflict between the work thread and the interface thread
2019-04-12	v2.0.9.9	1.Remove the zoom module, if the device is the feedback focal length value, the application layer can perform *100 on the field of view data. 2.Adjust the angle of the field of view c interface, increase the parameters of the visible light option 3.Fix the problem that Haikang drawing callback does not feedback 4.Solve the problem that the video cannot be previewed after updating the Univision SDK
2019-04-09	v2.0.9.8	1.Increase the transparent transmission and automatic connection function of Haikang equipment 2.Increase the type of equipment of Vision 3.Replace the sdk of Uniview 2.0.2 version 4.Increase the demo of PyQt 5.Fix known bugs
2019-03-01	v2.0.9.5	1.Add device configuration maintenance module, add device restart interface 2.Demo adds violent test and superimposed field of view functions 3.Fix known bugs

2018-12-26	v2.0.9.2	1.Optimize and upgrade modules, increase information printing 2.The demo does not change the baud rate when connecting 3.Add parameters to the Position3D function to control whether the angle positioning is not implemented but externally 4.Add tracking function module 5.Dynamic application for transparent data buffer, external applications can send about 255 bytes of data 6.Complete the sdk debugging 7.Fix known bugs
2018-11-29	v2.0.9.0	1.Add uniview sdk support 2.Fix known bugs
2018-11-16	v2.0.8.3	1.Increase the support of Shanghai Jiguang SDK 2.Optimize demo, adjust the path of video and capture to the directory where exe is located 3.Optimize the demo and switch the 3D positioning of soft and hard modes through 3D configuration items
2018-11-12	v2.0.8.1	1.Turn off Dahua's log output 2.Increase the ABF function of Dahua equipment 3.Connect dvs2 channel to connect to turntable tracking video 4.Fix the operation problem of 3D positioning mode 5.Do not do sleep processing during PTZ control in transparent transmission mode 6.Turn on the transparent transmission mode to perform zoom and focus positioning without opening related queries 7.Video SDK add temperature measurement module (special for test equipment) 8. Increase the 3D positioning function of the software 9.Increase fixed target Zoom function by distance
2018-10-31	v2.0.7.5	1.Development of Dahua equipment support 2.Fix the problem that changing the baud rate is invalid 3.Add upgrade function interface 4.Remove the abf function from the demo 5.The upgrade function is added to the demo (no

		need to manually click the [restart] button during the upgrade)
2018-10-26	v2.0.7.4	1.Adjust the demo interface of mfc 2.Adjust the demo of c# to support pointer parameters 3.Add ABF interface function 4.64-bit library files
2018-10-24	v2.0.7.0	1.Adjust video sdkdemo to enrich the functions of demo 2.Add click and locate interface 3.Increase the interface for setting the baud rate 4.Add interface for auto focus control 5.Add callback interface for transparent data transmission 6.Add upgrade function module
2018-09-12	v2.0.6.5	1.Block the support of Dahua SDK
2018-08-22	v2.0.6.3	1.Fix the problem of opening angle query
2018-08-20	v2.0.6.1	1.Increase the function of positioning speed setting 2.Update the parsing algorithm of serial data, which can parse the alarm sent back from a single video server 3.Fix thread safety issues that may be caused by disconnection 4.Fix known bugs
2018-05-30	v2.0.5.1	1.Complete the functional debugging of 3D positioning
2018-05-16	v2.0.4.4	1.In transparent mode, the lens control increases the distinction between visible light and thermal imaging
2018-05-15	v2.0.4.3	1.The preset position increases the distinction between visible light and thermal imaging 2.optimization demo
2018-04-24	v2.0.4.2	1.Solve the problem of not analyzing angular data 2.Optimize the type definition of callback function, support C# docking
2018-04-18	v2.0.4.0	1.Increase zoom positioning, focus positioning related functions 2.Add the function of field of view angle analysis and positioning 3.Add drawing interface callback 4.Add interface callback for video stream 5.Fix the problem of thread leakage

2018-03-02	v2.0.3.0	1.Adjust the header file namespace 2.Adjust the output file name
2018-02-26	v2.0.2.1	1. Reconstruct the video sdk to provide a clear functional interface 2. Increase the support of Dahua Agreement 3.Improve the c-style interface
2018-01-02	v1.0.3.6	1.Optimize the function of presets, and call presets in a compatible transparent channel 2.Add function interface to delete all preset positions 3.Add cruise related interfaces 4.Add a functional interface for setting the dwell time of the preset position 5.Compatible with the function of manually obtaining the angle 6.Control the time interval of serial data sending, only angle query is 200 milliseconds 7.Adjust the delete processing of accumulated serial commands
2017-11-27	v1.0.2.7	1.Add the function of setting the transparent transmission address code, compatible with the PTZ control of the laser camera transparent transmission mode
2017-11-24	v1.0.2.6	1.Fix the crash that may be caused when the device is disconnected abnormally 2.Fix the problem of crash when calling angle positioning when the serial port is not open 3. Fix the problem of unhandled device connection abnormality under special circumstances 4.Open access video preview status
2017-08-12	v1.0.2.2	1.Add angle positioning interface
2017-07-12	v1.0.2.1	1. Increase the interface of PTZ preset position
2017-06-07	v1.0.2.0	1.Add video function interface 2.Increase the interface of detection status 3.Increase the interface to obtain the angle 4.Fix the bug that the alarm start signal cannot be received when the alarm is turned off and turned on again。
2017-05-02	v1.0.1.0	1.Add C format interface file, the interface currently only contains video, cloud control, and angle functions.
2017-04-28	v1.0.0.5	1.Fix the crash problem under win10 system

2017-04-21	v1.0.0.4	1.Increase the function of network exception handling 2.Increase subscription to events from the PTZ perspective 3.Increase the channel control of the serial port 4.Fix the crash caused by releasing the sdk。
2017-04-19	v1.0.0.2	1.Increase the alarm function 2.Increase picture capture function
2017-04-14	v1.0.0.1	1.Increase device connection function 2.Add video preview function 3.Add PTZ direction control function 4.Add PTZ camera control function

3. Call follow

3.1 SDK Main Process

3.1.1 Introduce "HPVideoSdk. h" and "hpws.video.lib" files into the project.

3.1.2 After introducing the Hpws: Video namespace, use CHPVideoSdk::Create to initialize a device pointer.

example

```
using namespace Hpws::Video;
int main(int argc, char * argv[])
{
IDevice* m_device = CHPVideoSdk::Create(HPDEVICE_HIK);
}
```

3.1.3 After the device pointer is initialized, connect the device through the Connect function.

example

```
{ m_device->Connect(ip, port, user, pwd); }
```

3.1.4 After the device is connected, the pointer of each module can be obtained through the member method under IDevice, and then different business operations can be completed by returning the pointer of each module.

example

```
{
    // Get the pointer of the alarm module
    IAlarm* alarm = m_device->GetAlarmPtr();
    // Turn on the alarm
    alarm->Open();
    ......

    // Get the pointer of the angle module
    IAngle* angle = m_device->GetAnglePtr();
    ......
}
```

3.1.5 After the device is used, close the connection.

example

```
{ m_device->DisConnect(); }
```

3.1.6 Release device pointer.

example

```
{
    CHPVideoSdk::Release(m_device);
    m_device = nullptr;
}
```

3.2 SDKAnti-droneCallProcess

3.2.1 The basic process is that the radar first detects the target, then reports the target data, analyzes the target data, adjusts the photoelectric angle positioning and automatic zoom, and finally performs target tracking.
3.2.2 Connect the device. The equipment tracking video takes the thermal imaging channel, so it needs to connect to the thermal imaging ip. The default ip is 192.168.1.65 and the port number is 8000.
3.2.3 Turn on tracking video.
if (m_device->Connect(ip, port, user, pwd))

```
{
        if (m_device->GetVideoPtr()->Open(m_HWND, 2))
                SetWindowText("Tracking video connected");
        else
                ::MessageBox(this->GetSafeHwnd(), "Failed to start video", "prompt",
MB_OK);
}
```

3.2.4 Angle monitor. Angle monitoring is mainly to obtain the status of the photoelectric equipment, and to grasp the horizontal angle, pitch angle and field of view information of the photoelectric in real time.

```
if (!m_device->GetAnglePtr()->IsOpen())
{
      if (m_device->GetAnglePtr()->Open()) //Turn on angle monitoring
      {
            ((CEdit*)this->GetDlgItem(IDC_BTN_ANGLE))->SetWindowText("Angle
query");
            ((CEdit*)this->GetDlgItem(IDC_BTN_OPENRIAL))->SetWindowText("Close
passthrough");
            m_device->GetAnglePtr()->RegisterEvent(this, OnAngleEvent,
OnViewEvent); //Register angular callback event
            GetDlgItem(IDC_STATIC_CURMODE)->ShowWindow(SW_SHOW);
      }
      else
      {
            ::MessageBox(this->GetSafeHwnd(), "Failed to open angle return",
"prompt", MB_OK);
      }
}
```

3.2.5 Radar photoelectric guidance. The radar photoelectric guidance is the longitude and latitude of the target given by the radar, and then combined with the longitude and latitude of the photoelectric installation to convert it into the target's horizontal pitch angle to the photoelectric.

Then perform horizontal pitch angle positioning to realize the business function of radar photoelectric guidance.

```
if (strlen(x) > 0)
{
        m_device->GetAnglePtr()->PositionX(atof(x), m_speedP);
        Sleep(150);             // Must sleep, sleep time can be about 50ms
}
```

```
if (strlen(y) > 0)
{
        m_device->GetAnglePtr()->PositionY(atof(y), m_speedP);
   }
```

3.2.6 Target lock tracking. When the target enters the video of the camera, you need to call the lock command to trigger the lock tracking. In the case of multiple targets, it will be based on the target pixel size and distance from the center of the video.

Select the optimal lock tracking and other related factors. If you need to unlock the lock tracking, you need to call it once again. In the lock tracking state, ordinary pan/tilt control is invalid.

```
if (m_device->IsConnect())
      m_device->GetTrackPtr()->Trigger();
else
      ::MessageBox(this->GetSafeHwnd(), "Please connect to the tracking video first","prompt", MB_OK);
```

4. Basic Interface Definition

4.1 Sdk main interface

```
#pragma once
#ifdef LINUX
#include "CrossPlatForm.h"
#define HPVIDEOSDK_API
#else
#ifdef HPVIDEOSDK_EXPORTS
#define HPVIDEOSDK_API __declspec(dllexport)
#else
#define HPVIDEOSDK_API __declspec(dllimport)
#endif

#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN                    // Exclude rarely used data from the Windows header
#endif

#include

#endif

#ifndef HP_VIDEO_TYPE
```

```
#define HP_VIDEO_TYPE
#define HP_light    1           /*!< Visible light */
#define HP_ird      0           /*!< Thermal Imaging */
#endif

#include "interface/IDevice.h"

namespace Hpws
{
        namespace Video
        {
                //---------------------------------------------------------------------------
                // Interface definition
                //---------------------------------------------------------------------------
                /**
                * @brief sdk Interface Entry
                */
                class HPVIDEOSDK_API CHPVideoSdk
                {
                public:
                        /**
                        * @brief Create device pointer
                        * @param type Equipment type
                        * @return device pointer
                        */
                        static IDevice* Create(HPDeviceType type =
HPDEVICE_HIK);

                        /**
                        * @brief Release device pointer
                        * @param device Device pointer
                        */
                        static void Release(IDevice *device);
                };
        }
}
```

4.1.1 Create device pointer

```
IDevice* Create(HPDeviceType type = HPDEVICE_HIK);
```

Parameters

type

[in] Initialization parameter type, different values ??correspond to different parameters, see the table below for details

type	Type value	meaning
HPDEVICE_HIK	0	Haikang Agreement
HPDEVICE_DH	1	Dahua Agreement
HPDEVICE_FV	2	Light Collection Agreement
HPDEVICE_UV	3	Univision Agreement
HPDEVICE_SH	4	Vision Agreement
HPDEVICE_ASTAR	5	Ep Agreement
HPDEVICE_HPWS	6	Hepu Agreement

Return Values

Return the device pointer.

example

```
IDevice *m_device;
m_device = NULL;
if (m_device == NULL)
{
	//Initialize device pointer
	m_device = CHPVideoSdk::Create(HPDEVICE_HIK);
}
```

4.1.2 Release device pointer

```
void Release(IDevice *device);
```

Parameters

IDevice

[in]Device pointer

example

```
if (m_device)
{
	//Release device pointer
	CHPVideoSdk::Release(m_device);
	m_device = NULL
}
```

4.2 Device interface

```
#pragma once
#include "IAlarm.h"
#include "IAngle.h"
#include "ICapture.h"
#include "ICruise.h"
//#include "IDevice.h"
#include "IPlayback.h"
#include "IPreset.h"
#include "IPtz.h"
#include "IRecord.h"
#include "ISerial.h"
#include "IVideo.h"
#include "IFocusInfo.h"
#include "IPower.h"
#include "ITemperaturen.h"
#include "IConfig.h"
#include "IVoice.h"
#include "IExten.h"
#include "INvr.h"
#include "ITrack.h"

#include "HpIrdSdk.h"
using namespace Hpws::IRD;
#ifndef LINUX
#pragma comment(lib, "Hpws.IrdSDK.lib")
#endif
//---------------------------------------------------------------------------
// Type definition
//---------------------------------------------------------------------------
#ifndef HP_DEVICE_TYPE_DEF
#define HP_DEVICE_TYPE_DEF
/**
* @brief Equipment type
*/
enum HPDeviceType
{
	HPDEVICE_HIK	= 0,	/*!< Haikang Agreement */
#ifndef LINUX
	HPDEVICE_DH		= 1,	/*!< Dahua Agreement */
	HPDEVICE_FV		= 2,	/*!< Light Collection Agreement */
	HPDEVICE_UV		= 3,	/*!< Univision Agreement */
```

```
#endif
        HPDEVICE_SH             = 4,    /*!< Vision Agreement */
#ifndef LINUX
        HPDEVICE_ASTAR  = 5,    /*!< Aipu Agreement */
#endif
        HPDEVICE_HPWS   = 6,    /*!< Hepu Agreement */
        HPDEVICE_COUNT
};
#endif

#ifndef HP_EXCEPT_EVENT_DEF
#define HP_EXCEPT_EVENT_DEF
/**
* @brief Exception information
*/
enum HPExcept
{
        HPEXCEPT_CONNECT,               /*!< Device connection abnormal */
        HPEXCEPT_PREVIEW,               /*!< Video preview abnormal */
        HPEXCEPT_SERIAL,                /*!< Alarm channel abnormal */
        HPEXCEPT_OTHER,                 /*!< Other abnormalities */
        HPEXCEPT_PING                   /*!< ping failed, actively disconnect */
};
/**
* @brief Abnormal event
*/
typedef void (__stdcall *HPEXCEPT_EVENT)(void *arg, char *ip, HPExcept ex);

#endif

//---------------------------------------------------------------------------
// Interface definition
//---------------------------------------------------------------------------
namespace Hpws
{
        namespace Video
        {
                /**
                * @brief Logs related to device abnormalities
                */
                typedef void (__stdcall *HPDEV_LOG_EVENT)(void *arg, char *ip,
DWORD code, char *msg);
                /**
                * @brief Logs related to equipment alarms
```

```
		* @param level Warning level: 1 failure; 2 serious warning; 3
video warning; 4 general warning;
		*/
		typedef void (__stdcall *HPDEV_ALARM_EVENT)(void *arg, char
*ip, HPDevAlarmType type, char *msg, int level);

		/**
		* @brief Device interface
		*/
		interface IDevice
		{
		public:
			/**
			* @brief Connect the device
			* @param ip IP address of the device
			* @param port Port number of the device
			* @param user Username of the device
			* @param pwd Device password
			* @return Success or failure
			*/
			virtual BOOL Connect(char *ip, int port, char *user, char
*pwd) = 0;
			/**
			* @brief Disconn
			* @return Success or failure
			*/
			virtual BOOL DisConnect(void) = 0;
			/**
			* @brief Is it connected
			* @return Is it connected
			*/
			virtual BOOL IsConnect(void) = 0;
			/**
			* @brief Register abnormal events
			* @param arg Callback event subscription object
			* @param event Callback event
			* @return no
			*/
			virtual void RegisterEvent(void *arg, HPEXCEPT_EVENT
event) = 0;

			/**
			* @brief Set the baud rate of the device
			* @param rate Baud rate
```

```
                * @return Success or failure
                * @note
rate???[1200,2400,4800,9600,19200,38400,57600,115200]
                * @note The serial port will be closed before setting the
baud rate
                */
                virtual BOOL SetBaudrate(int rate) = 0;

                /**
                * @brief Get device type
                * @return Equipment type
                */
                virtual HPDeviceType GetDeviceType(void) = 0;

                /**
                * @brief Register log events
                * @param logEvent Device exception log
                * @param alogEvent Device alarm log
                * @return no
                */
                virtual void RegisterLogEvent(void *arg,
HPDEV_LOG_EVENT logEvent, HPDEV_ALARM_EVENT alogEvent) = 0;

                /**
                * @brief Set original video size
                * @param width Video width
                * @param height Video height
                * @return no
                * @note This interface is mainly used for the calculation
of alarm coordinates related to the thermal imaging movement
                */
                virtual void SetVideoSrcSize(int width, int height) = 0;

        public:
                /**
                * @brief Get the pointer of the alarm module
                * @return Alarm module pointer
                */
                virtual IAlarm* GetAlarmPtr(void) const = 0 ;
                /**
                * @brief Get the angle module pointer
                * @return Angle module pointer
                */
                virtual IAngle* GetAnglePtr(void) const = 0;
```

```
/**
* @brief Get the pointer of the capture module
* @return Snapshot module pointer
*/
virtual ICapture* GetCapturePtr(void) const = 0;
/**
* @brief Get pointer of cruise module
* @return Cruise module pointer
*/
virtual ICruise* GetCruisePtr(void) const = 0;
/**
* @brief Get preset module pointer
* @return Preset module pointer
*/
virtual IPreset* GetPresetPtr(void) const = 0;
/**
* @brief Get PTZ module pointer
* @return PTZ module pointer
*/
virtual IPtz* GetPtzPtr(void) const = 0;
/**
* @brief Get the pointer of the recording module
* @return Video module pointer
*/
virtual IRecord* GetRecordPtr(void) const = 0;
/**
* @brief Get the serial port module pointer
* @return Serial port module pointer
*/
virtual ISerial* GetSerialPtr(void) const = 0;
/**
* @brief Get video module pointer
* @return Video module pointer
*/
virtual IVideo* GetVideoPtr(void) const = 0;
/**
* @brief Get the pointer of the video playback module
* @return Video playback module pointer
*/
virtual IPlayback* GetPlaybackPtr(void) const = 0;
/**
* @brief Get the focus module pointer
* @return Focus module pointer
*/
```

```
        virtual IFocusInfo* GetFocusPtr(void) const = 0;
        /**
        * @brief Get power control interface
        * @return Power control interface
        */
        virtual IPower* GetPowerPtr(void) const = 0;
        /**
        * @brief Get temperature measurement interface
        * @return Temperature measurement interface
        */
        virtual ITemperature* GetTemperaturePtr(void) const =
0;
        /**
        * @brief Get device configuration interface
        * @return Configure interface
        */
        virtual IConfig* GetConfigPtr(void) const = 0;
        /**
        * @brief Get voice intercom interface
        * @return Voice intercom interface
        */
        virtual IVoice* GetVoicePtr(void) const = 0;

        /**
        * @brief Get peripheral expansion interface
        * @return Peripheral expansion interface
        */
        virtual IExten* GetExtenPtr(void) const = 0;

        /**
        * @brief Get the NVR interface
        * @return NVR interface
        */
        virtual INvr* GetNvrPtr(void) const = 0;

        /**
        * @brief Get trace interface
        * @return Trace interface
        */
        virtual ITrack* GetTrackPtr(void) const = 0;

    public:
        /**
```

```
                                * @brief Get thermal image IPC protocol interface
module
                                * @return IPC protocol interface
                                * @note Only valid with the general agreement, do not
call the disconnection function in the IHpIRD interface
                                *           Users can also use Hpws.IrdSDK alone, and
connect to the device when used alone
                                */
                                virtual IHpIRD* GetHpIrdPtr(void) const = 0;
                        };
            }
}
```

4.2.1 Connect the device

```
BOOL Connect(char *ip, int port, char *user, char *pwd);
```

Parameters

ip

[in] IP address of the device

port

[in] Port number of the device

user

[in] Username of the device

pwd

[in] Device password

Return Values

TRUE means success, FALSE means failure.

