# 1、数据结构

## struct login\_info 登陆信息

struct login\_info

{

char ip[256];

unsigned port;

char user[32];

char password[32];

};

## struct st\_xy\_user\_data用户数据

struct st\_xy\_user\_data

{

int id;

int type;

char\* data;

int len;

};

## struct st\_output\_info 输出信息

struct st\_output\_info

{

int id; //一个输出在整个系统的编号，一个输出对应一个物理端口

int index; //一个输出在一个解码器的内部的编号

int hasdisplay; //是否连接显示器

char resolution[256]; //支持的解析度,支持多个以逗号隔开,比如“1920x1080,1280x720”

};

## struct st\_matrix\_info 解码器信息

struct st\_matrix\_info

{

int id; //解码器的id号，在添加解码器时id值填为-1，由系统填充为添加成功后的值

int status; //解码器的状态，在线或离线？？？？？？？？？？？？？？？？？？？？？？？？？

char name[64]; //解码器名称

char ip[64]; //解码器ip

int port; //登陆端口

char user[64]; //登陆用户名

char password[64]; //登陆端口

struct st\_output\_info\* output\_info; //该解码器的输出信息，，不为NULL时用户须自己释放该指针

int output\_info\_cnt; //output\_info的条数

};

调用[nmc\_set\_matrix](#_struct st_matrix_info 解码器信息)添加解码器时，id值填为-1，name填自定义的名称，填入解码器ip，port，user，password即可。

调用[nmc\_set\_matrix](#_struct st_matrix_info 解码器信息)修改解码器时，id值填为获取到的解码器的id值，name，ip，port，user，password填入为想要修改的值即可，不用修改的部分填为原值即可。

调用[nmc\_set\_matrix](#_struct st_matrix_info 解码器信息)删除解码器时，id值填为获取到的解码器的id值，其余不用填写。

## struct st\_window\_info 窗口信息

struct st\_window\_info

{

int id; //窗口的全局id

int index; //窗口在本输出的索引，从0起

int status; //窗口状态，0空闲、1实时流（包括RTSP和录屏）、2录像、3单窗口轮巡、4分组轮巡、5GIS、6报警

int sourceid; //内部nvr的id

char SurveyUuid[256]; //暂时无用

char Extend[256]; //sourceid为内部nvr的id时，Extend为一个子设备id

};

## struct st\_output\_layout 一个输出上的布局信息

struct st\_output\_layout

{

int id; //该输出的id号

int layout; //布局方式，1代表一分屏，2代表2分屏，4代表4分屏，以此类推，目前可取的值为

struct st\_window\_info\* window\_info; //该布局下的窗口信息，参见[struct st\_window\_info定义](#_struct st_window_info  窗口信息)

int window\_info\_cnt; //窗口信息条数

};

调用[nmc\_set\_output](#_nmc_set_output设置一个输出的布局)设置布局时，id填为输出的id号，layout填为想要设置的布局值

调用[nmc\_get\_output](#_nmc_get_output获取解码器的输出当前布局信息)获取布局时，会填充[struct st\_window\_info](#_struct st_window_info  窗口信息)结构，得到该布局下的窗口信息

## struct st\_large\_screen\_info 拼接屏信息

struct st\_large\_screen\_info

{

int screenid; //拼接屏id

int screenstatus; //屏幕状态

int rows; //行

int cols; //列

char screenname[64]; //名字

int sourceid; //流源号

int subsourceid; //流源子号

struct st\_subscreen\_info \*psubscreen\_info; [//子屏信息参见struct st\_subscreen\_info](#_struct st_subscreen_info拼接屏子屏信息)

int subscreen\_info\_cnt; //拼接屏的子屏个数

};

## struct st\_subscreen\_info拼接屏子屏信息

struct st\_subscreen\_info

{

int OutputId; //输出id号

int Row; //在拼接屏中的位置 行

int Col; //在拼接屏中的位置 列

};

