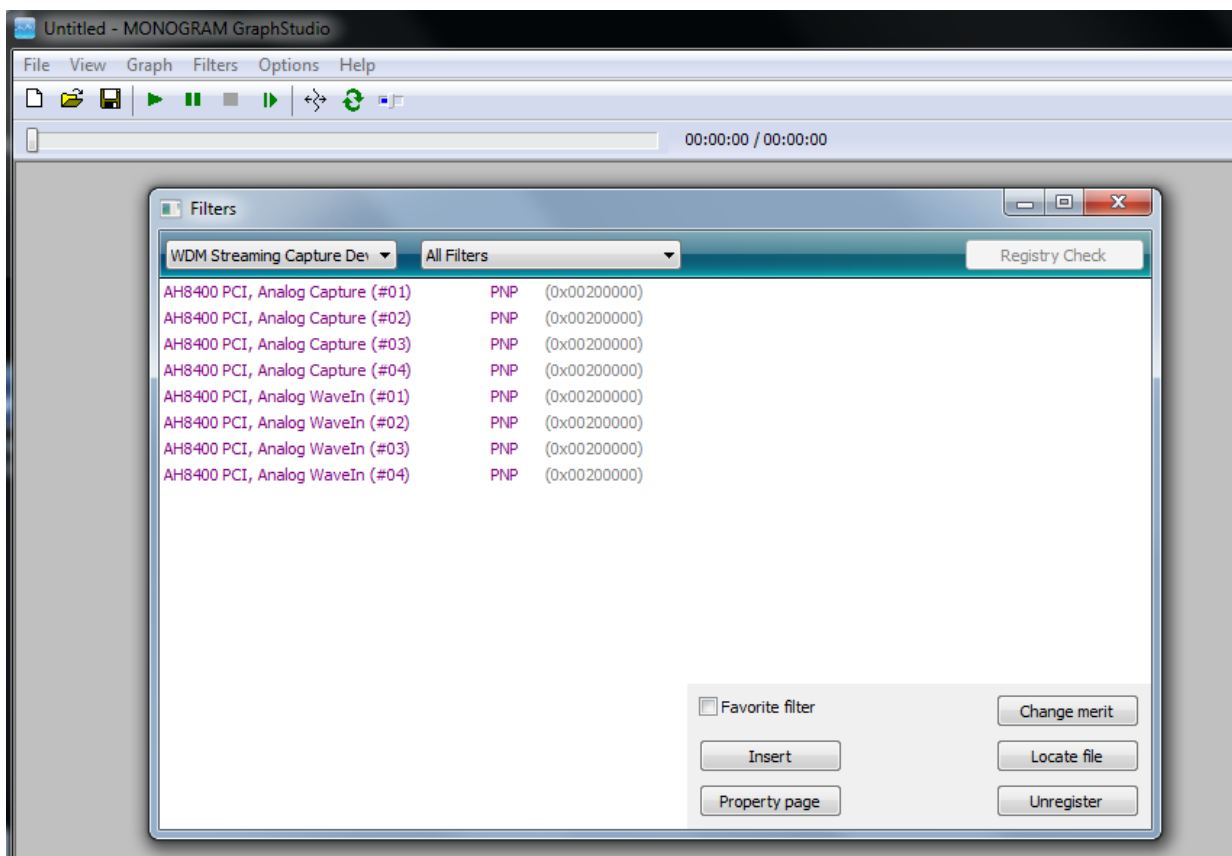


SC290 DirectShow Software Programming Guide

Customer uses DirectShow to develop software can bypass our SDK to access AH8400 directly. Majority of device properties is implemented by Microsoft DirectShow standard interface. Software developer can refer to Section 1 and Section 2 to control them. Other custom properties are implemented by **IKsPropertySet interface**. The interface can be queried from our capture source filter. Section 3 will describe how to access them in detail.

