DXNVT SDK 使用指南

# 前言

## 引言

DXNVT SDK是根据北京大讯科技有限公司研发的网络视频终端（包括网络视频服务器，网络数字硬盘录像机，网络摄像机，视频传输模块等）封装的应用软件开发套件。

DXNVT SDK内部采用多线程工作的方式，具有非常高的工作效率。Windows版本网络核心模块基于Windows的IOCP机制，Linux版本网络核心模块基于Epoll机制，协议数据解析（包括HTTP和XML）采用轻量级解析模块，利用缓冲区循环共享和智能搜索的方式达到非常高的效率。整个SDK采用纯C语言编写，上层应用程序可以采用C/C++，C#，JAVA等进行开发。

## SDK使用环境

* Windows XP SP2及以上版本。
* Windows 2003 Server 及以上版本。
* Linux 内核2.6以上版本。

## 读者对象

需要使用DXNVT SDK开发PC应用软件的研发人员。

## 修订记录

| 版本 | 作者 | 日期 | 说明 |
| --- | --- | --- | --- |
| 1.1 |  | 2009-12-18 |  |
| 1.2 |  | 2010-01-04 | 增加绑定本地IP功能。 |
| 1.3 |  | 2010-05-22 | 增加了抓拍接口函数。 |
| 1.4 |  | 2010-10-23 | DX\_NVT\_CB\_media\_arrive增加了param参数 |
| 1.5 |  | 2010-10-27 | 增加了应用举例。 |
| 1.6 |  | 2010-12-9 | 修改了数据结构DEVICE\_INFO的定义，增加对媒体通道ID的说明。 |
| 1.7 |  | 2011-01-02 | 修改了DX\_NVT\_add\_device。DX\_NVT\_CB\_device\_create接口，增加了context参数。 |
| 1.8 |  | 2011-01-07 | 增加了消息上下文参数，增加了本地监听操作函数。 |
| 1.9 |  | 2011-02-09 | 修改了DX\_NVT\_init\_media\_param接口参数，增加DX\_NVT\_get\_device\_base\_info接口。 |

# 声明

Copyright © 2009~2012北京大讯科技有限公司 版权所有，保留一切权利。

未经本公司书面许可，任何单位和个人不得擅自摘抄、复制本书内容的部分或者全部，并不得以任何形式传播。

由于SDK版本变更或者其他原因，本手册内容可能变更。大讯公司保留在没有通知或者提示的情况下对本手册的内容进行修改的权利。本手册仅作为使用指导，大讯公司尽全力在本手册中提供准确的信息，但是大讯公司并不确保手册内容完全没有错误，本手册中的所有陈述、信息和建议也不构成任何明示或暗示的担保。本手册说明中如有疑问或争议的，以公司最终解释为准。

# 目录

目录

[1 前言 1](#_Toc321584373)

[1.1 引言 1](#_Toc321584374)

[1.2 SDK使用环境 1](#_Toc321584375)

[1.3 读者对象 1](#_Toc321584376)

[1.4 修订记录 1](#_Toc321584377)

[2 声明 2](#_Toc321584378)

[3 目录 2](#_Toc321584379)

[4 数据类型定义 4](#_Toc321584380)

[4.1 DEVICE\_Handle 4](#_Toc321584381)

[4.2 MEDIA\_TYPE 4](#_Toc321584382)

[4.3 STREAM\_TRANS\_PROTOCOL 5](#_Toc321584383)

[4.4 STREAM\_TRASN\_DIR 6](#_Toc321584384)

[4.5 DEVICE\_CREATE\_MODE 6](#_Toc321584385)

[4.6 DEVICE\_DELETE\_REASON 7](#_Toc321584386)

[4.7 SNAP\_FORMAT 8](#_Toc321584387)

[4.8 RESULT\_CODE 8](#_Toc321584388)

[5 数据结构定义 9](#_Toc321584389)

[5.1 DEVICE\_PARAM 9](#_Toc321584390)

[5.2 DEVICE\_BASE\_INFO 10](#_Toc321584391)

[6 接口说明 11](#_Toc321584392)

[6.1 SDK控制函数 11](#_Toc321584393)

[6.1.1 DX\_NVT\_set\_callback 11](#_Toc321584394)

[6.1.2 DX\_NVT\_open 12](#_Toc321584395)

[6.1.3 DX\_NVT\_close 12](#_Toc321584396)

[6.2 TCP监听操作 13](#_Toc321584397)

[6.2.1 DX\_NVT\_start\_listen 13](#_Toc321584398)

[6.2.2 DX\_NVT\_stop\_listen 14](#_Toc321584399)

[6.2.3 DX\_NVT\_stop\_all\_listen 14](#_Toc321584400)

[6.3 设备操作基本函数 15](#_Toc321584401)

[6.3.1 DX\_NVT\_add\_device 15](#_Toc321584402)