## enum en\_nmc\_operator\_type 操作类型

enum en\_nmc\_operator\_type

{

nmc\_add=0,

nmc\_modify=1,

nmc\_remov=2,

};

0添加，1修改，2删除

## struct st\_jn\_equ 信号源设备信息

struct st\_jn\_equ

{

int EquId; //设备id号

char FactoryName[64]; //设备类型，参见[附录2](#_附录4)

char EquName[64]; //设备名称

char IP[256]; //设备ip地址

int Port; //设备端口

char UserName[64]; //登陆用户名

char Password[64]; //登陆密码

int ChannelNum; //通道数量，该设备能出的视频路数

int InputNum; //输入路数

int OutputNum; //输出路数

char Extend[256]; //扩展字段

int nOpSeq; //内部使用

int nOpType; //内部使用

int nOpResult; //内部使用

int nCtlgId; //内部使用

struct st\_jn\_sub\_equ \*stSubEqu; //子设备信息，参见[struct st\_jn\_sub\_equ定义](#_struct st_jn_sub_equ  子设备信息)

int stSubEquCnt; //子设备数量

};

调用[nmc\_add\_signal\_source](#_nmc_add_signal_source添加一个信号源到nmc上)添加信号源时，EquId填为-1，填入FactoryName，EquName，IP，Port，UserName，Password，ChannelNum，其余不用填写

调用[nmc\_get\_signal\_source](#_nmc_get_signal_source获取nmc上已有的信号源)获取信号源时，返回的struct st\_jn\_equ结构里stSubEqu字段会被填充，得到该信号源上的子设备信息，参见[struct st\_jn\_sub\_equ定义](#_struct st_jn_sub_equ  子设备信息)

## enum \_SUB\_EQU\_TYPE 子设备类型

typedef enum \_SUB\_EQU\_TYPE

{

SUB\_EQU\_TYPE\_VEDIO = 0x1100, //视频通道

SUB\_EQU\_TYPE\_INPUT = 0x1200, //开关量输入

SUB\_EQU\_TYPE\_OUTPUT = 0x1300, //开关量输出

}SUB\_EQU\_TYPE;

## struct st\_jn\_sub\_equ 子设备信息

struct st\_jn\_sub\_equ

{

int SubEquId; //子设备id号

int EquId; //子设备所属的设备id号

int CtlgId; //内部使用

char SubName[64]; //子设备名称

int SubSeq; //子设备在设备中的序号，从0起

int SubType; //子设备类型,参见[enum \_SUB\_EQU\_TYPE定义](#_enum _SUB_EQU_TYPE 子设备类型)

int SpecificType; //子设备类型，内部使用

int HasPTZ; //内部使用

int HasLamp; //内部使用

int HasDemist; //内部使用

int HasRainBrush; //内部使用

char Extend[256]; //内部使用

int nOpSeq; //内部使用

int nOpType; //内部使用

int nOpResult; //内部使用

};

## struct st\_video\_stream\_info 视频流信息

struct st\_video\_stream\_info

{

int len; //本结构体长度sizeof(struct st\_video\_stream\_info)

int codec\_type; //编码类型,目前只支持H.264,取值0x0080

int width; //宽

int height; //高

int fps; //帧率

int bitrate; //码率,单位bit

int frametype; //1：i帧, 2：b帧, 3：p帧

int res[4];

};

## struct st\_sw\_window\_info 漫游窗口当前状态

struct st\_sw\_window\_info

{

int sw\_window\_id; //漫游窗口id号，创建时该字段由系统填充为创建成功后的值

int sw\_window\_status; //窗口状态, 0:空闲 1:有实时流

int source\_id; //有实时流时的源id

int source\_sub\_id; //有实时流源子id

};

## struct st\_xy\_device\_info搜索到的设备信息

struct st\_xy\_device\_info

{

int len; //本结构体长度sizeof(struct st\_video\_stream\_info)

int device\_type; //设备类型

char ipv4[32]; //ip地址

char mask[32]; //掩码

char gateway[32]; //地址

char name[128]; //名字

int res[64];

};

