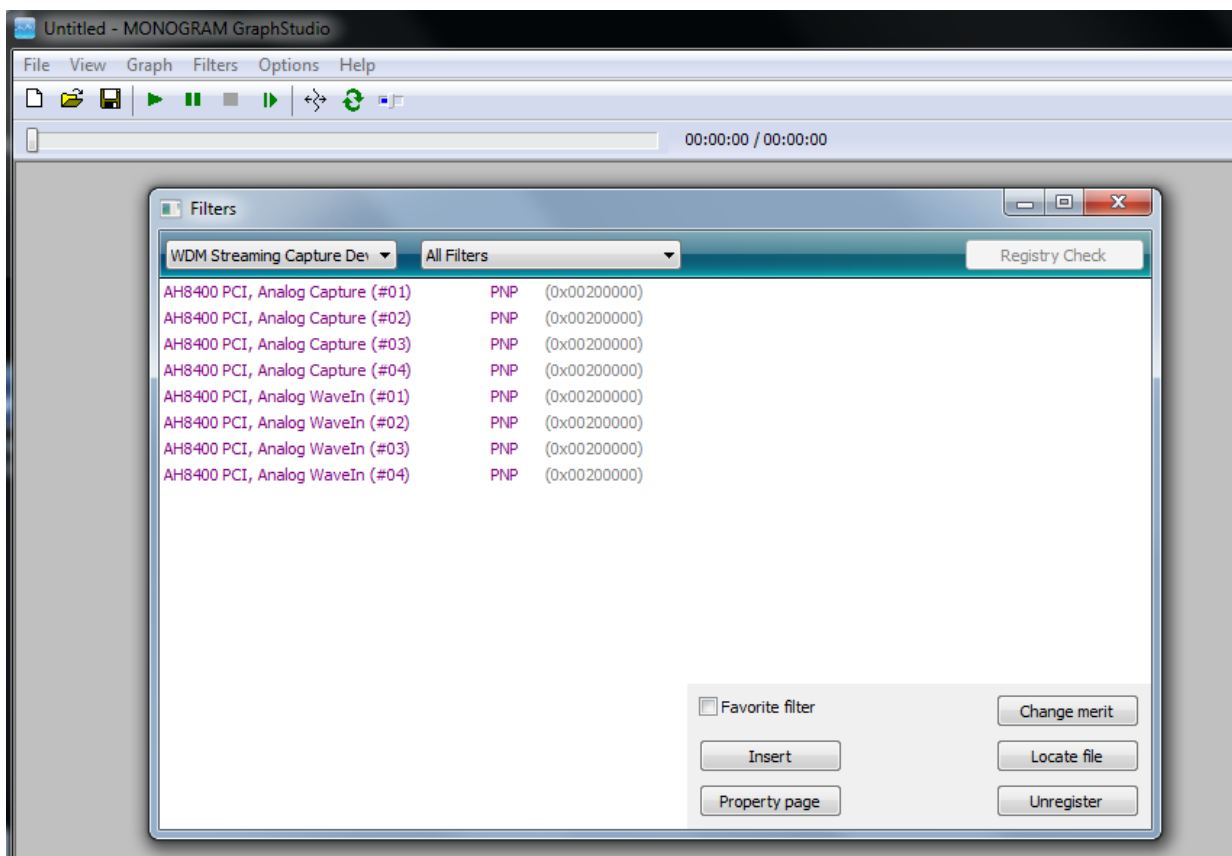


# SC290 DirectShow Software Programming Guide

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.