[6.3.2 DX\_NVT\_request\_delete\_device 16](#_Toc321584403)

[6.3.3 DX\_NVT\_release\_device\_handle 16](#_Toc321584404)

[6.3.4 DX\_NVT\_set\_device\_context 17](#_Toc321584405)

[6.3.5 DX\_NVT\_get\_device\_context 18](#_Toc321584406)

[6.4 设备功能控制函数 18](#_Toc321584407)

[6.4.1 DX\_NVT\_get\_device\_base\_info 18](#_Toc321584408)

[6.4.2 DX\_NVT\_set 19](#_Toc321584409)

[6.4.3 DX\_NVT\_get 20](#_Toc321584410)

[6.4.4 DX\_NVT\_exe 21](#_Toc321584411)

[6.4.5 DX\_NVT\_snap 22](#_Toc321584412)

[6.5 设备媒体流操作函数 23](#_Toc321584413)

[6.5.1 DX\_NVT\_init\_media\_param 23](#_Toc321584414)

[6.5.2 DX\_NVT\_open\_media 23](#_Toc321584415)

[6.5.3 DX\_NVT\_close\_media 24](#_Toc321584416)

[6.5.4 DX\_NVT\_set\_media\_receive\_buf 25](#_Toc321584417)

[6.5.5 DX\_NVT\_send\_media 26](#_Toc321584418)

[6.6 回调函数定义 27](#_Toc321584419)

[6.6.1 DX\_NVT\_CB\_device\_create 27](#_Toc321584420)

[6.6.2 DX\_NVT\_CB\_set\_response 28](#_Toc321584421)

[6.6.3 DX\_NVT\_CB\_get\_response 29](#_Toc321584422)

[6.6.4 DX\_NVT\_CB\_exe\_response 29](#_Toc321584423)

[6.6.5 DX\_NVT\_CB\_snap\_response 30](#_Toc321584424)

[6.6.6 DX\_NVT\_CB\_open\_media\_response 31](#_Toc321584425)

[6.6.7 DX\_NVT\_CB\_media\_arrive 31](#_Toc321584426)

[6.6.8 DX\_NVT\_CB\_delete 32](#_Toc321584427)

[7 开发步骤 33](#_Toc321584428)

[8 应用举例 33](#_Toc321584429)

[8.1 获取设备ID 33](#_Toc321584430)

[8.2 设置设备局域网网关 34](#_Toc321584431)

[8.3 获取MTC01温度离线数据 35](#_Toc321584432)

[9 部分参数说明 36](#_Toc321584433)

[9.1 媒体通道ID 36](#_Toc321584434)

[10 注意事项 37](#_Toc321584435)

# 数据类型定义

## DEVICE\_Handle

定义

typedef void \* DEVICE\_Handle;

描述

设备句柄，用来标识一个有效的设备。

## MEDIA\_TYPE

定义

typedef enum \_\_MEDIA\_TYPE

{

MEDIA\_TYPE\_INVALID = -1,

MEDIA\_TYPE\_COMPLEX = 0,

MEDIA\_TYPE\_VIDEO = 1,

MEDIA\_TYPE\_AUDIO,

MEDIA\_TYPE\_UART，

MEDIA\_TYPE\_DATA,

MEDIA\_TYPE\_PICTURE\_BMP = 9,

MEDIA\_TYPE\_PICTURE\_JPG = 10

}MEDIA\_TYPE;

取值

* MEDIA\_TYPE\_INVALID 无效的媒体类型；
* MEDIA\_TYPE\_COMPLEX 复合媒体类型；
* MEDIA\_TYPE\_VIDEO 视频；
* MEDIA\_TYPE\_AUDIO 音频；
* MEDIA\_TYPE\_UART 串口；
* MEDIA\_TYPE\_DATA 自定义数据；
* MEDIA\_TYPE\_PICTURE\_BMP BMP格式图片；
* MEDIA\_TYPE\_PICTURE\_JPG JPEG格式图片。

描述

媒体数据类型，最常用到的有视频，音频，串口和JPG图片，这几种类型是基本类型，DXV8000系列所有设备都支持。

## STREAM\_TRANS\_PROTOCOL

定义

typedef enum \_\_STREAM\_TRANS\_PROTOCOL

{

UDP\_UNICAST\_RAW = 1,

UDP\_UNICAST\_RTP,

UDP\_MULTICAST\_RAW,

UDP\_MULTICAST\_RTP,

TCP\_RAW,

TCP\_RTP,

TCP\_CUSTOM

}STREAM\_TRANS\_PROTOCOL;

取值

* UDP\_UNICAST\_RAW UDP 单播 原始流；
* UDP\_UNICAST\_RTP UDP 单播 RTP 封装；
* UDP\_MULTICAST\_RAW UDP 组播 原始流；
* UDP\_MULTICAST\_RTP UDP组播 RTP 封装；
* TCP\_RAW TCP 原始流；
* TCP\_RTP TCP RTP；
* TCP\_CUSTOM TCP 自定义格式。