# 2、函数

## 初始化

### nmc\_init sdk初始化

NMC\_API int NMC\_CALL\_TYPE nmc\_init(nmc\_status\_callback status\_callback, void\* userdata);

函数说明：sdk的全局初始化函数，在使用所有的sdk函数之前调用

参数说明：

[in] status\_callback：全局的事件回调函数，[参见nmc\_status\_callback定义](#_nmc_status_callback设备状态回调)

[in] userdata：设置的用户数据

返回值：

成功返回0，失败返回非0

### nmc\_status\_callback设备状态回调

int (NMC\_CALL\_TYPE \*nmc\_status\_callback)(long handle, int type, void\* data, int datalen, void \*userdata);

函数说明：获取设备的状态回调函数

参数说明：

[out] handle：设备句柄

[out] type：状态类型，参见[附录1](#_附录3)

[out] data：对应状态类型的数据，根据不同的状态类型，对应不同的数据结构

[out] datalen：对应状态类型的数据

返回值说明：正确返回0，不正确返回非0值

### nmc\_uninit sdk的反初始化

NMC\_API int NMC\_CALL\_TYPE nmc\_uninit(void\* data);

函数说明：sdk的反初始化，在所有的sdk使用之后调用，释放资源

参数说明：

[in] data：保留参数，暂未使用

返回值说明：成功返回0，失败返回非0

## 登陆

### nmc\_login登陆一个nmc服务器

NMC\_API long NMC\_CALL\_TYPE nmc\_login(struct login\_info \*info);

函数说明：登陆到一个nmc服务器，登陆后才能对nmc服务器进行操作

参数说明：

[in] info：登陆到一个nmc服务器所需的信息，参见[struct login\_info的定义](#_struct login_info)

返回值说明：成功返回登陆句柄，失败返回0值

### nmc\_logout 从一个nmc登出

NMC\_API int NMC\_CALL\_TYPE nmc\_logout(long handle);

函数说明：从一个nmc登出后，不能再对该nmc服务器进行操作

参数说明：

[in] handle：由nmc\_login函数返回得到的设备句柄，应为非0值

返回值说明：成功返回0，失败返回非0值

## 用户数据

### nmc\_get\_userdefinedata获取用户透明数据

NMC\_API int NMC\_CALL\_TYPE nmc\_get\_userdefinedata(long handle, int id, struct st\_xy\_user\_data \*\*ppuser\_data, int \*user\_data\_cnt);

函数说明：获取用户透明数据

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] id :数据id号，-1获取所有

[in] ppuser\_data：[struct st\_xy\_user\_data](#_struct st_xy_user_data用户数据) \*指针，返回用户数据信息

[in] user\_data\_cnt：返回用户数据数量

返回值说明：成功返回0，失败返回非0值

### nmc\_set\_userdefinedata设置用户透明数据

NMC\_API int NMC\_CALL\_TYPE nmc\_set\_userdefinedata(long handle, enum en\_nmc\_operator\_type operator\_type, struct st\_xy\_user\_data \*ppuser\_data);

函数说明：设置用户透明数据

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] operator\_type :对用户数据的操作类型

[in] ppuser\_data : 要操作的用户数据

返回值说明：成功返回0，失败返回非0值

## 解码器

### nmc\_get\_matrix 获取解码器信息

NMC\_API int NMC\_CALL\_TYPE nmc\_get\_matrix(long handle, int matrixid, struct st\_matrix\_info \*\*ppmatrix\_info, int \*matrix\_info\_cnt);

函数说明：获取由该nmc管理的解码器的信息

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] matrixid :想要获取解码器的解码器的id，-1表示获取所有该nmc上的所有解码器的信息

[in] matrix\_info：struct st\_matrix\_info \*指针，返回解码器信息，参见[struct st\_matrix\_info定义](#_struct st_matrix_info)