Example

```
if (!m_device->IsConnect())
{
        if (m_device->Connect(szIP, port, szUser, szPwd))
        {
                //Successful connection, business operation
                ......
        }
}
```

4.2.2 Disconnect

BOOL DisConnect(void);

Return Values

TRUE means success, FALSE means failure.

Example

bool res = m_device->Disconnect();

4.2.3 IsConnect

BOOL IsConnect(void);

Return Values

TRUE means success, FALSE means failure.

Example

bool res = m_device->IsConnect();

4.2.4 Register Event

void RegisterEvent(void *arg, HPEXCEPT_EVENT event);

Parameters

arg
[in] Callback event subscription object value
event
[in] Callback event

4.2.5 Set Baudrate

BOOL SetBaudrate(int rate);

Parameters

rate
[in] SetBaudrate value in
[1200,2400,4800,9600,19200,38400,57600,115200]

Return Values

TRUE means success, FALSE means failure.

```
if (!m_device->IsConnect())
{
	if (m_device->SetBaudrate(4800))
	{
		// Set the baud rate successfully
		.....
	}
}
```

4.2.6 Register Log Event

void Register Log Event(void *arg, HPDEV_LOG_EVENT log Event, HPDEV_ALARM_EVENT alog Event);

Parameters

arg
	[in] Callback event subscription object
logEvent
	[in] Device exception log
alogEvent
	[in] Device alarm log

4.2.7 Set Video Src Size

This interface is mainly used to resolve the alarm coordinates related to the thermal imaging movement.

void SetVideoSrcSize(int width, int height);

Parameters

width
	[in] Video width
height
	[in] Video height

example

```
if (!m_device->IsConnect())
{
	m_device->SetVideoSrcSize(600, 400);
}
```

4.3 Video interfaces

```
#pragma once

//---------------------------------------------------------------------------
// Type definition
//---------------------------------------------------------------------------
#ifndef HP_DRAW_EVENT_DEF
#define HP_DRAW_EVENT_DEF
/**
* @brief Drawing event
*/
typedef void (__stdcall *HPDRAW_EVENT)(void *arg, char *ip, HDC hdc);
#endif

#ifndef HP_REALDATA_EVENT_DEF
#define HP_REALDATA_EVENT_DEF
/**
* @brief Live video streaming events
*/
typedef void (__stdcall *HPREALDATA_EVENT)(void *arg, char *ip, DWORD type,
BYTE *buff, DWORD size);
#endif

#ifndef HP_DECODEDATA_EVENT_DEF
#define HP_DECODEDATA_EVENT_DEF
typedef struct{
        long nWidth;
        long nHeight;
        long nStamp;
        long nType;
        long nFrameRate;
        DWORD dwFrameNum;
}DECODE_METADATA;
/**
* @brief Decode video stream event
*/
typedef void (__stdcall *HPDECODEDATA_EVENT)(void *arg, BYTE *buff, DWORD
size, DECODE_METADATA * pMetadata);
#endif

//---------------------------------------------------------------------------
// Interface definition
//---------------------------------------------------------------------------
```

```
namespace Hpws
{
	namespace Video
	{
		/**
		* @brief Video interface
		*/
		interface IVideo
		{
		public:
			/**
			* @brief Open video
			* @param hwnd Video display area
			* @param channel Channel number
			* @param streamType Code stream: 0 main code
stream, 1 code stream, 2 code stream (there is no such code stream for thermal
imaging)
			* @return Success or failure
			*/
			virtual BOOL Open(HWND hwnd, int channel = 1, int
streamType = 0) = 0;
			/**
			* @brief Close video
			* @return Success or failure
			*/
			virtual BOOL Close(void) = 0;
			/**
			* @brief Whether the video is on
			* @return Is the video being previewed
			*/
			virtual BOOL IsOpen(void) = 0;
			/**
			* @brief Set the number of video playback buffer
frames
			* @int Number of buffered frames
			*/
			virtual void SetDisplayBufNum(int nBuffer) = 0;
			/**
			* @brief Get the current bit rate
			* @return Current bit rate
			* @note Unit kb/s
			*/
			virtual float GetCurrentBitrate(void) = 0;
```

```
                        /**
                        * @brief Register drawing events
                        * @param arg Event subscription object
                        * @param event Drawing event
                        * @return no
                        * @note If it is a Hikvision device, this interface call will
only work after Open
                        */
                        virtual void RegisterDrawEvent(void *arg,
HPDRAW_EVENT event) = 0;

                        /**
                        * @brief Register for live video streaming events
                        * @param arg Event subscription object
                        * @param event Video streaming event
                        * @return no
                        */
                        virtual void RegisterRealEvent(void *arg,
HPREALDATA_EVENT event) = 0;

                        /**
                        * @brief Register the decoded video stream event
                        * @param arg Event subscription object
                        * @param event Decode video stream event
                        * @return no
                        */
                        virtual void RegisterDecodeEvent(void *arg,
HPDECODEDATA_EVENT event) = 0;
                    };
        }
}
```

4.3.1 Open Video

BOOL Open(HWND hwnd, int channel = 1, int streamType = 0);

Parameters

Hwnd

[in] Video display area

channel

[in] Channel number

streamType

[in] Code stream: 0 main code stream, 1 code stream, 2 code stream (there is no such code stream for thermal imaging)

lpInBuffer

[in] Status acquisition condition buffer

Return Values

TRUE means success, FALSE means failure.

Example

```
{
	if (video->Open(hwnd, 1, 0))
	{
		// Successfully opened the video
		.....
	}
}
```

4.3.2 Close

BOOL Close(void);

Return Values

TRUE means success, FALSE means failure.

Example

```
{
	if (video->Close())
	{
		// Close video successfully
		.....
	}
}
```

4.3.3 Is Open

BOOL Is Open(void);

Return Values

TRUE means that the video is being previewed, and FALSE means that the video is not being previewed.

Example

```
{
	if (video->IsOpen())
	{
		// Video on
		......
	}
	else
	{
		// Video is not open
		......
	}
}
```

4.3.4 Set Display BufNum

```
void SetDisplayBufNum(int nBuffer);
```

Parameters

nBuffer
	[in] Number of buffered frames

Example

```
{
	video->SetDisplayBufNum(nBuffer);
}
```

4.3.5 Get Current Bit rate

```
float GetCurrentBitrate(void);
```

Return Values

The current bit rate.

Example

```
{
	float rate = video->GetCurrentBitrate();
}
```

Note

The unit is kb/s.

4.3.6 Registe rDraw Event

void Registe rDraw Event(void *arg, HPDRAW_EVENT event);

Parameters

arg

[in] Event subscription object

event

[in] Drawing event

Note

If it is a Hikvision device, this interface call will take effect after Open.

4.3.7 Register Real Event

void Register Rea lEvent(void *arg, HPREALDATA_EVENT event);

Parameters

arg

[in] Event subscription object

event

[in] Video streaming event

4.3.8 Register Decode Event

void RegisterDecodeEvent(void *arg, HPDECODEDATA_EVENT event);

Parameters

arg

[in] Event subscription object

event

[in] Decode video stream event

4.4Video playback Interface

#pragma once

```
//---------------------------------------------------------------------------
// Type definition
//---------------------------------------------------------------------------
#ifndef HP_RECORD_DEF
#define HP_RECORD_DEF
/**
* @brief Video file definition
*/
typedef struct _HPRecord
{
	char fileName[100];			/*!< file name */
	char startTime[16];			/*!< Starting time */
	char endTime[16];		/*!< deadline */
	DWORD fileSize;				/*!< File size */
	_HPRecord()
	{
		memset(this, 0, sizeof(_HPRecord));
	}
}HPRecord, *PHPRecord;
#endif


#ifndef HP_PLAYCTRLTYPE_DEF
#define HP_PLAYCTRLTYPE_DEF
/**
* @brief Play control
*/
enum HPPlayType
{
	HP_PCT_PLAY = 0,		/*!< Play */
	HP_PCT_PAUSE,				/*!< time out */
	HP_PCT_STOP,				/*!< stop */
	HP_PCT_SLOW,				/*!< Slow */
	HP_PCT_FAST,				/*!< Fast */
	HP_PCT_NEXTFRAME,			/*!< Next frame */
	HP_PCT_PREFRAME,			/*!< Previous frame */
	HP_PCT_PLAYNORMAL			/*!< Call this command after single
frame playback to return to normal */
};
#endif

//---------------------------------------------------------------------------
// Interface definition
//---------------------------------------------------------------------------
```

```
namespace Hpws
{
	namespace Video
	{
		/**
		* @brief Remote playback interface
		*/
		interface IPlayback
		{
		public:
			/**
			* @brief Find a video
			* @param channel Device channel number
			* @param startTime Search start time
			* @param endTime Search deadline2
			* @param count Number of video files retrieved
			* @retval Pointer of video file collection
			* @note The format of startTime/endTime is: 20180101
			*/
			virtual PHPRecord SearchRecord(int channel, char
*startTime, char *endTime, int &count) = 0;
			/**
			* @brief Play video files
			* @param file Videofile
			* @param hwnd Show handle
			* @return Success or failure
			*/
			virtual BOOL Play(char *file, HWND hwnd, BOOL local =
FALSE) = 0;
			/**
			* @brief Play video by time
			* @param channel Channel number
			* @param startTime PlayStarting time
			* @param endTime Playdeadline
			* @param hwnd Show handle
			* @return Success or failure
			*/
			virtual BOOL Play(int channel, char *startTime, char
*endTime, HWND hwnd) = 0;
			/**
			* @brief Whether to open playback
			* @return Success or failure
			*/
			virtual BOOL IsPlay(void) = 0;
```

```
/**
* @brief Is it playing
* @return Is it playing
*/
virtual BOOL IsPlaying(void) = 0;
/**
* @brief Play control
* @param type Control type
* @return Success or failure
*/
virtual BOOL Control(HPPlayType type) = 0;
/**
* @brief Set playback progress
* @param pos Progress value
* @return Success or failure
*/
virtual BOOL SetPos(float pos) = 0;
/**
* @brief Get playback progress
* @return Current playback progress
*/
virtual float GetPos(void) = 0;

/**
* @brief Set playback progress by absolute time
* @param tm Absolute time
* @return Success or failure
*/
virtual BOOL SetPosByTime(long tm) = 0;

/**
* @brief Get current playing time
* @return Current playing time
*/
virtual int GetCurrentPlayTime(void) = 0;
/**
* @brief Get total time
* @return Total time of recording file
*/
virtual int GetTotalTime(void) = 0;
/**
* @brief Get the current frame number
* @return The number of frames currently playing
*/
```

```
                virtual int GetCurrentFrame(void) = 0;
                /**
                * @brief Get the total number of frames
                * @return Total number of frames of video files
                */
                virtual int GetTotalFrame(void) = 0;

                //------------------------------------------------------------------------
---
                // Video download
                //------------------------------------------------------------------------
---
                /**
                * @brief Download video by file name
                * @param file Video file name
                * @param saveFile Saved local file name
                * @return Success or failure
                */
                virtual BOOL DownloadRecord(char *file, char *saveFile)
= 0;
                /**
                * @brief Download video by time
                * @param file Video file name
                * @param saveFile Saved local file name
                * @return Success or failure
                */
                virtual BOOL DownloadRecord(int channel, char
*startTime, char *endTime, char *saveFile) = 0;
                /**
                * @brief Get download progress
                * @return Current download progress
                */
                virtual int GetDownloadProgress(void) = 0;
                /**
                * @brief Stop download
                * @return Success or failure
                */
                virtual BOOL StopDownload(void) = 0;
                /**
                * @brief Is downloading
                * @return Success or failure
                */
                virtual BOOL IsDownloading(void) = 0;
        };
```

```
        }
}
```

4.4.1 SDK Play back Interface Data Type Definition

Video file definition.

```
typedef struct _HPRecord
{
  char fileName[100];          /*!< file name */
  char startTime[16];          /*!< Starting time */
  char endTime[16];            /*!< deadline */
  DWORD fileSize;              /*!< File size */
  _HPRecord()
  {
    memset(this, 0, sizeof(_HPRecord));
  }
} HPRecord, *PHPRecord;
```

Play control.

```
enum HPPlayType
{
  HP_PCT_PLAY = 0,             /*!< play */
  HP_PCT_PAUSE,                        /*!< pause */
  HP_PCT_STOP,                 /*!< stop */
  HP_PCT_SLOW,                 /*!< slow */
  HP_PCT_FAST,                 /*!< fast */
  HP_PCT_NEXTFRAME,            /*!< next frame */
  HP_PCT_PREFRAME,             /*!< front frame */
  HP_PCT_PLAYNORMAL            /*!< Call this command after single frame playback to
return to normal */
};
```

4.4.2 Search Record

```
PHPRecord SearchRecord(int channel, char *startTime, char *endTime, int &count);
```

Parameters

channel

[in] Device channel number

startTime

[in] Search start time

endTime

[in] Search deadline

count

[in] Number of video files retrieved

Return Values

A pointer to the collection of video files.

Note

The format of startTime/endTime is: 20180101.

4.4.3 Play video files

BOOL Play(char *file, HWND hwnd, BOOL local = FALSE);

Parameters

file

[in] Video file

hwnd

[in] Show handle

Return Values

Success or failure.

4.4.4 Play video by time

BOOL Play(int channel, char *startTime, char *endTime, HWND hwnd);

Parameters

channel

[in] Channel number

startTime

[in] Play start time

endTime

[in] Play deadline

hwnd

[in] Show handle

Return Values IsPlay

BOOL IsPlay(void);

Return Values

Success or failure.

Example

```
{
        if (play->IsPlay())
        {
                // Start playing
                ......
        }
}
```

4.4.6 Is Playing

BOOL IsPlaying(void);

Return Values

TRUE And FALSE

Example

```
{
        if (play->IsPlaying())
        {
                // is playing
                ......
        }
}
```

4.4.7 Play control

BOOL Control(HPPlayType type);

Parameters

type
        [in] Control type

Return Values

Success or failure.

Example

```
{
        HPPlayType type;
        if (play->Control(type))
        {
                // Set successfully
                ......
        }
}
```

4.4.8 Set Pos

BOOL SetPos(float pos);

Parameters

pos
        [in] Progress value

Return Values

Success or failure.

Example

```
{
        if (play->SetPos(0))
        {
                // Set successfully
                ......
        }
}
```

4.4.9 Get Pos

float GetPos(void);

Parameters

pos
        [in] Progress value

Return Values

Current playback progress.

Example

```
{
        float speed = play->GetPos();
}
```

4.4.10 Set Pos By Time

BOOL Set Pos By Time (long tm);

Parameters

tm
        [in] Absolute time

Return Values

Success or failure.

4.4.11 Get Current Play Time

int GetCurrent Play Time(void);

Return Values

Current playing time.

Example

```
{
        int time = play->GetCurrentPlayTime();
}
```

4.4.12 Get Total Time

int GetTotalTime(void);

Return Values

The total time of the video file.

Example

```
{
        int time = play->GetTotalTime();
}
```

4.4.13 Get Current Frame

int Get Current Frame(void);

Return Values

The number of frames currently playing.

Example

```
{
        int frame = play->GetCurrentFrame();
}
```

4.4.14 Get Total Frame

int GetTotalFrame(void);

Return Values

The total number of frames of the video file.

Example

```
{
        int frame = play->GetTotalFrame();
}
```

4.4.15 Download Record By File Name

BOOL Down load Record (char *file, char *saveFile);

Parameters

file
        [in] Video file name
saveFile
        [in] Saved local file name

Return Values

Success or failure.

4.4.16 Download Record By Time

BOOL DownloadRecord(int channel, char *startTime, char *endTime, char *saveFile);

Parameters

file
	[in] Video file name
saveFile
	[in] Saved local file name

Return Values

Success or failure.

4.4.17 Get Down load Progress

int Get Down load Progress(void);

Return Values

The progress of the current download.

Example

```
{
	int progress = play->GetDownloadProgress();
}
```

4.4.18 Stop Download

BOOL Stop Download (void);

Return Values

Success or failure.

Example

```
{
	if (play->StopDownload())
	{
		// Stop downloading successfully
		.....
	}
}
```

4.4.19 Is Downloading

BOOL IsDownloading(void);

Return Values

Success or failure.

Example

```
{
	if (play->IsDownloading())
	{
		// downloading
		.....
	}
}
```

4.5 Record interface

```
#pragma once
namespace Hpws
{
	namespace Video
	{

		/**
		* @brief Video interface
		*/
		interface IRecord
		{
		public:
			/**
			* @brief Start record
			* @param file The full path name of the video file, the format is .mp4
			* @return Success or failure
			*/
			virtual BOOL Start(char *file) = 0;
			/**
			* @brief Close record
			* @return Success or failure
			*/
			virtual BOOL Stop(void) = 0;
			/**
			* @brief Is recording
			* @return Success or failure
			*/
```

```
                virtual BOOL IsRecording(void) = 0;
            };
        }
}
```

4.5.1 Start record

```
BOOL Start (char *file);
```

Parameters

file
    [in] The full path name of the video file, the format is .mp4

Return Values

TRUE means success, FALSE means failure.

4.5.2 Stop record

```
BOOL Stop(void);
```

Return Values

TRUE means success, FALSE means failure.

Example

```
{
        if (record ->Stop())
        {
                // Close the recording successfully
                ......
        }
}
```

4.5.3 Is Recording

```
BOOL IsRecording(void);
```

Return Values

TRUE means success, FALSE means failure.