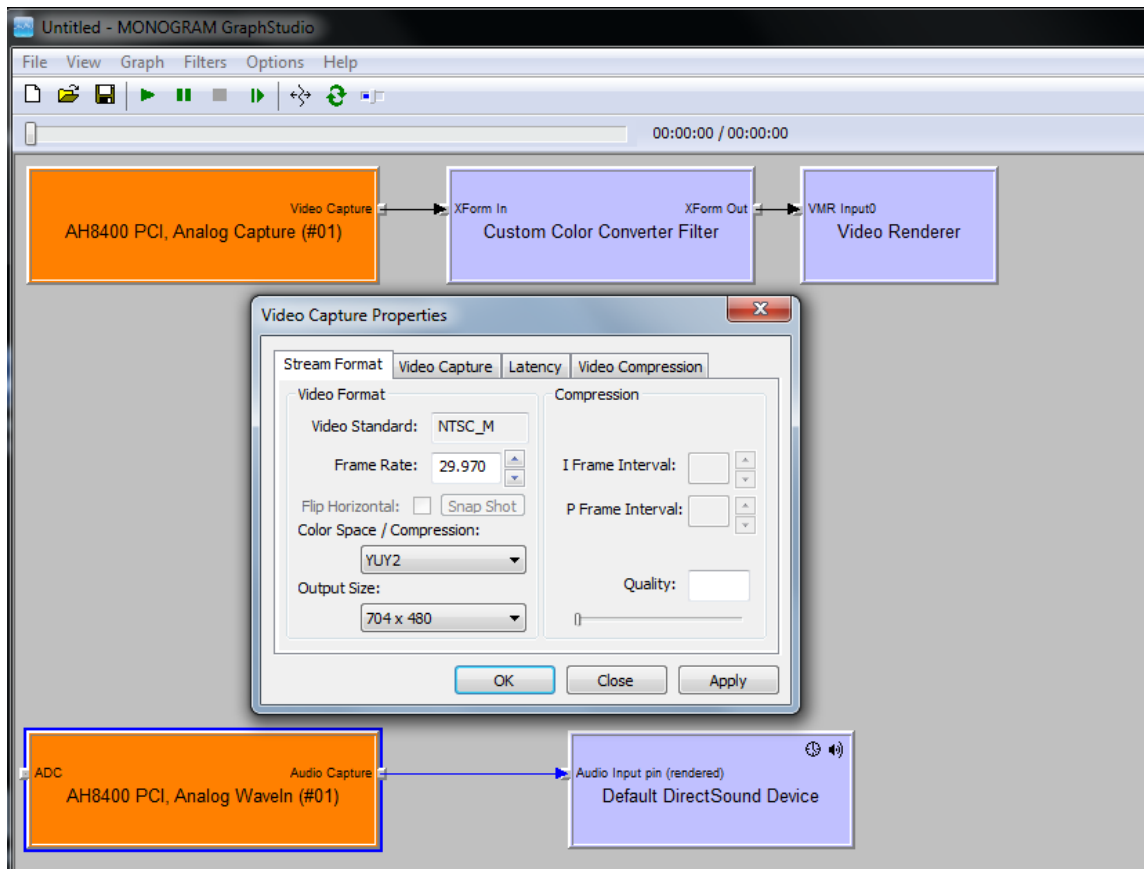
All filter names are "AH8400 PCI, Analog Capture (#XX)" for video, and "AH8400 PCI, Analog WaveIn (#XX)" for audio. They are registered at "WDM Streaming Captures Devices" category.



