Version1.2

2023-05-08



IPC Application interface

API reference

# IPC Application interface

## Preface

### Reader

* The reference is intended for programmers and describes various references for IPC application library development. Programmers using this reference should:
*  Familiar with C/C++ language
*  Master basic library function calls
* Familiar with Windows or Linux development environment

### Content introduction

The reference begins with an overview of the IPC application library API functions and their associations, and then details the various reference information separately. The reference is organized as follows.

| **Chapter** | **Content** |
| --- | --- |
| **Description** | The section describes the components of the IPC application library development kit and the software and hardware development environment.After reading this topic, you will have an overall understanding of the client-side IPC development library |
| **API Function** | This topic gives you access to the API reference information of the development library and describes each API interface function in detail |
| **Data Type** | This section describes the common data type definitions and structure definitions used by the API |
| **API Application Example** | The paper introduces how to use the development library API through examples. |

### Format convention

| **FORMAT** | **Description** |
| --- | --- |
| **Boldface** | Main body and title |
| **Regular Script** | Warnings, prompts and other content should be in italics, and lines and text isolation should be added before and after the content |
| **Aequilate** | The code is in monospaced fonts |

### Modification record

| **Version** | **Date** | **Modify Description** |
| --- | --- | --- |
| 1. **.2** | 2023-05-08 | Add PTZ control function declaration |
| **1.1** | 2022-12-08 | Add API function declaration |
| **1.0** | 2022-05-08 | The first version |

Contents

[IPC Application program interface 1](#_Toc121494686)

[Preface 1](#_Toc121494687)

[Reader 1](#_Toc121494688)

[Content introduction 1](#_Toc121494689)

[Format convention 1](#_Toc121494690)

[Modify record 1](#_Toc121494691)

[Description 3](#_Toc121494692)

[Description range 3](#_Toc121494693)

[Function list 4](#_Toc121494694)

[Function description way 4](#_Toc121494695)

[Strucure description way 5](#_Toc121494696)

[Function declaration 6](#_Toc121494697)

[NETSDK\_Init 6](#_Toc121494698)

[NETSDK\_Cleanup 6](#_Toc121494699)

[NETSDK\_SearchDevice 7](#_Toc121494700)

[NETSDK\_Login 8](#_Toc121494701)

[NETSDK\_Logout 9](#_Toc121494702)

[NETSDK\_CreateRealPlay 10](#_Toc121494703)

[NETSDK\_ DestoryRealPlay 10](#_Toc121494704)

[NETSDK\_ StartRealPlay 11](#_Toc121494705)

[NETSDK\_ SetRealStreamStatus 12](#_Toc121494706)

[NETSDK\_ StopRealPlay 14](#_Toc121494707)

Data type and data structure  [14](#_Toc121494708)

General data type description [14](#_Toc121494709)

[Data structure description 15](#_Toc121494710)

[API Application example 18](#_Toc121494711)

[Gain code stream flow chat 18](#_Toc121494712)

[Process example 19](#_Toc121494713)

## Description

### description range

IPC application library software provided by the company is a set of high performance, high reliability, good compatibility of the application development kit. It completes the main process of IPC communication and access to streaming data, and provides a flexible and simple API for users to quickly develop applications.

The software provides users with a variety of platforms under the static library call form, can be more convenient to develop applications. Table 1 lists the main components and their descriptions.

List 1

| **Component** | **Name** | **Explain** |
| --- | --- | --- |
| **API Interface** | ipsdk\_client.h  ipsdk\_data\_def.h  ipsdk\_debug.h  ipsdk\_def\_type.h  ipsdk\_erp.h  ipsdk\_server.h  ipsdk\_version.h | In the user project, you only need to include ipsdk\_client.h, and this file will contain other required files |
| **Static Library** | yssdk\_client.lib | Static libraries are multi-platform (Windows/Linux), multi-architecture (x86 and x64), multi-target (Debug/Release), and compile to link the correct version |
| **Example** |  |  |