Example

```
{
        if (record ->IsRecording())
        {
                // Recording
                ......
        }
        else
        {
                // Not recorded
                ......
        }
}
```

4.6 PTZ interface

```
#pragma once

//---------------------------------------------------------------------------
// Type definition
//---------------------------------------------------------------------------

#ifndef HP_VIDEO_TYPE
#define HP_VIDEO_TYPE
#define HP_light    1           /*!< light */
#define HP_ird      0           /*!< ird */
#endif

#ifndef HP_PTZ_TYPE_DEF
#define HP_PTZ_TYPE_DEF
/**
* @brief PTZ control type
*/
enum HPControlType
{
            HP_STATIC           = 0,      /*!< PTZ still */
            HP_UP                         = 1,      /*!< Head up */
            HP_DOWN                                 ,                       /*!< Gimbal */
            HP_LEFT                       ,                 /*!< Yuntai turn left */
            HP_RIGHT                      ,                 /*!< Yuntai turn right */
            HP_LEFT_UP                    ,                 /*!< Go up and turn left */
            HP_RIGHT_UP                   ,                 /*!< Go up and turn right */
            HP_LEFT_DOWN   ,                /*!< Pitch down and turn left */
            HP_RIGHT_DOWN ,                 /*!< PTZ down and turn right */
```

```
        HP_AUTO                                         /*!< The gimbal scans
around automatically at the speed of SS */
};
/**
* @brief Pan tilt lens control type
*/
enum HPCameraType
{
        HP_ZOOM_IN              = 1,    /*!< The focal length increases with
speed SS (magnification increases) */
        HP_ZOOM_OUT             ,               /*!< The focal length
becomes smaller at speed SS (magnification becomes smaller) */
        HP_FOCUS_NEAR  ,                /*!< Focus is adjusted forward at
speed SS */
        HP_FOCUS_FAR    ,               /*!< Focus is adjusted backward at
speed SS */
        HP_IRIS_OPEN    ,               /*!< Aperture expands at speed SS */
        HP_IRIS_CLOSE                   /*!< Aperture shrinks at speed SS */
};
#endif

//---------------------------------------------------------------------------
// Interface definition
//---------------------------------------------------------------------------
namespace Hpws
{
        namespace Video
        {
                /**
                * @brief PTZ interface
                */
                interface IPtz
                {
                public:
                        /**
                        * @brief Turntable control
                        * @param type Control type
                        * @param move Whether to rotate
                        * @param speed Value range: 1-63
                        * @param speedV Pitch speed: 1-63
                        * @return Success or failure
                        * @note The speedV parameter is valid in transparent
transmission mode
                        */
```

```
                virtual BOOL ControlMove(HPControlType type, BOOL
move, int speed = 50, int speedV = 50) = 0;
                /**
                * @brief Angular velocity turntable control
                * @param speed    A value of 0 stands for horizontal
stop; positive for right running and negative for left running
                * @param speedV A value of 0 stands for pitch stop;
positive for up running and negative for down running
                * @return Success or failure
                */
                virtual BOOL ControlMoveAngleSpeed(int speed, int
speedV) = 0;
                /**
                * @brief Lens control
                * @param type Control type
                * @param move Whether to rotate
                * @param video_type         Video type
                * - HP_light light
                * - HP_ird      ird
                * @return Success or failure
                * @note The light parameter is only valid after the
transparent transmission is turned on
                */
                virtual BOOL CameraMove(HPCameraType type, BOOL
move, int video_type = HP_ird) = 0;

                /**
                * @brief Hardware 3D positioning
                * @param video_type
                * - HP_light light
                * - HP_ird ird
                * @param video Video display area
                * @param select select area
                * @large
                * - TRUE enlarge
                * - FALSE Shrink
                * @return Success or failure
                */
                virtual BOOL Position3D(int video_type, RECT video,
RECT select, BOOL large) = 0;
                /**
                * @brief 3D positioning software implementation
```

```
                @param bPositionXY The internal realization is divided
into two parts: firstly locate the angle to the center point of the frame selection area,
and then locate the field of view angle
                        This variable controls whether to skip the angle
positioning step
                */
                virtual BOOL Position3D(int video_type, RECT video,
RECT select, BOOL large, float rate, BOOL bPositionXY = TRUE) = 0;

                /**
                * @brief Enable or disable the ABF function
                * @param enable Whether to enable
                * @return Success or failure
                */
                virtual BOOL CtlABF(BOOL enable) = 0;
                /**
                * @brief Determine whether the ABF function is enabled
                * @return Whether to enable
                */
                virtual BOOL IsEnableABF(void) = 0;
                /**
                * @brief Trigger ABF function
                * @return Success or failure
                */
                virtual BOOL FocusOnePush(void) = 0;


                /**
                * @brief Turn on the auxiliary switch
                * @param Which switch
                * @param video_type        Video type
                * - HP_light light
                * - HP_ird      ird
                * @return Success or failure
                * @note The video_type parameter is valid only when
transparent transmission is enabled
                * @note Turn on auxiliary switch 8: trigger/cancel
automatic tracking
                */
                virtual BOOL OpenAid(int preset, int video_type =
HP_ird) = 0;
                /**
                * @brief Turn off the auxiliary switch
                * @param Which switch
```

```
                                * @param video_type         Video type
                                * - HP_light light
                                * - HP_ird      ird
                                * @return Success or failure
                                * @note The video_type parameter is valid only when
transparent transmission is enabled
                                */
                                virtual BOOL CloseAid(int preset, int video_type =
HP_ird) = 0;

                                /**
                                * @brief Enable image stabilization
                                * @param orientation 0:horizontal 1:pitch
                                * @param bEnable Open or not
                                * @return Success or failure
                                */
                                virtual BOOL EnableImgStabilizer(int orientation, BOOL
bEnable) = 0;

                                /**
                                * @brief Setting image stabilization parameters
                                * @param orientation 0:horizontal 1:pitch
                                * @param value Zero bias correction -32767--32767
                                * @return Success or failure
                                */
                                virtual BOOL SetImgStabilizerPara(int orientation, int
value) = 0;
                        };
                }
}
```

4.6.1 PTZ interface data type definition

PTZ control type.

```
enum HPControlType
{
  HP_STATIC         = 0,    /*!< PTZ still */
  HP_UP             = 1,        /*!< Head up */
  HP_DOWN           ,           /*!< Gimbal */
  HP_LEFT           ,           /*!< Ptz turn left */
  HP_RIGHT          ,           /*!< Ptz turn right */
  HP_LEFT_UP        ,           /*!< Go up and turn left */
  HP_RIGHT_UP       ,           /*!< Go up and turn right */
  HP_LEFT_DOWN      ,           /*!< PTZ down and turn left */
```

```
 HP_RIGHT_DOWN          ,          /*!< PTZ down and turn right */
  HP_AUTO          /*!< The gimbal scans around automatically at the speed of SS */
};
```

PTZ camera control type.

```
enum HPCameraType
{
  HP_ZOOM_IN     = 1,        /*!< The focal length becomes larger at speed SS
(magnification becomes larger) */
  HP_ZOOM_OUT  ,          /*!< The focal length becomes smaller at the speed SS
(the magnification becomes smaller) */
  HP_FOCUS_NEAR          ,          /*!< Focus is adjusted forward at speed SS */
  HP_FOCUS_FAR  ,          /*!< Focus is adjusted backward at speed SS */
  HP_IRIS_OPEN    ,          /*!< Aperture expands at speed SS */
  HP_IRIS_CLOSE   /*!< Aperture shrinks at speed SS */
};
```

4.6.2 Control Move

```
BOOL ControlMove(HPControlType type, BOOL move, int speed = 50, int speedV =
50);
```

Parameters

type
: [in] Control type

move
: [in] Whether to rotate

speed
: [in] Ranges：1-63

speedV
: [in] Pitch speed：1-63

Return Values

TRUE means success, FALSE means failure.

Note

The speedV parameter is valid in the transparent transmission mode.

4.6.3 Control Move Angle Speed

BOOL ControlMoveAngleSpeed(int speed, int speedV);

Parameters

speed

[in] A value of 0 means horizontal stop; positive means right running, negative means left running

speedV

[in] A value of 0 means pitch stop; positive means up running, negative means down running

Return Values

TRUE means success, FALSE means failure.

4.6.4 Lens control

BOOL ControlMove(HPControlType type, BOOL move, int speed = 50, int speedV = 50);

Parameters

type

[in] Control type

move

[in] Whether to rotate

video_type

[in] Video type HP_light Visible light HP_ird Thermal Imaging

Return Values

TRUE means success, FALSE means failure.

Note

The light parameter is valid only after the transparent transmission is turned on.

4.6.5 Hardware 3D positioning

BOOL Position3D(int video_type, RECT video, RECT select, BOOL large);

Parameters

video

[in] Video display area

select
	[in] select area
large
	[in] TRUE enlarge FALSE Shrink
video_type
	[in] Video type HP_light Visible light HP_ird Thermal Imaging

Return Values

TRUE means success, FALSE means failure.

4.6.6 Software 3D positioning

BOOL Position3D(int video_type, RECT video, RECT select, BOOL large, float rate, BOOL bPositionXY = TRUE);

Parameters

video
	[in] Video display area
select
	[in] select area
large
	[in] TRUE enlarge FALSE Shrink
video_type
	[in] Video type HP_light Visible light HP_ird Thermal Imaging
bPositionXY

	[in] The internal implementation is divided into two parts, first angularly locate the center point of the frame selection area After the field of view angle positioning. This variable controls whether to skip the angle positioning step

Return Values

TRUE means success, FALSE means failure.

4.6.7 Enable or disable the ABF function

BOOL CtlABF(BOOL enable);

Parameters

enable
	[in] Whether to enable

Return Values

TRUE means success, FALSE means failure.

4.6.8 The ABF function is enabled

BOOL IsEnableABF(void);

Return Values

TRUE means open, FALSE means not open.

Example

```
{
        if (ptz ->IsEnableABF())
        {
                // ABF function is enabled
                ......
        }
        else
        {
                // ABF function is not enabled
                ......
        }
}
```

4.6.9 Trigger ABF function

BOOL FocusOnePush(void);

Return Values

TRUE means opening is successful, FALSE means opening failed.

Example

```
{
        if (ptz ->FocusOnePush())
        {
                // Open ABF function successfully
                ......
        }
}
```

4.6.10 Turn off the auxiliary switch

BOOL CloseAid(int preset, int video_type = HP_ird);

Parameters

preset
[in] Which switch
video_type
[in] Video type HP_light Visible light HP_ird Thermal Imaging

Return Values

TRUE means success, FALSE means failure.

Note

The video_type parameter is valid only when transparent transmission is enabled.

4.6.11 Turn on the auxiliary switch

BOOL OpenAid(int preset, int video_type = HP_ird);

Parameters

preset
[in] Which switch
video_type
[in] Video type HP_light Visible light HP_ird Thermal Imaging

Return Values

TRUE means success, FALSE means failure.

Note

The video_type parameter is valid only after the transparent transmission is turned on; turn on the auxiliary switch 8: trigger/cancel automatic tracking.

4.6.12 Enable ImgStabilizer

BOOL EnableImgStabilizer(int orientation, BOOL bEnable);

Parameters

orientation
[in] 0:Horizontal 1:Pitch
bEnable
[in] Whether to open

Return Values

TRUE means success, FALSE means failure.

4.6.13 Set ImgStabilizer Para

BOOL SetImgStabilizerPara(int orientation, int value);

Parameters

orientation
[in] 0:Horizontal 1:Pitch
value
[in] Zero offset correction value -32767--32767

Return Values

TRUE means success, FALSE means failure.

4.7 Alarm interface

```
#pragma once
#include "DataDefine.h"

namespace Hpws
{
	namespace Video
	{
		//---------------------------------------------------------------------------
		// Type definition
		//---------------------------------------------------------------------------
#ifndef HP_ALARM_EVENT_DEF
#define HP_ALARM_EVENT_DEF
		/**
		* @brief Alarm content definition
		*/
		typedef struct _HPAlarmInfo
		{
			float	angleX;		/*!< Horizontal angle */
			float	angleY;		/*!< Pitch angle */
			RECT	*rect;		/*!< Set pointer of alarm
area, business end will calculate according to original resolution */
```

```
                int             rcSize;         /*!< Number of
alarm areas */
        }HPAlarmInfo, *PHPAlarmInfo;

        /**
        * @brief Alarm event
        * @note Alarm on bAlarm:true, info:null
        * @note Alarm content bAlarm:true, info:!null
        * @note End of alarm bAlarm:false
        */
        typedef void (__stdcall *HPALARM_EVENT)(void *arg, char *ip,
BOOL bAlarm, PHPAlarmInfo info);

#endif

#ifndef HP_ALARM_V2_EVENT_DEF
#define HP_ALARM_V2_EVENT_DEF

        /**
        * @brief alarm type
        */
        enum HPDevAlarmType
        {
                HPDEV_ALARM_GPIO = 1,
        //IO input alarm                level:4
                HPDEV_ALARM_HUMAN_TEMP,
        //Human body temperature measurement alarm
        level:4
                HPDEV_ALARM_HEAT_TEMP,
        //Thermal warning                                       level:4
                HPDEV_ALARM_SD_FULL,
        //SD card full alarm                    level:2
                HPDEV_ALARM_SD_ERR,
        //SD card error alarm                   level:2
                HPDEV_ALARM_SD_NOSPACE,
        //SD card insufficient space alarm      level:2
                HPDEV_ALARM_VIDEO_LOST,
        //Video loss alarm              level:2
                HPDEV_ALARM_VIDEO_OCC,
        //Video block alarm                     level:2
                HPDEV_ALARM_IRDCORE_DIS,
        //The serial port of the thermal imaging movement is disconnected
        level:1
```

```
		HPDEV_ALARM_STALL_X,
//Horizontal motor locked		level:1
		HPDEV_ALARM_STALL_Y,
//Pitch motor blocked			level:1
		HPDEV_ALARM_ZOOM_LIGHT_FAULT,
//Visible light zoom failure		level:1
		HPDEV_ALARM_FOCUS_LIGHT_FAULT,
//Visible light focus failure		level:1
		HPDEV_ALARM_ZOOM_IRD_FAULT,
//Thermal imaging zoom failure		level:1
		HPDEV_ALARM_FOCUS_IRD_FAULT,
//Thermal imaging focus failure		level:1

		HPDEV_ALARM_MOVE = 100,
//Motion detection alarm		level:3
		HPDEV_ALARM_AREA,
//Area intrusion alarm			level:3
		HPDEV_ALARM_AREA_IN,
//Enter intrusion alarm			level:3
		HPDEV_ALARM_AREA_OUT,
//Exit zone alarm			level:3
		HPDEV_ALARM_LINE_IN,
//Mixing line intrusion alarm				level:3
		HPDEV_ALARM_TRACKING,
//Tracking target alarm			level:3
		HPDEV_ALARM_LINGER,
//Wandering alarm			level:3
		HPDEV_ALARM_GATER,
//Crowd gathering alarm			level:3
		HPDEV_ALARM_LEFT,
//Item left alarm			level:3
		HPDEV_ALARM_TAKE,
//Item pick up alarm			level:3
		HPDEV_ALARM_PARKING,
//Parking detection alarm		level:3
		HPDEV_ALARM_RUN
	//Fast motion alarm				level:3
	};

	typedef struct _HPAlarmTarget
	{
		int id;
		Generic::RECTD rect;
	}HPAlarmTarget;
```

```
		/**
		* @brief Alarm information
		*/
		typedef struct _HPAlarmInfo_V2
		{
			HPDevAlarmType  type;			/*!< Type of Police
*/
			HPAlarmTarget*    targets;		/*!< The set
pointers x, y, w, h of the alarm area are scale factors, and the scale can be multiplied
by the display size,

		If type is human body temperature measurement, the integer part of
w/100 is the highest temperature in the area */
			int					count;
	/*!< Number of alarm areas */

			_HPAlarmInfo_V2()
			{
				memset(this, 0, sizeof(_HPAlarmInfo_V2));
			}
			~_HPAlarmInfo_V2()
			{
				if(this->targets != NULL)
				{
					delete []this->targets;
					this->targets = NULL;
				}
			}
		}HPAlarmInfo_V2, *PHPAlarmInfo_V2;
		typedef void (__stdcall *HPALARM_V2_EVENT)(void *arg, char
*ip, PHPAlarmInfo_V2 info);

		/**
		* @brief Alarm information
		* @note Based on HPAlarmInfo_V2 structure information
encapsulated into a string for return: x, y, w, h; x, y, w, h;
		*/
		typedef void (__stdcall *HPALARM_V2_STR_EVENT)(void *arg,
char *ip, HPDevAlarmType type, const char *str);
#endif

		//---------------------------------------------------------------------
		// Interface definition
```

```
//---------------------------------------------------------------------------
/**
* @brief Alarm interface
*/
interface IAlarm
{
public:
	/**
	* @brief Enable alarm monitoring
	* @return Success or failure
	*/
	virtual BOOL Open(void) = 0;
	/**
	* @brief Turn off alarm monitoring
	* @return Success or failure
	*/
	virtual BOOL Close(void) = 0;
	/**
	* @brief Whether the alarm monitoring is turned on
	* @return Whether alarm monitoring is turned on
	*/
	virtual BOOL IsOpen(void) = 0;
	/**
	* @brief Are you calling the police
	* @return Are you calling the police
	*/
	virtual    BOOL IsAlarm(void) = 0;
	/**
	* @brief Register an alarm event
	* @param arg Callback event subscription object
	* @param event Callback event
	* @return no
	*/
	virtual void RegisterEvent(void *arg, HPALARM_EVENT
event) = 0;

	/**
	* @brief Enable intelligent analysis alarm monitoring
	* @return Success or failure
	*/
	virtual BOOL StartListenSmart(void) = 0;
	/**
	* @brief Turn off intelligent analysis alarm monitoring
	* @return Success or failure
```

```
                                */
                                virtual BOOL StopListenSmart(void) = 0;
                                /**
                                * @brief Set hotspot alarm threshold
                                * @param value Hotspot alarm threshold 1-255
                                * @return Success or failure
                                */
                                virtual BOOL SetHotThreshold(int value) = 0;

                                /**
                                * @brief Register an alarm event
                                * @param arg Callback event subscription object
                                * @param event Callback event
                                * @param strEvent Callback event (string format)
                                * @return no
                                */
                                virtual void RegisterAlarmEvent(void *arg,
HPALARM_V2_EVENT event, HPALARM_V2_STR_EVENT strEvent = NULL) = 0;
                        };
            }
}
```

4.7.1 SDK Alarm Interface

Definition of alarm content.

```
typedef struct _HPAlarmInfo
   {
     float angleX;    // Horizontal angle
     float angleY;              // Pitch angle
     RECT *rect;                // Set pointer of alarm area, business end will calculate
according to original resolution
     int  rcSize;               // Number of alarm areas 量
   }HPAlarmInfo, *PHPAlarmInfo;
```

alarm type.

```
enum HPDevAlarmType
   {
     HPDEV_ALARM_GPIO = 1,          //IO input alarm    level:4
     HPDEV_ALARM_HUMAN_TEMP,              //Human body temperature
measurement alarm          level:4
     HPDEV_ALARM_HEAT_TEMP,      //Thermal warning            level:4
```

```
  HPDEV_ALARM_SD_FULL,                    //SD card full alarm          level:2
    HPDEV_ALARM_SD_ERR,                   //SD card error alarm         level:2
    HPDEV_ALARM_SD_NOSPACE,    //SD card insufficient space alarm     level:2
    HPDEV_ALARM_VIDEO_LOST,    //Video loss alarm            level:2
    HPDEV_ALARM_VIDEO_OCC,     //Video block alarm           level:2
    HPDEV_ALARM_IRDCORE_DIS,   //The serial port of the thermal imaging
movement is disconnected  level:1
    HPDEV_ALARM_STALL_X,                  //Horizontal motor locked  level:1
    HPDEV_ALARM_STALL_Y,                  //Pitch motor blocked      level:1
    HPDEV_ALARM_ZOOM_LIGHT_FAULT,         //Visible light zoom failure  level:1
    HPDEV_ALARM_FOCUS_LIGHT_FAULT,        //Visible light focus failure  level:1
    HPDEV_ALARM_ZOOM_IRD_FAULT,           //Thermal imaging zoom failure
        level:1
    HPDEV_ALARM_FOCUS_IRD_FAULT,          //Thermal imaging focus failure
        level:1
    HPDEV_ALARM_MOVE = 100,    //Motion detection alarm   level:3
    HPDEV_ALARM_AREA,                     //Area intrusion alarm        level:3
    HPDEV_ALARM_AREA_IN,                  //Enter intrusion alarm       level:3
    HPDEV_ALARM_AREA_OUT,      //Exit zone alarm             level:3
    HPDEV_ALARM_LINE_IN,                  //Mixing line intrusion alarm
        level:3
    HPDEV_ALARM_TRACKING,      //Tracking target alarm       level:3
    HPDEV_ALARM_LINGER,                   //Wandering alarm             level:3
    HPDEV_ALARM_GATER,                    //Crowd gathering alarm       level:3
    HPDEV_ALARM_LEFT,                     //Item left alarm   level:3
    HPDEV_ALARM_TAKE,                     //Item pick up alarm          level:3
    HPDEV_ALARM_PARKING,                  //Parking detection alarm     level:3
    HPDEV_ALARM_RUN                       //Fast motion alarm           level:3
  };
```

4.7.2 Enable alarm monitoring

BOOL Open(void);

Return Values

TRUE means opening is successful, FALSE means opening failed.

example

```
{
        if (alarm->Open())
        {
                // Turn on alarm monitoring successfully
        }
```

}

4.7.3 Turn off alarm monitoring

BOOL Close(void);

Return Values

TRUE indicates that the shutdown is successful, and FALSE indicates that the shutdown failed.

example

```
{
	if (alarm->Close())
	{
		// Turn off the alarm monitor successfully
	}
}
```

4.7.4 Whether the alarm monitoring is on

BOOL IsOpen(void);

Return Values

TRUE means that the alarm monitoring is turned on, and FALSE means that the alarm monitoring is not turned on.

example

```
{
	if (alarm->IsOpen())
	{
		// Alarm monitoring is turned on
	}
}
```

4.7.5 Is Alarm

BOOL IsAlarm(void);

Return Values

TRUE means the alarm is in progress, FALSE means the alarm is not.

example

```
{
        if (alarm->IsAlarm())
        {
                // Alarming
        }
}
```

4.7.6 Register Event

void RegisterEvent(void *arg, HPALARM_EVENT event);

Parameters

arg
[in] Callback event subscription object value
event
[in] Callback event

4.7.7 Start Listen Smart

BOOL Start Listen Smart(void);

Return Values

TRUE means opening is successful, FALSE means opening failed.

example

```
{
        if (alarm->StartListenSmart())
        {
                // Smart analysis alarm monitoring is successful
        }
}
```

4.7.8 Stop Listen Smart

BOOL Stop Listen Smart(void);

Return Values

TRUE indicates that the shutdown is successful, and FALSE indicates that the shutdown failed.

example

```
{
        if (alarm->StopListenSmart())
        {
                // Turn off intelligent analysis alarm monitoring success
        }
}
```

4.7.9 Set Hot Threshold

BOOL SetHotThreshold(int value);

Parameters

value
[in] Hotspot alarm threshold 1-255

Return Values

TRUE means the setting is successful, FALSE means the setting failed.