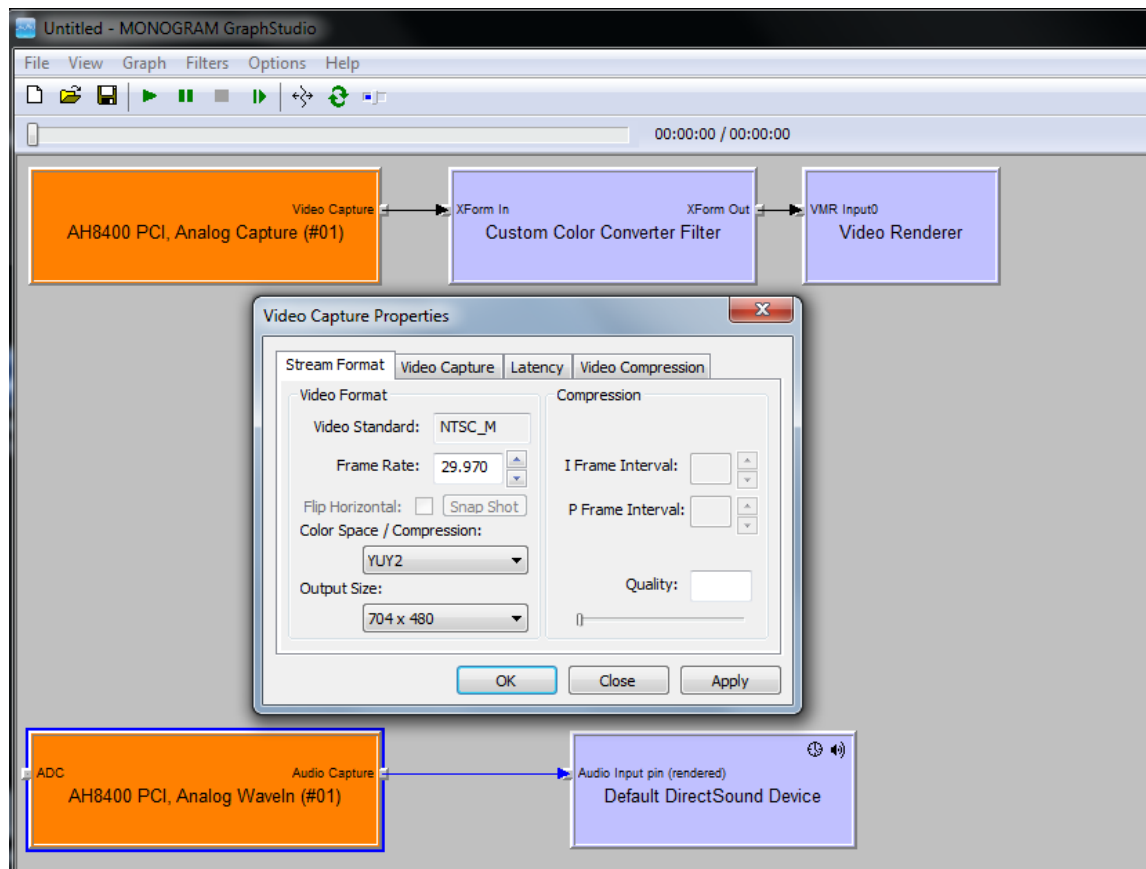
All filter names are "AH8400 PCI, Analog Capture (#XX)" for video, and "AH8400 PCI, Analog WaveIn (#XX)" for audio. They are registered at "WDM Streaming Captures Devices" category.



