NetSDK Manual for VDP

I. SDK Introduction

Network SDK is kit developed for NVR, NVS, network camera, speed dome and etc. This manual describes function, port and call among functions in each development kit with example.

This manual mainly introduces alarm host related function, support the following models：

DeviceType	DeviceName
VTO XXX	**VideoTalkOutDoor**
VTH XXX	**VideoTalkHome**

Device SDK under Windows

Functional Module	File	Note
Function Library	avglobal.h	head file
	dhassistant.h	head file
	dhnetsdk.h	head file
	dhnetsdk.lib	Lib file
	dhnetsdk.dll	Dll file
	avnetsdk.dll	Dll file
Config Library	dhconfigsdk.h	Config head file
	dhconfigsdk.lib	Config Lib file
	dhconfigsdk.dll	Config Dll file
avnetsdk.dll Aux Library	Infra.dll	function aux library
	Json.dll	function aux library
	Stream.dll	function aux library
	NetFramework.dll	function aux library
	StreamSvr.dll	function aux library

Device SDK under Linux

Functional Module	File	Note
Function Library 1	avglobal.h	head file
	dhassistant.h	head file
	dhnetsdk.h	head file
	libdhdvr.so	basic network library
	libdhnetsdk.so	function library 1
Function Library 2	libavnetsdk.so	function library 2
Config Library	dhconfigsdk.h	head file
	libdhconfigsdk.so	Config library
libavnetsdk.so Aux Library	libInfra.so	function aux library
	libJson.so	function aux library
	libStream.so	function aux library
	libNetFramework.so	function aux library

This SDK Function Library and config library are mandatory. If contain avnetsdk.dll，aux library 2 is mandatory too. If no other encoding tool, take play library and its aux library.

Function Library is main body of device network SDK, used mainly in communication among network client and products and is responsible to control, search, config, get and process stream data.

Config library is to pack and analyze structure of config function.

Aux library is mandatory library of avnetsdk. Without these files will influence usage of avnetsdk.

II. Terminology

Forced Card:

One of A&C card, used to unlock door when cardholder is threatened by someone. A&C system will identify forced card and alarm, activate other operations.

目录

1. Function Call Sequence

1） SDK initialization(**CLIENT_Init()**), login(**CLIENT_Login()**)and etc.
2） Configure device(**CLIENT_GetNewDevConfig()**, **CLIENT_SetNewDevConfig()** and etc.)
3） Monitor (**CLIENT_RealPlayEx**()、**CLIENT_StopRealPlay**())
4） Configure device(**CLIENT_GetNewDevConfig()**, **CLIENT_SetNewDevConfig()** and etc.)
5） Control device, listen events, query recordsets, query log(**CLIENT_ControlDevice()**, **CLIENT_SetDVRMessCallBack()**,**CLIENT_StartListenEx()**, **CLIENT_StopListen()**, **CLIENT_FindRecord()**,**CLIENT_FindNextRecord()**,**CLIENT_QueryRecordCount()**, **CLIENT_FindRecordClose()** , **CLIENT_QueryDeviceLog()**,**CLIENT_QueryDevLogCount()**)
6） Talk(**CLIENT_StartTalkEx()**、**CLIENT_StopTalkEx()**)
7） Logout, SDK anti-initialization(**CLIENT_Logout()**, **CLIENT_Cleanup()**)

2. Universal functions

1). Set network param

void **CLIENT_SetNetworkParam**(NET_PARAM* *pNetParam*);

Note：Set login network port. Call port before login is recommended. If not call, SDK uses default value.

Parameter：

[in] *pNetParam* network parameter structure indicator, please refer to NET_PARAM

Return value：N/A

NET_PARAM structure:

```
// Set login parameter
typedef struct
{
    int         nWaittime;          //  and etc. overtime waiting(unit is ms)，as 0 default 5000ms
    int         nConnectTime;       // overtime connection(unit is ms)，as 0 default 1500ms
    int         nConnectTryNum;     // connection attempts，as 0 default 1 time
    int         nSubConnectSpaceTime;  // sum of sub connection waiting time(unit is ms)，as 0 default 10ms
    int         nGetDevInfoTime;    // get device information overtime，as 0 default 1000ms
    int         nConnectBufSize;    // each connection receiving data buffer size(unit is byte)，as 0 default 250*1024
    int         nGetConnInfoTime;   // get sub connection information overtime(unit is ms)，as 0 default 1000ms
    int         nSearchRecordTime;  // by time search record file overtime(unit is ms),as 0 default is 3000ms
    int         nsubDisconnetTime;  // detect sub connection offline waiting time sum(unit is ms)，as 0 default is 60000ms
    BYTE        byNetType;          // network type, 0-LAN, 1-WAN
```

```
    BYTE        byPlaybackBufSize;      // playback data receiving buffer size（unit is M），
as 0 default is 4M
    BYTE        byReserved1[2];         // Align
 Int            nPicBufSize;            // real time picture receiving buffer size（unit is
 byte），as 0 default 2*1024*1024
    BYTE        bReserved[4];           // reserved text
} NET_PARAM;
```