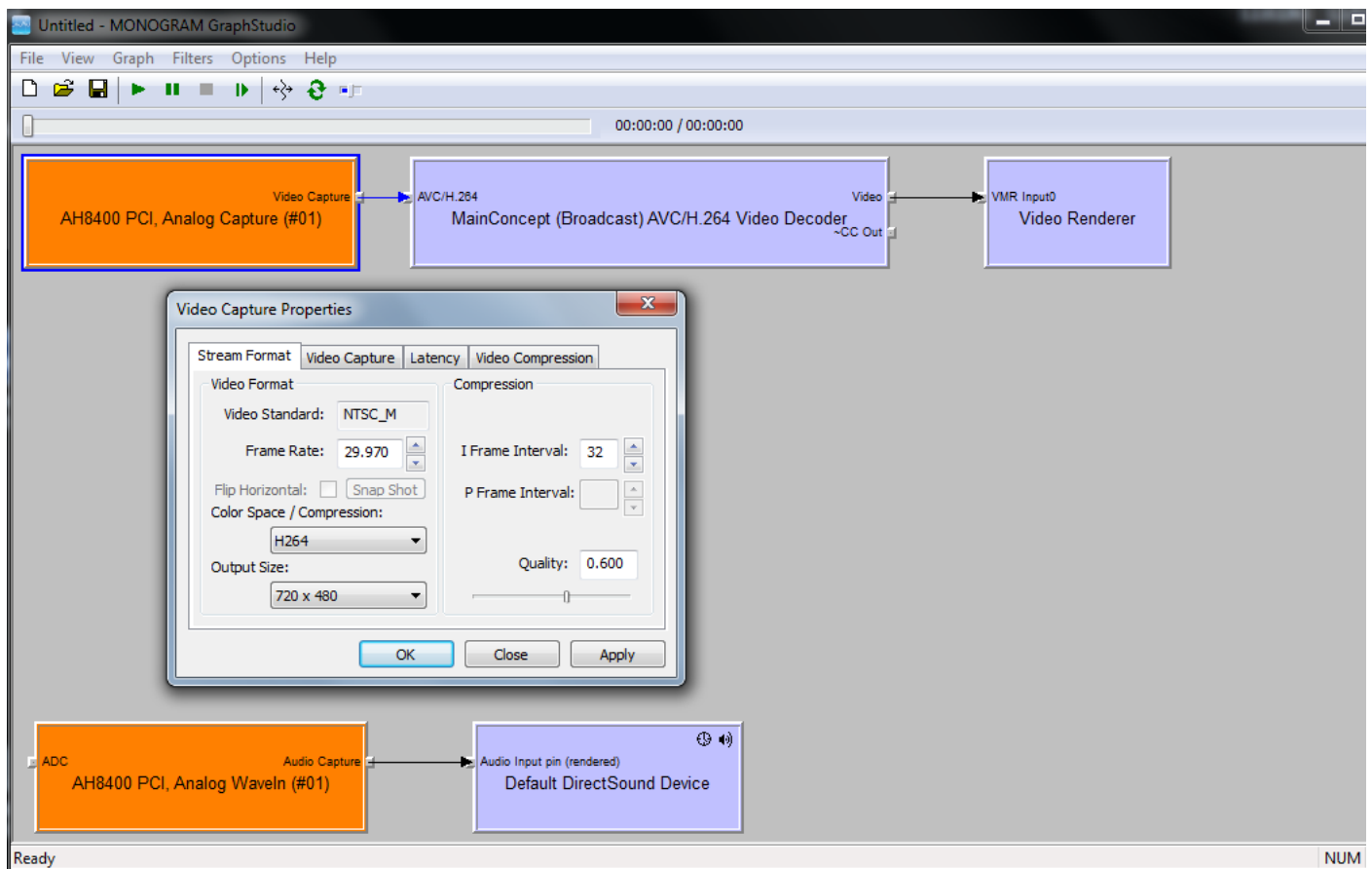
AH8400 is a hardware compression chip, it can output YUY2, and two H.264 streams at the same time. We use H.264 to stand for main stream.

```
#define MEDIASUBTYPE_H264 0x34363248, 0x0000, 0x0010, 0x80, 0x00, 0x00, 0xAA, 0x00, 0x38, 0x9B, 0x71
```

For the preview output, here, the video format is YV12 and audio format is PCM. The connection of filters is as:

