VIVOTEK NETWORK DEVELOPMENT PLATFORM

Data Broker

Version 5.0.0.24

2009/8/31

VIVOTEK may make changes to specifications and product descriptions at any time, without notice.

The following is trademarks of VIVOTEK Inc., and may be used to identify VIVOTEK products only: VIVOTEK. Other product and company names contained herein may be trademarks of their respective owners.

TABLE OF CONTENTS

1. OVERVIEW

1.1. Introduction

This document describes how to use DataBroker module to get video and audio frames from remote server. In the other hand, you can get video and audio packet using your own network module and use this DataBroker module to help unpacketize and parse the packets you have received.

1.2. File Structure

FILE	DESCRIPTION
doc\VNDP_DataBroker_API.pdf	This manual
lib\d_DataBroker.lib	The dynamic linking library
lib\DataBroker.dll	The dynamic runtime library
inc\DataBroker.h	Header file
inc\SrvTypeDef.h	Common definition file
inc\datapacketdef.h	Data packet definition file

2. PROGRAMMER'S GUIDE

2.1. Using DataBroker Module

You can use DataBroker module in two ways.

DataBroker and Connection

In the first situation, if you can't get packets from server through network by yourselves, you can use the network client embedded in DataBroker module to help you get packets from server. DataBroker would then unpacketize and parse the received packets automatically. So it's easy for you to get audio and video frames from server throughout the DataBroker module.

Call DataBroker_Initial function to acquire a DataBroker handle, you should specify the maximum connections that the DataBroker can handle when initializing. Using this DataBroker handle, you can call DataBroker_CreateConnection to create as many connections as you need (not exceed the maximum number of connections you specified). Every connection instance could establish a connection to a remote server (call DataBroker_Connect). Through the AV callback function, you can receive the unpacketized and parsed frames from the remote server. And the status callback function could tell you the status (connection established, starting to receive data, error conditions, etc…) when connecting to the server and receiving stream data from server.

Input Network Packet

The second situation is, if you want to implement the network client in your own way, using Input of DataBroker module to help you just unpacketize and parse the packets you have received.

Call DataBroker_CreateInputEx to acquire an Input handle. Call DataBroker_InputPacketEx using this Input handle and feeding packet data into it, then you can receive the unpacketized and parsed frames through the AV callback function. You also can get the status information from status callback function.

3. Sample Code

In this chapter we will introduce you the basic function of DataBroker. The sample code in the document are simpifiled, but can tell you the most important concepts of DataBroker.

If you want to create an application with DataBroker, you should refer to the sample code in the SDK package, too.

3.1. ConnectToDevice

DESCRIPTION

Connect to a video device, it could be a network camera or a video server.

SAMPLE CODE

STEP 1. Prepare the callback functions

You have to prepare 2 callback functions for DataBroker at first, one for Audio/Video packet callback and another for Status callback.

```
SCODE __stdcall DataBrokerAVCallBack(DWORD dwContext,TMediaDataPacketInfo
*ptMediaDataPacket)
{
  printf("Time stamp : %u %u \n",
          ptMediaDataPacket->dwFirstUnitSecond,
          ptMediaDataPacket->dwFirstUnitMilliSecond);
  return S_OK;
}

SCODE __stdcall DataBrokerStatusCallBack(DWORD dwContext, TDataBrokerStatusType
tStatusType, PVOID pvParam1, PVOID pvParam2)
{
   switch (tStatusType)
   {
          case eOnConnectionInfo:
          printf("Get eOnConnectionInfo, start to receive packet\n");
          break;

          case eOnStopped:
          printf("Get eOnStopped, stop to receive packet\n");
          break;
   }
  return S_OK;
}
```

STEP 2. Initial DataBroker

```
scRet = DataBroker_Initial(&hDataBroker, nMaxChannel, NULL,NULL,
                                mctALLCODEC, 0, DATABROKER_VERSION);
```

STEP 3. Create Connection

```
scRet = DataBroker_CreateConnection(hDataBroker, &hConn);
```

STEP 4. Set Options for Connection

```
TDataBrokerConnectionOptions tOpt;
memset(&tOpt, 0, sizeof(tOpt));
//Call the user-defined auxiliary function CreateConnectionOption() to construct the tOpt
CreateConnectionOption(&tOpt);
{
ptOpt->pzIPAddr = "192.168.1.100";
ptOpt->wHttpPort = 80;
ptOpt->pzUID = "root";
ptOpt->pzPWD = "XXXX";
ptOpt->dwProtocolType = eptTCP;
ptOpt->dwMediaType =  (emtVideo | emtAudio);
ptOpt->pzServerType = _T("auto");
ptOpt->dwStatusContext = NULL;
ptOpt->dwAVContext = NULL;
ptOpt->pfStatus = DataBrokerStatusCallBack;
ptOpt->pfAV = DataBrokerAVCallBack;
ptOpt->dwFlags = eConOptProtocolAndMediaType |
                 eConOptHttpPort |
                 (eConOptStatusCallback | eConOptStatusContext) |
                 (eConOptAVCallback | eConOptAVContext);
}
```

STEP 5. Connect to Device

```
scRet = DataBroker_Connect(hConn);
```

STEP 6. Wait until you want to stop

```
while( !_kbhit() )
{
    Sleep(16);
}
```

TIPS

The DataBroker_Connect() function is an asynchronous function, the connection information should be get in StatusCallback function rather than return code.

You can refer to the SDK package for full sample code.

3.2. ConnectToMultipleStreamDevice

DESCRIPTION

Connect to a device with multiple stream(for example, IP7161), and we will connect to the user specified stream for streaming.

SAMPLE CODE

STEP 1. Initial DataBroker

scRet = DataBroker_Initial(&hDataBroker, nMaxChannel, DataBrokerStatusCallBack, DataBrokerAVCallBack, mctALLCODEC, 0, DATABROKER_VERSION);

STEP 2. Create Connection

scRet = DataBroker_CreateConnection(hDataBroker, &hConn);

STEP 3. Set Options for Connection

```
TDataBrokerConnectionOptions tOpt;
memset(&tOpt, 0, sizeof(tOpt));
//Call the user-defined auxiliary function CreateConnectionOption() to construct the tOpt
CreateConnectionOption(&tOpt);
{
ptOpt->pzIPAddr = “192.168.1.100”;
ptOpt->wHttpPort = 80;
ptOpt->pzUID = “root”;
...
}
```

STEP 4 Set the Stream Index

scRet = DataBroker_SetConnectionExtraOption(hConn, eOptStreamIndex, dwStreamIndex, 0);

STEP 5. Connect to Device

scRet = DataBroker_Connect(hConn);

STEP 6. Wait until you want to stop

```
while( !_kbhit() )
{
    Sleep(16);
}
```

TIPS

The stream index is zero-based. The stream index is 0 for Stream 1 and so on.

3.3. ConnectToVideoServer

DESCRIPTION

Connect to a video server with multiple channels(for example, VS2403), and we will connect to the user specified channel for streaming.

SAMPLE CODE

STEP 1. Initial DataBroker

```
scRet = DataBroker_Initial(&hDataBroker, nMaxChannel, DataBrokerStatusCallBack,
DataBrokerAVCallBack, mctALLCODEC, 0, DATABROKER_VERSION);
```

STEP 2. Create Connection

```
scRet = DataBroker_CreateConnection(hDataBroker, &hConn);
```

STEP 3. Set Options for Connection

```
TDataBrokerConnectionOptions tOpt;
memset(&tOpt, 0, sizeof(tOpt));
//Call the user-defined auxiliary function CreateConnectionOption() to construct the tOpt
CreateConnectionOption(&tOpt);
{
ptOpt->pzIPAddr = “192.168.1.100”;
ptOpt->wHttpPort = 80;
ptOpt->pzUID = “root”;
...
ptOpt->wCam = 1;          // Channel 1
}
```

STEP 4. Connect to Device

```
scRet = DataBroker_Connect(hConn);
```

STEP 6. Wait until you want to stop

```
while( !_kbhit() )
{
    Sleep(16);
}
```

TIPS

The wCam is start from 1.

3.4. GetDIDONotify

DESCRIPTION

The DI/DO information will be report with the status callback.

SAMPLE CODE

STEP 1: Create connection and do connect as usual.

```
scRet = DataBroker_Initial(&hDataBroker, nMaxChannel, DataBrokerStatusCallBack,
DataBrokerAVCallBack, mctALLCODEC, 0, DATABROKER_VERSION);
scRet = DataBroker_CreateConnection(hDataBroker, &hConn);
TDataBrokerConnectionOptions tOpt;
memset(&tOpt, 0, sizeof(tOpt));
CreateConnectionOption(&tOpt);
tOpt.dwStatusContext = (DWORD) ptDIDOInfo;
scRet = DataBroker_Connect(hConn);
```

STEP 2: Process the case of eStatusType is eOnDiDo from StatusCallack

```
SCODE __stdcall DataBrokerStatusCallBack(DWORD dwContext, TDataBrokerStatusType
eStatusType, PVOID pvParam1, PVOID pvParam2)
{
switch (eStatusType)
{
    case eOnConnectionInfo:
    printf("Get eOnConnectionInfo, start to receive packet\n");
    break;

    case eOnStopped:
    printf("Get eOnStopped, stop to receive packet\n");
    break;
```

```
case eOnDiDo:
{
```

STEP 3: Cast dwContext to TDIDOInfo*

```
TDIDOInfo* ptDIDOInfo = (TDIDOInfo *) dwContext;
```

STEP 4: Get the DI and DO value from pvParam2

```
//The Low-word is represented the DI Value
WORD wDIVal = LOWORD((DWORD) pvParam2);
//The High-word is represented the DO Value
WORD wDOVal = HIWORD((DWORD) pvParam2);
```

STEP 5: Parse DI related bits

```
for (DWORD dwDIIdx = 0; dwDIIdx < ptDIDOInfo->dwDINum; dwDIIdx++)
{
//Bit 0 for DI1, Bit 1 for DI2..., we use bit-operator to check the value.
if (1 == (wDIVal & (1 << dwDIIdx)))
        printf("DI-%d is HIGH \n", (dwDIIdx + 1));
else
        printf("DI-%d is LOW \n", (dwDIIdx + 1));
}
```

STEP 6: Parse DO related bits

```
for (DWORD dwDOIdx = 0; dwDOIdx < ptDIDOInfo->dwDONum; dwDOIdx++)
{
//      Bit 0 for DO1, Bit 1 for DO2..., we use bit-operator to check the value.
if (1 == (wDOVal & (1 << dwDOIdx)))
        printf("DO-%d is HIGH\n", (dwDOIdx + 1));
else
        printf("DO-%d is LOW\n", (dwDOIdx + 1));
}
}
```

TIPS

The DI/DO value could be get from AVCallback with TMediaDataPacketInfo, too.