3. Device Basic Information

3.1 Device Type

LLONG CLIENT_Login(char *pchDVRIP, WORD wDVRPort, char *pchUserName, char *pchPassword, LPNET_DEVICEINFO lpDeviceInfo, int *error = 0);

Parameter：

[in] pchDVRIP	Device IP
[in] wDVRPort	Device Port
[in] pchUserName	Username
[in] pchPassword	Password
[out] lpDeviceInfo	Device Info
[out] error	If port failed, return error code

```
// Device Info
typedef struct
{
    BYTE        sSerialNumber[DH_SERIALNO_LEN];     // SN
    BYTE        byAlarmInPortNum;           // DVR alarm input no.
    BYTE        byAlarmOutPortNum;          // DVR alarm output no.
    BYTE        byDiskNum;                  // DVR HDD quantity
    BYTE        byDVRType;                  // DVR type, see    NET_DEVICE_TYPE
    BYTE        byChanNum;                  // DVR channel quantity
} NET_DEVICEINFO, *LPNET_DEVICEINFO;
```

3.2 Software Version

BOOL CLIENT_QueryDevState(LLONG lLoginID, int nType, char *pBuf, int nBufLen, int *pRetLen, int waittime);

Parameter：

[in]lLoginID	CLIENT_Login() return value
[in]nType	DH_DEVSTATE_SOFTWARE
[in] pBuf	Corresponding structure DHDEV_VERSION_INFO
[in] nBufLen	Structure Size
[in] pRetLen	Return structure size
[in] waittime	Overtime, unit is ms

```
typedef struct
{
    char        szDevSerialNo[DH_DEV_SERIALNO_LEN];    // SN
    char        byDevType;                  // Device Type, see NET_DEVICE_TYPE
    char        szDevType[DH_DEV_TYPE_LEN];  // Device detailed model, string format,
may be null
    int         nProtocalVer;           // Protocol Version
    char        szSoftWareVersion[DH_MAX_URL_LEN];
    DWORD       dwSoftwareBuildDate;
    char        szDspSoftwareVersion[DH_MAX_URL_LEN];
    DWORD       dwDspSoftwareBuildDate;
    char        szPanelVersion[DH_MAX_URL_LEN];
    DWORD       dwPanelSoftwareBuildDate;
    char        szHardwareVersion[DH_MAX_URL_LEN];
    DWORD       dwHardwareDate;
    char        szWebVersion[DH_MAX_URL_LEN];
    DWORD       dwWebBuildDate;
    char        reserved[256];
} DHDEV_VERSION_INFO;
```

3.3 Date Setup/Get

BOOL CLIENT_QueryDeviceTime(LLONG *lLoginID*, LPNET_TIME *pDeviceTime*, int *waittime*=1000);

Note：Apply to get current front-end device time for sync time

Parameter：

[in] lLoginID	CLIENT_Login() return value，when it is 0, it means abnormal login
[out] pDeviceTime	Receiving device time indicator, see LPNET_TIME
[in] waittime	and waiting overtime，default 1000ms

Return value：successfully return TRUE，failed to return FALSE. Port failed to return please call CLIENT_GetLastError()to get error code，find cause of error via error code.

BOOL CLIENT_SetupDeviceTime(LLONG *lLoginID*, LPNET_TIME *pDeviceTime*);

Note：Apply to sync current front-end device time with this device time.

Parameter：

[in] lLoginID	CLIENT_Login()return value，when it is 0, it means abnormal login
[in] pDeviceTime	Set device time indicator，see LPNET_TIME

Return value：successfully return TRUE，failed return to FALSE. Port failed to return please callCLIENT_GetLastError()to get error code，find error cause via error code.