Users can develop library based applications on a variety of compilation environments, compatible with Microsoft's Windows 7 or later mainstream Windows operating systems, and compatible with the vast majority of PC oriented CPU chipsets released by Intel and AMD since 2002. Table 1-2 describes the main development and operating environment.

| **Classify** | **Compatible configuration** | **Recommend Configuration** | **Explain** |
| --- | --- | --- | --- |
| **Compiler** | Visual Studio 2015 | Visual Studio 2017 |  |
| **Operation System** | Windows 7  Windows 8  Windows 10  Windows 11 | Windows 10+ |  |
| **Hardware** | Intel P3 Series  Intel P4 Series  Intel Core Series  AMD Athlon64 Series  AMD Sempron Series  AMD Athlon Series | The CPU frequency is more than 3.0GHz and Memory is larger than 512MB. |  |

### Function list

| **FUNCTION** | **Description** |
| --- | --- |
| **NETSDK\_Init** | Establish, initialize SDK library |
| **NETSDK\_Cleanup** | Clean、Destroy allocated memory |
| **NETSDK\_SearchDevice** | Search for IPC devices on the network |
| **NETSDK\_Login** | Login target IPC |
| **NETSDK\_Logout** | Log out |
| **NETSDK\_CreateRealPlay** | Create stream fetch object handle |
| **NETSDK\_DestoryRealPlay** | Close and destroy the stream fetch object handle |
| **NETSDK\_StartRealPlay** | Get real-time streaming data |
| **NETSDK\_SetRealStreamStatus** | Notifications of the corresponding real-time stream status |
| **NETSDK\_StopRealPlay** | Stop notifications of the corresponding real-time stream status |
| **NETSDK\_ProductTest** | PTZ operation |

### Function description way

The chapter describes API reference information in six domains.

| **Parameter field** | **Function** |
| --- | --- |
| **Purpose** | Briefly describe the main functionality of the API. |
| **Grammar** | Lists the syntax style of the API. |
| **Description** | Briefly describe how the API works. |
| **Parameter** | Lists the API parameters, parameter descriptions, and parameter properties. |
| **Returned value** | Lists the return value of the API and the return value description. |
| **Attention** | Considerations when using API |

### Structure description

| **Parameter filed** | **Function** |
| --- | --- |
| **Declare** | Briefly describe what the structure does. |
| **Definition** | Lists the definition of the structure |
| **Matters need attention** | Lists the considerations for structures |

## Function explain

### NETSDK\_Init

#### Purpose

Establish ,initialize

#### Grammar

*IPSDK\_Int* NETSDK\_Init();

#### Description

#### Initializes the SDK library and allocates context memory and environment

#### Parameter

Nothing

#### Returned value

| **Returned value** | **Description** |
| --- | --- |
| **0** | Success |
| **Is not 0** | Initialization failed, memory allocation failed, or the network is unavailable |

#### Attention

### The function must be called before all function calls, otherwise none of the functions will work properly

### NETSDK\_Cleanup

#### Purpose

Clean ,destroy SDK library

#### Grammar

*IPSDK\_Int* NETSDK\_Cleanup();

#### Description

#### After the end of use, destroy the memory space allocated when the SDK is working to prevent memory leaks

#### Parameter

Nothing

#### Returned value

| **Returned value** | **Description** |
| --- | --- |
| **0** | Success |
| **Is not 0** | Failure |

#### Attention

The function must be called at the end, and repeated destruction is not allowed.

### NETSDK\_SearchDevice

#### Purpose

Get basic information about all devices on the network

#### Grammar

*IPSDK\_Int* NETSDK\_SearchDevice(*fSearchDevCallBack* *fSearchCallback*,

*IPSDK\_PVoid* *lParam*, *IPSDK\_Int* *nTimeout*, *IPSDK\_ULong* *ulBindAddr*);

#### Description

#### You can use this function to view the number of devices in the current network and the basic information about each device.