3.5. TalkWithMicPhone

DESCRIPTION

Talk to a camera with micphone. With the PacketMaker module, user don't have to handle the audio capturing related issues.

SAMPLE CODE

START TALK

STEP 1: Create connection and do connect as usual.

```
scRet = DataBroker_Initial(&hDataBroker, nMaxChannel, DataBrokerStatusCallBack,
DataBrokerAVCallBack, mctALLCODEC, 0, DATABROKER_VERSION);
scRet = DataBroker_CreateConnection(hDataBroker, &hConn);
TDataBrokerConnectionOptions tOpt;
memset(&tOpt, 0, sizeof(tOpt));
CreateConnectionOption(&tOpt);
scRet = DataBroker_Connect(hConn);
```

STEP 2: Initial PacketMaker.

```
TPMACAPCHANNELOPTION tChOpt;
memset(&tChOpt, 0, sizeof(tChOpt));
//  Set other options
ptOpt->pfPacketCB = StaticAudioCapturePacketCB;
scRet = PacketMaker_Initial(&hPacketMaker, 0, PACKET_MAKER_VERSION);
```

STEP 3: Prepare a callback function for audio capturing callback

```
SCODE __stdcall StaticAudioCapturePacketCB(DWORD dwContent, TMediaDataPacketInfo
```

```
*ptMediaPacket, DWORD dwDataTimePeriod)
{
    HANDLE*       phConn = (HANDLE*) dwContent;
    if (*phConn != NULL)
    {
            // Input audio frame to connection for talk
            DataBroker_InputTxPacket(*phConn, ptMediaPacket, dwDataTimePeriod);
    }
    return S_OK;
}
```

STEP 4: Set audio capture option and create audio capture channel.

```
TPMACAPCHANNELOPTION tChOpt;
memset(&tChOpt, 0, sizeof(tChOpt));
CreateAudioCaptureOption(&tChOpt, tOpt.dwAudioEnc, &hConn);
scRet = PacketMaker_CreateAudioCaptureChannel(hPacketMaker, &hAudioCapCh, &tChOpt);
```

STEP 5: Start the audio capturing

```
// the audio frame should be get in StaticAudioCapturePacketCB.
scRet = PacketMaker_StartAudioCapture(hAudioCapCh);
```

STEP 6: Start talk to device

```
// the media be transmitted in DataBrokerTxCallBack.
DataBroker_StartTxConnection(hConn);
```

STOP TALK

STEP 1: Stop audio capture

```
PacketMaker_StopAudioCapture(hAudioCapCh);
```

STEP 2: Stop talk to device

```
DataBroker_StopTxConnection(hConn);
```

TIPS

3.6. TalkWithWAVFile

DESCRIPTION

Besides talk to camera with Micphone, user may want to play a WAV file to the device. Here we will introduce you how to play a WAV file and transmit the sound to device.

SAMPLE CODE

START TALK

STEP 1: Create connection and do connect as usual, but add the dwTxContext

```
scRet = DataBroker_Initial(&hDataBroker, nMaxChannel, DataBrokerStatusCallBack,
DataBrokerAVCallBack, mctALLCODEC, 0, DATABROKER_VERSION);
scRet = DataBroker_CreateConnection(hDataBroker, &hConn);
TDataBrokerConnectionOptions tOpt;
memset(&tOpt, 0, sizeof(tOpt));
CreateConnectionOption(&tOpt);
tOpt.dwTxContext = (DWORD) &tMakePacketInfo;
scRet = DataBroker_Connect(hConn);
```

STEP 2: Initial PacketMaker.

```
scRet = PacketMaker_Initial(&hPacketMaker, 0, PACKET_MAKER_VERSION);
```

STEP 4: Set audio capture option and create audio capture channel.

```
TPMACAPCHANNELOPTION tChOpt;
memset(&tChOpt, 0, sizeof(tChOpt));
//  Set other options
tChOpt.dwAudioFramesPerPacket = 1;
tChOpt.dwAudioSizePerFrame = 640;
tOpt.dwAudioCodec = mctG711;
tOpt.dwAudioSampleFreq = 8000;        //Must match with the WAV file
tOpt.dwAudioChannelNum = 1;           //Must match with the WAV file
CreateAudioCaptureOption(&tChOpt, tOpt.dwAudioEnc, &hConn);
PacketMaker_CreateChannel(hPacketMaker, &tMakePacketInfo.hAudioCh, &tChOpt);
```

STEP 5: Read the WAV file into memory

```
DWORD dwWAVHeaderSize = 44;
fseek(pfInput, 0, SEEK_END);
dwSize = ftell(pfInput) - dwWAVHeaderSize;
fseek(pfInput, dwWAVHeaderSize, SEEK_SET); //Ignore the WAV header
fread(pbyBuf, dwSize, 1, pfInput);
tMakePacketInfo.pbyCurrPos = pbyBuf;
tMakePacketInfo.pbyEndPos = pbyInBuffer + dwInputSize;
```

STEP 5: Prepare a DataBrokerTxCallback function.

```
SCODE __stdcall DataBrokerTxCallBack(DWORD dwContext, BYTE **ppbyData, DWORD
*pdwLen, DWORD *pdwDataTimePeriod)
{
    TMakePacketInfo* ptInfo = (TMakePacketInfo *) dwContext;
    DWORD dwAudioSampleFreq = 8000;        //Must match with the WAV file
    DWORD dwBytePerSample = 2;             //Must match with the WAV file
    if (ptInfo->pbyCurrPos <= ptInfo->pbyEndPos)
    {
            TMediaDataPacketInfo    tMediaPacket;
            DWORD                   dwUsedSize = 0;
            memset(&tMediaPacket, 0, sizeof(tMediaPacket));
            tMediaPacket.pbyBuff = ptInfo->pbyOutputBitstream;
            PacketMaker_MakeAudioPacket(ptInfo->hAudioCh,
                ptInfo->pbyCurrPos, ptInfo->pbyEndPos - ptInfo->pbyCurrPos,
                ptInfo->dwOutputBitstreamSize, &dwUsedSize, &tMediaPacket);
            ptInfo->pbyCurrPos += dwUsedSize;
            *pdwLen = tMediaPacket.dwBitstreamSize + tMediaPacket.dwOffset;
            *ppbyData = tMediaPacket.pbyBuff;
            *pdwDataTimePeriod = ((dwUsedSize * 1000 )/
                                  (dwBytePerSample * dwAudioSampleFreq));
            return S_OK;
    }
    return DATABROKER_S_NOMORE_CALLBACK;
}
```

STEP 6: Call DataBroker_StartTxConnection to start

DataBroker_StartTxConnection(hConn);

After calling the DataBroker_StartTxConnection, DataBroker will start to callback the DataBrokerTxCallBack() to get audio bitstream

TIPS

3.7. SaveToFile

DESCRIPTION

This sample we will introduce you how to store the TMediaDataPacketInfoV3 into file. You may playback this file in AVSynchronizer sample code “PlayRawFile”.

SAMPLE CODE

STEP 1: Create connection and do connect as usual.

```
scRet = DataBroker_Initial(&hDataBroker, nMaxChannel, DataBrokerStatusCallBack,
DataBrokerAVCallBack, mctALLCODEC, 0, DATABROKER_VERSION);
scRet = DataBroker_CreateConnection(hDataBroker, &hConn);
TDataBrokerConnectionOptions tOpt;
memset(&tOpt, 0, sizeof(tOpt));
CreateConnectionOption(&tOpt);
tOpt.dwAVContext = (DWORD) fp;        //pass the fp by context
scRet = DataBroker_Connect(hConn);
```

STEP 2: Save the TMediaDataPacketInfoV3 from AVCallack

```
SCODE __stdcall DataBrokerAVCallBack(DWORD dwContext,TMediaDataPacketInfo
*ptMediaDataPacket)
{
FILE* fp = (FILE *) dwContext;
TMediaDataPacketInfoV3* ptMediaDataPacketV3 = (TMediaDataPacketInfoV3*)
ptMediaDataPacket;
if (fp)
{
```

// STEP 3: write out pbyTLVExt

```
if (ptPacketV3->tIfEx.tRv1.tExt.pbyTLVExt)
{
   fwrite(ptPacketV3->tIfEx.tRv1.tExt.pbyTLVExt, 1, ptPacketV3->tIfEx.tRv1.tExt.
```

```
dwTLVExtLen, fp);
}
else
{
    DWORD dwDummyData = 0;
    fwrite(&dwDummyData, 1, sizeof(dwDummyData), fp);
}
```

// STEP 4: write out pbyBuff

```
if (ptPacketV3->tIfEx.tInfo.pbyBuff)
{
fwrite(ptPacketV3->tIfEx.tInfo.pbyBuff, 1,
        ptPacketV3->tIfEx.tInfo.dwOffset + ptPacketV3->tIfEx.tInfo.dwBitstreamSize, fp);
}
else
{
    DWORD dwDummyData = 0;
    fwrite(&dwDummyData, 1, sizeof(dwDummyData), fp);
}
```

// STEP 5: write out pbyVExtBuf

```
if (ptPacketV3->pbyVExtBuf)
{
    fwrite(&ptPacketV3->dwVExtLen, 1, sizeof(DWORD), fp);
    fwrite(ptPacketV3->pbyVExtBuf, 1, ptPacketV3->dwVExtLen, fp);
}
else
{
    DWORD dwDummyData = 0;
    fwrite(&dwDummyData, 1, sizeof(dwDummyData), fp);
}
```

TIPS

3.8. ConnectMoreThan32Channels

DESCRIPTION

The best number of connections is 32 for A DataBroker, here we will use 2 DataBrokers to connect more than 32 channels.

SAMPLE CODE

Please refer to the SDK package.

TIPS

A DataBroker can handle multiple connections simultaneously, so the DataBroker_ Initial should be called only once for initialization purpose and than use this handle to create the connection handles.

3.9. Reconnect

DESCRIPTION

The sample will auto re-connect the devices. Here we use a monitor thread to monitor the status of connections.