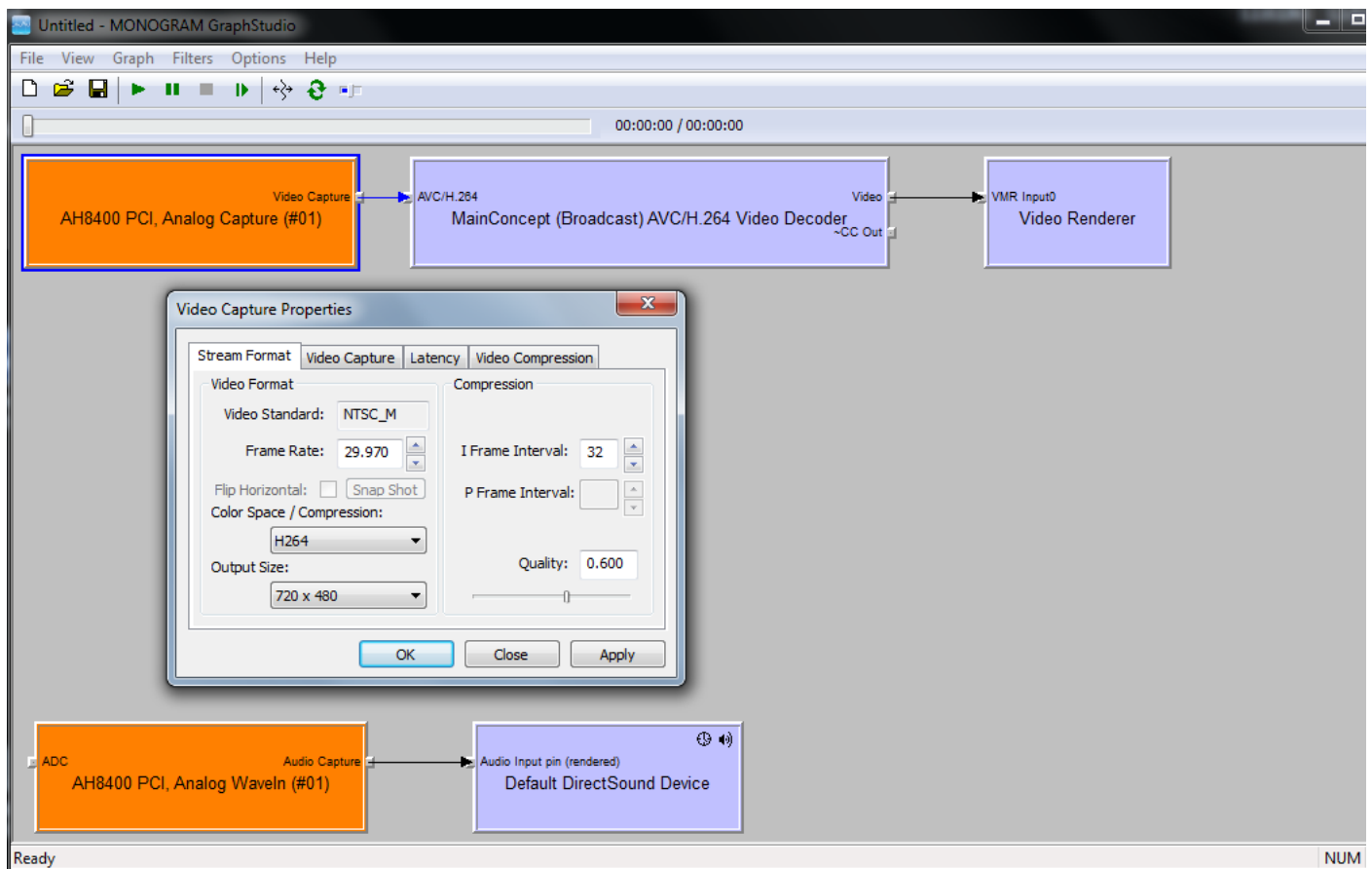
AH8400 is a hardware compression chip, it can output YUY2, and two H.264 streams at the same time. We use H.264 to stand for main stream.

```
#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:

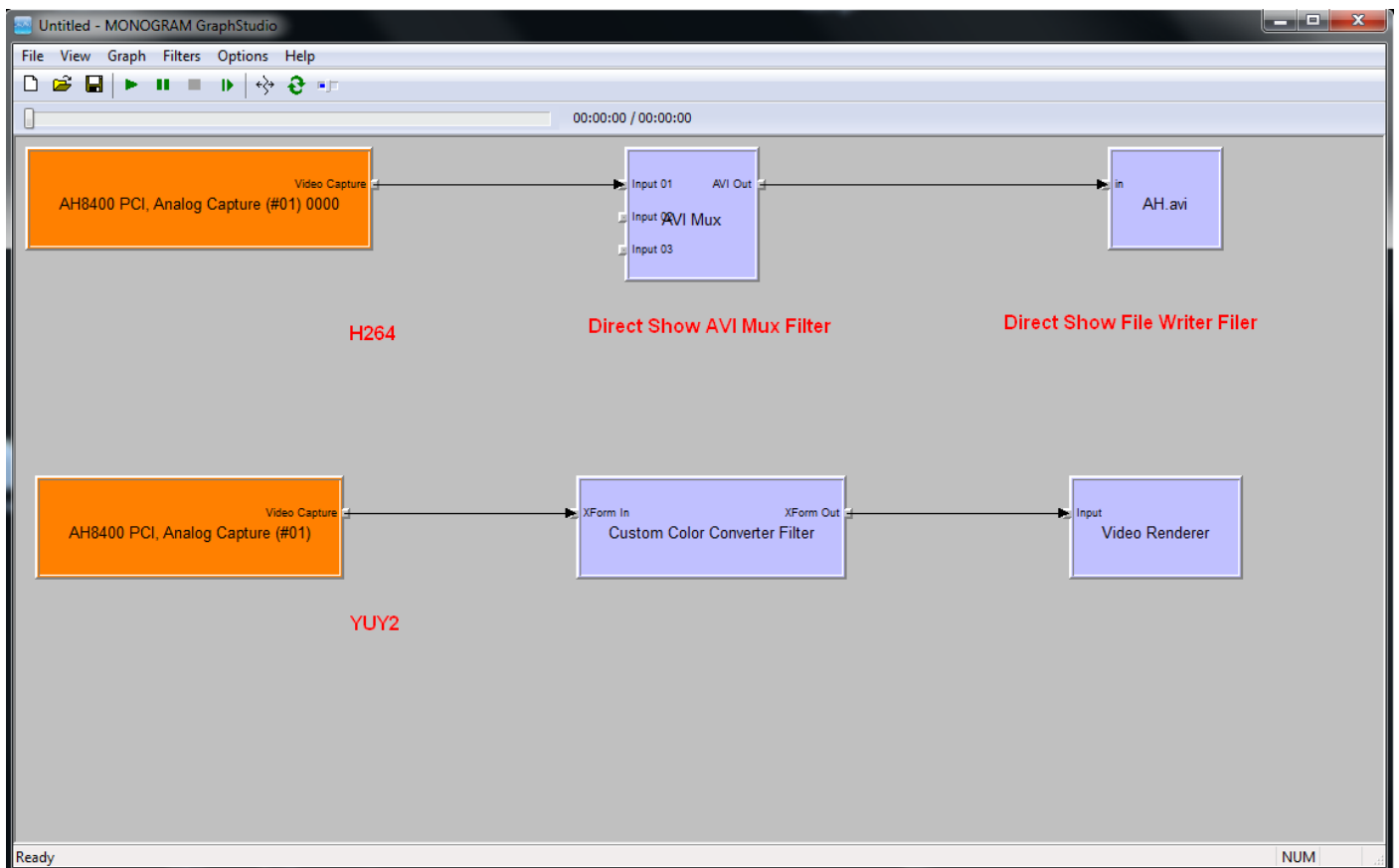


Main stream connection is as:



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_GET_DEVICE_SERIAL_NUMBER_INFO          = 0,  
    KSPROPERTY_CUSTOM_GET_DEVICE_BUS_NUMBER_INFO             = 2,  
    KSPROPERTY_CUSTOM_GET_ANALOG_VIDEO_MACROVISION            = 202,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_QUEUE_BUFFER_SIZE      = 216,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_FLEXIBLE_FPS_PATCH     = 218,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_FLEXIBLE_RESOLUTION_PATCH = 220,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_INPUT_AUTO_SCAN        = 232,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_VERTICAL_MIRROR         = 244,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_HORIZONTAL_MIRROR       = 245,  
    KSPROPERTY_CUSTOM_XET_PREVIEW_VIDEO_STREAM_POST_SKIP_FRAMERATE = 246,  
    KSPROPERTY_CUSTOM_XET_PREVIEW_VIDEO_STARAM_POST_AVG_FRAMERATE = 247,  
    KSPROPERTY_CUSTOM_XET_ANALOG_AUDIO_VOLUME                  = 251,  
    KSPROPERTY_CUSTOM_XET_PREVIEW_VIDEO_STERAM_POST_RESOLUTION = 350,  
    KSPROPERTY_CUSTOM_GET_PREVIEW_VIDEO_STARAM_FRAME_NUMBER_INFO = 351,  
    KSPROPERTY_CUSTOM_GET_PREVIEW_AUDIO_STARAM_FRAME_NUMBER_INFO = 361,  
    KSPROPERTY_CUSTOM_GET_ENCODER_VIDEO_DEFAULT_FRAME_NUMBER_INFO = 430,  
    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;
```