描述

媒体流网络传输协议。如果是广域网，无线网，或者网络情况比较复杂，推荐使用TCP\_CUSTOM，该协议为SDK内部协议。

## STREAM\_TRASN\_DIR

定义

typedef enum \_\_STREAM\_TRASN\_DIR

{

STREAM\_TRASN\_POSITIVE = 0,

STREAM\_TRASN\_NEGATIVE

}STREAM\_TRASN\_DIR;

取值

* STREAM\_TRASN\_POSITIVE SDK接收, 对应设备编码通道；
* STREAM\_TRASN\_NEGATIVE SDK发送，对应设备解码通道。

描述

媒体流传输方向。

## DEVICE\_CREATE\_MODE

定义

typedef enum \_\_DEVICE\_CREATE\_MODE

{

DEVICE\_CREATE\_ACTIVE = 0,

DEVICE\_CREATE\_PASSIVE

}DEVICE\_CREATE\_MODE;

取值

* DEVICE\_CREATE\_ACTIVE 主动连接创建，指调用DX\_NVT\_add\_device函数而创建的设备。
* DEVICE\_CREATE\_PASSIVE 被动连接创建，由设备向SDK监听端口主动发起连接而创建的设备。

描述

设备创建/上线方式。

## DEVICE\_DELETE\_REASON

定义

typedef enum \_\_DEVICE\_DELETE\_REASON

{

DEVICE\_DELETE\_ACTIVE = 1,

DEVICE\_DELETE\_PASSIVE = 2,

DEVICE\_DELETE\_TIMEOUT = 3,

DEVICE\_DELETE\_ERROR = 4

}DEVICE\_DELETE\_REASON;

取值

* DEVICE\_DELETE\_ACTIVE 由上层主动断开；
* DEVICE\_DELETE\_PASSIVE 设备主动断开；
* DEVICE\_DELETE\_TIMEOUT 网络超时导致设备断开；
* DEVICE\_DELETE\_ERROR 遇到错误而导致设备断开。

描述

设备/断开/下线原因。

## SNAP\_FORMAT

定义

typedef enum \_\_SNAP\_FORMAT

{

SNAP\_FORMAT\_BMP = 1,

SNAP\_FORMAT\_JPG = 2

}SNAP\_FORMAT;

取值

* SNAP\_FORMAT\_BMP BMP格式；
* SNAP\_FORMAT\_JPG JPEG格式。

描述

图片抓拍格式，当前版本只支持JPG(JPEG)格式。

## RESULT\_CODE

定义

typedef enum \_\_RESULT\_CODE

{

RESULT\_CODE\_OK = 0,

RESULT\_CODE\_ERR = -1,

RESULT\_CODE\_FAIL = -2,

RESULT\_CODE\_PAST = -301,

RESULT\_CODE\_REFUSE = -302,

RESULT\_CODE\_TIMEOUT = -401

}RESULT\_CODE;

取值

* RESULT\_CODE\_OK 操作成功；
* RESULT\_CODE\_ERR 出现未知错误；
* RESULT\_CODE\_FAIL 操作失败；
* RESULT\_CODE\_PAST 会话过期；
* RESULT\_CODE\_REFUSE 没有权限；
* RESULT\_CODE\_TIMEOUT 超时。

描述

设备操作返回结果代码。

# 数据结构定义

## DEVICE\_PARAM

定义

typedef struct \_\_DEVICE\_PARAM

{

struct \_\_DEVICE\_PARAM \*next;

int8 path[64];

int8 \*pvalue;

}DEVICE\_PARAM;

参数

* next 下一个设备参数，如果不存在，则为NULL；
* path 设备参数路径；
* pvalue 设备参数值，字符串形式。

描述

设备参数数据结构。

## DEVICE\_BASE\_INFO

定义

typedef struct \_\_DEVICE\_BASE\_INFO

{

int8 id[128];

int8 name[128];

int8 ver\_list[128];

int32 vin\_num;

int32 venc\_num;

int32 vdec\_num;

int32 ain\_num;

int32 aenc\_num;

int32 uart\_num;

}DEVICE\_BASE\_INFO;

参数

* id 设备ID；
* name 设备名称；
* ver\_list 设备固件程序版本；
* vin\_num 视频输入通道数；
* venc\_num 视频编码通道数；
* vdec\_num 视频解码通道数；
* ain\_num 音频输入通道数；
* aenc\_num 音频编码通道数；
* uart\_num 串口通道数。