#### Parameter

| **Parameter** | **Data range** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **fSearchCallback** | - | Input | Address of the callback function |
| **lParam** |  | Input | Parameters provided to the callback function |
| **nTimeout** | 0 – ∞ | Input | Network communication wait timeout limitation |
| **ulBindAddr** | - | Input | IP address of the local device |

#### Returned value

| **Returned value** | **Description** |
| --- | --- |
| **0** | Successful call |
| **Is not 0** | Function call error |

#### Attention

The function uses callbacks, so the fSearchCalback callback function must be declared and defined before it can be used. The callback function is declared as follows:

typedef *IPSDK\_Int*(IPSDK\_CALLBACK \**fSearchDevCallBack*)(*IPSDK\_Int* nStateCode, *IPSDK\_SearchDevice*\* pSearchDevice, *IPSDK\_PVoid* pUserParam);

##### Callback function

| **Parameter** | **Member** | **Description** |
| --- | --- | --- |
| **nStateCode** |  | Notice code |
| **pSearchDevice** | strDeviceId | Device ID |
|  | stLanInfo | Device network information (including network address, communication port, etc.) |
|  | stDeviceInfo | Device information (including version, hardware, cloud ID, etc.) |
| **pUserParam** |  | The value passed to lParam when the NETSDK SearchDevice is called |

When the function finds a device during execution, it notifies the user through the callback function, and the user can save the device information for use in subsequent execution.

### NETSDK\_Login

#### Target

#### Log in to the target IPC to create a valid session handle

#### Grammar

*IPSDK\_HLOGIN* NETSDK\_Login(const *IPSDK\_Char*\* *pstrHostAddr*, *IPSDK\_Int* *nSessionPort*, *IPSDK\_Int* *nStreamPort*, const *IPSDK\_Char*\* *pstrUserName*, const *IPSDK\_Char*\* *pstrPassword*, *IPSDK\_PVoid* *lParam*);

#### Description

#### The session is initialized, the communication channel with IPC is established, and the IPC verifies the validity of the user

#### Parameter

| **Parameter** | **Value range** | **Input/output** | **Description** |
| --- | --- | --- | --- |
| **pstrHostAddr** | - | Input | IP address of the target IPD |
| **nSessionPort** | 1-65535 | Input | Talk port |
| **nStreamPort** | 1-65535 | Input | Stream acquisition port |
| **pstrUserName** |  | Input | User name |
| **pstrPassword** |  | Input | User password |
| **lParam** |  |  | Reserved parameter |

#### Returned value

| **Returned value** | **Description** |
| --- | --- |
| **0** | Login failure |
| **Is not 0** | If the login succeeds, the returned value is session handle |

#### Attention

Nothing

### NETSDK\_Logout

#### Target

#### Destroy the session handle

#### Grammar

*IPSDK\_Int* NETSDK\_Logout(*IPSDK\_HLOGIN* *hUserId*);

#### Description

#### After the session ends, the allocated memory space is destroyed to prevent memory leaks

#### Parameter

| **Parameter** | **Value range** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **hUserId** | - | Input | Session handle to be destroyed |

#### Returned value

| **Returned value** | **Description** |
| --- | --- |
| **0** | Success |
| **Is not 0** | Failure |

### Attention

### The validity of the session handle is guaranteed by the user and repeated destruction is not allowed.

### Manually empty the destroyed handle.NETSDK\_CreateRealPlay

#### Target

#### Creates a valid channel handle

#### Grammar

*IPSDK\_HCHANNEL* NETSDK\_CreateRealPlay(*IPSDK\_HLOGIN* *hHandle*, const *IPSDK\_ChannelInfo*\* *pChannelInfo*);

#### Description

After establishing the session, establish and IPC channels to prepare for playback.

#### Parameter

| **Parameter** | **Member** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **hHandle** | - | Input | Session handle |
| **pChannelInfo** | nChannel | Input | Device channel |
|  | nStream | Input | Code stream channel |

#### Returned value

| **Returned value** | **Description** |
| --- | --- |
| **0** | Channel establishment failure |
| **Is not 0** | The channel is successfully created, and the return value is the channel handle |

#### Attention

### The user must ensure the validity of the session handle.

### Currently, the device channel is usually 0.

### There are currently two code channels, 0 representing the primary code stream and 1 representing the secondary code stream.