### 3.2. KSPROPERTY\_CUSTOM\_GET\_ANALOG\_VIDEO\_MACROVISION (202) (READ ONLY)

SUPPORT VALUE: 0 ~ 1 - UNLOCK ~ LOCK

```
LONG nLock = 0x00;
```

The property (202) allows you to detect if the input's media content owns HDCP or MarcoVision protection.

SUPPORT VALUE: 0, 1 - NO ~ YES

```
ULONG  HDCP = 0;
```

[illegible]

- 3.3. **KSPROPERTY\_CUSTOM\_GET\_ANALOG\_VIDEO\_SINGAL\_DEBUG\_INFO** (271) (READ ONLY)
- 3.3. **KSPROPERTY\_CUSTOM\_GET\_PREVIEW\_VIDEO\_STARAM\_FRAME\_NUMBER\_INFO** (351) (READ ONLY)
- 3.3. **KSPROPERTY\_CUSTOM\_GET\_PREVIEW\_AUDIO\_STARAM\_FRAME\_NUMBER\_INFO** (361) (READ ONLY)
- 3.3. **KSPROPERTY\_CUSTOM\_GET\_ENCODER\_VIDEO\_DEFAULT\_FRAME\_NUMBER\_INFO** (430) (READ ONLY)

The property **KSPROPERTY\_CUSTOM\_GET\_ANALOG\_VIDEO\_SINGAL\_DEBUG\_INFO** is used to get the debug information in capture card running state. The output information is 32-bit error numbers. If the number is 0, the device is working properly. You can call it in timer function to get current signal status regularly.

SUPPORT VALUE: 0: GOOD

OTHERS: ERROR BITS

EXAMPLE#01: TO GET CURRENT SINGAL DEBUG STATUS.

```
ULONG dwSingalDebugInfo = 0x00000000;
m_pKsPropertySet->Get( PROPSETID_AMEBDAD_CUSTOM_PROP,
                        KSPROPERTY_CUSTOM_GET_ANALOG_VIDEO_SINGAL_DEBUG_INFO,
                        NULL, 0,
                        &dwSingalDebugInfo, sizeof(ULONG), &temp );
```

The property **KSPROPERTY\_CUSTOM\_GET\_PREVIEW\_VIDEO\_STARAM\_FRAME\_NUMBER\_INFO** allows you to get the total number of frames in preview video. The property reads frame number information from video stream. You can call it in timer function to get current frame number regularly.

SUPPORT VALUE: FRAME NUMBER

EXAMPLE#02: TO GET VIDEO PREVIEW STREAM'S FRAME NUMBER.

```
ULONG dwPreviewVideoFrameNumber = 0;
m_pKsPropertySet->Get( PROPSETID_AMEBDAD_CUSTOM_PROP,
                        KSPROPERTY_CUSTOM_GET_PREVIEW_VIDEO_STARAM_FRAME_NUMBER_INFO,
                        NULL, 0,
                        &dwPreviewVideoFrameNumber, sizeof(ULONG), &temp );
```

The property **KSPROPERTY\_CUSTOM\_GET\_PREVIEW\_AUDIO\_STARAM\_FRAME\_NUMBER\_INFO** allows you to get the total number of frames in preview audio. The property reads frame number information from audio stream. You can call it in timer function to get current frame number regularly.

SUPPORT VALUE: FRAME NUMBER

EXAMPLE#03: TO GET AUDIO PREVIEW STREAM'S FRAME NUMBER.

```
ULONG dwPreviewAudioFrameNumber = 0;
m_pKsPropertySet->Get( PROPSETID_AMEBDAD_CUSTOM_PROP,
                        KSPROPERTY_CUSTOM_GET_PREVIEW_VIDEO_STARAM_FRAME_NUMBER_INFO,
                        NULL, 0,
                        &dwPreviewAudioFrameNumber, sizeof(ULONG), &temp );
```

The property **KSPROPERTY\_CUSTOM\_GET\_ENCODER\_VIDEO\_DEFAULT\_FRAME\_NUMBER\_INFO** allows you to get the total number of frames in video encoder. The property reads frame number information from compressed video stream. You can call it in timer function to get current frame number regularly.