描述

设备基本参数。

# 接口说明

## SDK控制函数

### DX\_NVT\_set\_callback

函数原型

void DX\_NVT\_set\_callback(

DX\_NVT\_CB\_device\_create cb\_create,

DX\_NVT\_CB\_set\_response cb\_set,

DX\_NVT\_CB\_get\_response cb\_get,

DX\_NVT\_CB\_exe\_response cb\_exe,

DX\_NVT\_CB\_snap\_response cb\_snap,

DX\_NVT\_CB\_open\_media\_response cb\_open\_media,

DX\_NVT\_CB\_media\_arrive cb\_meida\_arrive,

DX\_NVT\_CB\_delete cb\_delete);

参数

* cb\_create 设备创建/上线回调函数；
* cb\_set 设置设备参数完成回调函数；
* cb\_get 获取设备参数完成回调函数；
* cb\_exe 执行设备功能完成回调函数；
* cb\_exe 执行设备图片抓拍功能完成回调函数；
* cb\_open\_media 媒体流打开完成回调函数；
* cb\_meida\_arrive 媒体数据到达回调函数；
* cb\_delete 设备删除回调函数。

返回值

空。

功能

将上层定义的各个回调函数传入到SDK中。

### DX\_NVT\_open

函数原型

int32 DX\_NVT\_open(

int32 udp\_min\_port,

int32 udp\_max\_port);

参数

* udp\_min\_port 媒体流UDP本地端口最小值，0表示采用默认值，默认值为18000；
* udp\_max\_port 媒体流UDP本地端口最大值，0表示采用默认值，默认值为28000。

返回值

* GEN\_OK 成功；
* GEN\_ERR 失败。

功能

开启DXNVT SDK模块。

备注

DX\_NVT\_set\_callback，DX\_NVT\_open必须在SDK其他函数使用之前依次调用，以完成SKD的初始化工作。

### DX\_NVT\_close

函数原型

void DX\_NVT\_close(void);

参数

无。

返回值

空。

功能

关闭DXNVT SDK模块，当程序关闭，或者不再使用DXNVT SDK时，调用该函数关闭SDK，释放资源。

## TCP监听操作

### DX\_NVT\_start\_listen

函数原型

int32 DX\_NVT\_start\_listen(int32 port, int8 \*ip);

参数

* port 本地监听端口；
* ip 本地IP，NUL表示由SDK内部选择，推荐设置为NULL。

返回值

* GEN\_EXIST 监听已经存在；
* GEN\_UPPER\_LIMIT 监听端口数目超出限制，SDK限制为最多监听16个端口；
* GEN\_ERR 未知错误；
* GEN\_OK 监听成功。

功能

开启某个端口的TCP本地监听。

### DX\_NVT\_stop\_listen

函数原型

void DX\_NVT\_stop\_listen(int32 port);

参数

* port 监听端口。

返回值

空。

功能

停止本地某个端口TCP监听。

### DX\_NVT\_stop\_all\_listen

函数原型

void DX\_NVT\_stop\_all\_listen(void);

参数

无。

返回值

空。

功能

停止所有已开启的本地TCP监听。

## 设备操作基本函数

### DX\_NVT\_add\_device

函数原型

int32 DX\_NVT\_add\_device(int8 \*ip, int32 port, int8 \*local\_ip, void \*context);

参数

* ip 设备IP地址；
* port 设备HTTP监听端口；
* local\_ip 本地IP，一般填NULL。在多网卡情况下，如果要将设备的HTTP/TCP连接绑定到特定网卡，传入该参数。
* context 设备上下文，如果在DX\_NVT\_add\_device不指定context（即为NULL），则可以在设备成功创建后调用DX\_NVT\_set\_context进行设置；如果指定了context，也可以在设备成功连接后，调用DX\_NVT\_set\_context进行修改。

返回值

GEN\_OK 操作成功；

GEN\_UPPER\_LIMIT SDK连接数目达到上限；

GEN\_ERR 遇到错误，不能够连接到指定地址。

功能

添加一个指定网络地址的设备。

备注

返回成功只是表示成功提交了一个设备连接请求，并不意味设备成功创建，设备最终能否连接成功并且成功创建，需要看设备创建回调。

### DX\_NVT\_request\_delete\_device

函数原型

void DX\_NVT\_request\_delete\_device(DEVICE\_Handle handle);

参数

* handle 设备句柄。

返回值

空。

功能

请求删除/断开一个设备。

备注

该函数只是通知SDK，上层要断开一个设备。SDK将设备断开后，将回调设备删除回调。只有设备删除回调发生，才表示设备已经真正删除/断开。

### DX\_NVT\_release\_device\_handle

函数原型

void DX\_NVT\_release\_device\_handle(DEVICE\_Handle handle);