SAMPLE CODE

STEP 1: Create connection and do connect as usual

```
scRet = DataBroker_Initial(&hDataBroker, nMaxChannel, DataBrokerStatusCallBack,
DataBrokerAVCallBack, mctALLCODEC, 0, DATABROKER_VERSION);
scRet = DataBroker_CreateConnection(hDataBroker, &hConn);
TDataBrokerConnectionOptions tOpt;
memset(&tOpt, 0, sizeof(tOpt));
CreateConnectionOption(&tOpt);
```

STEP 2: Before calling DataBroker_Connect(), set the connection status first.

```
tDBInfo.tConInfo[i].eStatus = eConnecting;
scRet = DataBroker_Connect(hConn);
```

STEP 3: Prepare the StatusCallback function, and it should handle the connection status too

```
SCODE __stdcall DataBrokerStatusCallBack(DWORD dwContext, TDataBrokerStatusType
tStatusType, PVOID pvParam1, PVOID pvParam2)
{
    TConnectionInfo* ptConInfo = (TConnectionInfo *) dwContext;
    switch (tStatusType)
    {
            case eOnConnectionInfo:
            printf("Get eOnConnectionInfo, start to receive packet, %x\n", ptConInfo);
            ptConInfo->eStatus = eConnected;
            break;

            case eOnStopped:
            printf("Get eOnStopped, stop to receive packet, %x\n", ptConInfo);
            ptConInfo->eStatus = eDisconnectd;
            break;
```

```
            default:
            break;
    }
  return S_OK;
}
```

STEP 4: Prepare the monitor function to monitor the connection status

```
for (;;)
{
DWORD dwObj = WaitForSingleObject(ptDBInfo->hExit, 500);
if (dwObj == WAIT_OBJECT_0)  //exit the monitor
{
    break;
}
for (int i = 0; i < CONNECTION_SIZE; i++)
{
    if (ptDBInfo->tConInfo[i].eStatus == eDisconnectd)
    {
            DataBroker_DeleteConnection(ptDBInfo->hDataBroker,
                                        &ptDBInfo->tConInfo[i].hConn);
            DataBroker_CreateConnection(ptDBInfo->hDataBroker,
                                       &ptDBInfo->tConInfo[i].hConn);
            DataBroker_SetConnectionOptions(ptDBInfo->tConInfo[i].hConn,
            &ptDBInfo->tConInfo[i].tOpt);
            ptDBInfo->tConInfo[i].eStatus = eConnecting;
             DataBroker_Connect(ptDBInfo->tConInfo[i].hConn);
    }
}
```

TIPS

DON'T call the DataBroker API in the StatusCallback or AVCallback , such as DataBroker_ Disconnect(), DataBroker_DeleteConnection(), DataBroker_Release() and so on. It will cause the module error even crash.

3.10. GetV3Information

DESCRIPTION

The DataBroker always callback TMediaDataPacketInfoV3 now, you may cast the ptMediaDataPacket to TMediaDataPacketInfoV3 from AVCallback.

In this sample, we will introduce you the capture and cropping information. You could refer to the DataPacketParser document for more information.

SAMPLE CODE

STEP 1: Create connection and do connect as usual.

```
scRet = DataBroker_Initial(&hDataBroker, nMaxChannel, DataBrokerStatusCallBack,
DataBrokerAVCallBack, mctALLCODEC, 0, DATABROKER_VERSION);
scRet = DataBroker_CreateConnection(hDataBroker, &hConn);
TDataBrokerConnectionOptions tOpt;
memset(&tOpt, 0, sizeof(tOpt));
CreateConnectionOption(&tOpt);
scRet = DataBroker_Connect(hConn);
```

STEP 2: Process the V3 information from AVCallack

```
SCODE __stdcall DataBrokerAVCallBack(DWORD dwContext,TMediaDataPacketInfo
*ptMediaDataPacket)
{
  // Show capture window size in TMediaDataPacketInfoV3
  TMediaDataPacketInfoV3* ptMediaDataPacketV3 = (TMediaDataPacketInfoV3*)
```

```
ptMediaDataPacket;
  // Check if I frame and then show capture window information
  if (ptMediaDataPacketV3->tVUExt.tCapWinInfo.bWithInfo &&
      ptMediaDataPacketV3->tIfEx.tInfo.tFrameType == MEDIADB_FRAME_INTRA)
  {
      printf("x-axis = %u, y-axis = %u, capture height = %u, capture width = %u,
              cropping height = %u, cropping width = %u\n",
               ptMediaDataPacketV3->tVUExt.tCapWinInfo.wOffX,
               ptMediaDataPacketV3->tVUExt.tCapWinInfo.wOffY,
               ptMediaDataPacketV3->tVUExt.tCapWinInfo.wCapH,
               ptMediaDataPacketV3->tVUExt.tCapWinInfo.wCapW,
               ptMediaDataPacketV3->tVUExt.tCapWinInfo.wCropH,
               ptMediaDataPacketV3->tVUExt.tCapWinInfo.wCropW);
  }
return S_OK;
}
```

TIPS

3.11. SpecifyMediaType

DESCRIPTION

Connect to server and receive the specified media type only(Video only, Audio only or both).

SAMPLE CODE

STEP 1. Initial DataBroker

```
scRet = DataBroker_Initial(&hDataBroker, nMaxChannel, DataBrokerStatusCallBack,
DataBrokerAVCallBack, mctALLCODEC, 0, DATABROKER_VERSION);
```

STEP 2. Create Connection

```
scRet = DataBroker_CreateConnection(hDataBroker, &hConn);
```

STEP 3. Set Options for Connection

```
TDataBrokerConnectionOptions tOpt;
memset(&tOpt, 0, sizeof(tOpt));
//Call the user-defined auxiliary function CreateConnectionOption() to construct the tOpt
CreateConnectionOption(&tOpt);
{
ptOpt->pzIPAddr = “192.168.1.100”;
ptOpt->wHttpPort = 80;
ptOpt->pzUID = “root”;
...
```

```
printf("Please input connection media type (video = 1 , audio = 2 , AV = 3): ");
_tscanf("%u", &dwMediaType);
switch (dwMediaType)
{
   case 1:
    ptOpt->dwMediaType =  emtVideo;
    break;

    case 2:
    ptOpt->dwMediaType =  emtAudio;
    break;

    case 3:
    ptOpt->dwMediaType =  (emtVideo | emtAudio);
    break;

    default:
    printf("Input connection media type error %d \n", dwMediaType);
    ptOpt->dwMediaType =  (emtVideo | emtAudio);
    break;
}
}
```

STEP 4. Connect to Device

```
scRet = DataBroker_Connect(hConn);
```

STEP 5. Wait until you want to stop

```
while( !_kbhit() )
{
    Sleep(16);
}
```

TIPS

3.12. ForceIFrame

DESCRIPTION

We will force the camera to send I-Frame as soon as possible. If the resource of camera is shortage, it may ignore the force I-Frame request.

SAMPLE CODE

STEP 1. Initial DataBroker

scRet = DataBroker_Initial(&hDataBroker, nMaxChannel, DataBrokerStatusCallBack, DataBrokerAVCallBack, mctALLCODEC, 0, DATABROKER_VERSION);

STEP 2. Create Connection

scRet = DataBroker_CreateConnection(hDataBroker, &hConn);

STEP 3. Set Options for Connection

```
TDataBrokerConnectionOptions tOpt;
memset(&tOpt, 0, sizeof(tOpt));
//Call the user-defined auxiliary function CreateConnectionOption() to construct the tOpt
CreateConnectionOption(&tOpt);
{
ptOpt->pzIPAddr = “192.168.1.100”;
ptOpt->wHttpPort = 80;
ptOpt->pzUID = “root”;
...
}
```

STEP 5. Connect to Device

scRet = DataBroker_Connect(hConn);

STEP 6. Call DataBroker_ForceIFrame() function

```
while (1)
{
    gets(szbuffer);
    if (szbuffer[0] == 'i')     //force I Frame
    {
            scRet = DataBroker_ForceIFrame(hConn);
            if (scRet != S_OK)
            {
                    printf("DataBroker_ForceIFrame failed with error %X \n", scRet);
            }
    }
    if (szbuffer[0] == 's')    // stop
    {
            break;
    }
}
```

TIPS

3.13. SSLSupport

DESCRIPTION

Connect to a camera with SSL.

SAMPLE CODE

STEP 1. Initial DataBroker

scRet = DataBroker_Initial(&hDataBroker, nMaxChannel, DataBrokerStatusCallBack, DataBrokerAVCallBack, mctALLCODEC, 0, DATABROKER_VERSION);

STEP 2. Create Connection

scRet = DataBroker_CreateConnection(hDataBroker, &hConn);

STEP 3. Set Options for Connection

```
TDataBrokerConnectionOptions tOpt;
memset(&tOpt, 0, sizeof(tOpt));
//Call the user-defined auxiliary function CreateConnectionOption() to construct the tOpt
CreateConnectionOption(&tOpt);
{
ptOpt->pzIPAddr = “192.168.1.100”;
ptOpt->wHttpPort = 80;
ptOpt->pzUID = “root”;
...
}
```

STEP 4 Set the SSL option

scRet = DataBroker_SetConnectionExtraOption(hConn, eOptSSLEnable, TRUE, NULL);

STEP 5. Connect to Device

scRet = DataBroker_Connect(hConn);

STEP 6. Wait until you want to stop

```
while( !_kbhit() )
{
    Sleep(16);
}
```

TIPS

The option is work only for the device that support SSL.

4. API Reference

This chapter contains the API function calls for the Data Broker.

Note that in Windows CE all the interface functions use Unicode strings. And for Linux or Windows 32 platform, non-Unicode strings are used.

4.1. Enumeration

The enumeration used is depicted here.

EMediaCodecType

This enumeration indicates the media codec type. Please refer to Data Packet Parser document for detail definition.