4.7.10 RegisterAlarmEvent

void RegisterAlarmEvent(void *arg, HPALARM_V2_EVENT event, HPALARM_V2_STR_EVENT strEvent = NULL);

Parameters

arg
[in] Callback event subscription object value
event
[in] Callback event
strEvent
[in] Callback event (string format)

Return Values

TRUE means success, FALSE means failure.

4.8 Capture interface:

#pragma once

```
namespace Hpws
{
	namespace Video
	{

		//--------------------------------------------------------------------------
		// Interface definition
		//--------------------------------------------------------------------------
		/**
		* @brief Snapshot interface
		*/
		interface ICapture
		{
		public:
			/**
			* @brief Snap a video image
			* @param file The full path name of the image storage
			* @param inVideo Whether based on video
			* @note Currently only supports pictures in jpeg format
			* @channel Capture channel, inVideo = FALS valid
			* @return Success or failure
			*/
			virtual BOOL Capture(char *file, BOOL inVideo = FALSE,
int channel = 1) = 0;

			/**
			* @brief Snap a video image
			* @param buff Currently only supports pictures in jpeg
format
			* @param size Buffer size
			* @retval >0 Represents the size of the image
			* @retval 0 fail
			* @note buff is the memory address of external new
			*/
			virtual DWORD CaptureEx(char *buff, DWORD size) = 0;
		};
	}
}
```

4.8.1 Snap a video image

```
BOOL Capture(char *file, BOOL inVideo = FALSE, int channel = 1);
```

Parameters

file
[in] The full path name of the image storage
inVideo
[in] Whether based on video
channel
[in] Capture channel, inVideo = FALS valid

Return Values

TRUE means success, FALSE means failure.

Note

Currently only supports pictures in jpeg format.

4.8.2 Snap a video image

DWORD CaptureEx(char *buff, DWORD size);

Parameters

buff
[in] Image data buffer
size
[in] Buffer size

Return Values

> 0 Represents the size of the image, = 0 means failure.

Note

buff is the memory address of external new.

4.9 Angle interface

```
#pragma once
//----------------------------------------------------------------------------
// Type definition
//----------------------------------------------------------------------------

#ifndef HP_VIDEO_TYPE
#define HP_VIDEO_TYPE
#define HP_light    1           /*!< light */
#define HP_ird      0           /*!< ird */
```

```
#endif

#ifndef HP_ANGLE_EVENT_DEF
#define HP_ANGLE_EVENT_DEF
/**
* @brief Angular event
*/
typedef void (__stdcall *HPANGLE_EVENT)(void *arg, char *ip, float angleX, float
angleY);
/**
* @brief Field of view event
* @param video_type         Video type
* - HP_light light
* - HP_ird      ird
*/
typedef void (__stdcall *HPVIEW_EVENT)(void *arg, char *ip, float angleZ, int
video_type);
#endif

//--------------------------------------------------------------------------
// Interface definition
//--------------------------------------------------------------------------
namespace Hpws
{
        namespace Video
        {
                /**
                * @biref Float size
                */
                typedef struct _SIZED
                {
                        double    width;              /*!< width */
                        double    height;             /*!< height */

                        _SIZED(void)
                        {
                                this->width = 0;
                                this->height = 0;
                        }

                        _SIZED(double width, double height)
                        {
                                this->width = width;
                                this->height = height;
```

```
        }
}SIZED, *PSIZED;

/**
* @brief Angle interface
*/
interface IAngle
{
public:
        /**
        * @brief Turn on angle monitoring
        * @return Success or failure
        */
        virtual BOOL Open(void) = 0;
        /**
        * @brief Turn off angle monitoring
        * @return Success or failure
        */
        virtual BOOL Close(void) = 0;
        /**
        * @brief Whether angle monitoring is turned on
        * @return Whether angle monitoring is turned on
        */
        virtual BOOL IsOpen(void) = 0;

        /**
        * @brief Query horizontal angle
        * @return Success or failure
        * @note Query only once
        */
        virtual BOOL SearchX(void) = 0;
        /**
        * @brief QueryPitch angle
        * @return Success or failure
        * @note Query only once
        */
        virtual BOOL SearchY(void) = 0;
        /**
        * @brief Query field of view angle
        * @param video_type         Video type
        * - HP_light light
        * - HP_ird      ird
        * @return Success or failure
        * @note Query only once
```

```
                        */
                        virtual BOOL SearchView(int video_type) = 0;

                        /**
                        * @brief Set angular positioning speed
                        * @param speed Angle positioning speed, Speed range:
0 or 4-28
                        * @note When speed is equal to 0, reach the set angle
at the fastest speed. The larger the other values, the faster the speed
                        */
                        virtual BOOL SetSpeed(int speed) = 0;

                        /**
                        * @brief Horizontal angle positioning
                        * @param angle Horizontal angle
                        * @param speed Angle positioning speed, Speed range:
0 or 4-28
                        * @return Success or failure
                        * @note PositionX and PositionY directly need to sleep
for 150ms
                        */
                        virtual BOOL PositionX(double angle, int speed = 0) = 0;
                        /**
                        * @brief Pitch angle positioning
                        * @param angle Pitch angle
                        * @param speed Angle positioning speed, Speed range:
0 or 4-28
                        * @return Success or failure
                        * @note PositionX and PositionY directly need to sleep
for 150ms
                        */
                        virtual BOOL PositionY(double angle, int speed = 0) = 0;
                        /**
                        * @brief Field of view angle positioning
                        * @param angle Field of view angle
                        * @param video_type         Video type
                        * - HP_light light
                        * - HP_ird      ird
                        * @return Success or failure
                        */
                        virtual BOOL PositionZ(double angle, int video_type =
HP_ird) = 0;
                        /**
                        * @brief Click to locate
```

```
                                * @param video Video display area
                                * @param point coordinate
                                * @param video_type         Video type
                                * - HP_light light
                                * - HP_ird      ird
                                * @param rate Aspect ratio coefficient, the default is 3:4
                                * @return Success or failure
                                * @note Need to support the field of view angle
feedback function, if there is no angle feedback function, config.ini can be configured
                                *    [VIDEO_SDK]
                                *    LIGHT_VIEW=6.28
                                *    IRD_VIEW=6.28
                                *    EDGE_LEFT=32
                                *    EDGE_TOP=28
                                *    EDGE_RIGHT=39
                                *    EDGE_BOTTOM=31
                                */
                                virtual BOOL PositionPoint(RECT video, POINT point, int
video_type = HP_ird, FLOAT rate = 0.75) = 0;

                                /**
                                * @brief xyzAngle pos
                                * @param x Horizontal angle
                                * @param y Pitch angle
                                * @param z Field of view
                                * @return Success or failure
                                * @note Device angle xyz angle positioning, currently
only supports Univision dome camera
                                */
                                virtual BOOL PositionXYZ(double x, double y, double z) =
0;
                                /**
                                * @brief Automatic zoom according to distance
                                * @param dis Distance in m
                                * @param size Target size, unit m
                                * @param video_type         Video type
                                * - HP_light light
                                * - HP_ird      ird
                                * @param rate The proportion of the enlarged image
                                * @return Success or failure
                                */
                                virtual BOOL Zoom(int video_type, double dis, SIZED
&size, float rate = 0.2) = 0;
```

```
/**
* @brief Get the azimuth angle of a point in the video
* @param x The returned horizontal azimuth
* @param y Return pitch azimuth
* @param video_type         Video type
* - HP_light light
* - HP_ird      ird
*/
virtual void GetPointAngle(RECT video, POINT point,
double &x, double &y, int video_type = HP_ird, FLOAT rate = 0.75) = 0;

/**
* @brief Get the azimuth angle of a point in the video
* @param x The returned horizontal azimuth
* @param y Return pitch azimuth
* @param video_type         Video type
* - HP_light light
* - HP_ird      ird
*/
virtual void GetPointAngle(RECT video, POINT point,
double cameraX, double cameraY, double cameraZ, double &x, double &y, int
video_type = HP_ird, FLOAT rate = 0.75) = 0;
/**
* @brief Get the horizontal angle
* @return Horizontal angle
*/
virtual float GetAngleX(void) = 0;
/**
* @brief Get pitch angle
* @return Pitch angle
*/
virtual float GetAngleY(void) = 0;
/**
* @brief Get the angle of view
* @param video_type         Video type
* - HP_light light
* - HP_ird      ird
* @return Field of view angle
*/
virtual float GetAngleZ(int video_type = HP_ird) = 0;
/**
* @brief Register for angular events
* @param arg Callback event subscription object
```

```
                                * @param event Callback event
                                * @param view Field of view event
                                * @return no
                                */
                                virtual void RegisterEvent(void *arg, HPANGLE_EVENT
event, HPVIEW_EVENT view) = 0;

                                /**
                                * @brief Set a fixed angle of view
                                * @param viewIRD Thermal image fixed field of view
<0.1 invalid, can transmit 0 when not needed
                                * @param viewLight Visible light fixed field of view <0.1
is invalid, and can be transmitted when not needed
                                */
                                virtual void SetFixedView(float viewIRD, float viewLight)
= 0;
                        };
            }
}
```

4.9.1 Turn on angle monitoring

BOOL Open(void);

Return Values

TRUE means opening is successful, FALSE means opening failed.

example

```
{
        if (angle->Open())
        {
                // Turn on angle monitoring successfully
        }
}
```

4.9.2 Turn off angle monitoring

BOOL Close(void);

Return Values

TRUE means closing the angle is successful, FALSE means closing the angle failed.

example

```
{
	if (angle->Close())
	{
		// Close the angle monitor successfully
	}
}
```

4.9.3 Whether angle monitoring is turned on

BOOL IsOpen(void);

Return Values

TRUE means opening is successful, FALSE means opening failed.

example

```
{
	if (angle->IsOpen())
	{
		// Angle monitoring is turned on
	}
	else
	{
		// Angle monitoring is not turned on
	}
}
```

4.9.4 Query horizontal angle

BOOL SearchX(void);

Return Values

TRUE means success, FALSE means failure.

example

```
{
	if (angle->SearchX())
	{
		// Query the horizontal angle successfully
	}
}
```

4.9.5 Query pitch angle

BOOL SearchY(void);

Return Values

TRUE means success, FALSE means failure.

example

```
{
        if (angle->SearchY())
        {
                // Query the pitch angle successfully
        }
}
```

4.9.6 Query field of view angle

BOOL SearchView(int video_type);

Parameters

video_type
        [in] Video type

Return Values

TRUE means the query is successful, FALSE means the query failed.

example

```
{
        if (angle->SearchView(HP_light))
        {
                // Query the field of view angle successfully
        }
}
```

4.9.7 Query field of view angle

BOOL SearchView(int video_type);

Parameters

video_type

[in] Video type

Return Values

TRUE means the query is successful, FALSE means the query failed.

example

```
{
	if (angle->SearchView(HP_light))
	{
		// Query the field of view angle successfully
	}
}
```

4.9.8 Set angular positioning speed

BOOL SetSpeed(int speed);

Parameters

speed
	[in] Angle positioning speed, Speed ??range: 0 or 4-28

Return Values

TRUE means the setting is successful, FALSE means the setting failed

4.9.9 Horizontal angle positioning

BOOL PositionX(double angle, int speed = 0);

Parameters

angle
	[in] Horizontal angle
speed
	[in] Angle positioning speed, Speed ??range: 0 or 4-28

Return Values

TRUE means positioning is successful, FALSE means positioning failed.

4.9.10 Pitch angle positioning

BOOL PositionY(double angle, int speed = 0);

Parameters

angle
[in] Horizontal angle
speed
[in] Angle positioning speed, Speed ??range: 0 or 4-28

Return Values

TRUE means opening is successful, FALSE means opening failed.

4.9.11 Field of view angle positioning

BOOL PositionZ(double angle, int video_type = HP_ird);

Parameters

angle
[in] Field of view angle
video_type
[in] Video type HP_light Visible light HP_ird Thermal Imaging

Return Values

TRUE means success, FALSE means failure.

4.9.12 Click to locate

BOOL PositionPoint(RECT video, POINT point, int video_type = HP_ird, FLOAT rate = 0.75);

Parameters

video
[in] Video display area
point
[in] coordinate
video_type
[in] Video type HP_light Visible light HP_ird Thermal Imaging
rate
[in] Aspect ratio coefficient, default is 3:4

Return Values

TRUE means success, FALSE means failure.

Note

Need to support the field of view angle feedback function, if there is no angle feedback function, config.ini can be configured

[VIDEO_SDK]

LIGHT_VIEW=6.28

IRD_VIEW=6.28

EDGE_LEFT=32

EDGE_TOP=28

EDGE_RIGHT=39

EDGE_BOTTOM=31

4.9.13 xyz angle positioning

BOOL PositionXYZ(double x, double y, double z);

Parameters

x
: [in] Horizontal angle

y
: [in] Pitch angle

z
: [in] Field of view

Return Values

TRUE means success, FALSE means failure.

Note

Device angle xyz angle positioning, currently only supports Univision dome camera.

4.9.14 Automatic zoom according to distance

BOOL Zoom(int video_type, double dis, SIZED &size, float rate = 0.2);

Parameters

video_type
[in] Video type HP_light Visible light HP_ird Thermal Imaging
dis
[in] Distance in m
size
[in] Target size, unit m
rate
[in] The proportion of the enlarged image

Return Values

TRUE means success, FALSE means failure.

4.9.15 Get the azimuth in the video

void GetPointAngle(RECT video, POINT point, double &x, double &y, int video_type = HP_ird, FLOAT rate = 0.75);

Parameters

video_type
[in] Video type HP_light Visible light HP_ird Thermal Imaging
x
[in] Returned horizontal azimuth
y
[in] Return pitch azimuth

4.9.16 Get the azimuth in the video

void GetPointAngle(RECT video, POINT point, double cameraX, double cameraY, double cameraZ, double &x, double &y, int video_type = HP_ird, FLOAT rate = 0.75);

Parameters

video_type
[in] Video type HP_light Visible light HP_ird Thermal Imaging
x
[in] Returned horizontal azimuth
y
[in] Return pitch azimuth

4.9.17 Get the horizontal angle

float GetAngleX(void);

Return Values

Horizontal angle.

example

```
{
        float x = angle->GetAngleX())
}
```

4.9.18 Get pitch angle

float GetAngleY(void);

Return Values

Pitch angle.

example

```
{
        float y = angle->GetAngleY())
}
```

4.9.19 Get the angle of view

float GetAngleZ(int video_type = HP_ird);

Parameters

video_type
        [in] Video type HP_light Visible light HP_ird Thermal Imaging

Return Values

Field of view angle.

example

```
{
        float z = angle->GetAngleZ())
}
```

4.9.20 Set Fixed View

void SetFixedView(float viewIRD, float viewLight);

Parameters

viewIRD

[in] Thermal image fixed field of view < 0.1 invalid, Pass 0 when not needed

viewLight

[in] Fixed viewing angle of visible light < 0.1 invalid, Pass 0 when not needed

4.9.21 Register Event

void RegisterEvent(void *arg, HPANGLE_EVENT event, HPVIEW_EVENT view);

Parameters

arg

[in] Callback event subscription object

event

[in] Callback event

view

[in] Field of view angle event

4.10 Preset Interface

```
#pragma once

#ifndef HP_VIDEO_TYPE
#define HP_VIDEO_TYPE
#define HP_light    1           /*!< light */
#define HP_ird       0          /*!< ird */
#endif

namespace Hpws
{
            namespace Video
            {

                        /**
                        * @brief Preset interface
                        */
                        interface IPreset
                        {
                        public:
                                    /**
                                    * @brief Call presets
                                    * @param Preset number
                                    * @param video_type         Video type
                                    * - HP_light light
```

```
                                * - HP_ird      ird
                                * @return Success or failure
                                * @note The video_type parameter is valid only when
transparent transmission is enabled
                                */
                                virtual BOOL Call(int preset, int video_type = HP_ird) = 0;
                                /**
                                * @brief Set preset position
                                * @param Preset number
                                * @param video_type         Video type
                                * - HP_light light
                                * - HP_ird      ird
                                * @return Success or failure
                                * @note The video_type parameter is valid only when
transparent transmission is enabled
                                */
                                virtual BOOL Set(int preset, int video_type = HP_ird) = 0;
                                /**
                                * @brief Clear preset
                                * @param Preset number
                                * @param video_type         Video type
                                * - HP_light light
                                * - HP_ird      ird
                                * @return Success or failure
                                * @note The video_type parameter is valid only when
transparent transmission is enabled
                                */
                                virtual BOOL Clear(int preset, int video_type = HP_ird) =
0;
                                /**
                                * @brief Clear all presets
                                * @return Success or failure
                                */
                                virtual BOOL Clear(void) = 0;
                        };
                }
}
```

4.10.1 Call presets

BOOL Call(int preset, int video_type = HP_ird);

Parameters

preset

[in] Preset number

video_type

[in] Video type HP_light Visible light HP_ird Thermal Imaging

Remarks

The video_type parameter is valid only when transparent transmission is enabled.