参数

* handle 设备句柄。

返回值

空。

功能

删除一个设备句柄数据。

备注

当设备删除/断开后，要调用该函数释放设备相关的数据，调用该函数后，针对该设备相关的函数都不可以再调用。

设备删除回调发生后，设备已经断开/离线，如果设备句柄没有删除，针对该设备相关的函数还可以调用，但是会返回错误码。

### DX\_NVT\_set\_device\_context

函数原型

int32 DX\_NVT\_set\_device\_context(

DEVICE\_Handle handle,

void \*device\_context);

参数

* handle 设备句柄；
* device\_context设备上下文。

返回值

* GEN\_OK 成功；
* GEN\_ERR 失败。

功能

设置设备上下文，即将设备上下文与设备handle绑定。

### DX\_NVT\_get\_device\_context

函数原型

void \*DX\_NVT\_get\_device\_context(DEVICE\_Handle handle);

参数

* handle 设备句柄。

返回值

NULL 没有设置上下文；

非NULL 设备上下文。

功能

获取设备上下文。

## 设备功能控制函数

### DX\_NVT\_get\_device\_base\_info

函数原型

DEVICE\_BASE\_INFO \*DX\_NVT\_get\_device\_base\_info(DEVICE\_Handle handle);

参数

* handle 设备句柄；

返回值

DEVICE\_BASE\_INFO类型指针。

功能

获取设备基本参数。

备注

设备基本参数和设备其他参数，都可以用DX\_NVT\_get函数进行获取。

### DX\_NVT\_set

函数原型

int32 DX\_NVT\_set(

DEVICE\_Handle handle,

int8 \*path, int8 \*pvalue,

void \*msg\_context);

参数

* handle 设备句柄；
* path 设备参数PATH；
* pvalue 设备参数值；
* msg\_context 消息上下文，每次对设备的控制操作，都可以传入一个上下文，当操作完成后，回调函数会将上下文传回来，方便应用程序区分不同的操作。

返回值

GEN\_OK 成功；

GEN\_UPPER\_LIMIT 发送缓冲区已经满，可以稍后发送；

GEN\_ERR 内部错误。

功能

发送设置设备功能参数消息。

备注

返回成功只是表示成功提交了消息，并不意味着参数设置成功，参数最终是否设置成功，需要看参数设置回调返回的结果。

### DX\_NVT\_get

函数原型

int32 DX\_NVT\_get(DEVICE\_Handle handle, int8 \*path, void \*msg\_context);

参数

* handle 设备句柄；
* path 设备参数PATH；
* msg\_context 消息上下文，每次对设备的控制操作，都可以传入一个上下文，当操作完成后，回调函数会将上下文传回来，方便应用程序区分不同的操作。

返回值

GEN\_OK 成功；

GEN\_UPPER\_LIMIT 发送缓冲区已经满，可以稍后发送；

GEN\_ERR 内部错误。

功能

发送获取设备参数消息。

备注

返回成功只是表示成功提交了消息，并不意味着参数获取成功，参数获取最终是否成功，需要看设备参数获取回调。

### DX\_NVT\_exe

函数原型

int32 DX\_NVT\_exe(

DEVICE\_Handle handle,

int8 \*path[], int8 \*pvalue[],

int32 count, void \*msg\_context);

参数

* handle 设备句柄；
* path[] 设备参数PATH组；
* pvalue[] 设备参数值组；
* count 参数组参数数目；
* msg\_context 消息上下文，每次对设备的控制操作，都可以传入一个上下文，当操作完成后，回调函数会将上下文传回来，方便应用程序区分不同的操作。

返回值

GEN\_OK 成功；

GEN\_UPPER\_LIMIT 发送缓冲区已经满，可以稍后发送；

GEN\_ERR 内部错误。

功能

发送执行设备功能消息。

备注

返回成功只是表示成功提交了消息，并不意味着执行成功，执行最终是否成功，需要看设备执行回调返回的结果。

### DX\_NVT\_snap

函数原型

int32 DX\_NVT\_snap(

DEVICE\_Handle handle, int32 chn,

SNAP\_FORMAT format, int32 quality);

参数

* handle 设备句柄；
* chn 媒体通道ID；
* format 图片格式；
* quality 图像质量，1~100。

返回值

GEN\_OK 成功；

GEN\_UPPER\_LIMIT 发送缓冲区已经满，可以稍后发送；

GEN\_ERR 内部错误。

功能

发送执行抓拍功能消息。

备注

返回成功只是表示成功提交了消息，并不意味着执行成功，执行最终是否成功，需要看设备执行回调返回的结果。

## 设备媒体流操作函数

### DX\_NVT\_init\_media\_param

函数原型

int32 DX\_NVT\_init\_media\_param(

DEVICE\_Handle handle,

int8 \*local\_ip);