TDataBrokerChannelType

This enumeration indicates the channel types.

```
typedef enum
{
eAudioChannel,
eVideoChannel
}
TDataBrokerChannelType;
```

Values

eAudioChannel
audio channel

eVideoChannel
video channel

Remarks

Requirements

DataBroker.h

TDataBrokerConnOptionFlag

This enumeration indicates the flags that have to be turned on when you want to specify values to some fields of the TDataBrokerConnectionOptions. When the flag is set the corresponding field is checked and taken in function call. Otherwise, the field is ignored.

```
typedef enum
{
eConOptCam                      = 0x0001,
eConOptVSize                    = 0x0002,
eConOptQuality                  = 0x0004,
eConOptHttpPort                 = 0x0008,
eConOptProtocolAndMediaType     = 0x0010,
eConOptVideoCodec               = 0x0020,
eConOptAudioCodec               = 0x0040,
eConOptStatusCallback           = 0x0080,
eConOptAVCallback               = 0x0100,
eConOptStatusContext            = 0x0200,
eConOptAVContext                = 0x0400,
eConOptTxCallback               = 0x0800,
eConOptTxContext                = 0x1000,
eConOptAudioEncCodec            = 0x2000,
eConOptVideoCodecPri            = 0x4000,
eConOptAudioCodecPri            = 0x8000,
eConOptAudioSample              = 0x10000,
eConOptAudioEncSamle            = 0x20000
} TDataBrokerConnOptionFlag;
```

Values

eConOptCam

the flag for wCam field of TdataBrokerConnectionOptions.

eConOptVSize

the flag for zVSize field of TDataBrokerConnectionOptions

eConOptQuality

the flag for dwQuality field of TDataBrokerConnectionOptions

eConOptHttpPort

the flag for wHttpPort field of TDataBrokerConnectionOptions

eConOptProtocolAndMediaType
the flag for dwProtocolType and dwMediaType fields of TDataBrokerConnectionOptions. You should specify these two values at the same time.

eConOptVideoCodec
the flag for dwVideoCodec field of TDataBrokerConnectionOptions

eConOptAudioCodec
the flag for dwAudioCodec field of TDataBrokerConnectionOptions

eConOptStatusCallback
the flag for pfStatus field of TDataBrokerConnectionOptions

eConOptAVCallback
the flag for pfAV field of TDataBrokerConnectionOptions

eConOptStatusContext
the flag for dwStatusContext field of TDataBrokerConnectionOptions

eConOptAVContext
the flag for dwAVContext field of TDataBrokerConnectionOptions

eConOptTxCallback
the flag for pfTx field of TDataBrokerConnectionOptions

eConOptTxContext
the flag for dwTxContext field of TDataBrokerConnectionOptions

eConOptAudioEncCodec
the flag for dwAudioEnc field of TDataBrokerConnectionOptions

eConOptVideoCodecPri
the flag for dwVCodecOrder field of TDataBrokerConnectionOptions

eConOptAudioCodecPri
the flag for dwACodecOrder field of TDataBrokerConnectionOptions

eConOptAudioSample
the flag for dwAudioSample field of TDataBrokerConnectionOptions

eConOptAudioEncSample
the flag for dwAudioEncSample field of TDataBrokerConnectionOptions

Remarks

Requirements

DataBroker.h

TDataBrokerOptionFlag

This enumeration indicates the flags that have to be turned on when modify DataBroker handle's option in TDataBrokerOptions. When the flag is set the corresponding field is checked and taken in function call. Otherwise, the field is ignored.

```
typedef enum
{
eOptEnableProxy          = 0x0001,
eOptProxyPort            = 0x0002,
eOptProxyName            = 0x0004,
eOptEnableIPRestrict     = 0x0008,
eOptIPRestrictNum        = 0x0010,
eOptIPRestrictList       = 0x0020,
eOptConnUseIE            = 0x0040,
eOptConnNotUseIE         = 0x0080,
eOptConnTimeout          = 0x0100,
eOptRWTimeout            = 0x0200,
} TDataBrokerOptionFlag;
```

Values

eOptEnableProxy

the flag for bEnableProxy field of TDataBrokerOptions.

eOptProxyPort

the flag for dwProxyPort field of TDataBrokerOptions.

eOptProxyName

the flag for szProxyName field of TDataBrokerOptions.

eOptEnableIPRestrict

the flag for bEnableIPRestrict field of TDataBrokerOptions.

eOptIPRestrictNum

the flag for dwIPRestrictNum field of TDataBrokerOptions.

eOptIPRestrictList

the flag for pszIPRestrictList field of TDataBrokerOptions.

eOptConnUseIE

the flag to indicate that the connection will use Wininet to connect to server. Only works when in Windows 32 or Windows CE platform.

eOptConnNotUseIE

the flag to indicate the connection will NOT use Wininet to connect to server. Only works when in Windows 32 or Windows CE platform. If both this and eOptConnUseIE are defined, the not use is taken.

eOptConnTimeout

the flag for dwConnTimeout field of TDataBrokerOptions.

eOptRWTimeout

the flag for dwRWTimeout field of TDataBrokerOptions.

Remarks

Requirements

DataBroker.h

TDataBrokerStatusType

This enumeration indicates the status of DataBroker.

```
typedef enum
{
eOnConnectionInfo,
eOnAuthFailed,
eOnStartMediaChannel,
eOnChannelClosed,
eOnTimeout,
eOnProtocolChanged,
eOnPacketLoss,
eOnDiDo,
eOnLocationChanged,
eOnInputInfo,
eOnOtherError,
eOnStopped,
eOnAudioMode,
eOnChangeMediaType,
eOnAudioDisabled,
eOnAudioUpstreamOccupied,
eOnGetPrivilege,
eOnTxChannelStart,
eOnTxChannelClosed,
eOnControlChannelClosed,
eOnVideoSignalChange,
eOnServiceUnavailable,
eOnAudioUpstreamDisabled,
eOnMediaRange,
eOnMP4Vconfig,
eOnMP4Aconfig,
eOnGAMRConfig,
eOnConnectionOptionError,
eOnProxyAuthFailed,
eOnConnectionType,
} TDataBrokerStatusType;
```


Values

eOnConnectionInfo

Indicate connection info when connecting to the server

eOnAuthFailed
Can not pass the authorization of server

eOnStartMediaChannel
Begin to receive media stream

eOnChannelClosed
Audio or Video channel closed

eOnTimeout
Audio or Video channel receives data timeout

eOnProtocolChanged
The protocol of receiving media changed

eOnPacketLoss
Packet loss

eOnDiDo
Receiving the digital input alert and digital output status.

eOnLocationChanged
Detecting the change of location

eOnInputInfo
Indicate the width and height of the image when using Input to unpacketize and parse the receiving packets.

eOnOtherError
Other error occurs.

eOnStopped
The Connection stopped.

eOnAudioMode
Notify the audio mode set on server. This notification is only available when connecting to 6000 series servers. The pvParam1 contains the integer value that indicates the audio mod. Please refer to TMediaAudioMode. This status is also notified when someone changes the server audio mode. So it is not subjected to called when connecting.

eOnChangeMediaType
Notify that due to server settings or users' permission, the media is changed. This notification is only available when connecting to 6000 series servers. The pvParam1 contains reason. Please refer to TMediaChangeReason.

eOnAudioDisabled
This status code is similar to eOnChangeMediaType. But when users get this notification, it means only control channel is established. In such case, no audio or video data would be available.

eOnAudioUpstreamOccupied
When users start talk and DataBroker finds that the talk-channel is already used by

other user, this status code will be sent to users by callback. This notification is only available when connecting to 6000/7000/8000 series servers

eOnGetPrivilege

Notify the privilege for current user. This notification is only available when connecting to 6000 series servers. The privilege type is a ORed double word of the privilege define in TUserPrivilege.

eOnTxChannelStart

Notify the Talk connection is established. This notification is only available when connecting to 6000/7000/8000 series servers.

eOnTxChannelClosed

Notify the Talk connection is closed. This notification is only available when connecting to 6000/7000/8000 series servers.

eOnControlChannelClosed

Notify the control channel has been closed.

eOnVideoSignalChange

Notify that there is no signal for video input of Video server model. The parameter would either be signal lost or signal restored.

eOnServiceUnavailable

Notify that the server is now serving 10 streaming clients and no more new client could request for streaming now unless one of the previous connection closed.

eOnAudioUpstreamDisabled

Notify that the upstream channel is turned off by server. The parameter is the new audio mode that server is set currently.

eOnMediaRange,

This is the notification for RTSP server's playing range for media. This is only made if the requested media is file based. For live streaming, there will not be such status notified.

eOnMP4Vconfig

This is the MP4 CI value for RTSP. pvParam1 is the CI starting address and pvParam2 is the length for CI.

eOnMP4Aconfig

This is the AAC info value for RTSP. pvParam1 is the DWORD value for the type value.

eOnGAMRConfig

This is the GAMR info value for RTSP. pvParam1 is the sample rate.

eOnConnectionOptionError

This indicates that connection options are different from that in server.

eOnProxyAuthFailed

Notify AP authentication fail.

eOnConnectionType

Notify the connection type. pvParam1's upper 16 bits indicates if this is a dual-stream model or not. If pvParam1's lower 16 bits is 1 means this connection is RTSP and 2 means server push. pvParam2 is the model of server.

Remarks

Requirements

DataBroker.h

TExtraOptions

This enumeration lists the options available when calling DataBroker_SetConnectionExtraOption.

```
typedef enum
{
eOptQueueSize                          = 1,
eOptAutoForceI                         = 2,
eOptDIDOBySet                          = 3,
eOptSetDIDOValue                       = 4,
eOpt3KAudioDIDO                        = 5
eOptRtspCtrlPort                       = 6
eOptServerInfo                         = 7
eOptAACInfo                            = 8
eOptProxyAuthen                        = 9
eOptStreamIndex                        = 10
eOptRtspProxyEnable                    = 11
eOptRtspProxyInfo                      = 12
eOptProtocolMedia                      = 13
eOptProtocolRollNext                   = 14
eOptUpdUserNamePwd                     = 15
eOptRtspMcastProtocolRollingTimeout    = 16
eOptRtspUDPProtocolRollingTimeout      = 17
} TExtraOptions;
```

Values

eOptQueueSize

To set queue size for connection or input. It is effecttive before the queue is created. For input, please set this before DataBroker_SetInputOptions is called. For connection, set it before DataBroker_SetConnectionOptions is called. The dwParam1 is for Video Q size, and dwParam2 is for Audio Q Size. The maximum value for video is 60, and the maximum for Audio is 20.

eOptAutoForceI

To enable or disable the auto-force I function. It's only workable for 3000 server and the protocol must not be HTTP. When the dwParam1 is true means to enable and in such case, the dwParam2 is the period in milliseconds. The minimum value could be set is 100 ms. When disable period, param2 is not used. This option is only applicable for Connection, not input.

eOptDIDOBySet

For the old model of server, the video bit stream does not carry DI/DO information.

If application could get DI/DO from other channel (such as by http command), the application could set the value manually. This option enables the module to embed the set value into video packet. The dwParam1 contain True or False to enable or disable the function. It only works for 2000 and 3000 server. The caller must take care for if the firmware already sends DI/DO. If so, it's not necessary to use this path to set DI/DO value.

eOptSetDIDOValue

Set DI/DO value for a connection or input. When set, dwParam1 contains the DI value, and dwParam2 contains DO value. Each bit represents a single DI or DO. The LSB means DI 0 or DO 0, bit 1 means DI 1 or DO 1, and so on.

eOpt3KAudioDIDO

Should the module synchronize the DI/DO value in audio with that in video? For 3000 server, even if the sever sends DI/DO. The values are contains only in the video packets. With this option, the module will retrieve the DI/DO values from video and set them in audio packet. The dwParam1 contains True or False to enable or disable this function.

eOptRtspCtrlPort

Set the control port for the RTSP server and dwParam1 is the control port.

eOptServerInfo

Set server Information. dwParam1 points to a TSrvDepResource_SysInfo structure. dwParam2 is the options to set. See EoptSysInfo in SrvDepResource document.

eOptAACInfo

Set AAC information for Input-channel. dwParam1 points to TAACExtInfo.

eOptProxyAuthen

Settings for proxy authentication, dwParam1 is the user name, dwParam2 is password

eOptStreamIndex

For dual-stream models, indicates which connection to connect. dwParam1 is the index of stream. The index starts from zero.

eOptRtspProxyEnable

dwParam1 is TRUE for enable and FALSE for disable RTSP proxy.

eOptRtspProxyInfo

dwParam1 is proxy address, and dwParam2 is the proxy port for RTSP.

eOptProtocolMedia

Specifying the protocol and media type. dwParam1 is protocol, and dwParam2 is media type. Value of 0xFFFFFFFF means do not set it

eOptProtocolRollNext

Specifies the next protocol to be tried when protocol rolling. dwParam1 the protocol type.

eOptUpdUserNamePwd

dwParam1 is user name, and dwParam2 is password

eOptRtspMcastProtocolRollingTimeout

Assign the timeout value of multicast protocol rolling. dwParam1 is timeout value.

eOptRtspUDPProtocolRollingTimeout

Assign the timeout value of UDP protocol rolling. dwParam1 is timeout value.