### NETSDK\_ DestoryRealPlay

#### Target

#### Destroy the channel handle

#### Grammar

*IPSDK\_Int* NETSDK\_DestoryRealPlay(*IPSDK\_HCHANNEL* *hChannel*);

#### Description

#### After the fetching stream is complete, the allocated memory space is destroyed to prevent memory leaks

#### Parameter

| **Parameter** | **Value range** | **Input/output** | **Description** |
| --- | --- | --- | --- |
| **hChannel** | - | Input | The channel handle to be destroyed |

#### Returned value

| **Returned value** | **Description** |
| --- | --- |
| **0** | Success |
| **Is not 0** | Failure |

#### Attention

The validity of the channel handle is guaranteed by the user and repeated destruction is not allowed.

Manually empty the destroyed handle.

### NETSDK\_ StartRealPlay

#### Target

Get the data flow in IPC

#### Grammar

*IPSDK\_Int* NETSDK\_StartRealPlay(*IPSDK\_HCHANNEL* *hChannel*, *fRealStreamCallBack* *cbRealDataCallBack*, *IPSDK\_PVoid* *pUserData*, *IPSDK\_Int* *bBlocked*);

#### Description

Users can use this function to obtain the real-time IPC code stream.

#### Parameter

| **Parameter** | **Value range** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **hChannel** | - | Input | Channel handle |
| **cbRealDataCallBack** | - | Input | A callback function that receives streaming data |
| **pUserData** | - | Input | Parameters reserved for the user |
| **bBlocked** | TRUE/FALSE | Input | Synchronous/asynchronous mode |

#### Returned value

| **Returned value** | **Description** |
| --- | --- |
| **0** | Successful call |
| **Is not 0** | Function call error |

#### Attention

The function uses callbacks, so the fRealStreamCallBack callback function must be declared and defined before it can be used：

typedef *IPSDK\_Int*(IPSDK\_CALLBACK \**fRealStreamCallBack*)(*IPSDK\_HANDLE* lRealHandle, *IPSDK\_Int* nChannel, *IPSDK\_Int* nStream, *IPSDK\_Int* nDataType, *IPSDK\_PChar* pDataBuf, *IPSDK\_Int* nDataLen, *IPSDK\_PVoid* pUserBuf);

##### Description of callback functions

| **Parameter** | **Description** |
| --- | --- |
| **lRealHandle** | Channel handle |
| **nChannel** | Channel number |
| **nStream** | Code stream number |
| **nDataType** | Data type |
| **pDataBuf** | Start address of the stream data |
| **nDataLen** | Stream data length |
| **pUseBuf** | The value passed to pUserData when NETSDK\_StartRealPlay is called |

### In the execution of the function, every data packet obtained will be provided to the user through the callback function, and the user can save, decode and play, transcode and other operations after obtaining the data in the callback function.NETSDK\_ SetRealStreamStatus

#### Target

#### In the process of obtaining data flow, understand the current communication status in real time

#### Grammar

*IPSDK\_Int* NETSDK\_SetRealStreamStatus(*IPSDK\_HCHANNEL* *hChannel*, *fStatusEventCallBack* *fStatusEvent*, *IPSDK\_PVoid* *lParam*);

#### Description

#### Users can use this function to obtain the real-time communication status of IPC.

#### The user can exit by returning non-0 data in the fStatusEvent callback function.

#### Parameter

| **Parameter** | **Value range** | **Input/output** | **Description** |
| --- | --- | --- | --- |
| **hChannel** | - | Input | Channel handle |
| **fStatusEvent** | - | Input | A callback function that receives status notifications |
| **lParam** | - | Input | Parameters reserved for the user |

#### Returned value

| **Returned value** | **Description** |
| --- | --- |
| **0** | Successful call |
| **Is not 0** | Function call error |