LPNET_TIME Structure：

```
// Time
typedef struct
{
```

```
    DWORD       dwYear;     // Year
    DWORD       dwMonth;    // Month
    DWORD       dwDay;      // Day
    DWORD       dwHour;     // Hour
    DWORD       dwMinute;   // Minute
    DWORD       dwSecond;   // Second
} NET_TIME,*LPNET_TIME;
```

3.4 MAC

BOOL CLIENT_QueryDevState(LLONG *lLoginID*, int *nType*, char *pBuf, int *nBufLen*, int *pRetLen*, int *waittime*);

Parameter：

[in]lLoginID	CLIENT_Login() return value
[in]nType	DH_DEVSTATE_NETINTERFACE
[in] pBuf	Corresponding structure DHDEV_NETINTERFACE_INFO
[in] nBufLen	Structure size
[in] pRetLen	Return structure size
[in] waittime	Overtime, unit is ms

```
typedef struct tagDHDEV_NETINTERFACE_INFO
{
    int         dwSize;
    BOOL        bValid;         // Valid or not
    BOOL        bVirtual;       // Virtual network card
    int         nSpeed;         // Network card theoretical speed(Mbps)
    int         nDHCPState;     // 0-not enabled, 1-getting, 2-successfully got
    char        szName[DH_NETINTERFACE_NAME_LEN];   // Network Name
    char        szType[DH_NETINTERFACE_TYPE_LEN];// Network Type
    char        szMAC[DH_MACADDR_LEN]; // MAC address
    char        szSSID[DH_MAX_SSID_LEN]; // SSID, only valid for wireless network(szType
== "Wireless")
 char           szConnStatus[DH_MAX_CONNECT_STATUS_LEN]; // Wifi, 3G connection
 status， "Inexistence" : network inexists "Down"： off "Disconn"： not connected
 "Connecting"：connecting "Connected"： connected
    int         nSupportedModeNum;  // Actual 3G supported network mode number
    char        szSupportedModes[DH_MAX_MODE_NUM][DH_MAX_MODE_LEN];// 3G
supported network mode  "TD-SCDMA", "WCDMA", "CDMA1x", "EDGE", "EVDO"
} DHDEV_NETINTERFACE_INFO;
```

4. Listen to Event

0).Set call function to get event info

void **CLIENT_SetDVRMessCallBack**(fMessCallBack *cbMessage*, LDWORD *dwUser*);

Note: set call function to listen to event
Parameter:
[in] cbMessage message call function，may call device status，if alarm status can get via this call; when setup is 0, it means no call allowed.

typedef BOOL (CALLBACK *fMessageCallBack)(
LONG *lCommand*, Call type，see “supported event”
LLONG *lLoginID*, CLIENT_Login return value
Char *pBuf*, Receive alarm data buffer，and correspond to lCommand value，see ”supported event”
DWORD *dwBufLen*, pBuf length(unit is byte)
Char *pchDVRIP*, Device ip
LONG *nDVRPort*, Port
LDWORD *dwUser* User custom data
);

[in] dwUser User custom data
Return value: N/A

1).Enable listening to event
BOOL **CLIENT_StartListenEx**(LLONG *lLoginID*);
Note: enable listening to event
Parameter:
[in] lLoginID CLIENT_Login() return value
Return value: TRUE is successful，FALSE is failed

2).Stop listening to event
BOOL **CLIENT_StopListen**(LLONG *lLoginID*);
Note: stop listening to event
Parameter:
[in] lLoginID CLIENT_Login() return value
Return value: TRUE is successful，FALSE is failed

3). Supported Event

Event Name	Command Type	Info(usually as structure)
A&C event	DH_ALARM_ACCESS_CTL_EVENT	ALARM_ACCESS_CTL_EVENT_INFO
Intrusion event info	DH_ALARM_CHASSISINTRUDED	ALARM_CHASSISINTRUDED_INFO
Local alarm event	DH_ALARM_ALARM_EX2	ALARM_ALARM_INFO_EX2

5. Config Functions

a) BOOL **CLIENT_GetNewDevConfig**(LLONG *lLoginID*, char* *szCommand*, int *nChannelID*, char* *szOutBuffer*, DWORD *dwOutBufferSize*, int *error*, int *waittime*);

Note： get config，follow string format，each string included info is analyzed by CLIENT_ParseData.