Remarks

Requirements

DataBroker.h

TMediaAudioMode

This enumeration indicates the audio mode currently set on server. The notification of this mode is only available when connecting to 6000 series servers

```
typedef enum
{
eFullDuplex                = 0x0001,
eHalfDuplex                = 0x0002,
eTalkOnly                  = 0x0003,
eListenOnly                = 0x0004,
} TMediaAudioMode;
```

Values

eFullDuplex

Server is in full-duplex mode, which enables users to listen and talk simultaneously.

eHalfDuplex

Server is in half-duplex mode, which allows users to be either talk or listen. When talking, the downstream will be disabled.

eTalkOnly

Server is in talk-only mode, which allows users to talk but not listen. Users might get eOnChangeMediaType/ eOnAudioDisabled status callback if selects video/audio or audio only mode when connecting.

eListenOnly

Server is in listen-only mode, which allows users to listen but not talk.

Remarks

For 7000/8000 series servers, the audo mode is always in Full Duplex.

Requirements

DataBroker.h

TMediaChangeReason

This enumeration indicates the reason why media type is changed. This reason code is only available when connecting to 6000 series servers

```
typedef enum
{
eNoPermission
eModeNotSupport
} TMediaChangeReason;
```

Values

eNoPermission

The permission of current user is not allowed to open downstream or upstream audio connection.

eModeNotSupport

The server's audio mode is set to talk-only or audio disabled mode so users are not allowed to establish downstream audio connection. Or the server is set to listen-only or disabled mode so users are not allowed to establish upstream audio connection.

Remarks

Requirements

DataBroker.h

TsdrAudioCodec

This enumeration indicates the audio codec types.

```
typedef enum
{
eACodecNone             = 0x0000,
eACodecG7221            = 0x0100,
eACodecG729A            = 0x0200
eACodecAAC              = 0x0400
eACodecGAMR             = 0x0800
} TsdrAudioCodec;
```

Values

eACodecNone
no audio codec

eACodecG7221
G.722.1

eACodecG729A
G.729A

eACodecAAC
AAC (stereo)

eACodecGAMR
GAMR, used in RTSP IP camera and server

Remarks

Requirements

SrvTypeDef.h

TsdrMediaType

This enumeration indicates the request media types.

```
typedef enum
{
emtAudio                = 1,
emtVideo                = 2,
emtTransmitAudio        = 4
} TsdrMediaType;
```

Values

emtAudio

the media is received audio

emtVideo

the media is received video

emtTransmitAudio

the media is transmitted audio

Remarks

Requirements

SrvTypeDef.h

TsdrProtocolType

This enumeration indicates the protocol types.

```
typedef enum
{
eptHTTP,
eptTCP,
eptUDP
eptScalableMULTICAST =
eptMULTICAST,
eptBackchannelMULTICAST
} TsdrProtocolType;
```

Values

eptHTTP
HTTP

eptTCP
TCP

eptUDP
UDP

eptMULTICAST
MULTICAST

eptScalableMULTICAST
MULTICAST

eptBackchannelMULTICAST
MULTICAST

Remarks

Requirements

SrvTypeDef.h

TUserPrivilege

This enumeration indicates the privilege for a user. This privilege value is only available when connecting to 6000 series servers

```
typedef enum
{
ePrivelegeDIDO          = 0x00000001
eModeNotSupport         = 0x00000002
ePrivelegeTALK          = 0x00000004
ePrivelegeCAMCTRL       = 0x00000008
ePrivelegeCONF          = 0x00000080
ePrivelegeAll           =0xFFFFFFFF
} TUserPrivilege;
```

Values

ePrivilegeDIDO

Users could set DO and retrieve DI value from server. Note: this flag might be renamed to DO only because the DI is carried in video stream. So there is no reason to limit users to get DI value.

ePrivilegeLISTEN

Users could open the downstream audio connection to server. Hence they could listen to the live audio sent via server.

ePrivilegeTALK

Users could open the upstream audio connection to server. Hence they could send server audio data. Note: Talk capability is determined by this privilege and the server's audio mode. Both must be turn on to enable talking.

ePrivelegeCAMCTRL

Users could control the camera control. This includes Pan/Tilt/Zoom/Focus. It depends on the server model.

ePrivelegeCONF

Users could set the configuration of the servers.

ePrivelegeAll

Users have the full access to the camera.

Remarks

Requirements

DataBroker.h

TVideoSignalState

This enumeration indicates the signal state when callback notify that the video signal changes. The callback would be called whenever the signal state changes, so if the signal lost at certain time, the signal will be called once for signal lost, and would not be called until signal restored.

```
typedef enum
{
eSignalRestored          = 0
eSignalLost              = 1
} TVideoSignalState;
```

Values

eSignalRestored

The signal restored.

eSignalLost

The signal is lost.

Remarks

Requirements

DataBroker.h

4.2. Callback Function

The Callback function is depicted here.

TDataBrokerAVCallback

This callback function is used to receive audio and video frames.

Syntax

```
typedef SCODE
(*TDataBrokerAVCallback) (  DWORD dwContext,
                            TMediaDataPacketInfo *pMediaDataPacket  );
```

Parameters

dwContext

[in] the context value of the AV callback function

pMediaDataPacket

[in] received media data frame

Return Values

S_OK

Receive the audio or video frame successfully.

DATABROKER_S_FRAME_NOT_HANDLED

Could not handle this frame at this time.

Remarks

Requirements

DataBroker.h

TDataBrokerNetPacketCallback

This callback function is used to notify application about the arrival of network packet, and let users could direct access the packet data.

Syntax

```
typedef SCODE
(*TDataBrokerNetPacketCallback) (   DWORD dwContext,
                                    DWORD dwMediaType,
                                    DWORD dwLen,
                                    BYTE * pbyPacket        );
```

Parameters

dwContext

[in] the context value of the callback function

dwMediaType

[in] the media type for this packet, the possible types are defined in TsdrMediaType.

dwLen

[in] the data length of the packet

pbyPacket

[in] the packet data. Note the pointer is no longer valid after return, application needs to copy the content to his own buffer.

Return Values

Return S_OK if successful or an error value otherwise.

Remarks

TDataBrokerStatusCallback

This callback function is used to report the status of DataBroker object.

Syntax

```
typedef SCODE
(*TDataBrokerStatusCallback) (
                                DWORD dwContext,
                                TDataBrokerStatusType tStatusType,
                                PVOID pvParam1,
                                PVOID pvParam2                    );
```

Parameters

dwContext

[in] the context value of the status callback function

tStatusType

[in] the status type

pvParam1

[in] the first parameter getting from DataBroker. For the detail of this parameter, please refer to the remarks section.

pvParam2

[in] the second parameter getting from DataBroker. For the detail of this parameter, please refer to the remarks section.

Return Values

Return S_OK for most cases. When the status code is eOnProtocolChanged, return DATABROKER_S_STOPCONNECTION to inform DataBroker to stop the connection, else to allow server to change to HTTP.

Remarks

Status code	pvParam1	pvParam2
eOnConnectionInfo	Connection information (*TDataBrokerConnInfo)	(None)
eOnAuthFailed	(None)	(None)
eOnStartMediaChannel	(None)	(None)
eOnChannelClosed	Channel type (TDataBrokerChannelType)	(None)
eOnTimeout	Channel type (TDataBrokerChannelType)	(None)
eOnProtocolChanged	Original protocol (TsdrProtocolType)	New protocol (TsdrProtocolType)
eOnPacketLoss	Numbers lost (DWORD)	Media type (TsdrMediaType)
eOnDiDo	DI/DO changes (DWORD)	DI/DO values (DWORD)

eOnLocationChanged	New location (char*)	(None)
eOnInputInfo	Image width (DWORD)	Image height (DWORD)
eOnOtherError	Error code	(None)
eOnStopped	(None)	(None)
eOnAudioMode	Audio mode (TMediaAudioMode)	(None)
eOnChangeMediaType	Change reason (TMediaChangeReason)	(None)
eOnAudioDisabled	Disabled reason (TMediaChangeReason)	(None)
eOnAudioUpstreamOccupied	(None)	(None)
eOnGetPrivilege	Privilege (or-ed valued of TUserPrivilege)	(None)
eOnTxChannelStart	(None)	(None)
eOnTxChannelClosed	(None)	(None)
eOnControlChannelClosed	(None)	(None)
eOnVideoSignalChange	(None)	(None)
eOnServiceUnavailable	(None)	(None)
eOnAudioUpstreamDisabled	Audio mode (TMediaAudioMode)	(None)
eOnMediaRange	The start time for the period in second (relative value)	The end time for the period in second (relative value)
eOnMP4Vconfig	The stating address of CI.	The length for CI.
eOnMP4Aconfig	This is the AAC info value for RTSP	(None)
eOnGAMRConfig	This is the sample rate for GARM	(None)
eOnConnectionOptionError	(None)	(None)
eOnProxyAuthFailed	(None)	(None)
eOnConnectionType	**ConnectionType**(DWORD)	The model name for server.

DI/DO changes

Each bit is used to indicate the change of DI alert and DO, DI alert is the lower 16 bits, DO is the higher 16 bits. The LSB indicates the first one. It supports four digital input sources and two digital outputs at most in the present.

DI/DO values

Each bit is used to indicate the value (H/L) of DI alert and DO, DI alert is the lower 16 bits, DO is the higher 16 bits. The LSB indicates the first one. It supports four digital input sources and two digital outputs at most in the present.

Error Code

DATABROKER_E_HTTP_READ_ERROR

Read web page from server error.

DATABROKER_E_HTTP_CONNECT_FAILED

Could not connect to server by http protocol.