#### Attention

This function uses callbacks, so the fStatusEventCallBack callback function must be declared and defined before it can be used

typedef *IPSDK\_Int*(IPSDK\_CALLBACK \**fStatusEventCallBack*)(*IPSDK\_Int* nStateCode, *IPSDK\_PChar* pResponse, *IPSDK\_PVoid* pUserParam);

##### Description of callback function declaration

| **Parameter** | **Description** |
| --- | --- |
| **nStateCode** | Status code |
| **pResponse** | Information content |
| **pUserParam** | The value passed to lParam when calling NETSDK\_SetRealStreamStatus |

During the execution of the function, the callback function provides the current running status to the user, the user understands the real-time communication through the status code, and can immediately exit the running process of the current obtained code stream by returning a non-0 code.

### NETSDK\_ StopRealPlay

#### Target

#### Stop the stream fetching process

#### Grammar

*IPSDK\_Int* NETSDK\_StopRealPlay(*IPSDK\_HCHANNEL* *hChannel*);

#### Description

Stop the stream fetching process created by StartRealPlay

#### Parameter

| **Parameter** | **Value range** | **Input/output** | **Description** |
| --- | --- | --- | --- |
| **hChannel** | - | Input | Channel handle |

#### Returned value

| **Returned value** | **Description** |
| --- | --- |
| **0** | Success |
| **Is not 0** | Failure |

#### Attention

### The validity of the channel handle is guaranteed by the user.

### NETSDK\_ ProductTest

#### Target

#### Send instruction ID and parameters containing JSON content to the target IPC to achieve the effect of operating various functions of the PTZ

#### Grammar

*IPSDK\_Int* NETSDK\_ProductTest(*IPSDK\_HLOGIN* *hUserId*, *IPSDK\_Int nTestId*, *fProductTestCallBack* *cbProductTest*, *IPSDK\_CPChar* *pParamJson*, *IPSDK\_PVoid* *pUser*);

Description

The user can use this function to move the ptz (up and down left and right), stop, reset and adjust the lens (zoom, aperture, zoom).

Users get more feedback from IPC in the cbProductTest callback function.

#### Parameter

| **Parameter** | **Value range** | | **Input/Output** | **Description** |
| --- | --- | --- | --- | --- |
| **hUserId** | - | | Input | You can obtain the current Login handle of the user through NETSDK Login |
| **nTestId** | 80-85 | | Input | PTZ operation code |
| 80 | pParamJson = {"direction":"up","speed":1} | 云台移动   * direction: PTZ rotation direction，up/down/left/rightCorresponds to the up/down/left and right directions * speed:Rotation speed of the PTZ（1-4） |
| 81 | pParamJson = nullptr | PTZ stop |
| 82 | pParamJson = nullptr | PTZ reset |
| 83 | pParamJson = nullptr | Example Query the preset bit information about the cloud platform |
| 84 | pParamJson = {"operate":"set","preset":1} | * Preset operation of PTZ * operate: Operate with preset bits. * set: sets the preset bit * call: Calls the preset bit * clear: clears the preset bit * preset: Preset bit label of the current operation |
| 85 | pParamJson = {"operate":"zoom\_tele"} | PTZ lens operation   * operate: lens operation镜作。 * zoom\_tele: zoom in * zoom\_wide: zoom out * focus\_near: zoom（near） * focus\_far: zoom（far） * iris\_open: aperture（open aperture ） * iris\_close:aperture（close aperture） |
| **cbProductTest** | - | | Input | Callback function for receiving feedback of PTZ operation |
| **pParamJson** | -Refer to nTestId parameter | | Input | Additional parameters related to nTestId |
| **pUser** | - | | Input | Parameters reserved for the user |

#### Returned value

| **Returned value** | **Description** |
| --- | --- |
| **0** | Successful call |
| **Is not 0** | Function call error |

#### Attention

The function uses callbacks, so the fProductTestCallback callback function must be declared and defined before it can be used. The declaration of the callback function is:

typedef *IPSDK\_Int*(IPSDK\_CALLBACK \* *fProductTestCallback*)(*IPSDK\_Int* nTestId, *IPSDK\_Int* nCode, *IPSDK\_CPChar* pResultJson, *IPSDK\_PVoid* pUser);