Return Values

True means success, False means failure.

4.10.2 Set preset position

BOOL Set(int preset, int video_type = HP_ird);

Parameters

preset

[in] Preset number

video_type

[in] Video type HP_light Visible light HP_ird Thermal Imaging

Remarks

The video_type parameter is valid only when transparent transmission is enabled.

Return Values

True means success, False means failure.

4.10.3 Clear preset

BOOL Clear(int preset, int video_type = HP_ird);

Parameters

preset

[in] Preset number

video_type

[in] Video type HP_light Visible light HP_ird Thermal Imaging

Remarks

The video_type parameter is valid only when transparent transmission is enabled.

Return Values

True means success, False means failure.

4.10.4 Clear all presets

BOOL Clear(void);

Return Values

True means success, False means failure.

example

```
{
        if (preset->Clear())
        {
                // Clear all presets successfully
        }
}
```

4.11 Cruise interface

```
#pragma once

//---------------------------------------------------------------------------
// Type definition
//---------------------------------------------------------------------------
#ifndef HP_CRUISE_TYPE_DEF
#define HP_CRUISE_TYPE_DEF
/**
* @brief Boundary category
*/
enum HPLineType
{
        HPLINE_LEFT,            /*!< Left margin */
        HPLINE_RIGHT,           /*!< Right border */
        HPFAN_LEFTBOTTOM,       /*!< Lower left border */
        HPFAN_RIGHTTOP          /*!< Upper right border */
};
/**
* @brief Cruise category
*/
enum HPCruiseType
```

```
{
	HPCRUISE_NORMAL,	/*!< Normal cruise */
	HPCRUISE_LINE,		/*!< Line scan */
	HPCRUISE_FAN		/*!< Fan sweep */
};
#endif

//---------------------------------------------------------------------------
// Interface definition
//---------------------------------------------------------------------------
namespace Hpws
{
	namespace Video
	{
		/**
		* @brief Cruise interface
		*/
		interface ICruise
		{
		public:
			/**
			* @brief Start cruise
			* @param type Cruise category
			* @param num The first cruise, valid when type =
CRUISE_NORMAL
			* @return Success or failure
			* @note The number of cruise lines obtained by dividing
90 preset positions by the number of setpresetnum
			*/
			virtual BOOL Start(HPCruiseType type, int num = 1) = 0;
			/**
			* @brief Pause cruise, call again to continue
			* @return Success or failure
			*/
			virtual BOOL PauseOrContinue(void) = 0;
			/**
			* @brief Stop cruise
			* @return Success or failure
			*/
			virtual BOOL Stop(void) = 0;
			/**
			* @brief Set the number of preset positions included in
the cruise
			* @param num Value: (10,80)
```

```
                                * @return Success or failure
                                * @note When not called, 20 are used by default
                                */
                                virtual BOOL SetPresetNum(int num = 20) = 0;
                                /**
                                * @brief Set the dwell time of the preset position during
cruise
                                * @param time Preset position stay time
                                * @return Success or failure
                                * @note When not calling, the default preset dwell time
is 30s
                                */
                                virtual BOOL SetPresetElapse(int time = 30) = 0;
                                /**
                                * @brief Set the boundary line
                                * @param type Boundary type
                                * @return Success or failure
                                */
                                virtual BOOL SetDemarcationLine(HPLineType type) = 0;
                        };
                }
}
```

4.11.1 Cruise interface data type definition

Boundary category.

```
enum HPLineType
{
  HPLINE_LEFT,      /*!< Left margin */
  HPLINE_RIGHT,    /*!< Right border */
  HPFAN_LEFTBOTTOM,       /*!< Lower left border */
  HPFAN_RIGHTTOP            /*!< Upper right border */
};
```

Cruise category.

```
enum HPCruiseType
{
  HPCRUISE_NORMAL,         /*!< Normal cruise */
  HPCRUISE_LINE,  /*!< Line scan */
  HPCRUISE_FAN    /*!< Fan sweep */
};
```

4.11.2 Start cruise

BOOL Start(HPCruiseType type, int num = 1);

Parameters

type
[in] Cruise category
num
[in] The first cruise, valid when type = CRUISE_NORMAL

Return Values

True means success, False means failure.

Note

Divide 90 preset positions by the number of SetPresetNum to get the number of cruises

4.11.3 Pause cruise

BOOL PauseOrContinue(void);

Return Values

True means success, False means failure.

example

```
{
	if (cruise->PauseOrContinue())
	{
		// Suspended cruise successfully
	}
}
```

4.11.4 Stop cruise

BOOL Stop(void);

Return Values

True means success, False means failure.

example

```
{
	if (cruise->Stop())
	{
		// Stop cruise successfully
	}
}
```

4.11.5 Set the number of preset positions included in the cruise

BOOL SetPresetNum(int num = 20);

Parameters

num
	[in] Value:(10,80)

Return Values

True means success, False means failure.

Note

When not called, the default is 20

4.11.6 Set the dwell time of the preset position during cruise

BOOL SetPresetElapse(int time = 30);

Parameters

time
	[in] Preset position stay time

Return Values

True means success, False means failure.

Note

When not calling, the default preset dwell time is 30s

4.11.7 Set boundary line

BOOL SetDemarcationLine(HPLineType type);

Parameters

type

[in] Boundary type

Return Values

True means success, False means failure.

4.12 Config interface

```
#pragma once

//---------------------------------------------------------------------------
// Interface definition
//---------------------------------------------------------------------------
namespace Hpws
{
	namespace Video
	{
		//---------------------------------------------------------------------------
		// Type definition
		//---------------------------------------------------------------------------
#ifndef HP_VIDEO_CONFIG_DEF
#define HP_VIDEO_CONFIG_DEF
		/**
		* @brief Device Information
		*/
		typedef struct _HPDeviceBasicInfo
		{
			char	id[100];		/*!< Device ID */
			char	name[100];		/*!< Device name
*/
			char	type[100];		/*!< Equipment
type */
			char	version[100];	/*!< Device version */
			char	day[100];		/*!< Version date
*/
		}HPDeviceBasicInfo, *PHPDeviceBasicInfo;

		/**
		* @brief Device network information
		*/
		typedef struct _HPDeviceNetInfo
```

```
{
	BOOL	dhcp;			/*!< Whether DHCP is enabled */
	char	ip[100];		/*!< IP address */
	char	mask[100];		/*!< Subnet mask */
	char	gate[100];		/*!< Gateway */
}HPDeviceNetInfo, *PHPDeviceNetInfo;

/**
* @brief Equipment image parameter information
*/
typedef struct _HPDeviceVideoEffectInfo
{
	BOOL standard;			/*!< Standard mode */
	int bright;				/*!< Brightness, value range [0,255] */
	int contrast;			/*!< Contrast */
	int saturation;			/*!< saturation */
	int hue;			/*!< Chroma */
	int sharpness;			/*!< Sharpness */
}HPDeviceVideoEffectInfo, *PHPDeviceVideoEffectInfo;
#endif

/**
* @brief Device configuration maintenance interface
*/
interface IConfig
{
public:
	/**
	* @brief Dev reboot
	* @return Success or failure
	* @note The return value only represents the success or failure of the message sent, the real status of the device is obtained by the query module
	*/
	virtual BOOL Reboot() = 0;

	/**
	* @brief Set OSD information
	* @return Success or failure
	*/
```

```
                        virtual BOOL SetOSD(char *osd, int x, int y) = 0;

                        /**
                        * @brief Display text information
                        * @return Success or failure
                        * @note Including channel name, time, OSD and other
information
                        */
                        virtual BOOL ShowTextInfo(BOOL show) = 0;

                        /**
                        * @brief Set alarm output status
                        * @param enable TRUE normally open, FALSE normally
closed
                        * @return Success or failure
                        * @retval
                        * @note
                        */
                        virtual BOOL EnableAlarmOut(BOOL enable) = 0;

                        /**
                        * @brief Control equipment power
                        * @param ip The IP address of the serial server
                        * @param port The port number of the serial server,
usually 4001
                        * @param type Control type 1:turn on; 2:turn off;
3:restart
                        * @note When type is 1/2/3, it uses “power off all
except the switch board and serial server”
                        *           The type is: 11/12/13 adopts “all power on
and off except the IP box”
                        */
                        virtual BOOL CtrlPower(char *ip, int port, int type) = 0;

                        /**
                        * @brief Set the local time to the current time of the
device
                        * @return Success or failure
                        */
                        virtual BOOL SetTime(void) = 0;

                        /**
                        * @brief Get network information
                        * @param Device network information returned by info
```

```
                        * @return Success or failure
                        * @note Only support ipv4, non-DHCP mode
                        */
                        virtual BOOL GetNetAddress(HPDeviceNetInfo &info) =
0;
                        /**
                        * @brief Set network information
                        * @return Success or failure
                        * @note Only support ipv4, non-DHCP mode
                        */
                        virtual BOOL SetNetAddress(char *ip, char *mask, char
*gate) = 0;

                        /**
                        * @brief Get image parameters
                        * @param info Image parameter information
                        * @return Success or failure
                        */
                        virtual BOOL GetVideoEffect(HPDeviceVideoEffectInfo
&info) = 0;
                        /**
                        * @brief Set image parameters
                        * @param standard                Standard mode
                        * @param bright                  Brightness, value
range [0,255]
                        * @param contrast                Contrast, value
range [0,255]
                        * @param saturation              Saturation, value
range [0,255]
                        * @param hue                     Chromaticity, value
range [0,255]
                        * @param sharpness               Sharpness, value
range [0,255]
                        * @return Success or failure
                        */
                        virtual BOOL SetVideoEffect(BOOL standard, int bright,
int contrast, int saturation, int hue, int sharpness) = 0;

                        /**
                        * @brief Modify username and password
                        * @return Success or failure
                        */
                        virtual BOOL ChangePassword(char *user, char *pwd,
char *newUser, char *newPwd) = 0;
```

```
                    /**
                    * @brief Get basic device information
                    * @param Basic device information returned by info
                    * @return Success or failure
                    */
                    virtual BOOL GetDeviceBasicInfo(HPDeviceBasicInfo
&info) = 0;

                    /**
                    * @brief Whether the electronic image stabilization is
turned on
                    * @return Is it on
                    * @note Currently only supports Haikang
                    */
                    virtual BOOL IsOpenOIS() = 0;

                    /**
                    * @brief Turn on electronic image stabilization
                    * @param open On or off
                    * @return Success or failure
                    * @note Currently only supports Haikang
                    */
                    virtual BOOL OpenOIS(BOOL open = TRUE) = 0;

                public:
                    /**
                    * @brief Get video channel number (valid for single IP
device)
                    * @param video_type      Video type
                    *                  -HP_light           Visible
light channel
                    *                  -HP_ird             Thermal
imaging channel
                    * @return -0 No such channel
                    *                  >0 Corresponding channel number
                    * @note The video channel number starts from 1
                    */
                    virtual int GetVideoChannel(int video_type) = 0;
                };
        }
}
```

4.12.1 Configuration maintenance interface data type definition

Device Information。

```
typedef struct _HPDeviceBasicInfo
{
    char id[100];               /*!< Device ID*/
    char name[100];                     /*!< Equipment name */
    char type[100];             /*!< Equipment type */
    char version[100];          /*!< Device version */
    char day[100];              /*!< Version date */
} HPDeviceBasicInfo, *PHPDeviceBasicInfo;
```

Device network information。

```
typedef struct _HPDeviceNetInfo
{
    BOOL dhcp;                  /*!< Whether DHCP is enabled*/
    char ip[100];               /*!< IP address */
    char mask[100];             /*!< Subnet mask */
    char gate[100];             /*!< Gateway */
} HPDeviceNetInfo, *PHPDeviceNetInfo;
```

Equipment image parameter information。

```
typedef struct _HPDeviceVideoEffectInfo
{
  BOOL standard;                /*!< Standard mode */
    int bright;       /*!< Brightness, value range[0,255] */
    int contrast;     /*!< Contrast */
    int saturation;             /*!< saturation */
    int hue;                    /*!< Chroma */
    int sharpness;              /*!< Sharpness */
} HPDeviceVideoEffectInfo, *PHPDeviceVideoEffectInfo;
```

4.12.2 Device restart

```
BOOL Reboot();
```

Return Values

TRUE means success, FALSE means failure.

example

```
{
        if (config->Reboot())
        {
                // Device restart
        }
}
```

Note

The return value only represents the success or failure of sending the message, and the real status of the device is obtained by the query module

4.12.3 Set OSD information

BOOL SetOSD(char *osd, int x, int y);

Return Values

TRUE means success, FALSE means failure.

4.12.4 Display text information

BOOL ShowTextInfo(BOOL show);

Return Values

TRUE means success, FALSE means failure.

Note

Including channel name, time, OSD and other information.

4.12.5 Set alarm output status

BOOL EnableAlarmOut(BOOL enable);

Parameters

enable

[in] TRUE Normally open，FALSE Changguan

Return Values

TRUE means success, FALSE means failure.

Note

Currently, only Univision and Hepu are supported.

4.12.6 Control equipment power

BOOL CtrlPower(char *ip, int port, int type);

Parameters

ip
: [in] The IP address of the serial server

port
: [in] The port number of the serial server, usually 4001

type
: [in] Control type 1:turn on; 2:turn off; 3:restart

Return Values

TRUE means success, FALSE means failure.

Note

When type is 1/2/3, it uses "power off all except the switch board and serial server";

The type is: 11/12/13 adopts "power on and off all except the IP box".

4.12.7 Set the local time to the current time of the device

BOOL SetTime(void);

Return Values

TRUE means success, FALSE means failure.

4.12.8 Get network information

BOOL GetNetAddress(HPDeviceNetInfo &info);

Parameters

info
: [in] Returned device network information

Return Values

TRUE means success, FALSE means failure.

Note

Only supports ipv4, not DHCP mode.

4.12.9 Set network information

BOOL SetNetAddress(char *ip, char *mask, char *gate)

Parameters

ip
[in] ip address
mask
[in] Mask
gate
[in] Gateway

Return Values

TRUE means success, FALSE means failure.

Note

Only supports ipv4, non-DHCP mode.

4.12.10 Get image parameters

BOOL GetVideoEffect(HPDeviceVideoEffectInfo &info);

Parameters

info
[in] Image parameter information

Return Values

TRUE means success, FALSE means failure.

4.12.11 Set image parameters

BOOL SetVideoEffect(BOOL standard, int bright, int contrast, int saturation, int hue, int sharpness);

Parameters

standard
[in] Standard mode
bright
[in] Brightness, value range[0,255]
contrast
[in] Contrast, value range[0,255]
saturation
[in] Saturation, value range[0,255]
hue
[in] Chromaticity, value range[0,255]
sharpness
[in] Sharpness, value range[0,255]

Return Values

TRUE means success, FALSE means failure.

4.12.12 Modify username and password

BOOL ChangePassword(char *user, char *pwd, char *newUser, char *newPwd);

Parameters

user
[in] username
pwd
[in] user password
newUser
[in] new user name
newPwd
[in] New user password

Return Values

TRUE means success, FALSE means failure.

4.12.13 Get basic device information

BOOL GetDeviceBasicInfo(HPDeviceBasicInfo &info);

Parameters

info
[in] Basic device information returned

Return Values

TRUE means success, FALSE means failure.

4.12.14 Whether the electronic image stabilization is turned on

BOOL IsOpenOIS();

Return Values

TRUE means it is turned on, FALSE means it is not turned on.

Note

Currently only Haikang is supported.

4.12.15 Turn on electronic image stabilization

BOOL OpenOIS(BOOL open = TRUE);

Parameters

open
[in] On or off

Return Values

TRUE means success, FALSE means failure

Note

Currently only Haikang is supported.

4.12.16 Get the video channel number (valid for single IP device)

int GetVideoChannel(int video_type);

Parameters

video_type
[in] Video type HP_light Visible light channel HP_ird Thermal imaging channel

Return Values

0 No such channel;

> 0 Corresponding channel number.

Note

The video channel number starts from 1.

4.13 Serial interface

```
#pragma once
#include "IUpdate.h"

//----------------------------------------------------------------------------
// Type definition
//----------------------------------------------------------------------------
#ifndef HP_SERIALDATA_EVENT_DEF
#define HP_SERIALDATA_EVENT_DEF

/**
* @brief device status
*/
typedef struct _HPDeviceStateInfo
{
	BOOL	ccdPower;			/*!< Whether the visible light power is on */
	BOOL	ccdAF;				/*!< Whether the visible light auto focus is turned on */
	BOOL	irPower;		/*!< Whether the thermal image power is on */
	BOOL	irAF;				/*!< Whether the thermal image auto focus is turned on */
	BOOL	patrol;				/*!< Whether cruise is on */
	BOOL	autoSacn;			/*!< Whether automatic scanning is on */
	BOOL	applePeel;			/*!< Whether the apple peel is on */
	BOOL	defrog;				/*!< Whether heating defrosting is on */
	int		trackAlg;		/*!< Algorithm; 1: Centroid; 2: Correlation; 3: Centroid; 4: Correlation */
	BOOL	trackState;			/*!< Is optoelectronics tracking */
	BOOL	trackCCD;			/*!< TRUE:Light tracking；FALSE:Ird tracking */
	BOOL	trackAuto;			/*!< Is it auto tracking */
	BOOL	trackGuid;			/*!< Whether the radar linkage is turned on */
}HPDeviceStateInfo, *PHPDeviceStateInfo;
```

```
/**
* @brief Transparent data event
*/
typedef void (__stdcall *HPSERIALDATA_EVENT)(void *arg, unsigned char *data, int
len);

/**
* @brief Status event
*/
typedef void (__stdcall *HPDEVICE_STATE_EVENT)(void *arg, char *ip,
PHPDeviceStateInfo state);
#endif

//------------------------------------------------------------------------------
// Interface definition
//------------------------------------------------------------------------------
namespace Hpws
{
	namespace Video
	{

		/**
		* @brief Transparent channel interface
		*/
		interface ISerial
		{
		public:
			/**
			* @brief Open transparent channel
			* @return Success or failure
			*/
			virtual BOOL Open(void) = 0;
			/**
			* @brief Close transparent channel
			* @return Success or failure
			*/
			virtual BOOL Close(void) = 0;
			/**
			* @brief Whether the transparent channel is turned on
			* @return Open or not
			*/
			virtual BOOL IsOpen(void) = 0;
			/**
```

```
                        * @brief Set as a single address device
                        * @note TC/RC equipment needs to call this interface
                        */
                        virtual void SetSingleAddress(void) = 0;
                        /**
                        * @brief Is it a single address device
                        */
                        virtual BOOL IsSingleAddress(void) = 0;

                        /**
                        * @brief Send serial port (Pelc) data
                        * @param data Data content
                        * @param len Data length
                        * @param elapse Send waiting time
                        * @return Success or failure
                        */
                        virtual BOOL SendData(char *data, int len = 7, int elapse
= 30) = 0;

                        /**
                        * @brief Register transparent data event
                        * @param arg Callback event subscription object
                        * @param event Callback event
                        * @param original Whether to return the original data
                        * @return no
                        */
                        virtual void RegisterEvent(void *arg,
HPSERIALDATA_EVENT event, BOOL original) = 0;

                        /**
                        * @brief Send status query data
                        * @return Success or failure
                        */
                        virtual BOOL SeachState(void) = 0;

                        /**
                        * @brief Register status callback event
                        * @param arg Callback event subscription object
                        * @param stateEvent          State callback event
                        * @return no
                        */
                        virtual void RegisterStateEvent(void *arg,
HPDEVICE_STATE_EVENT event) = 0;
```

```
                    /**
                    * @brief Get device upgrade interface
                    * @return Upgrade interface
                    */
                    virtual IUpdate* GetUpdatePtr(void) const = 0;
                };
        }
}
```

4.13.1 SDK Transparent channel interface

device status.

```
typedef struct _HPDeviceStateInfo
{
  BOOL ccdPower;            /*!< Whether the visible light power is on */
  BOOL ccdAF;               /*!< Whether the visible light auto focus is turned on */
  BOOL irPower;    /*!< Whether the thermal image power is on */
  BOOL irAF;                /*!< Whether the thermal image autofocus is turned on
*/
  BOOL patrol;      /*!< Whether cruise is on */
  BOOL autoSacn;   /*!< Whether automatic scanning is on */
  BOOL applePeel;           /*!< Whether the apple peel is on */
  BOOL defrog;      /*!< Whether heating defrosting is on */
  int  trackAlg;      /*!< Algorithm; 1: Centroid; 2: Correlation; 3: Centroid; 4:
Correlation */
  BOOL trackState;          /*!< Is the photoelectric tracking */
  BOOL trackCCD;   /*!< TRUE: visible light tracking; FALSE: thermal imaging tracking
*/
  BOOL trackAuto;           /*!< Whether it is automatic tracking */
  BOOL trackGuid;           /*!< Whether the radar linkage is turned on */
} HPDeviceStateInfo, *PHPDeviceStateInfo;
```

4.13.2 Open transparent channel

```
BOOL Open(void);
```

Return Values

TRUE means success, FALSE means failure.