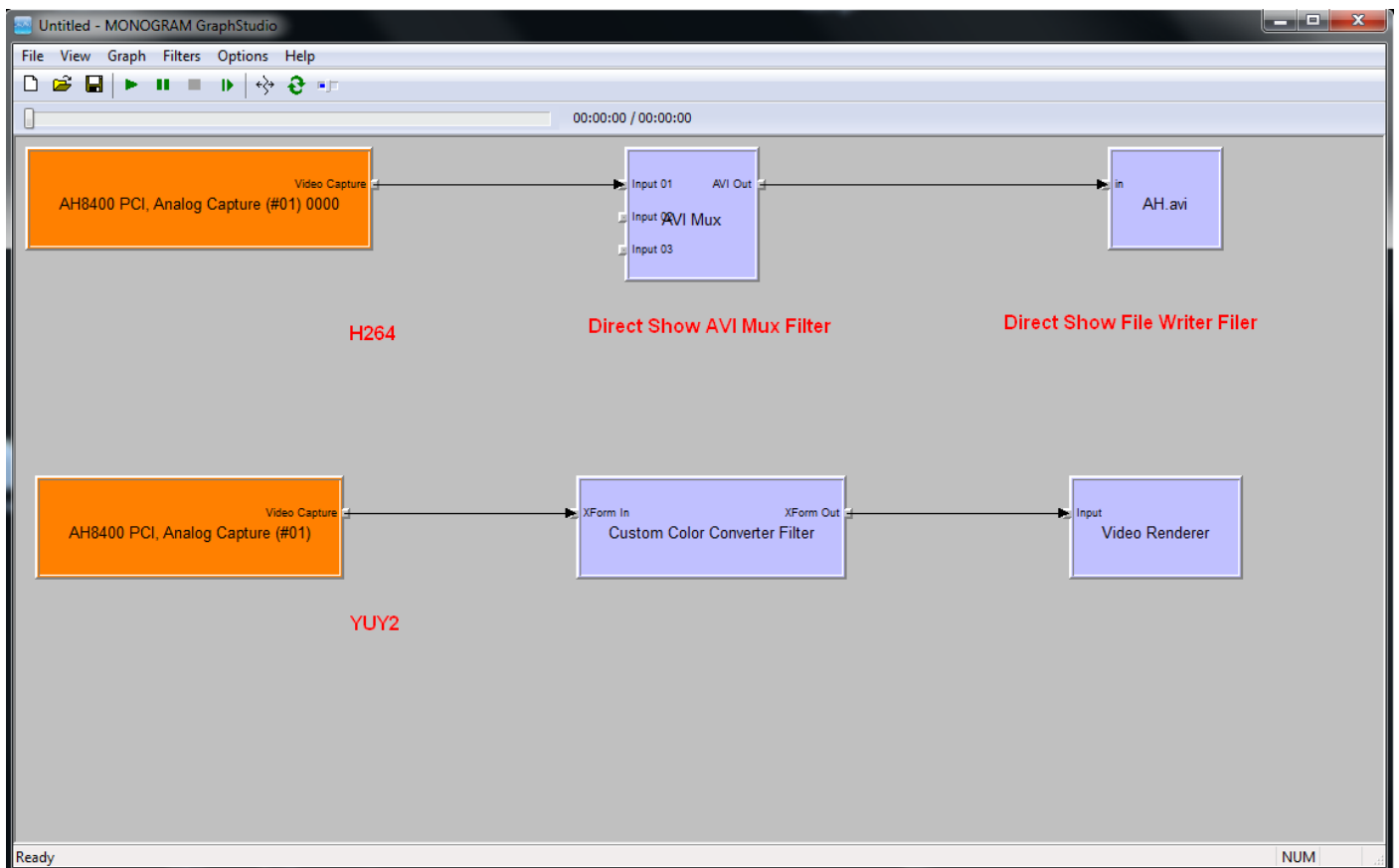


Main stream connection is as:



Moreover, customer wants to use graphedit to save H.264 stream into AVI can reference as below:

The graph demonstrates how to save AVI file. Here, YUY2 stream is used as preview function.



1. ACCESS VIDEO STANDARD (IAMAnalogVideoDecoder)

The video standard is implemented by IAMAnalogVideoDecoder interface. Customer must to setup the correct standard before accessing video format. For example, the 720X480@30fps format is only implemented under NTSC, and the 720x576@25fps format is only implemented under PALB.

EXAMPLE#01: SET STANDARD TO NTSC.

```
m_pCommonCaptureGraphBuilder2->FindInterface( NULL,
                                                NULL,
                                                m_pVideoCaptureSourceBaseFilter,
                                                IID_IAMAnalogVideoDecoder,
                                                (VOID **) (&m_pAMAnalogVideoDecoder) );

m_pAMAnalogVideoDecoder->put_TVFormat( AnalogVideo_NTSC_M );
```

2. ACCESS OUTPUT FORMAT OF CAPTURE PIN (IAMStreamConfig)

To get/set output format of capture pin, customer can use IAMStreamConfig interface.