参数

* handle 设备句柄；
* local\_ip本地IP，一般填NULL。在多网卡情况下，如果要将媒体流使用的UDP连接绑定到特定网卡，才传入该参数。

返回值

* GEN\_OK 操作成功；
* GEN\_ERR 内部错误。

功能

初始化设备媒体参数。

备注

该函数应该在设备其他媒体流操作函数之前调用。

### DX\_NVT\_open\_media

函数原型

int32 DX\_NVT\_open\_media(

DEVICE\_Handle handle,

MEDIA\_TYPE type, int32 chn,

int32 dir, int32 protocol,

void \*msg\_context);

参数

* handle 设备句柄；
* type 媒体类型，分为视频，音频，串口数据等；
* chn 媒体通道ID；
* dir 媒体流传输方向；
* protocol 媒体流传输协议；
* msg\_context 消息上下文。

返回值

* GEN\_OK 操作成功；
* GEN\_UPPER\_LIMIT 内部缓冲区已经满；
* GEN\_ERR 内部错误。

功能

打开设备一个流媒体通道。

备注

流媒体通道打开后，就可以接收或者发送媒体流数据了。

### DX\_NVT\_close\_media

函数原型

int32 DX\_NVT\_close\_media(

DEVICE\_Handle handle,

MEDIA\_TYPE type, int32 chn,

int32 dir, int32 protocol);

参数

* handle 设备句柄；
* type 媒体类型，分为视频，音频，串口数据等；
* chn 媒体通道ID；
* dir 媒体流传输方向；
* protocol 媒体流传输协议。

返回值

* GEN\_OK 操作成功；
* GEN\_UPPER\_LIMIT 内部缓冲区已经满；
* GEN\_ERR 内部错误。

功能

关闭设备流媒体通道。

### DX\_NVT\_set\_media\_receive\_buf

函数原型

int32 DX\_NVT\_set\_media\_receive\_buf(

DEVICE\_Handle handle,

MEDIA\_TYPE type, int32 chn,

int8 \*buf, int32 buf\_len);

参数

* handle 设备句柄；
* type 媒体类型，分为视频，音频，串口数据等；
* chn 媒体通道ID；
* buf 接收媒体数据的缓冲区；
* buf\_len 接收缓冲区长度。

返回值

* GEN\_OK 操作成功；
* GEN\_INVALID\_PARAM 参数无效；
* GEN\_ERR 内部错误。

功能

设置媒体数据接收缓冲区。

备注

调用DX\_NVT\_open\_media成功后，接收媒体数据之前，先要设置接收缓冲区，该缓冲区由上层传入到SDK内部，当数据达到时，SDK将数据写入缓冲区，同时执行数据到达回调，在数据到达回调中上层将缓冲区移走，同时再设置一个新的缓冲区（如果数据很快处理完毕，也可以设置与前一个相同的缓冲区，但这样效率比较低），以开启下一次媒体数据接收。

### DX\_NVT\_send\_media

函数原型

int32 DX\_NVT\_send\_media(

DEVICE\_Handle handle,

MEDIA\_TYPE type, int32 chn,

int8 \*buf, int32 len);

参数

* handle 设备句柄；
* type 媒体类型，分为视频，音频，串口数据等；
* chn 媒体通道ID；
* buf 待发送的媒体数据缓冲区；
* len 数据长度。

返回值

* > 0 操作成功，返回值为成功发送的数据长度；
* =0 网络繁忙，可以稍后再发；
* GEN\_INVALID\_PARAM 参数无效；
* GEN\_ERR 内部错误。

功能

发送媒体流数据。

备注

该函数调用为同步非阻塞方式。

## 回调函数定义

### DX\_NVT\_CB\_device\_create

函数定义

typedef void (\* DX\_NVT\_CB\_device\_create)(

DEVICE\_Handle handle,

void \*context,

int8 \*ip, int32 port,

DEVICE\_CREATE\_MODE mode);

参数

* handle 设备创建成功则为有效的设备句柄，创建失败则为NULL；
* context 设备上下文，为添加设备时指定的上下文，如果添加设备时没有指定或者设备是被动连接创建的，则context 为NULL；
* ip 设备IP；
* port 设备端口；
* mode 设备创建模式，分为主动、被动两种。主动指SDK主动发起对设备的连接，被动指设备连接SDK的监听端口而建立的连接。

返回值

空。

功能

设备创建/上线回调。

备注

只有主动连接的设备创建失败时，才会有handle为NULL。

设备创建成功后，应该设置设备上下文。

### DX\_NVT\_CB\_set\_response

函数定义

typedef void (\* DX\_NVT\_CB\_set\_response)(

DEVICE\_Handle handle,

int32 result\_code,

void \*msg\_context);

参数

* handle 设备句柄。
* result\_code 操作结果代码，参见数据类型RESULT\_CODE的定义；
* msg\_context 消息上下文。

返回值

空。

功能

设备参数设置回调。

### DX\_NVT\_CB\_get\_response

函数定义

typedef void (\* DX\_NVT\_CB\_get\_response)(

DEVICE\_Handle handle,

DEVICE\_PARAM \*param\_list,

void \*msg\_context);