4.13.3 Close transparent channel

```
BOOL Close(void);
```

Return Values

TRUE means success, FALSE means failure.

4.13.4 Whether the transparent channel is turned on

BOOL IsOpen(void);

Return Values

TRUE means success, FALSE means failure.

4.13.5 Set as a single address device

void SetSingleAddress(void);

Note

TC/RC equipment needs to call this interface.

4.13.6 Is it a single address device

BOOL IsSingleAddress(void);

Return Values

TRUE means yes, FALSE means no.

4.13.7 Send serial port (Pelc) data

BOOL SendData(char *data, int len = 7, int elapse = 30);

Parameters

date
[in] Data content
len
[in] Data length
elapse
[in] Send waiting time

Return Values

TRUE means success, FALSE means failure.

4.13.8 Register Event

void RegisterEvent(void *arg, HPSERIALDATA_EVENT event, BOOL original);

Parameters

arg
[in] Callback event subscription object
event
[in] Callback event
original
[in] Whether to return the original data

4.13.9 Send status query data

BOOL SeachState(void);

Return Values

TRUE means success, FALSE means failure.

4.13.10 Register State Event

void RegisterStateEvent(void *arg, HPDEVICE_STATE_EVENT event);

Parameters

arg
[in] Callback event subscription object
event
[in] State callback event

4.13.11 Get device upgrade interface

IUpdate* GetUpdatePtr(void) const;

Return Values

Upgrade interface.

4.14 Focus query interface

#pragma once

//---

```
// Type definition
//---------------------------------------------------------------------------

#ifndef HP_VIDEO_TYPE
#define HP_VIDEO_TYPE
#define HP_light    1           /*!< light */
#define HP_ird      0           /*!< ird */
#endif

#ifndef HP_FOCUS_EVENT_DEF
#define HP_FOCUS_EVENT_DEF
/**
* @brief Focus event
* @param video_type         Video type
* - HP_light light
* - HP_ird      ird
*/
typedef void (__stdcall *HPFOCUS_EVENT)(void *arg, char *ip, int focus, int
video_type);
#endif

//---------------------------------------------------------------------------
// Interface definition
//---------------------------------------------------------------------------
namespace Hpws
{
        namespace Video
        {
                /**
                * @brief Focus query interface
                */
                interface IFocusInfo
                {
                public:
                        /**
                        * @brief Enable focused query function
                        * @param arg Event subscription object
                        * @param event event
                        * @return Success or failure
                        */
                        virtual BOOL Open(void *arg, HPFOCUS_EVENT event) =
0;
                        /**
                        * @brief Disable the focus query function
```

```
                        * @return Success or failure
                        */
                        virtual BOOL Close(void) = 0;
                        /**
                        * @brief Whether to enable the focus query function
                        * @return Success or failure
                        */
                        virtual BOOL IsOpen(void) = 0;
                        /**
                        * @brief Focus
                        * @param value Focus value
                        * @param video_type         Video type
                        * - HP_light light
                        * - HP_ird      ird
                        * @return Success or failure
                        */
                        virtual BOOL Focus(int value, int video_type = HP_ird) =
0;

                        /**
                        * @brief Auto focus
                        * @param open Whether to turn on auto focus
                        * @param video_type         Video type
                        * - HP_light light
                        * - HP_ird      ird
                        * @return Success or failure
                        */
                        virtual BOOL AutoFocus(BOOL open, int video_type =
HP_ird) = 0;

                        /**
                        * @brief Query focus
                        * @return Success or failure
                        * @note Query only once
                        */
                        virtual BOOL Search(int video_type) = 0;

                        /**
                        * @brief Register for focusm events
                        * @param arg Event subscription object
                        * @param event event
                        * @return no
                        */
```

```
                                virtual void RegisterEvent(void *arg, HPFOCUS_EVENT
event) = 0;
                        };
            }
}
```

4.14.1 Enable focused query function

BOOL Open(void *arg, HPFOCUS_EVENT event);

Parameters

arg
[in] Event subscription object
event
[in] event

Return Values

Success or failure.

4.14.2 Disable the focus query function

BOOL Close(void);

Return Values

Success or failure.

4.14.3 Whether to enable the focus query function

BOOL IsOpen(void);

Return Values

Success or failure.

4.14.4 Focus

BOOL Focus(int value, int video_type = HP_ird);

Parameters

value
[in] Focus value
video_type

[in] Video type HP_light Visible light HP_ird Thermal Imaging

Return Values

Success or failure.

4.14.5 Auto focus

BOOL AutoFocus(BOOL open, int video_type = HP_ird);

Parameters

open
[in] Whether to turn on auto focus
video_type
[in] Video type HP_light Visible light HP_ird Thermal Imaging

Return Values

Success or failure.

4.14.6 Query focus

BOOL Search(int video_type);

Note

Query only once

Return Values

Success or failure.

4.14.7 Register for focusm events

void RegisterEvent(void *arg, HPFOCUS_EVENT event);

Parameters

arg
[in] Event subscription object
event
[in] event

4.15 Power status interface

```
#pragma once

//------------------------------------------------------------------------
// Type definition
//------------------------------------------------------------------------

#ifndef HP_VIDEO_TYPE
#define HP_VIDEO_TYPE
#define HP_light    1           /*!< light */
#define HP_ird       0          /*!< ird */
#endif

#ifndef HP_POWER_EVENT_DEF
#define HP_POWER_EVENT_DEF
/**
* @brief Power state
*/
enum HPPowerState
{
        POWER_ON,                   /*!< Power on state */
        POWER_OFF,                  /*!< Breakpoint state */
        POWER_STANDBY /*!< standby mode */
};
/*Abnormal event*/
typedef void (__stdcall *HPPOWER_EVENT)(void *arg, char *ip, HPPowerState
state);
#endif

//------------------------------------------------------------------------
// Interface definition
//------------------------------------------------------------------------
namespace Hpws
{
        namespace Video
        {
                /**
                * @brief Power status interface
                */
                interface IPower
                {
                public:
                        /**
```

```
                * @brief Enable status query function
                * @param arg Event subscription object
                * @param event event
                * @return Success or failure
                */
                virtual BOOL Open(void *arg, HPPOWER_EVENT event) =
0;
                /**
                * @brief Disable status query function
                * @return Success or failure
                */
                virtual BOOL Close(void) = 0;
                /**
                * @brief Whether to enable the status query function
                * @return Success or failure
                */
                virtual BOOL IsOpen(void) = 0;

                /**
                * @brief Determine whether it is in standby
                * @return Is it in standby
                * @note Thermal imaging equipment is effective
                */
                virtual BOOL IsOnstandby(void) = 0;

        public:
                /**
                * @brief Power on the device
                * @param video_type         Video type
                * - HP_light light
                * - HP_ird      ird
                * @return Success or failure
                * @note The return value only represents the success or
failure of the message sent, the real status of the device is obtained by the query
module
                */
                virtual BOOL PowerOn(int video_type) = 0;
                /**
                * @brief Device power off
                * @param video_type         Video type
                * - HP_light light
                * - HP_ird      ird
                * @return Success or failure
```

```
                                * @note The return value only represents the success or
failure of the message sent, the real status of the device is obtained by the query
module
                                */
                                virtual BOOL PowerOff(int video_type) = 0;
                        };
                }
}
```

4.15.1 Power state

```
enum HPPowerState
{
 POWER_ON,        /*!< Power on state */
  POWER_OFF,      /*!< Breakpoint state */
  POWER_STANDBY           /*!< standby mode */
};
```

4.15.2 Enable status query

```
BOOL Open(void *arg, HPPOWER_EVENT event);
```

Parameters

arg

[in] Event subscription object

event

[in] event

Return Values

TRUE means success, FALSE means failure.

4.15.3 Disable status query

```
BOOL Close(void);
```

Return Values

TRUE means success, FALSE means failure.

4.15.4 Determine whether it is in standby

```
BOOL IsOnstandby(void);
```

Return Values

Whether it is in standby state.

Note

Thermal imaging equipment is effective.

4.15.5 Power on the device

BOOL PowerOn(int video_type);

Parameters

video_type
[in] Video type HP_light Visible light HP_ird Thermal Imaging

Return Values

TRUE means success, FALSE means failure.

Note

The return value only represents the success or failure of the message transmission, and the real status of the device is obtained by the query module.

4.15.6 Device power off

BOOL PowerOff(int video_type);

Parameters

video_type
[in] Video type HP_light Visible light HP_ird Thermal Imaging

Return Values

TRUE means success, FALSE means failure.

Note

The return value only represents the success or failure of the message transmission, and the real status of the device is obtained by the query module.

4.16 ITemperature interface

```
#pragma once

#ifndef HP_TEMPERATURE_EVENT_DEF
#define HP_TEMPERATURE_EVENT_DEF



#endif // !HP_TEMPERATURE_EVENT_DEF

namespace Hpws
{
	namespace Video
	{
		/**
		* @brief Temperature measurement interface
		*/
		interface ITemperature
		{
		public:
			/**
			* @brief Display temperature information
			* @param bSHow Whether to display temperature
information
			* @return
			*/
			virtual BOOL ShowTemperInfo(BOOL bShow) = 0;
		};
	}
}
```

4.16.1 Display temperature information

```
BOOL ShowTemperInfo(BOOL bShow);
```

Parameters

bShow

[in] Whether to display temperature information

Return Values

TRUE means success, FALSE means failure.

4.17 Voice intercom interface

```
#pragma once
namespace Hpws
{
	namespace Video
	{

		/**
		* @brief Voice intercom interface
		*/
		interface IVoice
		{
		public:
			/**
			* @brief Turn on voice intercom
			* @param channel Voice intercom channel number
			* @return Success or failure
			*/
			virtual BOOL StartVoice(int channel = 1) = 0;
			/**
			* @brief Turn off voice intercom
			* @return Success or failure
			*/
			virtual BOOL StopVoice(void) = 0;
			/**
			* @brief Whether it is voice intercom
			* @return Success or failure
			*/
			virtual BOOL IsVoicinging(void) = 0;
			/**
			* @brief Set the intercom volume
			* @param volume Client volume 0-1000
			* @return Success or failure
			*/
			virtual BOOL SetAudioVolume(int volume) = 0;
		};
	}
}
```

4.17.1 Turn on voice intercom

```
BOOL StartVoice(int channel = 1);
```

Parameters

channel
[in] Voice intercom channel number

Return Values

TRUE means success, FALSE means failure.

4.17.2 Turn off voice intercom

BOOL StopVoice(void);

Return Values

TRUE means success, FALSE means failure.

4.17.3 Whether it is voice intercom

BOOL IsVoicinging(void);

Return Values

TRUE means voice intercom is in progress, FALSE means not voice intercom is in progress.

4.17.4 Set the intercom volume

BOOL SetAudioVolume(int volume);

Parameters

volume
[in] Client volume 0-1000

Return Values

TRUE means success, FALSE means failure.

4.18 Exten interface

#pragma once

//----------

```
// Type definition
//---------------------------------------------------------------------------
#ifndef HP_EXTEN_EVENT_DEF
#define HP_EXTEN_EVENT_DEF

/**
* @brief Ranging event
* @param dis Distance in m
*/
typedef void (__stdcall *HPRANGING_EVENT)(void *arg, char *ip, int dis);

/**
* @brief Electronic compass event
*/
typedef void (__stdcall *HPELC_COMPASS_EVENT)(void *arg, char *ip, float angleX,
float angleY);

#endif

//---------------------------------------------------------------------------
// Interface definition
//---------------------------------------------------------------------------
namespace Hpws
{
	namespace Video
	{

		/**
		* @brief Peripheral expansion interface
		*/
		interface IExten
		{
		public:

			/**
			* @brief Ranging
			* @param arg Callback event subscription object
			* @param event Callback event
			* @return Whether the request is sent successfully
			*/
			virtual BOOL Ranging(void *arg, HPRANGING_EVENT
event) = 0;

			/**
```

```
                                * @brief Query electronic compass
                                * @param arg Callback event subscription object
                                * @param event Callback event
                                * @return Whether the request is sent successfully
                                */
                                virtual BOOL ElcCompass(void *arg,
HPELC_COMPASS_EVENT event) = 0;
                        };
            }
}
```

4.18.1 Ranging

BOOL Ranging(void *arg, HPRANGING_EVENT event);

Parameters

arg

[in] Callback event subscription object

event

[in] Callback event

Return Values

Whether the request was sent successfully.

4.18.2 Query electronic compass

BOOL ElcCompass(void *arg, HPELC_COMPASS_EVENT event);

Parameters

arg

[in] Callback event subscription object

event

[in] Callback event

Return Values

Whether the request was sent successfully.

4.19 NVR interface

```
#pragma once

namespace Hpws
{
	namespace Video
	{

		//---------------------------------------------------------------------------
		// Type definition
		//---------------------------------------------------------------------------
#ifndef HP_NVR_CHN_DEF
#define HP_NVR_CHN_DEF
		/**
		* @brief NVR channel information definition
		*/
		typedef struct _HPNvrChnInfo
		{
			char ip[20];			/*!< ip */
			int port;			/*!< port */
			char user[100];			/*!< account number */
			char pwd[100];			/*!< Pwd */
			int index;			/*!< Channel number,
starting from 1 */
		}HPNvrChnInfo, *PHPNvrChnInfo;
#endif

		//---------------------------------------------------------------------------
		// Interface definition
		//---------------------------------------------------------------------------
		/**
		* @brief NVR interface
		*/
		interface INvr
		{
		public:
			/**
			* @brief Get NVR channel information
			* @param count Number of channels
			* @return Channel information collection
			*/
			virtual HPNvrChnInfo* GetChnInfos(int &count) = 0;
```

```
                    };
            }
}
```

4.19.1 SDK NVR Interface

NVR channel information definition.

```
typedef struct _HPNvrChnInfo
{
    char ip[20];     /*!< ip */
    int port;                   /*!< port */
    char user[100];             /*!< account number */
    char pwd[100];              /*!< password */
    int index;       /*!< Channel number, starting from 1 */
} HPNvrChnInfo, *PHPNvrChnInfo;
```

4.19.2 Get NVR channel information

```
HPNvrChnInfo* GetChnInfos(int &count);
```

Parameters

count

[in] Number of channels

Return Values

Channel information collection.

4.20 Handware trace interface

```
#pragma once

#ifndef HP_VIDEO_TYPE
#define HP_VIDEO_TYPE
#define HP_light    1           /*!< light */
#define HP_ird       0          /*!< ird */
#endif

namespace Hpws
{
            namespace Video
            {
                        /**
                        * @brief Track status event
```

```
		*/
		typedef void (__stdcall *HPDEVICE_TRACK_STATE_EVENT)(void
*arg, char *ip, BOOL tracking);
		/**
		* @brief Track target information event
		* @param angleX Target horizontal angle
		* @param angleY Target pitch angle
		* @param data Missing information
		*/
		typedef void (__stdcall *HPDEVICE_TRACK_AIM_EVENT)(void
*arg, char *ip, float angleX, float angleY, PMissData data);

		/**
		* @brief Hardware trace interface
		*/
		interface ITrack
		{
		public:
			/**
			* @brief Trigger/Cancel Tracking
			* @return The instruction is sent successfully or failed
			* @note Call this interface repeatedly to switch between
tracking and non-tracking
			*		Thermal image IPC does not support
			*/
			virtual BOOL Trigger(void) = 0;

			/**
			* @brief Start tracking
			* @note Not supported by devices in 2019 and before
			*/
			virtual BOOL StartTrack(void) = 0;
			/**
			* @brief Stop tracking
			* @note Not supported by devices in 2019 and before
			*/
			virtual BOOL StopTrack(void) = 0;

			/**
			* @brief Hardware frame selection tracking
			* @param video Video display area
			* @param select select area
			* @return The instruction is sent successfully or failed
			* @note Not supported by devices in 2019 and before
```

```
                                */
                                virtual BOOL Select(RECT video, RECT select) = 0;

                                /**
                                * @brief Switch tracking video
                                * @param video_type         Video type
                                *                           -HP_light    Switch to visible light
tracking video
                                *                           -HP_ird Switch to thermal imaging
tracking video
                                * @return The instruction is sent successfully or failed
                                * @note Thermal imaging IPC only tracks the control
channel, and is called in advance before tracking
                                */
                                virtual BOOL ChangeVideo(int video_type) = 0;


                                /**
                                * @brief Boot tracking
                                * @param angleX Horizontal angle
                                * @param angleY Pitch angle
                                * @param dis Distance, unit m, do not perform distance
zoom when it is 0
                                * @param aimWidth Target width, unit m
                                * @param rate The proportion of the enlarged image
                                * @param autoTrack Whether to track automatically
after booting
                                * @param speed Angle positioning speed, Speed range:
0 or 4-28
                                * @return Success or failure
                                * @note Not supported by devices in 2019 and before
                                */
                                virtual BOOL GuideTrack(double angleX, double angleY,
double dis, double aimWidth, float rate = 0.02, BOOL autoTrack = TRUE, int speed =
0) = 0;


                                /**
                                * @brief Set scan parameters
                                * @param video_type         Video type
                                * - HP_light light
                                * - HP_ird      ird
                                * @param difvalue Determine the error value of the
angle in place. Default 0.3
```

```
                * @param timeout The angle is in place overtime
judgment, if it is not in place within 30s, it will be regarded as a failure, stop rotating
and search
                * @note Set the type of visible light or thermal image,
the angle of view is different, so it needs to be set, and the angle error value and
timeout judgment
                */
                virtual void SetSearchConfig(int video_type, float
difValue = 0.3, int timeout = 30) = 0;

                /**
                * @brief Start range search
                * @param angleX The horizontal angle of the search
center
                * @param angleY Pitch angle of search center
                * @param valueX Horizontal search range (degrees)
                * @param valueY Pitch search range (degrees)
                * @param search Whether to start search
                * @param moveSpeed The search speed is 1-30, the
greater the speed, the greater the error. Recommendations 5-15
                * @param speed Speed to the center, Speed range: 0
(fastest) or 4-28
                * @return Success or failure
                */
                virtual BOOL StartSearch(double angleX, double angleY,
float valueX, float valueY, BOOL search , int moveSpeed = 12, int speed = 0) = 0;

                /**
                * @brief guidesearch
                * @param angleX The horizontal angle of the search
center
                * @param angleY Pitch angle of search center
                * @param valueX Horizontal search range (degrees)
                * @param valueY Pitch search range (degrees)
                * @param search Whether to start search
                * @param moveSpeed The search speed is 1-30, the
greater the speed, the greater the error. Recommendations 5-15
                * @param speed Speed to the center, Speed range: 0
(fastest) or 4-28
                * @param dis Distance in m
                * @param aimWidth Target width, unit m
                * @param rate The proportion of the enlarged image
                * @return Success or failure
                */
```

```
                virtual BOOL GuidSearch(double angleX, double angleY,
float valueX, float valueY, BOOL search , int moveSpeed = 12,
                                        double dis = 0, double aimWidth = 1, float rate
= 0.02,    int speed = 0) = 0;

                /**
                * @brief Stop search
                * @return Success or failure
                */
                virtual BOOL StopSearch(void) = 0;

                /**
                * @brief Register for tracking events
                * @note Not supported by devices in 2019 and before
                */
                virtual void RegisterTrackEvent(void *arg,
HPDEVICE_TRACK_STATE_EVENT event, HPDEVICE_TRACK_AIM_EVENT aimEvent) =
0;

                /**
                * @brief Is tracking
                */
                virtual BOOL IsTracking(void) = 0;

                /**
                * @brief Set the threshold for automatic tracking
                * @param threshold Threshold:1-255
                * @note Not supported by devices in 2019 and before
                */
                virtual BOOL SetAutoTrackThreshold(int threshold) = 0;

                /**
                * @brief Switch tracking mode
                * @note Thermal imaging IPC does not support this
interface
                */
                virtual BOOL ChangeScene(void) = 0;

                /**
                * @brief Switch tracking algorithm
                * @note Thermal imaging IPC does not support this
interface
                */
                virtual BOOL ChangeAlgorithm(void) = 0;
```

```
        public:
            /**
            * @brief Switch tracking mode
            * @param mode Target tracking mode
            *                  -0: semi-automatic against air; -1: automatic against air;
            *                  -2: Manual to the ground; -3: Semi-automatic to the ground; -4: Automatic to the ground;
            * @note Only thermal image IPC support
            */
            virtual BOOL ChangeScene(int mode) = 0;
        };
    }
}
```