EXAMPLE#01: SET VIDEO OUTPUT FORAMT TO 704X480 AT 30FPS.

```
m_pCommonCaptureGraphBuilder2->FindInterface( &LOOK_DOWNSTREAM_ONLY,
                                                NULL,
                                                m_pVideoCaptureSourceBaseFilter,
                                                IID_IAMStreamConfig,
                                                (VOID **)( &m_pAMStreamConfig ) );

AM_MEDIA_TYPE * pmt = NULL;
m_pAMStreamConfig->GetFormat( &pmt );
((VIDEOINFOHEADER *) (pmt->pbFormat))->bmiHeader.biCompression = MAKEFOURCC('Y', 'U', 'Y', '2');
((VIDEOINFOHEADER *) (pmt->pbFormat))->bmiHeader.biHeight = 704;
((VIDEOINFOHEADER *) (pmt->pbFormat))->bmiHeader.biWidth = 480;
((VIDEOINFOHEADER *) (pmt->pbFormat))->bmiHeader.biBitCount = 16;
((VIDEOINFOHEADER *) (pmt->pbFormat))->bmiHeader.biSizeImage = 704 * 480 * 16 / 8;
((VIDEOINFOHEADER *) (pmt->pbFormat))->AvgTimePerFrame = (ULONG) (INT) (10000000.0 / 30.000);
((VIDEOINFOHEADER *) (pmt->pbFormat))->dwBitRate = (ULONG) (INT) (704 * 480 * 16 * 30.000);
m_pAMStreamConfig->SetFormat( pmt );
DeleteMediaType( pmt );
```

EXAMPLE#02: SET AUDIO OUTPUT FORAMT TO MONO, 16BITS, AND 8000HZ.

```
m_pCommonCaptureGraphBuilder2->FindInterface( &LOOK_DOWNSTREAM_ONLY,
                                                NULL,
                                                m_pAudioCaptureSourceBaseFilter,
                                                IID_IAMStreamConfig,
                                                (VOID **)( &m_pAMStreamConfig ) );

AM_MEDIA_TYPE * pmt = NULL;
m_pAMStreamConfig->GetFormat( &pmt );
((WAVEFORMATEX *) (pmt->pbFormat))->nChannels = (USHORT) (1);
((WAVEFORMATEX *) (pmt->pbFormat))->wBitsPerSample = (USHORT) (16);
((WAVEFORMATEX *) (pmt->pbFormat))->nSamplesPerSec = (ULONG) (8000);
((WAVEFORMATEX *) (pmt->pbFormat))->nBlockAlign = (USHORT) (1 * 16 / 8);
((WAVEFORMATEX *) (pmt->pbFormat))->nAvgBytesPerSec = (ULONG) (1 * 16 * 8000 / 8);
m_pAMStreamConfig->SetFormat( pmt );
DeleteMediaType( pmt );
```

3 Customer Property Access

Customer can access all custom properties by IKsPropertySet, the parameter rguidPropSet of IKsPropertySet::Set/Get function, is defined as below:

```
GUID PROPSETID_AMEBDAD_CUSTOM_PROP =  
{ 0xD1E5209F, 0x68FD, 0x4529, 0xBE, 0xE0, 0x5E, 0x7A, 0x1F, 0x47, 0x92, 0x16 };
```

All custom properties are defined as below:

```
typedef enum {  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_QUEUE_BUFFER_SIZE    = 216,  
    KSPROPERTY_CUSTOM_SET_OSD_LINE                          = 920  
    KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING                  = 921  
    KSPROPERTY_CUSTOM_SET_OSD_COLOR                        = 929  
    KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION                   = 940,  
    KSPROPERTY_CUSTOM_XET_GPIO_DATA                       = 941,  
    KSPROPERTY_CUSTOM_XET_GPIO_SUPPORT                     = 942,  
} KSPROPERTY_AMEBDAD_CUSTOM;
```

3.1. KSPROPERTY_CUSTOM_SET_OSD_LINE (920) (WRITE ONLY)

3.1. KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING (921) (WRITE ONLY)

3.1. KSPROPERTY_CUSTOM_SET_OSD_COLOR (929) (WRITE ONLY)

The properties allow you to change AH8400's OSD context. The property KSPROPERTY_CUSTOM_SET_OSD_COLOR allows you to change string's color. Here, there are 16 kinds of colors can be selected by you. Also, you can modify it dynamically during recording.