[in] matrix\_info\_cnt：struct st\_matrix\_info \*指针，返回的解码器数量

返回值说明：成功返回0，失败返回非0值

返回值说明：成功返回0，失败返回非0值

### nmc\_set\_matrix设置解码器信息

NMC\_API int NMC\_CALL\_TYPE nmc\_set\_matrix(long handle, enum en\_nmc\_operator\_type operator\_type, struct st\_matrix\_info \*matrix\_info);

函数说明：对nmc上的解码器进行操作，可以添加一个解码器，删除一个解码器，修改解码器，修改解码器时

参数说明：

[in] operator\_type：对解码器的操作类型，参见[enum en\_nmc\_operator\_type定义](#_enum en_nmc_operator_type 操作类型)

[in] matrix\_info：要操作的解码器的信息，参见s[truct st\_matrix\_info定义](#_struct st_matrix_info 解码器信息)

返回值说明：成功返回0，失败返回非0值

## 输出

### nmc\_get\_output获取解码器的输出当前布局信息

NMC\_API int NMC\_CALL\_TYPE nmc\_get\_output(long handle, int outputid, struct st\_output\_layout \*\*ppoutput\_layout, int \*output\_layout\_cnt);

函数说明：获取解码器输出的布局信息

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] outputid :想要获取的输出的id号，-1表示获取所有该nmc上的所有的输出信息

[in] output\_layout：struct st\_output\_layout \*指针，返回输出分屏信息，参见[struct st\_output\_layout定义](#_struct st_output_layout  一个输出上的布局信息)

[in] output\_layout\_cnt：struct st\_output\_layout \*指针，返回的分屏信息条数

返回值说明：成功返回0，失败返回非0值

### nmc\_set\_output设置一个输出的布局

NMC\_API int NMC\_CALL\_TYPE nmc\_set\_output(long handle, struct st\_output\_layout \*output\_layout);

函数说明：设置解码器输出的布局，设置布局时，st\_output\_layout.id填为输出的id号，st\_output\_layout.layout填为想要设置的布局值

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] output\_layout :要设置的输出的内容参见[struct st\_output\_layout定义](#_struct st_output_layout  一个输出上的布局信息)

返回值说明：成功返回0，失败返回非0值

## 信号源

### nmc\_get\_signal\_source获取nmc上已有的信号源

NMC\_API int NMC\_CALL\_TYPE nmc\_get\_signal\_source(long handle, struct st\_jn\_equ \*\*equ, int \*equcnt);

函数说明：获取nmc上已有信号源，信号源表示视频源，可以从信号源上取视频流出来

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[out] equ :传入指针，返回目前已有的信号源，参见[struct st\_jn\_equ定义](#_struct st_jn_equ 信号源设备信息)

[out] equcnt :返回一个目前已有的信号源的数量

返回值说明：成功返回0，失败返回非0值

### nmc\_add\_signal\_source添加一个信号源到nmc上

NMC\_API int NMC\_CALL\_TYPE nmc\_add\_signal\_source(long handle, struct st\_jn\_equ \*equ);

函数说明：添加一个信号源到nmc上

参数说明：

[in] handle：由[nmc\_login](#_nmc_login登陆一个nmc服务器)函数创建出得设备对象的句柄

[in out] equ :传入想要添加的struct st\_jn\_equ数据结构的指针，参见[struct st\_jn\_equ定义](#_struct st_jn_equ 信号源设备信息)

返回值说明：成功返回0，失败返回非0值，目前可以添加的信号源参见[附录2](#_附录2)

### mc\_rmv\_signal\_source 将一个信号源从nmc上删除

NMC\_API int NMC\_CALL\_TYPE nmc\_rmv\_signal\_source(long handle, int sub\_equ\_id);

函数说明：将一个信号源从nmc上删除

参数说明：

[in] handle：由[nmc\_login](#_nmc_login登陆一个nmc服务器)函数创建出得设备对象的句柄

[in] sub\_equ\_id:想要删除的一个子设备的id号，对应struct st\_jn\_sub\_equ结构中的SubEquId

，参见[struct st\_jn\_sub\_equ定义](#_struct st_jn_sub_equ  子设备信息)