##### Description of callback functions

| **Parameter** | **Description** |
| --- | --- |
| **nTestId** | Like nTestId in NETSDK\_ProductTest, it is easy for the callback function to distinguish the current operation |
| **nCode** | Status code, 0 indicates success |
| **pResultJson** | Content returned to the user |
| When nTestId == 83, the message content is the current preset information, for example: {" list ":[1,2,3]} |
| **pUser** | Values passed to pUser when NETSDK ProductTest is called |

The function obtains return information through this callback function during execution.

## dATA TYPE AND STRUCTURE

Generic data type description

API The main data types used are defined below

typedef     *char*                *IPSDK\_Int8*;

typedef     *unsigned* *char*       *IPSDK\_UInt8*;

typedef     *short*               *IPSDK\_Int16*;

typedef     *unsigned* *short*      *IPSDK\_UInt16*;

typedef     *int*                 *IPSDK\_Int32*;

typedef     *unsigned* *int*        *IPSDK\_UInt32*;

typedef     *unsigned* *long* *long*  *IPSDK\_UInt64*;

typedef     *unsigned* *long*       *IPSDK\_ULong*;

typedef     *long*                *IPSDK\_Long*;

typedef     *char*                *IPSDK\_Char*;

typedef     *char*\*               *IPSDK\_PChar*;

typedef     const *char*\*         *IPSDK\_CPChar*;

typedef     *unsigned* *char*       *IPSDK\_Byte*;

typedef     *unsigned* *char*\*      *IPSDK\_PByte*;

typedef     *int*                 *IPSDK\_Int*;

typedef     *void*                *IPSDK\_Void*;

typedef     *void*\*               *IPSDK\_PVoid*;

typedef     *void*\*               *IPSDK\_HANDLE*;

typedef     *int*                 *IPSDK\_Boolean*;

typedef     *float*               *IPSDK\_Float*;

typedef     *int*                 *IPSDK\_ERR*;

typedef     *IPSDK\_Int32*         *IPSDK\_Size*;

### Data structure description

#### IPSDK\_SearchDevice

##### Explain

##### Description of device information returned to the user during device search

##### Definition

typedef *struct* *tag\_ipsdk\_device\_info\_*

{

*IPSDK\_Char* strDevModel[32];         /// Equipment type definition。

*IPSDK\_Char* strDevMagic[64];         /// Unique number of the device platform

*IPSDK\_Char* strDevName[32];          /// Device Name

*IPSDK\_Char* strDevBuildDate[32];     /// Device release date

*IPSDK\_Char* strDevHardVer[32];       /// Hardware version number

*IPSDK\_Char* strDevSoftVer[32];       /// Software versioning

*IPSDK\_Char* strDevSerialId[32];      /// Factory serial number

*IPSDK\_Char* strDevCloudId[32];       /// Cloud ID of the device.

*IPSDK\_Char* strDevOemName[32];       /// Name of the device OEM

*IPSDK\_Char* strDevOemId[32];         /// ID of the device OEM

*IPSDK\_Char* strDevOemSN[64];         /// Unique number of the device platform.

*IPSDK\_Char* res[64];                 /// Extended structure

} *IPSDK\_DeviceInfo*;

typedef *struct* *tag\_ipsdk\_lan\_info\_*

{

*IPSDK\_Int* nSessionPort;             /// Session port

*IPSDK\_Int* nStreamPort;              /// Live port

*IPSDK\_Char* strDns1[64];             /// dns domain name

*IPSDK\_Char* strDns2[64];             /// dns domain name

*struct*

    {

*IPSDK\_Char* strNetType[16];      /// Network type: Wired/Wireless

*IPSDK\_Char* strIpAddr[16];

*IPSDK\_Char* strMask[16];

*IPSDK\_Char* strGateway[16];

*IPSDK\_Char* strMacAddr[24];