Parameter：

[in] lLoginID CLIENT_Login() return value，when it is 0, it means abnormal login

[in] szCommand Command parameter， see following CLIENT_ParseData command parameter note

[in] nChannelID Channel no.

[out] szOutBuffer Output buffer

[out] dwOutBufferSize Output buffer size

[out] error Error code

0	**Successful**
1	Failed
2	Invalid data
3	Temporarily cannot set
4	No right

[in] waittime waiting overtime

b) BOOL **CLIENT_ParseData**(char* *szCommand*, char* *szInBuffer*, LPVOID *lpOutBuffer*, DWORD *dwOutBufferSize*, void* *pReserved*);

Note： This port is used with CLIENT_GetNewDevConfig()， after using CLIENT_GetNewDevConfig() to get string config info，use this port get info from config info.

Parameter：

[in] szCommand command parameter

命令参数	相应结构体
CFG_CMD_BUILDING	VTO Building(CFG_BUILDING_INFO)
CFG_CMD_BUILDING_EXTERNAL	VTO Building External(CFG_BUILDING_EXTERNAL_INFO)
CFG_CMD_DIALRULE	Dialrule(CFG_DIALRULE_INFO)
CFG_CMD_NASEX	NAS(CFG_NAS_INFO_EX)
CFG_CMD_VTNOANSWER_FORWARD	VT no answer forward(CFG_VT_NOANSWER_FORWARD_INFO)
CFG_CMD_VTO_CALL	VTO Call(CFG_VTO_CALL_INFO)

[in] szInBuffer Input buffer, string config buffer

[out] lpOutBuffer Output buffer, corresponding to szCommand structure type

[out] dwOutBufferSize Output buffer size

[out] pReserved Return data length

c) BOOL **CLIENT_PacketData**(char* *szCommand*, LPVOID *lpInBuffer*, DWORD *dwInBufferSize*, char* *szOutBuffer*, DWORD *dwOutBufferSize*);

Note：This port is used with CLIENT_SetNewDevConfig(). After using CLIENT_PacketData()，set packed string info via CLIENT_SetNewDevConfig() to device.

Parameter：

[in]	szCommand	Command parameter， see above CLIENT_ParseData chart
[in]	lpInBuffer	Input buffer，structure see CLIENT_ParseData chart
[in]	dwInBufferSize	Input buffer size
[out]	szOutBuffer	Output buffer
[out]	dwOutBufferSize	Output buffer size

d) BOOL **CLIENT_SetNewDevConfig**(LLONG *lLoginID*, char* *szCommand*, int *nChannelID*, char* *szInBuffer*, DWORD *dwInBufferSize*, int* *error*, int* *restart*, int *waittime*=500);

Note：Set config，by string format，each string pack info is formed by CLIENT_PacketData.

Parameter：

[in]	lLoginID	CLIENT_Login() return value，when it is 0, it means abnormal login
[in]	szCommand	Please refer to CLIENT_ParseData note
[in]	nChannelID	Channel No.
[in]	szInBuffer	Input buffer
[in]	dwInBufferSize	Input buffer size
[out]	error	Error code

0	**Successful**
1	Failed
2	Data invalid
3	Temporarily cannot set
4	No right

[out]	restart	After config, you shall reboot device，1 means reboot，0 means no reboot
[in]	waittime	and waiting overtime

6. Control Function

BOOL **CLIENT_ControlDevice**(LLONG *lLoginID*, CtrlType *type*, void * *param*, int *waittime* = 1000);

Note：device control port function

Parameter：

[in]	lLoginID	CLIENT_Login() return value, when it is 0, it means abnormal login

[in] type Control type，see following chart CtrlType

[in] param During control, input parameter structure，matching type

Control Type	Corresponding Structure
DH_CTRL_ACCESS_OPEN	A&C –unlock (corresponding structure NET_CTRL_ACCESS_OPEN)

[in] waittime and waiting overtime，default 1000ms

Return value：successful return TRUE，failed return FALSE. Port return please call CLIENT_GetLastError() to get error code，find error cause via error code.