返回值说明：成功返回0，失败返回非0值

### nmc\_set\_window\_signal\_source 将一个信号源设置到一个窗口上播放

NMC\_API int NMC\_CALL\_TYPE nmc\_set\_window\_signal\_source(long handle, int windowtype, int windowid, int sourceid, int subsourceid);

函数说明：设置解码窗口信号

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] windowtype :窗口类型，0非拼接，1拼接，由？？？？得到

[in] windowid :要设置的输出窗口，对应struct st\_window\_info结构中的id号[in] sourceid :要设置的输出信号源id，目前固定设置为1

[in] subsourceid :当sourceid（如NVR）包含多个子设备时候，subsourceid表示子信号源Id，对应struct st\_jn\_sub\_equ结构中的SubEquId，参见[struct st\_jn\_sub\_equ定义](#_struct st_jn_sub_equ  子设备信息)

返回值说明：成功返回0，失败返回非0值

### nmc\_clear\_window\_signal\_source清除窗口信号

NMC\_API int NMC\_CALL\_TYPE nmc\_clear\_window\_signal\_source(long handle, int windowtype, int outputid, int windowid);

函数说明：清除窗口信号

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] windowtype :窗口类型，0非拼接，1拼接

[in] windowid :要设置的输出窗口id, 当为-1时，清除outputid上所有的信号

[in] outputid :要设置的输出id

[in] subsourceid :当sourceid（如NVR）包含多个子设备时候，subsourceid表示子信号源Id

返回值说明：成功返回0，失败返回非0值

## 拼接

### nmc\_get\_large\_screen获取当前已有的拼接屏的信息

函数说明：获取当前已有的拼接屏的信息

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] screenid:想要获取的大屏的id号，-1表示当前nmc上所有大屏

[in] pplarge\_screen\_info：返回拼接屏信息，[参见struct st\_large\_screen\_info](#_struct st_large_screen_info 拼接屏信息)，

[in] large\_screen\_info\_cnt：返回拼接屏信息数量

返回值说明：成功返回0，失败返回非0值

NMC\_API int NMC\_CALL\_TYPE nmc\_get\_large\_screen(long handle, int screenid, struct st\_large\_screen\_info \*\*pplarge\_screen\_info, int \*large\_screen\_info\_cnt);

### nmc\_add\_large\_screen添加多解码器输出拼接

函数说明：添加多解码器输出拼接,如果该屏上由漫游窗口，设置无效

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] plarge\_screen\_info：struct st\_large\_screen\_info \*, 想要添加的拼接的信息，由用户填写，参见[struct st\_large\_screen\_info](#_struct st_large_screen_info 拼接屏信息)，结构体里的struct st\_subscreen\_info须由用户填写

返回值说明：成功返回0，失败返回非0值

NMC\_API int NMC\_CALL\_TYPE nmc\_add\_large\_screen(long handle, struct st\_large\_screen\_info \*plarge\_screen\_info);

### nmc\_rmv\_large\_screen删除多解码器输出拼接

函数说明：删除多解码器输出拼接

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] screenid：想要删除的的拼接屏的id号

返回值说明：成功返回0，失败返回非0值

NMC\_API int NMC\_CALL\_TYPE nmc\_rmv\_large\_screen(long handle, int screenid);

## 漫游

### nmc\_create\_sw\_windows 创建漫游窗口

NMC\_API int NMC\_CALL\_TYPE nmc\_create\_sw\_windows(long handle, int width, int height, struct st\_create\_sw\_window\_info \*pinfo, int info\_cnt);

函数说明：创建漫游窗口

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] matrixid：解码器id

[in] type：0显示编号，1显示背景图

返回值说明：成功返回0，失败返回非0值

### nmc\_delete\_sw\_windows 删除漫游窗口

NMC\_API int NMC\_CALL\_TYPE nmc\_delete\_sw\_windows(long handle, int \*sw\_window\_id\_list, int sw\_window\_id\_list\_len);