4.20.1 Trigger to cancel tracking

BOOL Trigger(void);

Return Values

The instruction was sent successfully or failed.

Note

Call this interface repeatedly to switch between tracking and non-tracking. Thermal IPC does not support it.

4.20.2 Start tracking

BOOL StartTrack(void);

Note

Not supported by devices in 2019 and before.

4.20.3 Stop tracking

BOOL StopTrack(void);

Note

Not supported by devices in 2019 and before.

4.20.4 Hardware frame selection tracking

BOOL Select(RECT video, RECT select);

Parameters

video
[in] Video display area
select
[in] select area

Return Values

The instruction was sent successfully or failed.

Note

Not supported by devices in 2019 and before.

4.20.5 Switch tracking video

BOOL ChangeVideo(int video_type);

Parameters

video_type
[in] Video type HP_light Switch to visible light tracking video HP_ird Switch to thermal imaging tracking video

Return Values

The instruction was sent successfully or failed.

Note

Thermal imaging IPC only tracks the control channel and calls it in advance before tracking.

4.20.6 Boot tracking

BOOL GuideTrack(double angleX, double angleY, double dis, double aimWidth, float rate = 0.02, BOOL autoTrack = TRUE, int speed = 0);

Parameters

angleX
[in] Horizontal angle
angleY

[in] Pitch angle

dis

[in] Distance, unit m, do not perform distance zoom when it is 0

aimWidth

[in] Target width, unit m

rate

[in] The proportion of the enlarged image

autoTrack

[in] Whether to track automatically after booting

speed

[in] Angle positioning speed, Speed ??range: 0 or 4-28

Return Values

Success or failure.

Note

Not supported by devices in 2019 and before.

4.20.7 Set scan parameters

void SetSearchConfig(int video_type, float difValue = 0.3, int timeout = 30);

Parameters

video_type

[in] Video type HP_light Visible light HP_ird Thermal Imaging

difvalue

[in] Determine the error value of the angle in place. Default 0.3

timeout

[in] The angle is in place overtime judgment, if it is not in place within 30s, it will be regarded as a failure, stop rotating and search

Note

Set the visible light or thermal image type, the field of view is different, so it needs to be set, as well as the angle error value and timeout judgment.

4.20.8 Range search

BOOL StartSearch(double angleX, double angleY, float valueX, float valueY, BOOL search , int moveSpeed = 12, int speed = 0);

Parameters

angleX
[in] The horizontal angle of the search center
angleY
[in] Pitch angle of search center
valueX
[in] Horizontal search range (degrees)
valueY
[in] Pitch search range (degrees)
search
[in] Whether to start search
moveSpeed
[in] The search speed is 1-30, the greater the speed, the greater the error. Recommendations 5-15
speed
[in] Speed ??to the center, Speed ??range: 0 (fastest) or 4-28

Return Values

Success or failure.

4.20.9 Guided search

BOOL GuidSearch(double angleX, double angleY, float valueX, float valueY, BOOL search , int moveSpeed = 12, double dis = 0, double aimWidth = 1, float rate = 0.02, int speed = 0);

Parameters

angleX
[in] The horizontal angle of the search center
angleY
[in] Pitch angle of search center
valueX
[in] Horizontal search range (degrees)
valueY
[in] Pitch search range (degrees)
search
[in] Whether to start search
moveSpeed
[in] The search speed is 1-30, the greater the speed, the greater the error. Recommendations 5-15
speed
[in] Speed ??to the center, Speed ??range: 0 (fastest) or 4-28
dis
[in] Distance in m

aimWidth
[in] Target width, unit m
rate
[in] (Image ratio after magnification)

Return Values

Success or failure.

4.20.10 Stop searching

BOOL StopSearch(void);

Return Values

Success or failure.

4.20.11 Register for tracking events

void RegisterTrackEvent(void *arg, HPDEVICE_TRACK_STATE_EVENT event, HPDEVICE_TRACK_AIM_EVENT aimEvent);

Note

Not supported by devices in 2019 and before.

4.20.12 Is tracking

BOOL IsTracking(void);

Return Values

True means tracking, False means not tracking.

4.20.13 Set the threshold for automatic tracking

BOOL SetAutoTrackThreshold(int threshold);

Parameters

threshold
[in] Threshold：1-255

Return Values

Success or failure.

Note

Not supported by devices in 2019 and before.

4.20.14 Switch tracking mode

BOOL Change Scene(void);

Note

Thermal IPC does not support this interface.

4.20.15 Switch tracking mode

BOOL ChangeAlgorithm(void);

Note

Thermal IPC does not support this interface.

4.20.16 Switch tracking mode

BOOL ChangeScene(int mode);

Parameters

mode

[in] Target tracking mode

-0：Semi-auto air；-1：Air-to-air；

-2：Manual on the ground；-3：Semi-automatic；-4：Automatic on the ground；

Return Values

Success or failure.

Note

Only thermal image IPC support.

5 Infrared thermal imaging IPC device interface

5.1 Infrared thermal imaging IPC device interface

```
#pragma once
#include "IRDSdkDefine.h"
#include "IHpIRDCore.h"
#include "IHpPtzCfg.h"
#include "IHpDeviceCfg.h"
#include "IHpSmartCfg.h"
#include "IHpAlarmCfg.h"
#include "IHpTrackCfg.h"
#include "IHpTempCfg.h"
#include "IHpAngle.h"
#include "IHpPreset.h"

namespace Hpws
{
	namespace IRD
	{
		/**
		* @brief Infrared thermal imaging ipc device interface
		* @note Does not support multithreading
		*/
		class IHpIRD
		{
		public:
			/**
			* @brief Initiate connection
			* @retv  The connection ID corresponds to the linkID parameter in the callback
			*                      -0    Connection failed
			*                      >0    Connection number
			*/
			virtual LONG Connect(char* ip, int port, char* user, char* pwd) = 0;
			/**
			* @brief Disconn
			*/
			virtual BOOL Disconnect(void) = 0;
			/**
			* @brief Is it connected
			*/
```

```
                virtual BOOL Isconnected(void) = 0;
                /**
                * @brief Register abnormal events
                * @param arg Callback event subscription object
                * @param event Callback event
                * @return no
                */
                virtual void RegisterExceptEvent(void* arg,
HPIRD_EXCEPT_EVENT event) = 0;
                /**
                * @brief Get connection number
                */
                virtual LONG GetLinkID(void) = 0;

        public:
                /**
                  * @briaf Register transparent data event
                  */
                virtual void RegisterSerialEvent(void* arg,
HPIRD_SERIAL_EVENT event) = 0;
                /**
                  * @briaf Binary data event
                  */
                virtual void RegisterBinaryEvent(void* arg,
HPIRD_BINARAY_EVENT binaryEvent) = 0;
                /**
                  * @briaf Send transparent data
                  */
                virtual BOOL SendSerialData(char* data, int len) = 0;
                /**
                  * @briaf Send request data
                  */
                virtual BOOL SendRequestData(BYTE type, BYTE flag,
char* data, int len) = 0;


                /**
                * @brief Get the movement configuration interface
                */
                virtual IHpIRDCore* GetIRDCorPtr(void) = 0;

                /**
                * @brief Get PTZ configuration interface
                */
```

```
                virtual IHpPtzCfg* GetPtzPtr(void) = 0;

                /**
                * @brief Get device configuration interface
                */
                virtual IHpDeviceCfg* GetDeviceCfgPtr(void) = 0;

                /**
                * @brief Get smart analysis configuration interface
                */
                virtual IHpSmartCfg* GetSmartCfgPtr(void) = 0;

                /**
                * @brief Get alarm configuration interface
                */
                virtual IHpAlarmCfg* GetAlarmCfgPtr(void) = 0;

                /**
                * @brief Get trace configuration interface
                */
                virtual IHpTrackCfg* GetTrackCfgPtr(void) = 0;

                /**
                * @brief Get temperature measurement configuration
interface
                */
                virtual IHpTempCfg* GetTempCfgPtr(void) = 0;

                /**
                * @brief get angle interface
                */
                virtual IHpAngle* GetAnglePtr() = 0;

                /**
                * @brief get Preset position interface
                */
                virtual IHpPreset* GetPresetPtr() = 0;
        };
    }
}
```

5.1.1 Create device pointer

```
IHpIRD* Create();
```

Return Values

Return the device pointer.

example

```
{
	IHpIRD* ird = Create();
	if (ird != nullptr)
	{
		//Business operations
		......
	}
}
```

5.1.2 Connection

LONG Connect(char* ip, int port, char* user, char* pwd);

Parameters

ip
	[in] ip address
port
	[in] The port number
user
	[in] username
pwd
	[in] password

Return Values

The connection ID corresponds to the linkID parameter in the callback
0 Connection failed
>0 Connection number

example

```
{
	if (ird->Connect(ip, port, user, pwd) != 0)
	{
		// connection succeeded
		.......
	}
}
```

5.1.3 Is connected

BOOL Isconnected(void);

Return Values

True means connected, False means not connected.

example

```
{
	if (ird->Isconnected())
	{
		// Already connected
	}
	else
	{	 // not connected

	}
}
```

5.1.4 Get connection number

LONG GetLinkID(void);

Return Values

Returns the connection number.

example

```
{
	//id is the connection number
	int id = ird->GetLinkID()
}
```

5.1.5 Register Except Event

void RegisterExceptEvent(void* arg, HPIRD_EXCEPT_EVENT event);

Parameters

arg
	[in] Callback event subscription object
event
	[in] Callback event

5.1.6 Register Serial Event

void Register Serial Event(void* arg, HPIRD_SERIAL_EVENT event);

Parameters

arg
[in] Callback event subscription object
event
[in] Callback event

5.1.7 Register Binary Event

void Register Binary Event(void* arg, HPIRD_BINARAY_EVENT binary Event);

Parameters

arg
[in] Callback event subscription object
event
[in] Callback event

5.1.8 Send Reques tData

BOOL SendRequestData(BYTE type, BYTE flag, char* data, int len);

Return Values

Success or failure.

5.1.9 Send Serial Data

BOOL SendSerialData(char* data, int len);

Return Values

Success or failure.

5.2 Get the movement configure interface

IHpIRDCore* GetIRDCorPtr(void);

Return Values

Return to the movement configuration interface.

example

```
{
    IHpIRDCore* core = GetIRDCorPtr(void);
    if (core != nullptr)
    {
        //Business operations
        ......
    }
}
```

5.2.1 Get image parameters

BOOL GetImgConfig(HPIRD_ImgConfigInfo* config)

Parameters

config

[out] The returned image parameter information (externally provided structure address).

Return Values

Success or failure.

example

```
{
    if (core->GetImgConfig())
    {
        // Get image parameter information
    }
}
```

5.2.2 Get the default image parameters

BOOL GetImgConfig(HPIRD_ImgConfigInfo* config)

Parameters

config

[out] The returned image parameter information (externally provided structure address).

Return Values

Success or failure.

example

```
{
	if (core->GetImgDefaultCfg())
	{
		// Get image parameter information
	}
}
```

5.2.3 Set image parameters

BOOL SetImgConfig(HPIRD_ImgConfigInfo* config)

Parameters

config
	[in] Image parameter information.

Return Values

Success or failure.

example

```
{
	if (core->SetImgConfig())
	{
		// Set image parameters
	}
}
```

5.2.4 Trigger movement self-test

BOOL TriggerSelfCheck(void)

Return Values

Success or failure.

example

```
{
	if (core->TriggerSelfCheck())
	{
```

```
        // Trigger movement self-test
    }
}
```

5.2.5 Trigger auto focus

BOOL TriggerAutofocus(void)

Return Values

Success or failure.

example

```
{
    if (core->TriggerSelfCheck())
    {
        // Trigger auto focus
    }
}
```

5.2.6 Trigger background correction

BOOL TriggerCalibBg(void)

Return Values

Success or failure.

example

```
{
    if (core->TriggerCalibBg())
    {
        // Trigger background correction
    }
}
```

5.2.7 Bad pixel correction selection coordinates

BOOL DefectiveChoose(int x, int y)

Parameters

x

[in] The coordinates of the x axis.

y

[in] The coordinates of the y axis.

Return Values

Success or failure.

example

```
{
    if (core->DefectiveChoose())
    {
        // Dead pixel correction selected coordinate successfully
    }
}
```

5.2.8 Dead pixel correction cancels the current coordinates

BOOL DefectiveCancel(void)

Return Values

Success or failure.

example

```
{
    if (core->DefectiveCancel())
    {
        // Dead pixel correction canceled the current coordinates successfully
    }
}
```

5.2.9 Bad pixel correction confirmation

BOOL DefectiveConfirm(void)

Return Values

Success or failure.

example

```
{
    if (core->DefectiveConfirm())
    {
```

```
        // Dead pixel correction canceled the current coordinates successfully
    }
}
```

5.2.10 Dead pixel correction save

BOOL DefectiveConfirm(void)

Return Values

Success or failure.

example

```
{
    if (core->DefectiveSave())
    {
        // Dead pixel correction saved successfully
    }
}
```

5.2.11 Reset factor has parameters

BOOL IRDRecovery(void)

Return Values

Success or failure.

example

```
{
    if (core->IRDRecovery())
    {
        // Reset successfully
    }
}
```

5.2.12 Get movement parameter information

BOOL GetIRDStatus(HPIRD_CoreInfo* status)

Parameters

status

[out] Returned movement status information (externally provided structure address).

Return Values

Success or failure.

example

```
{
	HPIRD_CoreInfo* status = new HPIRD_CoreInfo;
	if (core->GetIRDStatus(status))
	{
		// Get movement parameter information
	}
	delete status;
}
```

5.3 Get PTZ configuration interface

IHpPtzCfg* GetPtzPtr(void);

Return Values

Return to the PTZ configuration interface.

example

```
{
	IHpPtzCfg* ptz = GetPtzPtr();
	if (ptz != nullptr)
	{
		//Business operations
		......
	}
}
```

5.3.1 Turntable control

BOOL ControlMove(int type, BOOL move, int speed, int speedV)

Parameters

type

[in] The control command id, with a value range of 0-8, represents static, up, down, left, right, top left, bottom left, top right, and bottom right, respectively.

move

[in] TRUE or FALSE

speed

[in] Horizontal speed, value range：1-63

speedV

[in] Vertical speed, value range：1-63

Return Values

Success or failure

example

```
{
    if (ptz->ControlMove(HP_UP, TURE, 30, 30))
    {
        // Adjust the tilting turntable
        .......
    }
}
```

5.3.2 Lens control

BOOL CameraMove(int type, BOOL move, int video_type = FALSE)

Parameters

type

[in] The control command id, with a value range of 1-6, represents lens zoom in, lens zoom out, focus near, focus far, aperture zoom, and aperture zoom.

move

[in] TRUE or FALSE

video_type

[in] Video type: HP_light for visible light, HP_ird for thermal imaging

Return Values

Success or failure

example

```
{
```

```
    if (ptz->CameraMove(HP_ZOOM_IN, TURE))
    {
        // Visible lens zoom
        .......
    }
}
```

5.3.3 3D positioning

BOOL Position3D(int video_type, RECT select, BOOL large)

Parameters

light
[in] TRUE visible light, FALSE thermal imaging.
select
[in] select area
large
[in] TRUE zooms in, FALSE zooms out.