7. RecordSets Control Function

BOOL CLIENT_ControlDevice(LLONG *lLoginID*, CtrlType *type*, void * *param*, int *waittime* = 1000);

Note：Device control port function

Parameter：

[in] lLoginID CLIENT_Login() return value，when it is 0, it means abnormal login

[in] type Control type，see following chart CtrlType

[in] param During control, input parameter structure info，marching type

Control Type	Corresponding Structure
DH_CTRL_RECORDSET_INSERT	Add record, get record set no.(corresponding NET_CTRL_RECORDSET_INSERT_PARAM)
DH_CTRL_RECORDSET_UPDATE	Update come record no.(corresponding NET_CTRL_RECORDSET_PARAM)
DH_CTRL_RECORDSET_REMOVE	According to record set no., delete record(corresponding NET_CTRL_RECORDSET_PARAM)
DH_CTRL_RECORDSET_CLEAR	Clear all record set info(corresponding NET_CTRL_RECORDSET_PARAM)

[in] waittime and waiting overtime，default 1000ms

Return value：successfully return TRUE，failed return FALSE. Port return failed please call CLIENT_GetLastError() to get error code，find error cause via error code..

BOOL CLIENT_QueryDevState(LLONG *lLoginID*, int *nType*, char* *pBuf*, int *nBufLen*, int* *pRetLen*, int *waittime*=1000);

Note：search device status port

Parameter：

[in] lLoginID CLIENT_Login()return value，when it is 0, it means abnormal login

[in] nType Search info type

[out] pBuf Output parameter, used to get search return data buffer. According to search type, return data structures are different, see

Search Info Type	Corresponding Structure

DH_DEVSTATE_DEV_RECORDSET	Search device record info(corresponding NET_CTRL_RECORDSET_PARAM)

[in] nBufLen Buffer length，unit is byte

[out] pRetLen Output parameter，actual return data length，unit is byte

[in] waittime Search status and waiting time， default 1000ms

Return value：successful return TRUE，failed return FALSE. Port return please call CLIENT_GetLastError() to get error code，find error cause via error code..

RecordSet Type	Corresponding Structure
NET_RECORD_ACCESSCTLCARD	AccessControl Card Info(NET_RECORDSET_ACCESS_CTL_CARD)
NET_RECORD_ACCESSCTLCARDREC_EX	AccessControl Card Record(swiping info)(NET_RECORDSET_ACCESS_CTL_CARDREC)
NET_RECORD_ANNOUNCEMENT	Announcement Info(NET_RECORD_ANNOUNCEMENT_INFO)
NET_RECORD_ALARMRECORD	Alarm Event Info(NET_RECORD_ALARMRECORD_INFO)

8. RecordSet Search

BOOL CLIENT_QueryNewSystemInfo (LLONG *lLoginID*, char* *szCommand*, int *nChannelID*, char* *szOutBuffer*, DWORD *dwOutBufferSize*, int *error*, int *waittime*=1000);

Note：get record search capacity.

Parameter：

[in]	lLoginID	CLIENT_Login()return value, when it is 0, it means abnormal login
[in]	szCommand	Command parameter, CFG_CAP_CMD_RECORDFINDER
[in]	nChannelID	Channel no.
[out]	szOutBuffer	Output buffer, see following CFG_CAP_RECORDFINDER_INFO
[in]	dwOutBufferSize	Output buffer size
[out]	error	Error Code
[in]	waittime	Overtime

```
typedef struct tagCFG_CAP_RECORDFINDER_INFO
{
    int nMaxPageSize; //  Max page size
}CFG_CAP_RECORDFINDER_INFO;
```

BOOL CLIENT_FindRecord(LLONG *lLoginID*, NET_IN_FIND_RECORD_PARAM* *pInParam*, NET_OUT_FIND_RECORD_PARAM* *pOutParam*, int *waittime*=1000);

Note：Search record by filter，generate search handle.

Parameter：

[in]	lLoginID	CLIENT_Login()return value
[in]	pInParam	Search record parameter，see NET_IN_FIND_RECORD_PARAM
[out]	pOutParam	Return search handle，see NET_OUT_FIND_RECORD_PARAM
[in]	waittime	waiting overtime

Return value：successful return TRUE，failed return FALSE. Port return please call CLIENT_GetLastError() to get error code，find error cause via error code..

Input parameter：

```
typedef struct _NET_IN_FIND_RECORD_PARAM
{
    DWORD                   dwSize;
    EM_NET_RECORD_TYPE      emType;             // Record type to search
    void*                   pQueryCondition; // Search type corresponding filter
} NET_IN_FIND_RECORD_PARAM;
```

Output parameter:

```
typedef struct _NET_OUT_FIND_RECORD_PARAM
{
    DWORD                   dwSize;
    LLONG                   lFindeHandle;      // Search record handle，SN search
} NET_OUT_FIND_RECORD_PARAM;
```

BOOL CLIENT_FindNextRecord(NET_IN_FIND_NEXT_RECORD_PARAM* *pInParam*, NET_OUT_FIND_NEXT_RECORD_PARAM * *pOutParam*, int *waittime*=1000);

Note：get record，search handle as CLIENT_FindRecord port to get.