参数

* handle 设备句柄
* param\_list 参数列表，如果获取参数成功，则为有效的设备参数列表，如果获取失败，则为NULL；
* msg\_context 消息上下文。

返回值

空。

功能

获取设备参数回调。

### DX\_NVT\_CB\_exe\_response

函数定义

typedef void (\* DX\_NVT\_CB\_exe\_response)(

DEVICE\_Handle handle,

int32 result\_code,

void \*msg\_context);

参数

* handle 设备句柄；
* result\_code 操作结果代码；
* msg\_context 消息上下文。

返回值

空。

功能

执行设备功能回调。

### DX\_NVT\_CB\_snap\_response

函数定义

typedef void (\* DX\_NVT\_CB\_snap\_response)(

DEVICE\_Handle handle,

SNAP\_FORMAT format, int32 chn,

int8 \*pdata, int32 len);

参数

* handle 设备句柄；
* format 图片格式；
* chn 通道ID；
* pdata 图片数据；
* len 图片数据长度。

返回值

空。

功能

图片抓拍回调。

### DX\_NVT\_CB\_open\_media\_response

函数定义

typedef void (\* DX\_NVT\_CB\_open\_media\_response)(

DEVICE\_Handle handle,

int32 result\_code,

void \*msg\_context);

参数

* handle 设备句柄；
* result\_code 操作结果代码；
* msg\_context 消息上下文。

返回值

空。

功能

打开设备媒体流回调。

### DX\_NVT\_CB\_media\_arrive

函数定义

typedef void (\* DX\_NVT\_CB\_media\_arrive)(

DEVICE\_Handle handle,

MEDIA\_TYPE type, int32 chn,

int8 \*buf, int32 len

uint32 param);

参数

* handle 设备句柄；
* type 媒体类型；
* chn 媒体流通道ID；
* buf 媒体数据接收缓冲区，该缓冲区由上层应用程序设定；
* len 媒体数据长度；
* param 媒体附加参数。一般情况可以忽略该参数。

返回值

空。

功能

媒体流数据达到。

### DX\_NVT\_CB\_delete

函数定义

typedef void (\* DX\_NVT\_CB\_delete)(

DEVICE\_Handle handle,

DEVICE\_DELETE\_REASON reason);

参数

* handle 设备句柄；
* reason设备删除/下线原因代码。

返回值

空。

功能

设备删除回调，该回调发生后，表示设备已经删除/离线。

# 开发步骤

实现各个回调函数。

调用DX\_NVT\_set\_callback函数，将回调函数传入SDK。

调用DX\_NVT\_open初始化SDK。

调用DX\_NVT\_start\_listen开启本地监听。

如果需要主动连接设备，则调用DX\_NVT\_add\_device函数添加设备。

如果设备创建回调执行，表示有设备成功创建/上线，则调用DX\_NVT\_set\_context函数设置设备上下文。

按照需要执行设备功能操作，如设备参数设置，设备参数获取等。

调用DX\_NVT\_init\_media\_param函数，初始化媒体数据。

调用DX\_NVT\_open\_media函数打开媒体流通道。

调用DX\_NVT\_set\_media\_receive\_buf函数，设置媒体数据接收缓冲区。

注意：DX\_NVT\_set\_media\_receive\_buf必须在DX\_NVT\_open\_media函数之后调用。只有DX\_NVT\_set\_media\_receive\_buf调用后才开始启动码流接收。为了使获取的图像数据可以迅速解码，最好在第一次调用DX\_NVT\_set\_media\_receive\_buf后，请求一次图像I帧。

在媒体数据达到回调执行时，处理收到的数据，

数据处理完，调用DX\_NVT\_set\_media\_receive\_buf函数启动下一次接收。

设备不再使用时，依次调用DX\_NVT\_close\_media、DX\_NVT\_delete\_device函数。

程序退出时，调用DX\_NVT\_close函数释放资源。

# 应用举例

## 获取设备ID

获取设备参数调用“DX\_NVT\_get”函数，当调用成功，在回调函数“DX\_NVT\_CB\_get\_response”中可以取到相应的设备参数值。

根据“网络设备交互参数”文档可知，设备ID的path为“/base/id”，则调用方法为：

int32 ret；

ret = DX\_NVT\_get(handle, “/base/id”, NULL);

/\*\*

\* 其中，handle为连接成功的设备句柄。

\* 返回值ret如果为0，表示函数执行成功。

\*/

当上述函数正确执行后，在获取设备参数回调函数中可以获取设备id：

void Example\_DX\_NVT\_CB\_get\_response(

DEVICE\_Handle handle,

DEVICE\_PARAM \*param\_list,

void \*msg\_context)

