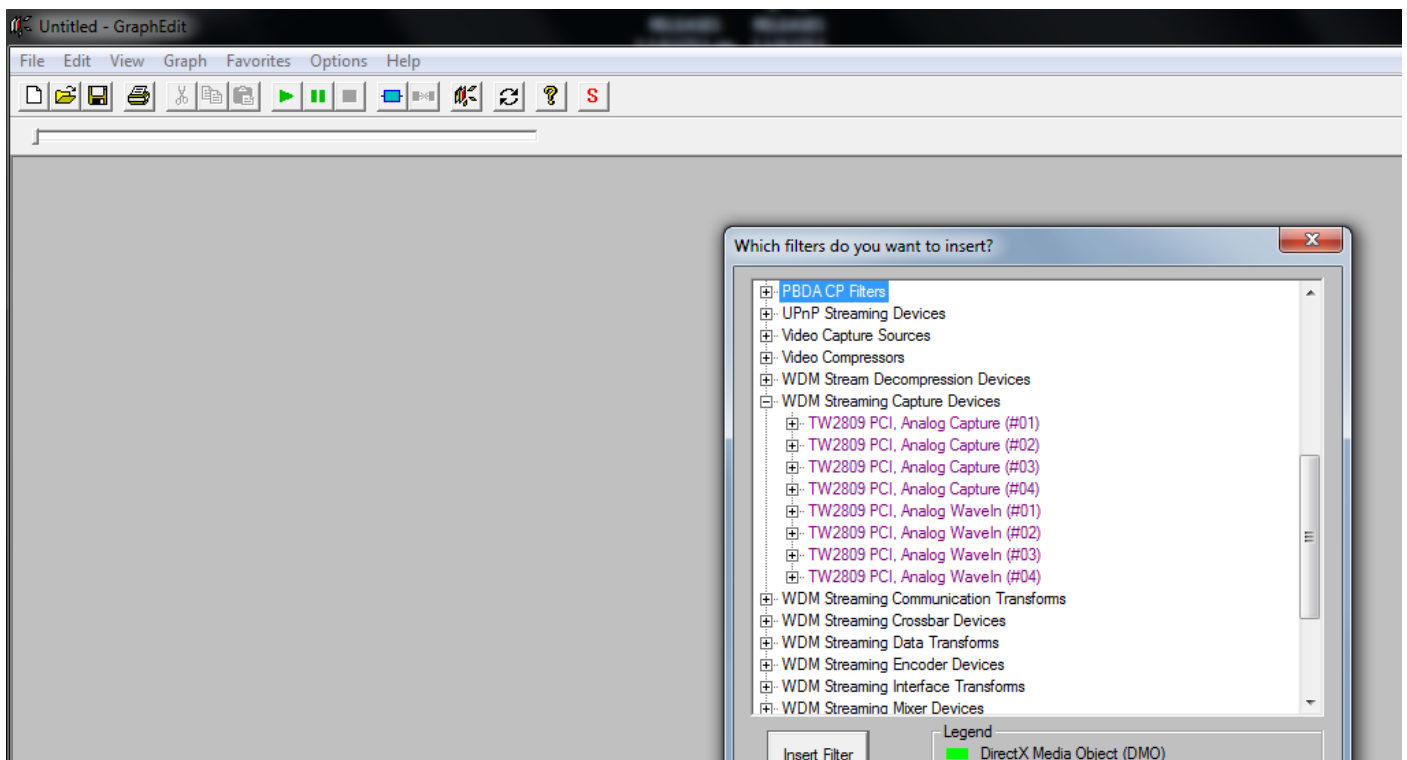


SC590 DirectShow Software Programming Guide

Customer uses DirectShow to develop software can bypass our SDK to access TW2809 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.