Parameter：

[in]	lLoginID	CLIENT_Login()return value，when it is 0, it means abnormal login
[in]	pInParam	Search record parameter，see NET_IN_FIND_NEXT_RECORD_PARAM
[out]	pOutParam	Return record，see NET_OUT_FIND_NEXT_RECORD_PARAM
[in]	waittime	and waiting overtime

Return value：successful return TRUE，failed return FALSE. Port return please call CLIENT_GetLastError() to get error code，find error cause via error code..

Input parameter：

```
typedef struct _NET_IN_FIND_NEXT_RECORD_PARAM
{
    DWORD                   dwSize;
    LLONG                   lFindeHandle;      // Search handle
    int                     nFileCount;        // Current search record item number
} NET_IN_FIND_NEXT_RECORD_PARAM;
```

Output parameter:

```
typedef struct _NET_OUT_FIND_NEXT_RECORD_PARAM
```

```
{
    DWORD                   dwSize;
    void*                   pRecordList;        // Record parameter，user allocate memory
    int                     nMaxRecordNum;      // List record number
    int                     nRetRecordNum;      // Searched record item，when searched item is less then designated item quantity, search ends
} NET_OUT_FIND_NEXT_RECORD_PARAM;
```

BOOL CLIENT_QueryRecordCount(NET_IN_QUEYT_RECORD_COUNT_PARAM* *pInParam*, NET_OUT_QUEYT_RECORD_COUNT_PARAM* *pOutParam*, int *waittime*=1000);

Note：get record number，search handle as CLIENT_FindRecord port to get.

Parameter：

[in] lLoginID	CLIENT_Login()return value，when it is 0, it means abnormal login
[in] pInParam	Search record parameter，see NET_IN_QUEYT_RECORD_COUNT_PARAM
[out] pOutParam	Retuen record number，see NET_OUT_QUEYT_RECORD_COUNT_PARAM
[in] waittime	and waiting overtime

Return value：successful return TRUE，failed return FALSE. Port return please call CLIENT_GetLastError() to get error code，find error cause via error code..

Input parameter：

```
typedef struct _NET_IN_QUEYT_RECORD_COUNT_PARAM
{
    DWORD                   dwSize;
    LLONG                   lFindeHandle;       // Search handle
} NET_IN_QUEYT_RECORD_COUNT_PARAM;
```

Output parameter:

```
typedef struct _NET_OUT_QUEYT_RECORD_COUNT_PARAM
{
    DWORD                   dwSize;
    int                     nRecordCount;                   //Device return record item
} NET_OUT_QUEYT_RECORD_COUNT_PARAM;
```

BOOL CLIENT_FindRecordClose(LLONG *lFindHandle*);

Note：end search.

Parameter：

[in] lFindHandle	CLIENT_FindRecord () return value

Return value：successful return TRUE，failed return FALSE. Port return please call CLIENT_GetLastError() to get error code，find error cause via error code..

9. Device Actively Report IP(under equivalent network)

1. Enable listening

LLONG CLIENT_ListenServer (char* *ip*, WORD *port*, int *nTimeout*, fServiceCallBack *cbListen*, LDWORD *dwUserData*);

[in] ip It intends to start service ip, if only one card, then fill the machine ip often listen

[in] port Port Number intends to start service

[in] nTimtout This parameter is invalid, be random value

[in] cbListen Listening call function

[in] dwUserData Call function parameter

Return Value: Successful returns TRUE, otherwise returns FALSE. Interface returns failed call CLIENT_GetLastError () to get the error code, determine the cause of error by the error code.

typedef int(CALLBACK fServiceCallBack)(

LLONG	*lHandle*,	server handle
char	**ip*,	Device end IP connected to server
WORD	*port*,	Connected to the device side port services
LONG	*lCommand*,	Device request command, here is NET_DEV_NOTIFY_IP_RETURN
void	**pParam*,	Device request command parameter, device SN
DWORD	*dwParamLen*,	pParam valid length, unit byte
LDWORD	*dwUserData*	*Call custom parameter*

);

Note：Field *lCommand* reported to NET_DEV_NOTIFY_IP_RETURN, ip for the device now can be found at the address, pParam identify the device serial number. Get ip can be found in the usual type of procedure.