函数说明：删除漫游窗口

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] sw\_window\_id\_list：要删除的漫游窗口id列表

[in] sw\_window\_id\_list\_len：要删除的漫游窗口id个数

返回值说明：成功返回0，失败返回非0值

### nmc\_stick\_sw\_windows 置顶漫游窗口

NMC\_API int NMC\_CALL\_TYPE nmc\_stick\_sw\_windows(long handle, int sw\_window\_id);

函数说明：置顶漫游窗口

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] sw\_window\_id\_list：要置顶的漫游窗口id

返回值说明：成功返回0，失败返回非0值

### nmc\_get\_sw\_windows\_info 获取漫游窗口信息

NMC\_API int NMC\_CALL\_TYPE nmc\_get\_sw\_windows\_info(long handle, int sw\_window\_id, struct st\_sw\_window\_info\*\* info, int \*info\_cnt);

函数说明：获取漫游窗口信息

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] sw\_window\_id：要获取的漫游窗口id

[in] sw\_window\_info：要获取的漫游窗口信息，参见[struct st\_sw\_window\_info](#_struct st_sw_window_info 漫游窗口当前状态)

返回值说明：成功返回0，失败返回非0值

## 轮询

### nmc\_get\_survey\_plan获取轮询方案信息

NMC\_API int NMC\_CALL\_TYPE nmc\_get\_survey\_plan(long handle, int survey\_id, struct st\_xy\_survey\_info \*\*ppsurvey\_info, int \*survey\_info\_cnt);

函数说明:获取轮询方案信息

参数说明：

[in] survey\_id 轮询id，-1，表示获取所有

[in] ppsurvey\_info 设备信息

[in] len 设备信息长度

返回值说明：成功返回0，失败返回非0

### nmc\_set\_survey\_plan设置轮询方案

NMC\_API int NMC\_CALL\_TYPE nmc\_set\_survey\_plan(long handle, enum en\_nmc\_operator\_type operator\_type, struct st\_xy\_survey\_info \*ppsurvey\_info);

函数说明:设置轮询方案

参数说明：

[in] survey\_id 轮询id，-1，表示获取所有

[in] ppsurvey\_info 设备信息

[in] len 设备信息长度

返回值说明：成功返回0，失败返回非0

### nmc\_ctrl\_window\_survey控制单窗口轮巡

NMC\_API int NMC\_CALL\_TYPE nmc\_ctrl\_window\_survey(long handle, int ctrl\_type, int survey\_id, int wndtype, int wndid);

函数说明:控制单窗口轮巡

参数说明：

[in] ctrl\_type 0 启动单窗口轮巡， 1 停止单窗口轮巡

[in] survey\_id 轮询id

[in] wndtype 窗口类型,目前只支持单窗口,填为0

[in] wndid 窗口id

返回值说明：成功返回0，失败返回非0

## 搜索

### nmc\_search\_device在网络里搜索设备

NMC\_API int NMC\_CALL\_TYPE nmc\_search\_device(struct st\_xy\_device\_info \*\*device\_info, int \*len);

函数说明:在网络里搜索设备，由nmc\_init的回调函数获得，对应[NDM\_GET\_SEARCH\_DEVCIE](#_附录1)

消息

参数说明：

[in] device\_info 设备信息,目前设置为NULL

[in] len 设备信息长度,目前设置为0

返回值说明：成功返回0，失败返回非-1

## 推流模式

### nmc\_create\_user\_stream\_source 创建一个用户推流源

NMC\_API long NMC\_CALL\_TYPE nmc\_create\_user\_stream\_source(long handle, void\* arg);

函数说明：创建一个用户推流源

参数说明：

[in] handle：由nmc\_login函数创建出得设备对象的句柄

[in] arg：保留参数, 设置为NULL

返回值说明：成功返回非0标示流源句柄，失败返回0

### nmc\_delete\_user\_stream\_source 删除用户推流源

NMC\_API int NMC\_CALL\_TYPE nmc\_delete\_user\_stream\_source(long streamhandle);