*IPSDK\_Int* bEnableDhcp;

*EN\_IPSDK\_AllNetConnect* enAllNetConnect; /// pass all networks

    } Lan[2];

*IPSDK\_Char* res[64];                 /// Extended structure

} *IPSDK\_LanInfo*;

typedef *struct* *tag\_IPSDK\_UserInfo*

{

*IPSDK\_Char* strUserName[IPSDK\_USER\_MAX\_NAME\_SZ];     /// User name

*IPSDK\_Char* strPassword[IPSDK\_USER\_MAX\_PASS\_SZ];     /// Password

*IPSDK\_Char* res[64];                 /// Extended structure

} *IPSDK\_UserInfo*;

typedef *struct* *tag\_IPSDK\_SearchDevice*

{

*IPSDK\_Char* strDeviceId[64];     /// Automatically gets the image ID of the fill device, which is used to identify the device on the device side

*EN\_IPSDK\_DeviceMode* nDeviceMode;    /// Type of equipment (lighting)

*IPSDK\_DeviceInfo* stDeviceInfo;  /// Device information

*IPSDK\_LanInfo* stLanInfo;

*IPSDK\_UserInfo* stUserInfo;

*IPSDK\_UInt64* enLicenseMask;     /// License mask, which is a combination of EN IPSDK LicenseMask

*EN\_IPSDK\_ProductCommand* enProductCmd;   /// Product test command

} *IPSDK\_SearchDevice*;

#### IPSDK\_ChannelInfo

##### Explain

Channel description information

##### Definition

typedef *struct* *tag\_ipsdk\_channel\_info\_*

{

*IPSDK\_Int* nChannel;     /// Device channel

*IPSDK\_Int* nStream;      /// main stream[0], sub stream[1]

} *IPSDK\_ChannelInfo*;

#### IPSDK\_MediaFrame

##### Explain

##### Media stream structure

##### Definition

typedef *struct* *tag\_ipsdk\_media\_frame\_*

{

*EN\_DATA\_TYPE* enMediaType;   /// Media data type

*IPSDK\_UInt64* u64SysTime;    /// Media frame system time (hour minute second)ms

*IPSDK\_UInt64* u64Timstmp;    /// Media frame time stamp

*IPSDK\_UInt32* u32DataLen;    /// Media raw data size

*IPSDK\_UInt32* u32AlarmType;  /// Judge alarm type: EN\_IPSDK\_STORE\_TYPE

*union*

    {

*struct*

        {

*IPSDK\_UInt32* u32FrameType;  /// Frame Type: EN\_IPSDK\_FRAME\_TYPE

*IPSDK\_UInt32* u32FrameSeq;   /// ref frame counter

*IPSDK\_UInt32* u32FrameRate;

*IPSDK\_UInt32* u32Width;

*IPSDK\_UInt32* u32Height;

        }video, h264, h265;

*struct*

        {

*IPSDK\_UInt32* u32SampleRate;     /// Sampling rate: 4000,8000,16000,32000

*IPSDK\_UInt32* u32SampleWidth;    ///Sampling bits: 8 bits, 16 bits

*IPSDK\_UInt32* u32SampleChannel;  /// Sampling channels: 1 mono,2 dual

*IPSDK\_UInt32* u32Ratio; /// if g711a.u == 2.0

*IPSDK\_UInt32* res;

        }audio, pcm, g711a, g711u;

    };

*struct*

    {

        ///H265 have vps[32],sps[33],pps[34],idr[19] ==> [Store in order]

        ///H264 have     sps[07],pps[08],idr[05] ==>[Store in order]

*IPSDK\_Char*  vvps\_buf[128]; /// Only for video messages

*IPSDK\_UInt32* vvps\_len;

*IPSDK\_Char*  vpps\_buf[128];

*IPSDK\_UInt32* vpps\_len;

*IPSDK\_Char*  vsps\_buf[128];

*IPSDK\_UInt32* vsps\_len;

    }media\_info;

} *IPSDK\_MediaFrame*;