2. Turn off listening

BOOL CLIENT_StopListenServer (LLONG *lServerHandle*);

10. Audio Talk(Call or Called)

0. Note：must integrate VTHStack library

1. Register

LLONG **CLIENT_VT_RegisterVto** (const char *pszIp, int nPort, int nWaitTime);

Note： register to VTO where VTH locates

Parameter：

[in] pszIp VTO IP where VTH locates

[in] nPort VTO port

[in] nWaitTime Max overtime to wait for VTO response ， unit is ms

Return Value: Returns a non-zero success, failed to return to zero.

2. Set talk parameter

BOOL **CLIENT_SetDeviceMode**(LLONG lLoginID, EM_USEDEV_MODE emType, void*

pValue)；

Note：specific talk mode, certain audio encode mode, talk parameter and other to audio talk.

Parameter：

[in] lLoginID CLIENT_Login() return value， when it is 0, means abnormal login

[in] emType Enumeration value， here is DH_TALK_VT_PARAM type

Enumeration Value	Definiton
DH_TALK_VT_PARAM	Set video intercom related parameters (corresponding to NET_VT_TALK_PARAM)

[in] pValue Here is emType corresponding structure， is NET_VT_TALK_PARAM

Return Value: Successful returns TRUE, otherwise returns FALSE. Interface returns failed call CLIENT_GetLastError () to get the error code, determine the cause of error by the error code.

3. Enable talk

LLONG **CLIENT_StartTalkEx**(LLONG lLoginID, pfAudioDataCallBack pfcb, LDWORD dwUser)；

Note：send audio talk request to device.

Parameter：

[in] lLoginID CLIENT_Login() return value， when it is 0, means abnormal login

[in] pfcb

(typedef void (**CALLBACK *pfAudioDataCallBack**)(

LLONG lTalkHandle, // handle returned by the local interface;

char* pDataBuf, // for the content of the audio data,

DWORD dwBufSize, // for the length of the audio data (unit: byte),

BYTE byAudioFlag, // audio data ownership flag

LDWORD dwUser);) // user-defined data

Label	Date Type
0	Indicates that the local audio data recording library collection
1	Device indicates the received audio data sent by device

[in] dwUser User define info, return to user via call function

Return Value: Successful returns nonzero, speaking for the device handle, else return 0.

4. Turn on the local recording (only in client mode, the NetSDK call PlaySDK achieve encoding and decoding audio collection)

BOOL **CLIENT_RecordStartEx**(LLONG lLoginID)；

Note: Start a local recording function, recording audio data collected by CLIENT_StartTalkEx out callback function callback to the user, the corresponding action is CLIENT_RecordStopEx (LLONG lLoginID).

Parameter：

[in] lLoginID login return device handle

Return Value: Successful returns TRUE, otherwise returns FALSE. Interface returns failed call CLIENT_GetLastError () to get the error code, determine the cause of error by the error code.

5. Turn off local recording

   BOOL **CLIENT_RecordStopEx**(LLONG lLoginID);

   Note：stop local recording，corresponding operation is LIENT_RecordStartEx(LLONG lLoginID)。

   Parameter：

   [in] lLoginID login return device handle

   Return Value: Successful returns TRUE, otherwise returns FALSE. Interface returns failed call CLIENT_GetLastError () to get the error code, determine the cause of error by the error code.

6. Turn off talk

   BOOL **CLIENT_StopTalkEx**(LLONG lTalkHandle);

   Port note：stop audio talk.

   Parameter note：

   [in] lTalkHandle Audio talk handle，CLIENT_StartTalkEx return

   Return Value: Successful returns TRUE, otherwise returns FALSE. Interface returns failed call CLIENT_GetLastError () to get the error code, determine the cause of error by the error code.

7. Unregister

   BOOL **CLIENT_VT_UnRegisterVto**(LLONG lLoginID);

   Note：lagout from VTO.

   Parameter：

   [in] lLoginID CLIENT_VT_RegisterVto return registration handle

   Return Value: Successful returns TRUE, otherwise returns FALSE. Interface returns failed call CLIENT_GetLastError () to get the error code, determine the cause of error by the error code.

11. To do