SUPPORT VALUE: FRAME NUMBER

EXAMPLE#04: TO GET VIDEO ENCODER STREAM STREAM'S FRAME NUMBER.

```
ULONG dwEncoderVideoFrameNumber = 0;
m_pKsPropertySet->Get( PROPSETID_AMEBDAD_CUSTOM_PROP,
                        KSPROPERTY_CUSTOM_GET_ENCODER_VIDEO_DEFAULT_FRAME_NUMBER_INFO,
                        NULL, 0,
                        &dwEncoderVideoFrameNumber, sizeof(ULONG), &temp );
```

### 3.4. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_VERTICAL\_MIRROR (244)

### 3.4. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_HORIZONTAL\_MIRROR (245)

The two properties (244/245) are used to set mirror function. When mirror function is enabled, the vertical or horizontal video frame is inverted on display window. Same as deinterlacing, the property is used for display engine only.

SUPPORT VALUE: 0 ~ 1 - DISABLE ~ ENABLE

EXAMPLE#02: ENABLE THE VERTICAL MIRROR FUNCTION ON DISPLAY WINDOW

```
LONG enable = 0x01;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_VERTICAL_MIRROR, NULL, 0,  
                        &enable, sizeof(LONG) );
```

EXAMPLE#03: ENABLE THE HORIZONTAL MIRROR FUNCTION ON DISPLAY WINDOW

```
LONG enable = 0x01;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_HORIZONTAL_MIRROR, NULL, 0,  
                        &enable, sizeof(LONG) );
```

### 3.5. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_INPUT\_AUTO\_SCAN (232)

This property allows you to enable or disable the automatic scan video input signal source. If this function detects the actual video input source and format on capture card, it will automatically set the correct video input source and format.

SUPPORT VALUE: 0 ~ 1 - DISABLE ~ ENABLE

EXAMPLE#01: ENABLE THE AUTO INPUT SCAN FUNCTION

```
LONG enable = 0x01;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_INPUT_AUTO_SCAN,  
                        NULL, 0,  
                        &enable, sizeof(ULONG), &temp );
```

EXAMPLE#02: DISENABLE THE AUTO INPUT SCAN FUNCTION

```
LONG disable = 0x00;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_INPUT_AUTO_SCAN,  
                        NULL, 0,  
                        &disable, sizeof(ULONG), &temp );
```

### 3.6. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_VERTICAL\_MIRROR (244)

### 3.6. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_HORIZONTAL\_MIRROR (245)

This property is used to set mirror function. When mirror function is enabled, the vertical or horizontal video frame is inverted on display window. Same as deinterlacing, the property is used for display engine only.

SUPPORT VALUE: 0 ~ 1 - DISABLE ~ ENABLE

EXAMPLE#01: ENABLE THE VERTICAL MIRROR FUNCTION ON DISPLAY WINDOW

```
LONG enable = 0x01;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_VERTICAL_MIRROR, NULL, 0,  
                        &input, sizeof(LONG) );
```

EXAMPLE#02: ENABLE THE HORIZONTAL MIRROR FUNCTION ON DISPLAY WINDOW

```
LONG enable = 0x01;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_HORIZONTAL_MIRROR, NULL, 0,  
                        &input, sizeof(LONG) );
```

### 3.7. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_AUDIO\_VOLUME (251)

The property is used to control the current audio ADC's volume on the capture card.

SUPPORT VALUE: 0 (Mute): ~ 255 (Full)

**Note!! The property is enabled only by HDMI, DVI-D, and SDI input mode.**

EXAMPLE#01: TO SET THE AUDIO VOLUME AMPLITUDE.

```
LONG volume = 128;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_XET_ANALOG_AUDIO_VOLUME,  
                        NULL, 0,  
                        &volume, sizeof(ULONG), &temp );
```

### 3.8. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_QUEUE\_BUFFER\_SIZE (216)

The property allows 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]



### 3.9. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_FLEXIBLE\_RESOLUTION\_PATCH (220)

The property allows you to adjust the video's resolution from hardware board. If it is disabled, the output resolution is equal to input signal's resolution. If it is enabled, we will enable one auto scalar to output customized format. For example, input resolution is 704x480 and capture output pin's resolution is 352x240.