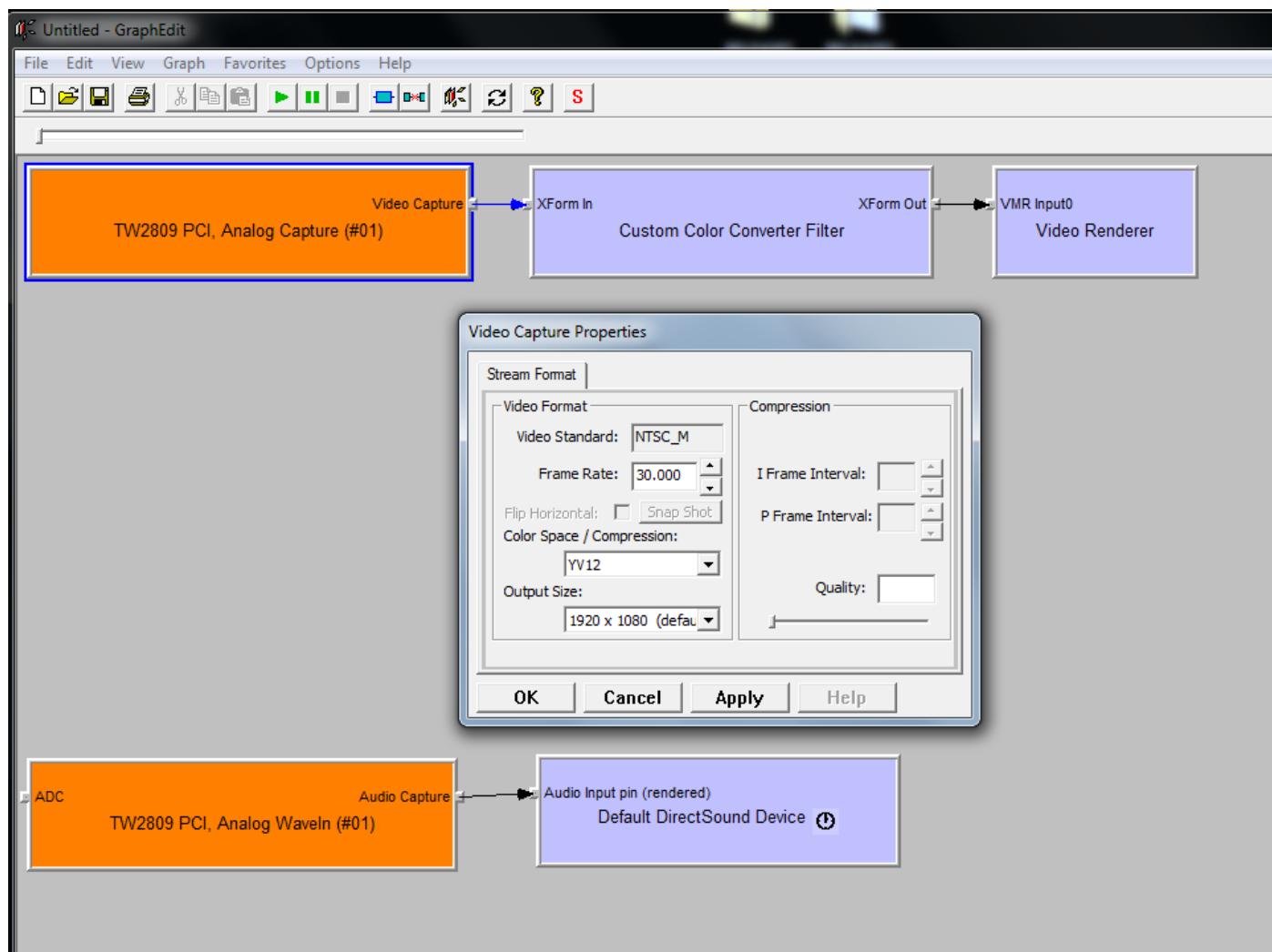
All filter names are "TW2809 PCI, Analog Capture (#XX)" for video, and "TW2809 PCI, Analog WaveIn (#XX)" for audio. They are registered at "WDM Streaming Captures Devices" category.



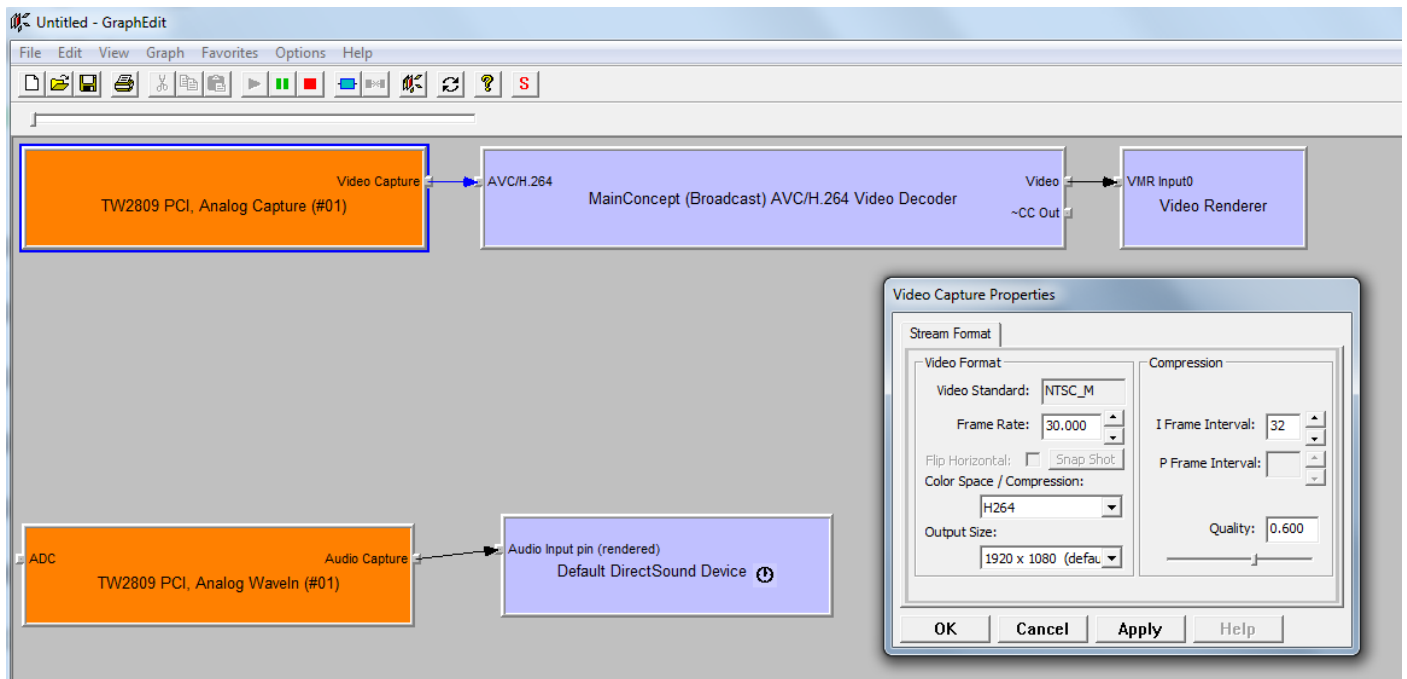
TW2809 is a hardware compression chip, it can output YV12, and two H.264 streams at the same time. We use H.264 to stand for main stream, X.264 for sub stream.

```
#define MEDIASUBTYPE_H264 0x34363248, 0x0000, 0x0010, 0x80, 0x00, 0x00, 0xAA, 0x00, 0x38, 0x9B, 0x71
#define MEDIASUBTYPE_X264 0x34363258, 0x0000, 0x0010, 0x80, 0x00, 0x00, 0xAA, 0x00, 0x38, 0x9B, 0x71
```