{

DEVICE\_PARAM \*param = param\_list;

while (param != NULL){

printf("%s: %s\n", param->path, param->pvalue);

param = param->next;

}

}

## 设置设备局域网网关

设置设备参数调用“DX\_NVT\_set”函数，当调用成功，在回调函数“DX\_NVT\_CB\_set\_response”中可以获取到相应的设置结果。

根据“网络设备交互参数”文档可知，设备局域网网关的path为“/lan/gateway”，则调用方法为：

int32 ret；

ret = DX\_NVT\_set(handle, “/lan/gateway”, “192.168.0.1”, NULL);

/\*\*

\* 该语句将设备局域网网关设置为“192.168.0.1”。

\* 其中，handle为连接成功的设备句柄。

\* 返回值ret如果为0，表示函数执行成功。

\* 注意，函数执行成功并不意味着设备参数设置成功，

\* 设备参数是否设置成功要在回调函数中才知道。

\*/

当上述函数正确执行后，在设置设备参数回调函数中可以知道设置结果：

void Example\_DX\_NVT\_CB\_set\_response(

DEVICE\_Handle handle,

int32 result\_code,

void \*msg\_context)

{

printf("result: %d\n", result\_code);

}

## 获取MTC01温度离线数据

离线数据属于自定义媒体数据类型，其数据传输走自定义数据通道。每个设备有且仅有1个自定义数据通道。

**第一步**：如果自定义数据通道没有打开，则需要执行下面函数打开自定义数据通道：

DX\_NVT\_open\_media(

handle,

STREAM\_MEDIA\_DATA, 1,

STREAM\_TRASN\_POSITIVE, TCP\_CUSTOM,

NULL);

DX\_NVT\_set\_media\_receive\_buf(

handle,

STREAM\_MEDIA\_DATA, 1,

buf, buf\_len);

**第二步**：打开MTC01离线温度数据传输：

int8 path[4][128];

int8 value[4][128];

int8 \*ppath[4];

int8 \*pvalue[4];

int32 i;

for(i=0; i<4; i++){

ppath[i] = path[i];

pvalue[i] = value[i];

}

sprintf(path[0], "dataExe/getOfflineData/duration");

sprintf(value[0], "60");

sprintf(path[1], "dataExe/getOfflineData/interval");

sprintf(value[1], "0");

sprintf(path[2], "dataExe/getOfflineData/type");

sprintf(value[2], "11");

sprintf(path[3], "dataExe/getOfflineData/idMask");

sprintf(value[3], "-1");

DX\_NVT\_exe(handle, ppath, pvalue, 4, NULL);

/\*\*

\* 关于获取离线数据的path请参见“网络设备交互参数”中的“自定义数据执行”章节。

\* 离线数据类型11表示MTC01温度数据。

\* 上述代码执行成功后，设备将开始传输相应的离线数据。

\*/

第三步：获取离线数据：

void Example\_DX\_NVT\_CB\_media\_arrive(

DEVICE\_Handle handle, STREAM\_MEDIA\_TYPE type, int32 chn,

int8 \*buf, int32 len, uint32 param)

{

if(type == STREAM\_MEDIA\_DATA){

/\*\* 在此处进行数据解析 \*/

}

DX\_NVT\_set\_media\_receive\_buf(

handle, type, chn,

stream\_rcv\_buf, stream\_rcv\_buf\_len);

}

/\*\*

\* 当SDK收到离线数据后，将回调DX\_NVT\_CB\_media\_arrive函数。

\* 在回调函数中，如果type的类型为STREAM\_MEDIA\_DATA，则为自定义数据（包括离线数据）。

\* 具体离线数据的结构定义，请参见相关文档。

\*/

# 部分参数说明

## 媒体通道ID

媒体通道ID为一个整型数，表示媒体的通道序号，从1开始。下面针对不同的媒体类型对其进行说明：

* 视频数据：媒体通道ID为视频编码通道ID。
* 音频数据：媒体通道ID为音频编码通道ID。
* 串口数据：媒体通道ID为串口通道ID。
* BMP/JPG图片数据：媒体通道ID为对应的视频编码通道ID。
* 复合音视频数据：媒体通道ID为对应的视频编码通道ID。

**注意**：如果视/音频编码为多码流模式，那么视/音频编码通道ID与视/音频输入通道ID不是一一对应的关系。比如视频编码为三码流模式时，视频编码通道1，2，3对应视频输入通道1，视频编码通道4，5，6对应视频输入通道2，其他以此类推。

# 注意事项

1. 应用程序在Linux 平台编译时，要加上“-pthread”项和“-L/path -ldxnvt”项，其中path为libdxnvt.so存放的目录。
2. 在Linux平台运行应用程序时，如果应用程序没有执行加载libdxnvt.so，则可以将libdxnvt.so拷入系统库目录如“/usr/lib/”，这样系统在启动时会自动加载libdxnvt.so库。