SUPPORT VALUE: 0 ~ 1 - DISABLE ~ ENABLE

EXAMPLE#01: TO ENABLE RESOLUTION SCALER.

```
LONG enable = 0x01;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_FLEXIBLE_RESOLUTION_PATCH, 0,  
                        &enable, sizeof(LONG) );
```

### 3.10. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_FLEXIBLE\_FPS\_PATCH (218)

The property allows you to control the output format from one video capture filter. It allows you to adjust the video's frame rate from driver side. If it is disabled, the output frame rate is equal to input signal's frame rate.

SUPPORT VALUE: 0 ~ 1 - DISABLE ~ ENABLE

EXAMPLE#01: TO ENABLE FRAMERATE SCALER.

```
LONG enable = 0x01;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_FLEXIBLE_FPS_PATCH, 0,  
                        &enable, sizeof(LONG) );
```

### 3.11. KSPROPERTY\_CUSTOM\_XET\_PREVIEW\_VIDEO\_STERAM\_POST\_RESOLUTION (350)

The property allows you to adjust current video resolution dynamically. The driver will re-allocate memory during changing video format on capture card running state.

SUPPORT VALUE:     RESOLUTION = (WIDTH << 16) | (HEIGHT << 0)

EXAMPLE#01: TO SET PREVIEW VIDEO RESOLUTION DYNAMICALLY.

```
LONG resolution = (WIDTH << 16) | (HEIGHT << 0);
```

[illegible]

### 3.12. KSPROPERTY\_CUSTOM\_XET\_PREVIEW\_VIDEO\_STREAM\_POST\_SKIP\_FRAMERATE (246)

The property (246) allows you to adjust current video skip frame rate dynamically. The range of the property is from 1 to 255. It is identical to the skip number of frame. For example, the value 1 will generate the preview frame rate, 15.000fps.

EXAMPLE#01: TO SET PREVIEW VIDEO SKIP FRAMERATE DYNAMICALLY.

The property (247) allows you to adjust current video average frame rate dynamically. The range of the property is from 1 to 85. To enable it, our driver will follow the setting value to output one average fps. For example, 9 mean 9.00fps.

EXAMPLE#01: TO SET PREVIEW VIDEO AVERAGE FRAMERATE DYNAMICALLY.

### 3.14. KSPROPERTY\_CUSTOM\_SET\_OSD\_LINE (920) (WRITE ONLY)

### 3.14. KSPROPERTY\_CUSTOM\_SET\_OSD\_TEXT\_STRING (921) (WRITE ONLY)

### 3.14. 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 )
```

### 3.15. KSPROPERTY\_CUSTOM\_XET\_GPIO\_DIRECTION (940)

### 3.15. KSPROPERTY\_CUSTOM\_XET\_GPIO\_DATA (941)

### 3.15. 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) );
```





- 3.16. **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_MODE** (407)
- 3.16. **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_QUALITY** (404)
- 3.16. **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_BITRATE** (403)
- 3.16. **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_MAX\_BITRATE** (409)
- 3.16. **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_MIN\_BITRATE** (410)

The property **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_MODE** allows you to get/set record mode on hardware-compressed capture device. There are 3 kinds of encoder mode: variable bitrate (VBR), constant bitrate (CBR) and average bitrate (ABR).

SUPPORT VALUE: 0: VBR ( FQP )  
                   1: CBR  
                   2: ABR

In the VBR mode, you choose the desired quality going from 0 (lowest quality) to 1000 (highest quality). The encoder tries to maintain the given quality for your video file. The main advantage is that you are able to specify the quality level that you want to reach, but the disadvantage is that the video size is unpredictable.

The property **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_QUALITY** allows you to set a suitable quality in VBR mode.

SUPPORT VALUE: 0 ~ 10000

EXAMPLE#01: TO SET VIDEO ENCODER QUALITY.

```
ULONG RecordMode = 0;
Encoder->m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,
                                KSPROPERTY_CUSTOM_XET_ENCODER_VIDEO_STERAM_RECORD_MODE, 0,
                                &RecordMode, sizeof(ULONG) );

ULONG Quality = 8000;
Encoder->m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,
                                KSPROPERTY_CUSTOM_XET_ENCODER_VIDEO_STERAM_RECORD_QUALITY, 0,
                                &Quality, sizeof(ULONG) );
```

In the CBR mode, the bitrate will be the same for the whole video file. The quality of your video is variable. The main advantage is that final video size can be accurately predicted, but the disadvantage is that the complex video parts will be a lower quality.