DATABROKER_E_CONTROL_CHANNEL_CONNECT_FAILED

Control channel could not be established

DATABROKER_E_INVALID_ID

the remote ID passed in is not a correct ID assigned by 3000 servers. (Only applied to 3000 servers)

DATABROKER_E_OUT_OF_MEMORY

Could not allocated memory for internal usage.

DATABROKER_E_CONTROL_CHAN_VER

The server's control channel supports older message version from the version this module is used.

Requirements

DataBroker.h

TDataBrokerTxCallback

This callback function is used to transmit media stream. It's only used for 6000/7000/8000 servers

Syntax

```
typedef SCODE
(*TDataBrokerTxCallback) (
                                    DWORD dwContext,
                                    BYTE **ppbyDataBuffer,
                                    DWORD *pdwLen,
                                    DWORD *pdwDataTimePeriod    );
```

Parameters

dwContext

[in] the context value of the Tx callback function

ppbyDataBuffer

[out] the data to be sent

pdwLen

[out] the length of the valid data held in ppbyDataBuffer

pdwDataTimePeriod

[out] The total time length for the data held in ppbyDataBuffer. This value is used to control the data rate sent to server

Return Values

Return S_OK for normal case. If the return value is DATABROKER_S_NOMORE_CALLBACK, this module won't make callback any more. And the ppbyDataBuffer, pdwLen, and pdwDataTimePeriod are all ignored. Users must push the to be transmitted data by calling DataBroker_InputTxPacket.

Remarks

Users must give correct pdwDataTimePeriod to avoid server from being flooded by audio data packet. The value is counted like this: the buffer contains 5 frames and each frame contains 10 milliseconds encoded audio data. Then the pdwDataTimePeriod should be 5 * 10 = 50 milliseconds.

4.3. Data Structure

The data structure is depicted here.

TDataBrokerConnectionOptions

This structure indicates the connection options.

```
typedef struct
{
WORD                        wCam;
TCHAR                       zVSize[10];
DWORD                       dwQuality;
TDataBrokerStatusCallback   pfStatus;
TDataBrokerAVCallback       pfAV;
TDataBrokerTxCallback       pfTx;
DWORD                       dwStatusContext;
DWORD                       dwAVContext;
DWORD                       dwTxContext,
WORD                        wHttpPort;
DWORD                       dwProtocolType;
DWORD                       dwMediaType;
DWORD                       dwVideoCodec;
DWORD                       dwAudioCodec;
DWORD                       dwAudioSample;
DWORD                       dwAudioEnc;
DWORD                       dwAudioEncSample;
TCHAR*                      pzServerType;
TCHAR*                      pzIPAddr;
TCHAR*                      pzUID;
TCHAR*                      pzPWD;
DWORD                       adwVCodecOrder[MAX_VIDEO_CODEC];
DWORD                       adwACodecOrder[MAX_AUDIO_CODEC];
DWORD                       dwFlags;
} TDataBrokerConnectionOptions, *PTDataBrokerConnectionOptions;
```

Members

wCam

the camera index

zVSize

the Vsize (For 2K series only)

dwQuality
the quality value (For 2K series only)

pfStatus
the pointer to the status callback function.

pfAV
the pointer to the callback function of receiving AV frames.

pfTx
the pointer to the callback function of transmitting media.

dwStatusContext
the context value for status callback function.

dwAVContext
the context value for AV callback function.

dwTxContext
the context value for Tx callback function.

wHttpPort
the HTTP port number

dwProtocolType
the protocol type

dwMediaType
the requested media type

dwVideoCodec
the video codec type

dwAudioCodec
the audio codec type

dwAudioSample
the audio sample rate

dwAudioEnc
the audio encoding codec type

dwAudioEncSample
the audio encoding sample rate

pzServerType
Server friendly name. This option must be specified.

pzIPAddr
Remote IP address you want to connect to, the maximum length is 128 bytes. This option must be specified.

pzUID
User login ID, the maximum length is 40 bytes. This option must be specified.

pzPWD

User login password, the maximum length is 40 bytes. This option must be specified.

adwVCodecOrder

Server supports video codecs following this order. It's only valid for 6K server.

adwACodecOrder

Server supports audio codecs following this order. It's only valid for 6K server.

dwFlags

Flags for the optional fields.

Remarks

pzServerType, pzIPAddr, pzUID, and pzPWD fields must be specified values. Others are optional. If you want to specify values to these optional fields, turn on the corresponding flags on dwFlags. Otherwise, they would use default values.

You can specify status and AV callback function when calling DataBroker_Initial to give the same callback functions among all connections. If you want to give different callback functions for each connection, specify the pfStatus and pfAV fields in this structure when calling DataBroker_SetConnectionOptions. But anyway, these two callbacks are must.

For 3000 servers, the audio codec type could be only two modes: one is eACodecG7221, the other is eACodecG7221|eACodecG729A. Because client must always support this codec, eACodecG7221.

If user wants DataBroker to detect the server information, please set pzServerType to "Auto" and input pzIPAddr, pzUID, pzPWD, and wHttpPort.

Requirements

DataBroker.h

TDataBrokerConnInfo

This structure indicates the information of a connection.

```
typedef struct
{
DWORD                   dwWidth;
DWORD                   dwHeight;
TCHAR                   zLanguage[8];
DWORD                   dwAudioCodec;
DWORD                   dwVideoCodec;
DWORD                   dwMediaType;
DWORD                   dwProtocol;
WORD                    wVideoPort;
WORD                    wAudioPort;
TCHAR                   szServerType[20];
} TDataBrokerConnInfo, *PTDataBrokerConnInfo;
```

Members

dwWidth

the width of the image. Note that this value is only for reference because for HTTP mode of 3000 server, the value is not retrieved actually, so some reference value is returned. To get the exact value, please use the decoder callback from AVSynchronizer.

dwHeight

the height of the image. Note that this value is only for reference because for HTTP mode of 3000 server, the value is not retrieved actually, so some reference value is returned. To get the exact value, please use the decoder callback from AVSynchronizer.

zLanguage

the language type of the server

dwAudioCodec

the audio codec type of the server

dwVideoCodec

the video codec type of the server

dwMediaType

the media type of the server

dwProtocol

the actual protocol used by the connection.

wVideoPort

the server side video port used by the connection.

wAudioPort

the server side audio port used by the connection.

szServerType

the server's friendly name.

Remarks

The connection information would be passed to your application through the status callback function in eConOptConnectionInfo status.

Requirements

DataBroker.h

TDataBrokerInputOptions

This structure indicates the Input options.

```
typedef struct
{
TDataBrokerStatusCallback      pfStatus;
TDataBrokerAVCallback          pfAV;
DWORD                          dwStatusContext;
DWORD                          dwAVContext;
TCHAR*                         pzServerType;
DWORD                          dwAudioCodec;
DWORD                          dwVideoCodec;
DWORD                          dwProtocolType;
TCHAR*                         zVSize;
} TDataBrokerInputOptions, *PTDataBrokerInputOptions;
```

Members

pfStatus
the pointer to the status callback function

pfAV
the pointer to the AV callback function

dwStatusContext
the context value for the status callback function

dwAVContext
the context value for the AV callback function

pzServerType
the server friendly name.

dwAudioCodec
the audio codec type

dwVideoCodec
the video codec type

dwProtocolType
the protocol type

zVSize
the Vsize (For 2K series only)

Remarks

Requirements

DataBroker.h

TDataBrokerOptions

This structure indicates the proxy options for DataBroker.

```
typedef struct
{
BOOL                    bEnableProxy;
DWORD                   dwProxyPort;
TCHAR                   szProxyName[128];
BOOL                    bEnableIPRestrict;
DWORD                   dwIPRestrictNum;
TCHAR                   (*pszIPRestrictList)[128];
DWORD                   dwFlags;
DWORD                   dwConnTimeout;
DWORD                   dwRWTimeout;
} TDataBrokerOptions;
```

Members

bEnableProxy

Enable proxy when connect to server by HTTP.

dwProxyPort

Proxy port.

szProxyName

IP of proxy server.

bEnableIPRestrict

Enable IP restriction check.

dwIPRestrictNum

The number of IP restriction strings you want to apply.

pszIPRestrictList

The pointer to the list of IP restriction strings. Proxy will not be applied to the IP starting by these strings.

dwFlags

Flags for the optional field.

dwConnTimeout

This is the socket connection timeout in seconds. The default value is 20 seconds. If you are connecting to 6000 servers, the timeout value should not be less than 20 seconds.

dwRWTimeout

This is the socket read/write timeout in seconds. The default value is 20 seconds

Remarks

Requirements

DataBroker.h

TMediaDataPacketInfoEx

This data structure is used in AV callback. When using in AV callback, please cast the TMediaDataPacketInfo* to TMediaDataPacketInfoEx*.

```
typedef struct
{
TMediaDataPacketInfo    tInfo;
char                    szReserved[16];
DWORD                   dwWidth;
DWORD                   dwHeight;
DWORD                   dwWidthPadLeft;
DWORD                   dwWidthPadRight;
DWORD                   dwHeightPadTop;
DWORD                   dwHeightPadBottom;
char                    szReserved2[16];
} TMediaDataPacketInfoEx;
```

Members

tInfo

tInfo contains information of the data packet.

szReserved

Reserved for future use.

dwWidth

dwWidth contains the width of the video frame if it is a video. If the frame has padding, it is included.

dwHeight

dwHeight contains the height of the video frame if it is a video. If the frame has padding, it is included.

dwWidthPadLeft

If the frame has padding, this is its left width.

dwWidthPadRight

If the frame has padding, this is its right width.

dwHeightPadTop

If the frame has padding, this is its height on top.

dwHeightPadBottom

If the frame has padding, this is its height on bottom.

szReserved2

Reserved for future use.

Remarks

Requirements

DataBroker.h

TAACExtInfo

This data structure is used for AAC Input-channel

```
typedef struct
{
DWORD                   dwSamplingFrequency;
DWORD                   dwChannelNumber;
} TAACExtInfo;
```

Members

dwSamplingFrequency
the sampling frequency

dwChannelNumber
the channel number

Remarks

Requirements

DataBroker.h

4.4. API Definition

The API definition is depicted here.

DataBroker_CheckIfLive

Check if a RTSP connection is live or file playback.

Syntax

```
SCODE DataBroker_CheckIfLive (      HANDLE hConn      );
```

Parameters

hConn

[in] the handle of a Connection instance

Return Values

DATABROKER_S_OK

The connection is live.

DATABROKER_E_INVALID_HANDLE

The Connection handle is invalid.

DATABROKER_E_FAIL

The connection is not a live channel.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_Connect

Establish the connection to the remote server.