SUPPORT VALUE: 0 ~ 15 - COLOR#0 ~ COLOR#15

The properties *SET_OSD_LINE and *SET_OSD_TEXT_STRING both help you to change string context. Note!! When you set the custom string into device, our driver will auto disable default time OSD.

SUPPORT VALUE: 0 ~ 2 - LINE#0 ~ LINE#2

SUPPORT LINE#0 STRING: 32 CHARACTERS

SUPPORT LINE#1 STRING: 15 CHARACTERS

SUPPORT LINE#2 STRING: 15 CHARACTERS

EXAMPLE#01: TO CHANGE OSD COLOR TO COLOR#1.

```
ULONG color = 0x0001;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_SET_OSD_COLOR, NULL, 0,  
                        &color, sizeof(ULONG) );
```

SUPPORT VALUE: 0 ~ 2 - LINE#0 ~ LINE#2

EXAMPLE#01: TO CHANGE LINE#0'S STRING.

```
ULONG line = 0x0000;
```

```
CHAR string[] = "1234567890ABCDEF";
```

```
CHAR psz[ 256 ];
```

```
memset( psz, 0x00, 64 );
```

```
sprintf( psz, "%s", string );
```

```
psz[ 63 ] = 0x00;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_SET_OSD_LINE, NULL, 0,  
                        &line, sizeof(ULONG) );
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 921, NULL, 0, psz + 0, 16 )
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 922, NULL, 0, psz + 16, 16 )
```



```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 923, NULL, 0, psz + 32, 16 )
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 924, NULL, 0, psz + 48, 16 )
```

EXAMPLE#02: TO CHANGE LINE#1'S STRING.

```
ULONG line = 0x0001;
CHAR string[] = "ABCDEF1234567890";
CHAR psz[ 256 ];
memset( psz, 0x00, 64 );
sprintf( psz, "%s", string );
psz[ 63 ] = 0x00;
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,
                      KSPROPERTY_CUSTOM_SET_OSD_LINE, NULL, 0,
                      &line, sizeof(ULONG) );
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 921, NULL, 0, psz + 0, 16 )
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 922, NULL, 0, psz + 16, 16 )
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 923, NULL, 0, psz + 32, 16 )
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 924, NULL, 0, psz + 48, 16 )
```

3.2. KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION (940)

3.2. KSPROPERTY_CUSTOM_XET_GPIO_DATA (941)

3.2. KSPROPERTY_CUSTOM_GET_GPIO_SUPPORT (942) (READ ONLY)

The property allows you to access AH8400's GPIO interface. The property KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION allows you to control its direction. Here, writing 1 to bit enables this pin as output pin. Usually, the GPIO is controlled by the first chipset in one board.

SUPPORT VALUE: 0 ~ 1 - INPUT ~ OUTPUT

The property KSPROPERTY_CUSTOM_XET_GPIO_DATA allows you to access GPIO's data.

SUPPORT VALUE: 0 ~ 1 - LOW ~ HIGH

The property KSPROPERTY_CUSTOM_XET_GPIO_SUPPORT allows you to obtain GPIO's information (pin size) on hardware board. Developer can use it to check if the device can support GPIO access.

SUPPORT VALUE: 0 IS NON-SUPPORT

EXAMPLE#01: TO DEFINE GPIO AS 8 OUTPUT PINS [0:7] AND 8 INPUT PINS [8:15].

```
ULONG input = 0x00FF;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION, NULL, 0,  
                        &input, sizeof(ULONG) );
```

EXAMPLE#02: TO DEFINE GPIO AS 16 OUTPUT PINS [0:15] THEN PULL HIGH FOR ALL.

```
ULONG input = 0xFFFF;
```

```
ULONG data = 0xFFFF;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION, NULL, 0,  
                        &input, sizeof(ULONG) );
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_XET_GPIO_DATA, NULL, 0,  
                        &data, sizeof(ULONG) );
```

EXAMPLE#03: TO DEFINE GPIO AS 16 INPUT PINS [0:15] THEN READ DATA FROM IT.

```
ULONG input = 0x0000;
```

```
ULONG data = 0x0000;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,
                        KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION, NULL, 0,
                        &input, sizeof(ULONG) );
```

[illegible]

3.3. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_QUEUE_BUFFER_SIZE (216)

The property allow you to specify the number of the rendered video frame in the queue buffer for a preview or hardware-encoded (main, sub) stream. By the default, the queue size of the corresponding a preview and hardware-encoded stream is set 10 and 16. Here we recommended use the size by default because this is implicated in many resource issues. For example, the unexpected signal error may occur when you try to adjust the queue buffer size of which exceeds your system resource.