函数说明:删除用户推流源

参数说明：

[in] streamhandle：由nmc\_create\_user\_stream\_source函数创建出得流对象的句柄

返回值说明：成功返回0，失败返回非0

### nmc\_set\_user\_stream\_source\_info 设置用户推流源信息

NMC\_API int NMC\_CALL\_TYPE nmc\_set\_user\_stream\_source\_info(long streamhandle, struct st\_video\_stream\_info \*info);

函数说明:设置用户推流源信息

参数说明：

[in] streamhandle：由nmc\_create\_user\_stream\_source函数创建出得流对象的句柄

[in] struct st\_video\_stream\_info \*info, 流源信息, [参见struct st\_video\_stream\_info](#_struct st_video_stream_info  视频流信息)

[in] len, struct st\_video\_stream\_info结构体长度

返回值说明：成功返回0，失败返回非0

### nmc\_set\_user\_stream\_source\_to\_window 设置用户推流源到窗口

NMC\_API int NMC\_CALL\_TYPE nmc\_set\_user\_stream\_source\_to\_window(long streamhandle, int windowtype, int windowid);

函数说明:设置用户推流源到窗口

参数说明：

[in] streamhandle：由nmc\_create\_user\_stream\_source函数创建出得流对象的句柄

[in] windowtype 窗口类型

[in] windowid 窗口id

返回值说明：成功返回0，失败返回非0

### nmc\_clear\_user\_stream\_source\_from\_window 从窗口清除用户推流源

NMC\_API int NMC\_CALL\_TYPE nmc\_clear\_user\_stream\_source\_from\_window(long streamhandle, int windowtype, int windowid);

函数说明:从窗口清除用户推流源

参数说明：

[in] streamhandle：由nmc\_create\_user\_stream\_source函数创建出得流对象的句柄

[in] windowtype 窗口类型

[in] windowid 窗口id, id为0，清除该流之前设置的所有窗口

返回值说明：成功返回0，失败返回非0

### nmc\_push\_user\_stream\_video\_data 推送一帧视频流数据,阻塞发送

NMC\_API int NMC\_CALL\_TYPE nmc\_push\_user\_stream\_video\_data(long streamhandle, struct st\_video\_stream\_info \*info, char\* pstreamdata, int streamdatalen);

函数说明:推送一帧视频流数据,阻塞发送

参数说明：

[in] streamhandle：由nmc\_create\_user\_stream\_source函数创建出得流对象的句柄

[in] info 流流源信息 [参见struct st\_video\_stream\_info](#_struct st_video_stream_info  视频流信息)

[in] pstreamdata 流数据指针

[in] streamdatalen 流数据长度，注意:流数据必须是完整的一帧数据

返回值说明：成功返回0，失败返回非0

### 搜索

# 附录

## 附录1

enum

{

NMC\_EVENT\_DISCONNECT =1000,//连接断线

NDM\_EVENT\_DISCONNECT =1001,//解码器断线

NDM\_EVENT\_CONNECTED =1002,//解码器连接

NDM\_GET\_SEARCH\_DEVCIE =1003,//搜索到设备

};

nmc\_status\_callback中不同type状态取值data对应的结构体将会不同，具体如下表:

|  |  |  |
| --- | --- | --- |
| type | 定义 | data |
| NMC\_EVENT\_DISCONNECT | 连接断线 | 无结构体对应 |
| NDM\_EVENT\_DISCONNECT |  |  |
| NDM\_EVENT\_CONNECTED |  |  |
| NDM\_GET\_SEARCH\_DEVCIE |  | [struct st\_xy\_device\_info](#_struct st_xy_device_info搜索到的设备信息) |

## 附录2

“RTSP-TCP” rtsp协议tcp模式的设备

“RTSP-UDP” rtsp协议udp模式的设备

“AMPONXM” 雄迈协议的设备

“haikangweishi” 海康协议的设备

“DH” 大华协议的设备

“ONVIF” onvif协议的设备

“LKDesktop” 桌面协议的设备