The property **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_BITRATE**



[illegible]

```
ULONG RecordMode = 2;
```

```
ULONG Bitrate = 1 * 1024 *1024;
```

[illegible]

### 3.17. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STARAM\_POST\_AVG\_FRAMERATE (422)

```
SUPPORT VALUE:  0: DISABLE
                1, 2, 3, 4, ... SKIP
```

```
ULONG FrameRate = 1;
```

The property (422) allows you to adjust encoding average frame rate dynamically. The range of the property is from 1 to 85. To enable it, our driver will follow the setting value to output one average fps. For example, 9 mean 9.00fps.

EXAMPLE#01: TO SET VIDEO ENCODING AVERAGE FRAMERATE DYNAMICALLY.

```
ULONG FrameRate = 9;
```

[illegible]

### 3.18. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_POST\_RESOLUTION (401)

The property allows you to adjust video encoding resolution dynamically. The driver will re-allocate memory during changing video format on capture card running state.

SUPPORT VALUE:     RESOLUTION = (WIDTH << 16) | (HEIGHT << 0)

EXAMPLE#01: TO SET VIDEO ENCODING RESOLUTION DYNAMICALLY.

```
ULONG RESOLUTION = (WIDTH << 16) | (HEIGHT << 0);
```

[illegible]

### 3.19. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_PROFILE (412)

The property allows you to adjust video encoder profile on hardware-compressed capture device. There are 2 kinds of profile value.

```
SUPPORT VALUE:    0: DEFAULT (MAIN)
                  1: BASELINE
                  2: MAIN
```

EXAMPLE#01: TO SET VIDEO ENCODER PROFILE.

```
ULONG PROFILE = 2;
```

[illegible]

### 3.20. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_ENTROPY (415)

The property allows you to adjust video encoder entropy on hardware-compressed capture device. There is only one kind of entropy value.

```
SUPPORT  VALUE:    0: DEFAULT  (CAVLC)
                  1: CAVLC
```

EXAMPLE#01: TO SET VIDEO ENCODER ENTROPY.

```
ULONG ENTROPY = 1;
```

[illegible]





### 3.22. KSPROPERTY\_CUSTOM\_SET\_ENCODER\_VIDEO\_STERAM\_FORCE\_KEY\_FRAME (406)

The property allows you to set video encoder force key frame on hardware-compressed capture device. The property puts key frame on the next frame or forcing a key frame at specified timestamp.

SUPPORT VALUE: 1: FROCE KEYFRAME

EXAMPLE#01: TO SET VIDEO ENCODER FORCE KEYFRAME.

```
ULONG FROCE KEYFRAME = 1;
```

[illegible]

### 3.23. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_QPSTEP (408)

The property allows you to set video encoder QPStep on hardware-compressed capture device. For the QPStep property, it is used in CBR and ABR mode. When the QPStep is 1, the encoder will do the fps-checking during every frame. Here, the output bitrate of encoder will be very close to your target bitrate. When the QPStep is set to 30, the encoder will do the fps-checking per 30 frames. It will offer bigger tolerance on bitrate controlling, but it will cause the output bitrate is not accurate.

SUPPORT VALUE: 1 ~ 255 ( DEFAULT : 1 )

EXAMPLE#01: TO SET VIDEO ENCODER QPSTEP.

```
ULONG QPSTEP = 1;
```

[illegible]

### 3.24. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_FRAME\_QUEUE\_LENGTH (424)

The property allows you to specify the number of the rendered video frame in the queue buffer for video encoded stream. By the default, the queue size of the corresponding video encoded stream is set 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 if the total buffer sizes you want to set exceed the system capabilities.

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

EXAMPLE#01: TO SET THE VIDEO ENCODER QUEUE SIZE TO 16 FRAMES.

```
ULONG QUEUE_FRAMES = 16;
```

[illegible]

### 3.25. 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.