Note: Setting queue buffer size will involve in dynamically allocated memory.

EXAMPLE#01: TO SET THE PREVIEW QUEUE SIZE TO 10 FRAMES

[illegible]

EXAMPLE#02: TO SET THE HARDWARE-ENCODED QUEUE (MAIN) SIZE TO 16 FRAMES

[illegible]

EXAMPLE#03: TO SET THE HARDWARE-ENCODED QUEUE(SUB) SIZE TO 16 FRAMES

[illegible]

3.4 Video Encoder Property:

Please reference the two functions to get/set all video encoder's parameters.

```
static const GUID GUID_KPS_AH8400 = { 0xD1E5209F, 0x68FD, 0x4529, 0xBE, 0xE0, 0x5E, 0x7A, 0x1F, 0x47, 0x92, 0x16 };

BOOL OnGetVideoCompressionProperty( ULONG nProperty, ULONG * pValue )
{
    if( NULL == m_pAMVideoCompression ) { FALSE; }

    if( NULL == m_pKsPropertySet ) { FALSE; }

    if( nProperty == 0x00000000 ) { // KEY.FRAME.RATE (GOP)

        if( S_OK != m_pAMVideoCompression->get_KeyFrameRate( (LONG *) (pValue) ) ) { return FALSE; }
    }
    if( nProperty == 0x00000001 ) { // QUALITY

        double fQuality = 0.0f;

        if( S_OK != m_pAMVideoCompression->get_Quality( &fQuality ) ) { return FALSE; }

        *pValue = (ULONG) (fQuality * 10000.0f);
    }
    if( nProperty == 0x00000003 ) { // BIT.RATE.MODE

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 407, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x00000004 ) { // BIT.RATE

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 403, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x00000008 ) { // POST.RESOLUTION

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 401, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x00000009 ) { // POST.SKIP.FRAME RATE

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 402, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x0000000D ) { // POST.AVG.FRAME RATE

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 422, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x0000000A ) { // B.FRAME

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 411, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x0000000B ) { // PROFILE

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 412, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x0000000C ) { // ASPECT.RATIO

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 413, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    return TRUE;
}
```

```

BOOL OnSetVideoCompressionProperty( ULONG nProperty, ULONG nValue )
{
    if( NULL == m_pAMVideoCompression ) { return FALSE; }

    if( NULL == m_pKsPropertySet ) { return FALSE; }

    if( nProperty == 0x00000000 ) { // KEY.FRAME.RATE (GOP)
        if( S_OK != m_pAMVideoCompression->put_KeyFrameRate( nValue ) ) { return FALSE; }
    }
    if( nProperty == 0x00000001 ) { // QUALITY
        double fQuality = nValue;

        fQuality /= 10000.0f;

        if( S_OK != m_pAMVideoCompression->put_Quality( fQuality ) ) { return FALSE; }
    }
    if( nProperty == 0x00000002 ) { // OVERRIDE.KEY.FRAME
        if( S_OK != m_pAMVideoCompression->OverrideKeyFrame( nValue ) ) { return FALSE; }
    }
    if( nProperty == 0x00000003 ) { // BIT.RATE.MODE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 407, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000004 ) { // BIT.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 403, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000008 ) { // POST.RESOLUTION
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 401, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000009 ) { // POST.SKIP.FRAMERATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 402, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000D ) { // POST.AVG.FRAMERATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 422, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000A ) { // B.FRAME
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 411, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000B ) { // PROFILE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 412, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000C ) { // ASPECT.RATIO
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 413, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    return TRUE;
}

```

4. Application Note for DirectShow Developer

The developer who uses DirectShow to access our capture source filter need check the frame size in the callback function of your SampleGrabber class. If the frame size is 0 bytes, it means the frame is one bad frame. You should drop it. More detail, please check with our engineer team directly.