Syntax

```
SCODE DataBroker_Connect (      HANDLE hConn            );
```

Parameters

hConn

[in] the handle of a Connection instance

Return Values

DATABROKER_S_OK

Establish the connection to the remote server successfully.

DATABROKER_E_INVALID_HANDLE

The Connection handle is invalid.

DATABROKER_E_RUNNINGCONNECTION

The connection is running.

DATABROKER_E_NOSETTING

No setting about this Connection.

DATABROKER_E_FAIL

Failed to establish the connection to the remote server.

Remarks

The DataBroker_SetConnectionOptions must be called at least once before calling this function.

Requirements

DataBroker.h

See Also

DataBroker_CreateConnection

Create a Connection instance.

Syntax

```
SCODE DataBroker_CreateConnection (     HANDLE hDataBroker,
                                        HANDLE *phConn          );
```

Parameters

hDataBroker

[in] the handle of DataBroker instance

***phConn**

[out] the pointer to the handle of a Connection instance.

Return Values

DATABROKER_S_OK

Create a Connection instance successfully.

DATABROKER_E_INVALID_HANDLE

The DataBroker handle is invalid.

DATABROKER_E_OUT_OF_MEMORY

Memory allocate failed.

DATABROKER_E_TOO_MANY_CONNECTIONS

Can’t create any more connection.

DATABROKER_E_FAIL

Failed to create a Connection instance.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_CreateInput

Create an Input instance. This function is obsolete please use DataBroker_CreateInputEx instead.

Syntax

```
SCODE DataBroker_ CreateInput (     HANDLE *phInput
                                    DWORD dwVersion     );
```

Parameters

***phInput**

[out] a pointer to receive the handle of Input instance.

dwVersion

[in] the library version.

Return Values

DATABROKER_S_OK

Create an Input instance successfully.

ERR_INVALID_VERSION

Library version is invalid.

DATABROKER_E_OUT_OF_MEMORY

Memory allocate failed.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_CreateInputEx

Create an Input instance. This function is meant to replace DataBroker_CreateInput.

Syntax

```
SCODE DataBroker_ CreateInputEx (    HANDLE hDataBroker,
                                     HANDLE *phInput          );
```

Parameters

hDataBroker

[in] the handle of DataBroker instance

***phInput**

[out] a pointer to receive the handle of Input instance.

Return Values

DATABROKER_S_OK

Create an Input instance successfully.

ERR_INVALID_VERSION

Library version is invalid.

DATABROKER_E_OUT_OF_MEMORY

Memory allocate failed.

DATABROKER_E_TOO_MANY_CONNECTIONS

Can't create any more input due to maximum connection limitation is reached. To solve this problem, it should be assigned larger connection number when initialize Databroker object.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_DeleteConnection

Delete a Connection instance.

Syntax

```
SCODE DataBroker_DeleteConnection (     HANDLE hDataBroker,
                                        HANDLE *phConn          );
```

Parameters

hDataBroker

[in] the handle of DataBroker instance.

***phConn**

[in] the pointer to the handle of a Connection instance.

Return Values

DATABROKER_S_OK

Delete the Connection instance successfully.

DATABROKER_E_INVALID_HANDLE

The DataBroker handle or Connection handle is invalid.

Remarks

A connection could be used to connect to different servers (of course, only connect to one server at the same moment) by setting different options. So it is more efficient to create all the connections when program starts and does not delete those connection until program ends.

Requirements

DataBroker.h

See Also

DataBroker_DeleteInput

Delete an Input instance. This function is obsolete please use DataBroker_DeleteInputEx

Syntax

```
SCODE DataBroker_DeleteInput (    HANDLE *phInput    );
```

Parameters

***phInput**

[in] a pointer to the handle of an Input instance.

Return Values

DATABROKER_S_OK

Delete the Input instance successfully.

DATABROKER_E_INVALID_HANDLE

The Input handle is invalid.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_DeleteInputEx

Delete an Input instance. This function is to replace DataBroker_DeleteInput

Syntax

```
SCODE DataBroker_DeleteInput (      HANDLE hDataBroker,
                                    HANDLE *phInput          );
```

Parameters

hDataBroker

[in] the handle of DataBroker instance

***phInput**

[in] a pointer to the handle of an Input instance.

Return Values

DATABROKER_S_OK

Delete the Input instance successfully.

DATABROKER_E_INVALID_HANDLE

The Input handle is invalid.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_Disconnect

Disconnect from remote server.

Syntax

```
SCODE DataBroker_Disconnect (      HANDLE hConn      );
```

Parameters

hConn

[in] the handle of a Connection instance.

Return Values

DATABROKER_S_OK

Disconnect from remote server successfully.

DATABROKER_E_INVALID_HANDLE

The Connection handle is invalid.

Remarks

The connection is not yet closed after this function returned. The application should wait for the eOnStopped status called to ensure the connection is really torn down.

Requirements

DataBroker.h

See Also

DataBroker_ForceIFrame

Force server to send an I-Frame immediately.

Syntax

```
SCODE DataBroker_ ForceIFrame (      HANDLE hConn    );
```

Parameters

hConn

[in] the handle of a Connection instance.

Return Values

DATABROKER_S_OK

Force server to send a I-Frame successfully.

DATABROKER_E_INVALID_HANDLE

The Connection handle is invalid.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_GetVersionInfo

Get the version information of the DataBroker library.

Syntax

```
SCODE DataBroker_ GetVersionInfo (      BYTE *byMajor,
                                        BYTE *byMinor,
                                        BYTE *byBuild,
                                        BYTE *byRevision        );
```

Parameters

***byMajor**

[out] the pointer to the Major byte of the version.

***byMinor**

[out] the pointer to the Minor byte of the version.

***byBuild**

[out] the pointer to the Build byte of the version.

***byRevision**

[out] the pointer to the Revision byte of the version.

Return Values

DATABROKER_S_OK

Get the version information successfully.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_Initial

Create a DataBroker instance.

Syntax

```
SCODE DataBroker_Initial (    HANDLE *phDataBroker,
                              DWORD dwMaxConn,
                              TDataBrokerStatusCallback pfStatus,
                              TDataBrokerAVCallback pfAV,
                              DWORD dwSupportCodec,
                              DWORD dwFlag,
                              DWORD dwVersion                    );
```

Parameters

***phDataBroker**

[out] a pointer to receive the handle of DataBroker object

dwMaxConn

[in] the maximum number of connections that DataBroker can handle.

pfStatus

[in] the pointer to the status callback function. For most case, this pointer is set to NULL.

pfAV

[in] the pointer to the AV callback function. For most case, this pointer is set to NULL.

dwSupportCodec

[in] specify the codec types which client supports. For most case, this value is set to mctALLCODEC.

dwFlag

[in]

dwVersion

[in] the library version.

Return Values

DATABROKER_S_OK

Create the DataBroker instance successfully.

ERR_INVALID_VERSION

Library version is invalid.

DATABROKER_E_OUT_OF_MEMORY
Memory allocate failed.

DATABROKER_E_FAIL
Failed to create the DataBroker instance.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_InputPacket

Input stream packet from network client to the media unpacketizer and parser. This function is obsolete please use DataBroker_InputPacketEx instead.

Syntax

```
SCODE DataBroker_ InputPacket (    HANDLE hInput,
                                   TsdrMediaType dwMediaType,
                                   BYTE *pbyData,
                                   DWORD dwLength                );
```

Parameters

hInput

[in] the handle of an Input instance.

dwMediaType

[in] the media type of the stream data.

***pbyData**

[in] the stream data received from network.

dwLength

[in] the length of the input stream data.

Return Values

DATABROKER_S_OK

Input stream data from network client to unpacketizer and parser successfully.

DATABROKER_E_OUT_OF_MEMORY

Memory allocate failed.

DATABROKER_E_FAIL

Failed to input stream data from network client to unpacketizer and parser.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_InputPacketEx

Input stream packet from network client to the media unpacketizer and parser. This function solves the problem for DataBroker_InputPacket that sometimes the frame could lose if the data size is small. Please use the hInput handle created by DataBroker_CreateInputEx. Do not use the handle created by DataBroker_CreateInput.

Syntax

```
SCODE DataBroker_ InputPacket (     HANDLE hInput,
                                    TsdrMediaType dwMediaType,
                                    BYTE *pbyData,
                                    DWORD dwLength                );
```

Parameters

hInput

[in] the handle of an Input instance.

dwMediaType

[in] the media type of the stream data.

***pbyData**

[in] the stream data received from network or from the callback TDataBrokerNetPacketCallback.

dwLength

[in] the length of the input stream data.

Return Values

DATABROKER_S_OK

Input stream data from network client to unpacketizer and parser successfully.

DATABROKER_E_OUT_OF_MEMORY

Memory allocate failed.

DATABROKER_E_FAIL

Failed to input stream data from network client to unpacketizer and parser.

Remarks

Requirements

DataBroker.h

DataBroker_InputTxPacket

Input upstream packet into network client and it will be sent to server side. This is only available for 6000/7000/8000 series servers

Syntax

```
SCODE
DataBroker_ InputTxPacket (     HANDLE hConn,
                                TMediaDataPacketInfo *ptMediaPacketInfo
                                DWORD dwDataTimePeriod                  );
```

Parameters

hConn

[in] the handle of a connection

ptMediaPacketInfo

[in] the media packet to be sent. Currently, only audio packets encoded by eACodecG729A are allowed.

dwDataTimePeriod

[in] the time period that the data in ptMediaPacketInfo needs when playing back. This time is in milliseconds unit, and it is used for DataBroker to control the data rate sent to server.

Return Values

DATABROKER_S_OK

Input upstream packet successfully.

DATABROKER_E_INVALID_HANDLE

The connection handle is not correct.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_JumpMediaStreaming

For RTSP connection and the connection is a file playback, users could use this function to jump to arbitrary point in the period.

Syntax

```
SCODE
DataBroker_JumpMediaStreaming (     HANDLE hConn,
                                    DWORD dwPercentage    );
```

Parameters

hConn

[in] the handle of a Connection instance.

dwPercentage

[in] This is the percentage to be set. The percentage is 10000 based. That means, when set 100, the module will move the play location to 100/10000 = 0.01 position.

Return Values

DATABROKER_S_OK

Force server to send a I-Frame successfully.

DATABROKER_E_INVALID_HANDLE

The Connection handle is invalid.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_PauseMediaStreaming

For RTSP connection, pause the non-live connection. The server will not stream any media after pause. Users could use DataBroker_ResumeMediaStreaming to resume the streaming.

Syntax

```
SCODE DataBroker_PauseMediaStreaming (      HANDLE hConn    );
```

Parameters

hConn

[in] the handle of a Connection instance

Return Values

DATABROKER_S_OK

The connection is paused.