Return Values

Success or failure

example

```
{
    if (ptz->Position3D(TRUE, select, TRUE))
    {
        // Zoom in and locate the frame selection area of ??select
        .......
    }
}
```

5.3.4 Set focus speed

BOOL CameraMove(int type, BOOL move, int video_type = FALSE)

Parameters

speed
[in] Focus speed:0-100

Return Values

Success or failure

example

```
{
	if (ptz->SetFocusSpeed(50))
	{
		// Set focus speed successfully
		.......
	}
}
```

5.3.5 Enable image stabilization

BOOL EnableImgStabilizer(int orientation, BOOL bEnable)

Parameters

orientation
	[in] 0: horizontal 1: pitch
bEnable
	[in] bEnable Whether to open

Return Values

Success or failure

example

```
{
	if (ptz->EnableImgStabilizer(1, TRUE))
	{
		// Image stabilization enabled successfully
		.......
	}
}
```

5.3.6 Set image stabilization parameters

BOOL SetImgStabilizerPara(int orientation,int value)

Parameters

orientation
	[in] 0: horizontal 1: pitch
value
	[in] Zero offset correction value -32767--32767

Return Values

Success or failure

example

```
{
	if (ptz->SetImgStabilizerPara(1, 0))
	{
		// Successfully set image stabilization parameters
		.......
	}
}
```

5.4 Get device configuration interface

IHpDeviceCfg* GetDeviceCfgPtr(void);

Return Values

Return to the device configuration interface.

example

```
{
	IHpDeviceCfg* device = GetDeviceCfgPtr();
	if (device != NULL)
	{
		// Business operations
		.......
	}
}
```

5.4.1 Get device capability set

BOOL GetDevAbility(HPIRD_AbilityInfo* ability)

Parameters

ability

[out] Returned device capability set information (externally provided structure address)

Return Values

Success or failure

example

```
{
    HPIRD_AbilityInfo* ability = new HPIRD_AbilityInfo;
    if (device->GetDevAbility(ability))
    {
        // Successfully acquire device capability set
        .....
    }
}
```

5.4.2 Get basic device information

BOOL GetDevInfo(HPIRD_DeviceInfo* info)

Parameters

info

[out] Basic information of the device returned (externally provided structure address)

Return Values

Success or failure

example

```
{
    HPIRD_DeviceInfo* ability = new HPIRD_DeviceInfo;
    if (device->GetDevInfo(ability))
    {
        // Succeeded in obtaining basic device information
        .....
    }
}
```

5.4.3 Set basic device information

BOOL SetDevInfo(HPIRD_DeviceInfo* info)

Parameters

info

[in] Basic equipment information

Return Values

Success or failure

example

```
{
	if (device->SetDevInfo(info))
	{
		// Device basic information is set successfully
		.......
	}
}
```

5.4.4 Get device clock information

BOOL GetDevTimeInfo(HPIRD_TimeInfo* info)

Parameters

info

[out] Returned device clock information (externally provided structure address)

Return Values

Success or failure

example

```
{
	HPIRD_TimeInfo* info = new HPIRD_TimeInfo;
	if (device->GetDevTimeInfo(info))
	{
		// Obtain device clock information successfully
		.......
	}
}
```

5.4.5 Set device clock information

BOOL SetDevTimeInfo(HPIRD_TimeInfo* info)

Parameters

info

[in] Device clock information

Return Values

Success or failure

example

```
{
    HPIRD_TimeInfo* info = new HPIRD_TimeInfo;
    if (device->SetDevTimeInfo(info))
    {
        // Set device clock information successfully
        .....
    }
}
```

5.4.5 Get device network information

BOOL GetDevNetInfo(HPIRD_NetInfo* info)

Parameters

info

[out] Returned device network information (externally provided structure address)

Return Values

Success or failure

example

```
{
    HPIRD_NetInfo* info = new HPIRD_NetInfo;
    if (device->GetDevNetInfo(info))
    {
        // Successfully obtain device network information
        .....
    }
}
```

5.4.6 Set device network information

BOOL SetDevNetInfo(HPIRD_NetInfo* info)

Parameters

info

[in] Device network information

Return Values

Success or failure

example

```
{
    if (device->SetDevNetInfo(info))
    {
        // Set device network information successfully
        .....
    }
}
```

5.4.7 Change device password

BOOL ChangePassword(char* user, char* pwd, char* newUser, char* newPwd)

Parameters

user

[in] User

pwd

[in] Password

newUser

[in] New User

newPwd

[in] New Password

Return Values

Success or failure

example

```
{
    if (device->ChangePassword(user, pwd, newUser, newPwd))
    {
        // Successfully modify the device password
        .....
    }
```

}

5.4.8 Get the thermal image channel number

int GetIrdChannel(void)

Return Values

0 No such channel
>0 Corresponding channel number

example

```
{
	int res = device->GetIrdChannel(info);
	if (res > 0)
	{
		// Get the thermal image channel number successfully
		.....
	}
}
```

5.4.9 Obtain the visible light channel number

int GetLightChannel(void)

Return Values

0 No such channel
>0 Corresponding channel number

example

```
{
	int res = device->GetLightChannel(info);
	if (res > 0)
	{
		// Obtain the visible light channel number successfully
		.....
	}
}
```

5.4.10 Get the video channel number

int GetVideoChannel(int video_type)

Parameters

video_type
[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel

Return Values

0 No such channel
>0 Corresponding channel number

example

```
{
    int res = device->GetVideoChannel(HP_light);
    if (res > 0)
    {
        // Get video channel number successfully
        ......
    }
}
```

5.4.11 Get video type

int GetVideoType(int channel)

Parameters

channel
[in] Video channel number

Return Values

<0 failure

example

```
{
    int res = device->GetVideoType(1);
    if (res == 0 || res == 1)
    {
        // Get video type
        ......
    }
}
```

5.4.12 Power control

BOOL PowerCtl(int video_type, BOOL open)

Parameters

video_type
[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel
open
[in] On or off

Return Values

<0 failure

example

```
{
    if (device->PowerCtl(1, TRUE);)
    {
        // Control power on
        ......
    }
}
```

5.5 Get smart analysis configuration interface

IHpSmartCfg* GetSmartCfgPtr(void);

Return Values

Return to the intelligent analysis configuration interface.

example

```
{
    IHpSmartCfg* smart = GetSmartCfgPtr();
    if (smart != NULL)
    {
        // Business operations
        .......
    }
}
```

5.5.1 Get smart analysis configuration information 息

BOOL GetIVPCfg(int video_type, HPIRD_AICfgInfo** aiCfg, int type)

BOOL GetDefaultIVPCfg(int video_type, HPIRD_AICfgInfo** aiCfg, int type)

Parameters

video_type

[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel

aiCfg

[in] The returned intelligent analysis configuration information (empty structure address provided externally, aiCfg must not be released externally).

type

[in] 0: Leftover detection; 1: Object removal detection; 2: Tripwire detection; 3: Intrusion detection; 4: Target tracking;
5: Enter the area; 6: Leaving the area; 7: Wandering detection; 8: Crowd gathering detection; 9: Fast moving; 10: Target classification
11: face detection; 12: temperature measurement; 13: thermal alarm; 14: motion detection

Return Values

Success or failure

example

```
{
    HPIRD_AICfgInfo** aiCfg;
    if (smart->GetIVPCfg(HP_light, aiCfg, 0))
    {
        // Successfully obtain intelligent analysis configuration information
        .....
    }
}
```

5.5.2 Set intelligent analysis configuration information 息

BOOL SetIVPCfg(int video_type, HPIRD_AICfgInfo* config, int type)

Parameters

video_type

[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel

Return Values

Success or failure

example

```
{
    HPIRD_AICfgInfo** aiCfg;
    if (smart->GetIVPCfg(HP_light, aiCfg, 0))
    {
        // Successfully obtain intelligent analysis configuration information
        .....
    }
}
```

5.5.3 Set function switch information

BOOL SetGateCfg(int video_type, PHPIRD_AIGateInfo gateCfg, int type)

Parameters

video_type
[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel
gateCfg
[in] Intelligent analysis function switch configuration information definition
type
[in] 0: Leftover detection; 1: Object removal detection; 2: Tripwire detection; 3: Intrusion detection; 4: Target tracking;
5: Enter the area; 6: Leaving the area; 7: Wandering detection; 8: Crowd gathering detection; 9: Fast moving; 10: Target classification
11: face detection; 12: temperature measurement; 13: thermal alarm; 14: motion detection

Return Values

Success or failure

example

```
{
    PHPIRD_AIGateInfo gateCfg;
```

```
    // Set the parameters of the gateCfg object
    gateCfg->enable = 1;
    ......

    if (smart->SetGateCfg(HP_light, gateCfg, 0))
    {
        // Successfully set intelligent analysis configuration information
        .......
    }
}
```

5.5.4 Set preset scene information

BOOL SetSceneCfg(int video_type, int preset, PHPIRD_AIPresetCfgInfo sceneCfg, int type)

Parameters

video_type
: [in] Video type: HP_light visible light channel, HP_ird thermal imaging channel

preset
: [in] When preset = 0, it is the default scene

sceneCfg
: [in] Intelligent analysis of preset scene configuration information definition

type
: [in] 0: Leftover detection; 1: Object removal detection; 2: Tripwire detection; 3: Intrusion detection; 4: Target tracking;
5: Enter the area; 6: Leaving the area; 7: Wandering detection; 8: Crowd gathering detection; 9: Fast moving; 10: Target classification
11: face detection; 12: temperature measurement; 13: thermal alarm; 14: motion detection

Return Values

Success or failure

example

```
{
    if (smart->SetSceneCfg(HP_light, 0, sceneCfg, 0))
    {
        // Set the preset scene information successfully
        .......
    }
```

}

5.5. 5 Set intelligent analysis configuration information

void RegisterAlarmIvpEvent(void* arg, HPIRD_ALARMIVP_EVENT ivpEvent)

Parameters

arg
[in] Callback event subscription object value
event
[in] Intelligent analysis of alarm events

Get alarm configuration interface

IHpAlarmCfg* GetAlarmCfgPtr(void);

Return Values

Return to the alarm configuration interface.

example

```
{
    IHpAlarmCfg* alarm = GetAlarmCfgPtr();
    if (alarm != NULL)
    {
        // Business operations
        .....
    }
}
```

5.6 Get alarm configuration interface

IHpAlarmCfg* GetAlarmCfgPtr(void);

Return Values

Return to the alarm configuration interface.

example

```
{
    IHpAlarmCfg* alarm = GetAlarmCfgPtr();
    if (alarm != NULL)
```

```
    {
        // Business operations
        .......
    }
}
```

5.6.1 Set alarm output

BOOL SetAlarmOut(BOOL enable)

Parameters

enable
[in] TRUE turns on the alarm output, FALSE turns off the alarm output

Return Values

Success or failure

example

```
{
    if (alarm->SetAlarmOut(TRUE))
    {
        // Turn on the alarm output
        .......
    }
}
```

5.6.2 Temperature data output

BOOL OutputTempInfo(HPIRD_TempOutputInfo* tempInfo)

Parameters

tempInfo
[in] Temperature information

Return Values

Success or failure

example

```
{
    HPIRD_TempOutputInfo* tempInfo;
```

```
    if (alarm->OutputTempInfo(tempInfo))
    {
        // Get temperature data output
        .......
    }
}
```

5.6.3 Register Alarm Event

void Register Alarm Event (void* arg, HPIRD_ALARM_EVENT event)

Parameters

arg
[in] Callback event subscription object value
event
[in] Callback event

5.7 Get trace configuration interface

IHpTrackCfg* GetTrackCfgPtr(void);

Return Values

Return to the trace configuration interface.

example

```
{
    IHpTrackCfg* track = GetTrackCfgPtr();
    if (track != NULL)
    {
        // Business operations
        .......
    }
}
```

5.7.1 Start tracking

BOOL StartTrack(int video_type)

Parameters

video_type

[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel

Return Values

Success or failure

example

```
{
    if (track->StartTrack(HP_light))
    {
        // Start tracking visible light
        .....
    }
}
```

5.7.2 Frame tracking

BOOL Select(int video_type, RECT video, RECT select)

Parameters

video_type
[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel
video
[in] Video display area
select
[in] select area

Return Values

Success or failure

5.7.3 Stop tracking

BOOL StartTrack(int video_type)

Parameters

video_type
[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel

Return Values

Success or failure

example

```
{
	if (track->StopTrack(HP_light))
	{
		// Stop tracking visible light
		.......
	}
}
```

5.7.4 Is tracking

BOOL IsTracking()

Return Values

Success or failure

example

```
{
	if (track->IsTracking())
	{
		// Following
		.......
	}
	else
	{
		// Not tracked
		.......
	}
}
```

5.7.5 Switch tracking mode

BOOL ChangeScene(int video_type, int mode)

Parameters

video_type
	[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel
mode

[in] Target tracking mode: 0: semi-automatic against air; 1: automatic against air;
2: Manual to the ground; 3: Semi-automatic to the ground; 4: Automatic to the ground

Return Values

Success or failure

example

```
{
    if (track->ChangeScene(HP_light, 0))
    {
        // Switch the visible light target tracking mode to air semi-automatic
        .......
    }
}
```

5.7.6 Get current tracking mode

int GetCurrentScene(int video_type)

Parameters

video_type
[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel

Return Values

-1 Get failed

example

```
{
    if (track->GetCurrentScene(HP_light) != -1)
    {
        // Get the current tracking mode of visible light
        .......
    }
}
```

5.7.8 Register to track related events

void Registe rEvent(void* arg, HPIRD_TRACK_EVENT event Track, HPIRD_MISS_EVENT event Miss)

Parameters

arg
[in] Callback event subscription object value
eventTrack
[in] Track status event
eventMiss
[in] Missing information event

5.7.9 Enable/disable the tracking ability of the channel

BOOL EnableTrackAbility(int video_type, BOOL enable)

Parameters

video_type
[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel
enable
[in] TRUE or FALSE

Return Values

Success or failure

example

```
{
    if (track->EnableTrackAbility(HP_light, TRUE))
    {
        // Enable visible light channel tracking capability
        .....
    }
}
```

5.8 Get temperature measurement configuration interface

IHpTempCfg* GetTempCfgPtr(void);

Return Values

Return to the temperature measurement configuration interface.

example

```
{
	IHpTempCfg* temp = GetTempCfgPtr();
	if (temp != NULL)
	{
		// Business operations
		.......
	}
}
```

5.8.1 Set function switch information

BOOL SetGateCfg(int video_type, PHPIRD_AIGateInfo gateCfg)

Parameters

video_type
[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel
gateCfg
[in] Intelligent analysis function switch configuration information

Return Values

Success or failure

example

```
{
	PHPIRD_AIGateInfo gateCfg;
	if (temp->SetGateCfg(HP_light, gateCfg))
	{
		// Set function switch information successfully
		.......
	}
}
```

5.8.2 Set preset scene information

BOOL SetSceneCfg(int video_type, int preset, PHPIRD_AIPresetCfgInfo sceneCfg)

Parameters

video_type

[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel

preset

[in] = 0: the default scene

sceneCfg

[in] Intelligent analysis of preset scene configuration information

Return Values

Success or failure

example

```
{
    PHPIRD_AIGateInfo gateCfg;
    if (temp->SetSceneCfg(HP_light, 0, sceneCfg))
    {
        // Set the preset scene information successfully
        .......
    }
}
```

5.8.3 Set temperature configuration information

BOOL SetTempCfg(int video_type, PHPIRD_AITempCfgInfo tempCfg)

Parameters

video_type

[in] Video type: HP_light visible light channel, HP_ird thermal imaging channel

tempCfg

[in] Temperature measurement related information

Return Values

Success or failure

example

```
{
    PHPIRD_AITempCfgInfo tempCfg;
    if (temp->SetTempCfg  (HP_light, tempCfg))
    {
        // Set the temperature configuration information successfully
```

```
            .......
        }
}
```

5.8.4 Register regional temperature measurement alarm events

void Register Event (void* arg, HPIRD_AREATEMP_EVENT event)

Parameters

arg
[in] Callback event subscription object value
event
[in] Regional temperature measurement alarm event

5.9 Get the angle management interface

IHpAngle* GetAnglePtr();

Return Values

Return to the angle management interface.

example

```
{
    IHpAngle* angle = GetAnglePtr();
    if (angle != NULL)
    {
        // Business operations
        .......
    }
}
```

5.9.1 Angle positioning / PTZ moving speed

BOOL SetSpeed(int speed)

Parameters

speed
[in] range：0~63

Return Values

Success or failure

example

```
{
	if (angle->SetSpeed(10))
	{
		// Movement speed set successfully
		.......
	}
}
```

5.9.2 Horizontal angle positioning

BOOL PositionX(double angle, int speed)

Parameters

angle
	[in] Horizontal angle
speed
	[in] range：0~63

Return Values

Success or failure

example

```
{
	if (angle->PositionX(30, 10))
	{
		// The horizontal angle positioning is set successfully
		.......
	}
}
```

5.9.3 Pitch angle positioning

BOOL PositionY(double angle, int speed)

Parameters

angle
	[in] Pitch angle
speed

[in] range：0~63

Return Values

Success or failure

example

```
{
    if (angle->PositionY(30, 10))
    {
        // The pitch angle positioning is set successfully
        .......
    }
}
```

5.9.4 Field of view angle positioning

BOOL PositionZ(double angle, int video_type)

Parameters

angle
[in] Field of view angle
video_type
[in] Video type: HP_light visible light, HP_ird thermal imaging

Return Values

Success or failure

example

```
{
    if (angle->PositionZ(30, HP_light))
    {
        // The field of view angle positioning is set successfully
        .......
    }
}
```

5.9.5 Register angular events

void RegisterEvent(void* arg, HPIRD_ANGLEX_EVENT eventX,
HPIRD_ANGLEY_EVENT eventY

, HPIRD_ANGLEZ_IRD_EVENT eventZIrd,
HPIRD_ANGLEZ_LIGHT_EVENT eventZLight)

Parameters

arg
[in] Callback event subscription object value
event
[in] Regional temperature measurement alarm event
view
[in] Field of view angle event

5.10 Get preset management interface

IHpPreset* GetPresetPtr();

Return Values

Return to the preset management interface.

example

```
{
    IHpPreset* preset = GetPresetPtr();
    if (preset != NULL)
    {
        // Business operations
        .....
    }
}
```

5.10.1 Call presets

BOOL Call (int preset)

Parameters

preset
[in] Preset number

Return Values

Success or failure

example

```
{
	if (preset->Call(1))
	{
		// Call preset position 1
		.......
	}
}
```

5.10.2 Set preset position

BOOL Set (int preset)

Parameters

preset
	[in] Preset number

Return Values

Success or failure

example

```
{
	if (preset->Set(1))
	{
		// Set preset position 1
		.......
	}
}
```

5.10.3 Clear preset

BOOL Clear(int preset)

Parameters

preset
	[in] Preset number

Return Values

Success or failure

example

```
{
	if (preset->Clear(1))
	{
		// Clear preset position 1
		.......
	}
}
```

5.10.4 Clear all presets

BOOL Clear(void)

Return Values

Success or failure

example

```
{
	if (preset->Clear())
	{
		// Clear all presets successfully
		.......
	}
}
```

5.11 Disconnect

BOOL Disconnect(void);

Return Values

Success or failure.

example

```
{
	if (ird->Disconnect())
	{
		// Disconnect
	}
}
```

5.12 Release device pointer

void Release(IHpIRD* ird)

Parameters

ird
[in] Device pointer

example

```
{
    if (ird != nullptr)
    {
        Release(ird);
    }
}
```

6 Precautions

SDK Use Matters

6.1 For the anti-drone service, after the device is connected, the channel needs to be set to 2 when opening the tracking video, the first channel is the conventional thermal imaging video, and the second channel is the device Tracking video, if the device is a cooling device, the thermal imaging video is turned off by default, and the thermal imaging needs to be powered on in advance.

6.2 The callback function of the field of view angle includes the field of view angles of the two lenses of visible light and thermal imaging, which are distinguished by whether the parameter light is TRUE.

6.3 The angle monitoring module involves opening the transparent transmission connection of the device. At present, conventional devices only support opening once at the same time. Once one is opened successfully, the other places cannot be opened again. Open unless the first place is closed. If the transparent transmission is not turned off before disconnecting at the first place, the second place cannot be opened either. Either the first place is connected again, closed, or restarted The device can only be turned on in the second place.

6.4 During tracking, because the target will be lost during the focusing process, it is recommended to turn off the auto focus during the tracking process; before tracking, find the farthest target and focus clearly, so that the tracking Focusing is no

longer needed during the process, and the whole process is clear. Avoid losing the target due to focusing.

Disconnect

BOOL Disconnect(void);

Return Values

Success or failure.

example

```
{
	if (ird->Disconnect())
	{
		// Disconnect
	}
}
```

Release device pointer

void Release(IHpIRD* ird)

Parameters

ird

[in] Device pointer

example

```
{
	if (ird != nullptr)
	{
		Release(ird);
	}
}
```