DATABROKER_E_INVALID_HANDLE

The Connection handle is invalid.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_Release

Release a DataBroker instance.

Syntax

```
SCODE DataBroker_Release(    HANDLE *phDataBroker        );
```

Parameters

***phDataBroker**

[in/out] the pointer to a DataBroker object

Return Values

DATABROKER_S_OK

Release the DataBroker instance successfully.

DATABROKER_E_INVALID_HANDLE

The DataBroker handle is invalid.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_ResumeMediaStreaming

For RTSP connection, resume the paused non-live connection. The server will continue the streaming.

Syntax

```
SCODE DataBroker_ResumeMediaStreaming (    HANDLE hConn    );
```

Parameters

hConn

[in] the handle of a Connection instance

Return Values

DATABROKER_S_OK

The connection resume successfully.

DATABROKER_E_INVALID_HANDLE

The Connection handle is invalid.

Remarks

If the connection already timeout, the module will try to re-establish the connection and set the playing point to where paused.

Requirements

DataBroker.h

See Also

DataBroker_SetCodecPriority

Set the codec using priority for all connections.

Syntax

```
SCODE DataBroker_SetCodecPriority (   HANDLE hDataBroker,
                                      DWORD *pdwVideoCodec,
                                      DWORD *pdwAudioCodec,        );
```

Parameters

hDataBroker

[in] the handle of DataBroker instance

pdwVideoCodec

[in] the pointer to an array which contains the priority order of Video codecs.

pdwAudioCodec

[in] the pointer to an array which contains the priority order of Audio codecs.

Return Values

DATABROKER_S_OK

Set options to a DataBroker instance successfully.

DATABROKER_E_INVALID_HANDLE

The Databroker handle is invalid.

Remarks

The priorities are only used for 6K servers.

Requirements

DataBroker.h

See Also

DataBroker_SetConnectionExtraOption

Set more option for a connection. The new option could be adjusted at any time during the connection is active (connected).

Syntax

```
SCODE DataBroker_SetConnectionExtraOption (     HANDLE hConn,
                                                DWORD dwOption,     );
                                                DWORD dwParam1,
                                                DWORD dwParam2
```

Parameters

hConn

[in] the handle of a Connection instance.

dwOption

[in] The option listed in TExtraOptions.

dwParam1

[in] the first parameter if needed. Please refer to TExtraOptions for more information.

dwParam2

[in] the second parameter if needed. Please refer to TExtraOptions for more information.

Return Values

DATABROKER_S_OK

Set the Connection options successfully.

DATABROKER_E_INVALID_HANDLE

The Connection handle is invalid.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_SetConnectionNetPacketCallback

Set network packet callback function and the related parameters. These settings could be changed during connection, but it is recommended to set them before connecting.

Syntax

```
SCODE
DataBroker_SetConnectionNetPacketCallback    HANDLE hConn,
(
                                             TDataBrokerNetPacketCallback
                                             pfNetPacketCallback,
                                             DWORD dwContext,
                                             BOOL bCallbackOnly
                                                                        );
```

Parameters

hConn

[in] the handle of a Connection instance.

pfNetPacketCallback

[in] the network packet callback function that would be got called whenever there comes the new network packet from server.

dwContext

[in] the context associated with the callback function.

bCallbackOnly

[in] if this flag is set, the module would not call back the frame data. That is the TDataBrokerAVCallback would not get called. This would save some CPU resources if the computer runs as a proxy server.

Return Values

DATABROKER_S_OK

Set the Connection option successfully.

DATABROKER_E_INVALID_HANDLE

The Connection handle is invalid.

Remarks

Requirements

DataBroker.h

DataBroker_SetConnectionOptions

Set options of a Connection instance.

Syntax

```
SCODE
DataBroker_ SetConnectionOptions (    HANDLE hConn,
                                      TDataBrokerConnectionOptions
                                      *ptOption                    );
```

Parameters

hConn

[in] the handle of a Connection instance.

***ptOption**

[in] the pointer to a TDataBrokerConnectionOptions object.

Return Values

DATABROKER_S_OK

Set the Connection options successfully.

DATABROKER_E_INVALID_HANDLE

The Connection handle is invalid.

DATABROKER_E_INVALID_ARG

The input argument is invalid.

DATABROKER_E_OUT_OF_MEMORY

Memory allocate failed.

DATABROKER_E_WRONG_CONNECTION_SETTING

Wrong settings.

DATABROKER_E_RUNNINGCONNECTION

The connection is running. You can't set a connection's options when it is running. Stop it before calling this function.

DATABROKER_E_FAIL

Failed to set the Connection options.

Remarks

The function will return DATABROKER_E_WRONG_CONNECTION_SETTING if the setting is not correct, the following list the possible reasons for this error code:

- szServerType is not specified
- For 2000 or 3000 video only model (VS3101) the protocol is not HTTP or the media includes audio.

- For 3000 series (A/V models) the protocol is HTTP but the media includes audio or the protocol is TCP or UDP but the media is not A/V.
- For 6000 servers, the protocol is TCP.
- For 7000 servers, the protocol is not TCP or not UDP.
- The szServerType contains model that is not listed in SrvDepResource module.

Requirements

DataBroker.h

See Also

DataBroker_SetConnectionUrlsExtra

Set extra information for the connection. The URLs will take effect only if it's called before calling DataBroker_Connect.

Syntax

```
SCODE
DataBroker_SetConnectionUrlsExtra (    HANDLE hConn,
                                       const char * pszVideoUrl,
                                       const char * pszVideoExtra,
                                       const char * pszTxUrl,
                                       const char * pszTxExtra,
                                       const char * pszRxUrl,
                                       const char * pszRxExtra,          );
```

Parameters

hConn

[in] the handle of a Connection instance.

pszVideoUrl

[in] the actual video URL when connecting to the server. Usually it is not necessary to set the URL when connecting to the server. But if you are connecting to the IIS solution for 2000 server, the URL should be updated so that DataBroker could connect to it correctly. For 7000 servers or other RTSP streaming server, the URL could be different from those specified in SrvDepResource module. In such case, please use this function to set the video URL. The URL should not contain the IP/host name part, and should not contain the parameters part (put parameters in pszVideoExtra instead).

pszVideoExtra

[in] the extra parameters for the specific video URL.

pszTxUrl

[in] this is reserved for future 6K servers

pszTxExtra

[in] this is reserved for future 6K servers

pszRxUrl

[in] this is reserved for future 6K servers

pszRxExtra

[in] this is reserved for future 6K servers

Return Values

DATABROKER_S_OK

Set the Connection options successfully.

DATABROKER_E_INVALID_HANDLE
The Connection handle is invalid.

DATABROKER_E_INVALID_ARG
The input argument is invalid.

DATABROKER_E_OUT_OF_MEMORY
Memory allocate failed.

DATABROKER_E_WRONG_CONNECTION_SETTING
Wrong settings.

DATABROKER_E_RUNNINGCONNECTION
The connection is running. You can't set a connection's options when it is running.

DATABROKER_E_FAIL
Failed to set the Connection options.

Remarks

Requirements

DataBroker.h

DataBroker_SetInputExtraOption

Set more option of an Input instance. The property could be changed at any time.

Syntax

```
SCODE DataBroker_SetInputExtraOption (      HANDLE hInput,
                                            DWORD dwOption,
                                            DWORD dwParam1,
                                            DWORD dwParam2,       );
```

Parameters

dwOption

[in] The option listed in TExtraOptions.

dwParam1

[in] the first parameter if needed. Please refer to TExtraOptions for more information.

dwParam2

[in] the second parameter if needed. Please refer to TExtraOptions for more information.

Return Values

DATABROKER_S_OK

Set the input option successfully.

DATABROKER_E_INVALID_HANDLE

The input handle is invalid.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_SetInputOptions

Set options of an Input instance.

Syntax

```
SCODE
DataBroker_SetInputOptions (    HANDLE hInput,
                                TDataBrokerInputOptions *ptInputOptions    );
```

Parameters

hInput

[in] the handle of an Input instance.

***ptInputOptions**

[in] the pointer to a TDataBrokerInputOptions structure.

Return Values

DATABROKER_S_OK

Set options of an Input instance successfully.

DATABROKER_E_INVALID_HANDLE

The Input handle is invalid.

DATABROKER_E_INVALID_ARG

The argument is invalid.

DATABROKER_E_OUT_OF_MEMORY

Memory allocate failed.

DATABROKER_E_WRONG_CONNECTION_SETTING

Wrong settings.

DATABROKER_E_FAIL

Failed to set options of an Input instance.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_SetOptions

Set proxy and timeout options to a DataBroker instance.

Syntax

```
SCODE DataBroker_SetOptions (     HANDLE hDataBroker,
                                  TDataBrokerOptions *ptOptions    );
```

Parameters

hDataBroker

[in] the handle of DataBroker instance

ptOptions

[in] the pointer to a TDataBrokerOptions object

Return Values

DATABROKER_S_OK

Set options to a DataBroker instance successfully.

DATABROKER_E_INVALID_ARG

The input argument is invalid.

DATABROKER_S_FAIL

Failed to set options to a DataBroker instance.

Remarks

You can set which servers should be applied proxy.

Note: If proxy is applied to a 3000 series video only server or a 3000 series AV server connected by HTTP, you can not get VSize and Language from callback function.

Requirements

DataBroker.h

See Also

DataBroker_StartTxConnection

Start to establish the upstream connection. This is only available for 6000/7000/8000 series servers. When the connection is established, the eOnTxChannelStart status code will be called back. If the channel is already occupied, eOnAudioUpstreamOccupied will be called back. It might happen that server already changes the audio mode, and the new audio mode does not support talk, in such case, the status eOnAudioUpstreamDisabled will be called.

Syntax

```
SCODE DataBroker_StartTxConnection (    HANDLE hConn    );
```

Parameters

hConn

[in] the handle of a connection.

Return Values

DATABROKER_S_OK

Input upstream packet successfully.

DATABROKER_E_INVALID_HANDLE

The connection handle is not correct.

DATABROKER_E_FAIL

Unable to start the upstream connection. It might because the control connection to server is not established yet.

Remarks

Requirements

DataBroker.h

See Also

DataBroker_StopTxConnection

Close the upstream connection. This is only available for 6000/7000/8000 series servers. When the connection is closed, the eOnTxChannelClosedstatus code will be called back.

Syntax

```
SCODE DataBroker_StopTxConnection (    HANDLE hConn     );
```

Parameters

hConn

[in] the handle of a connection

Return Values

DATABROKER_S_OK

Input upstream packet successfully.

DATABROKER_E_INVALID_HANDLE

The connection handle is not correct.

Remarks

Requirements

DataBroker.h

See Also

None