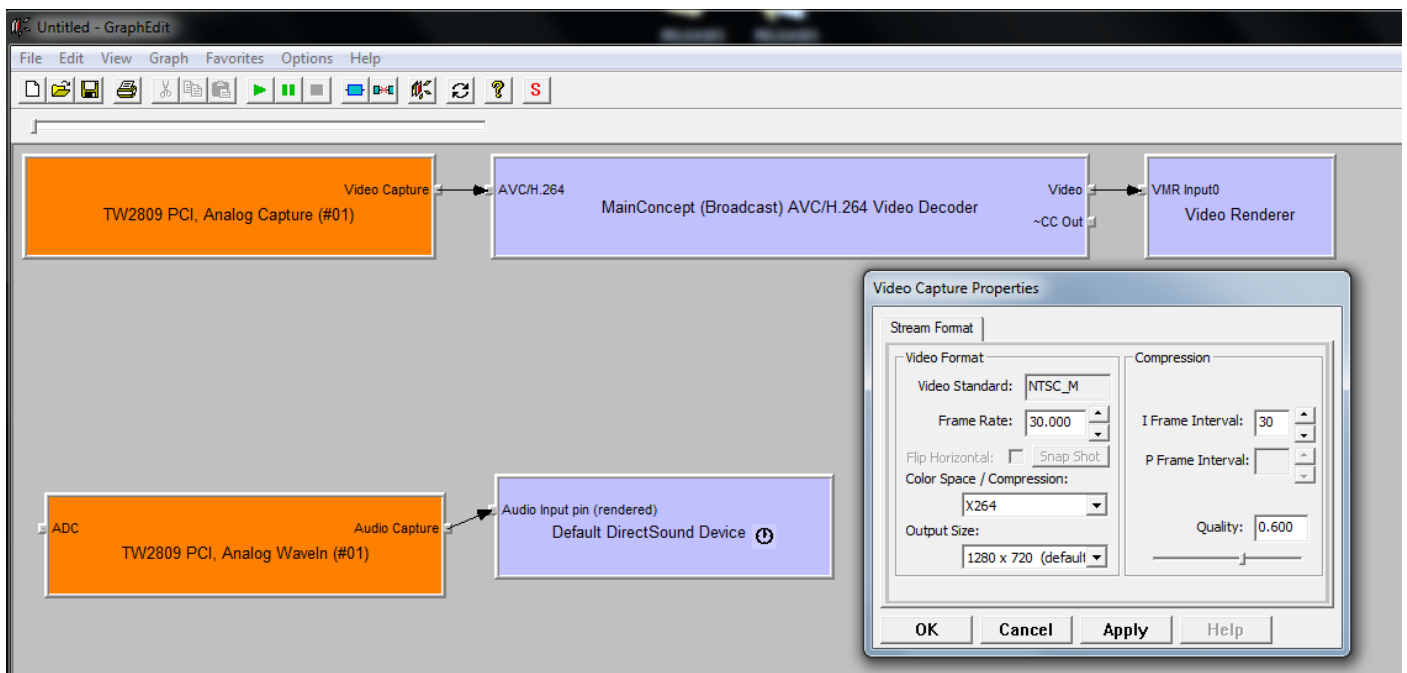
For the preview output, here, the video format is YV12 and audio format is PCM. The connection of filters is as:



Main stream connection is as:

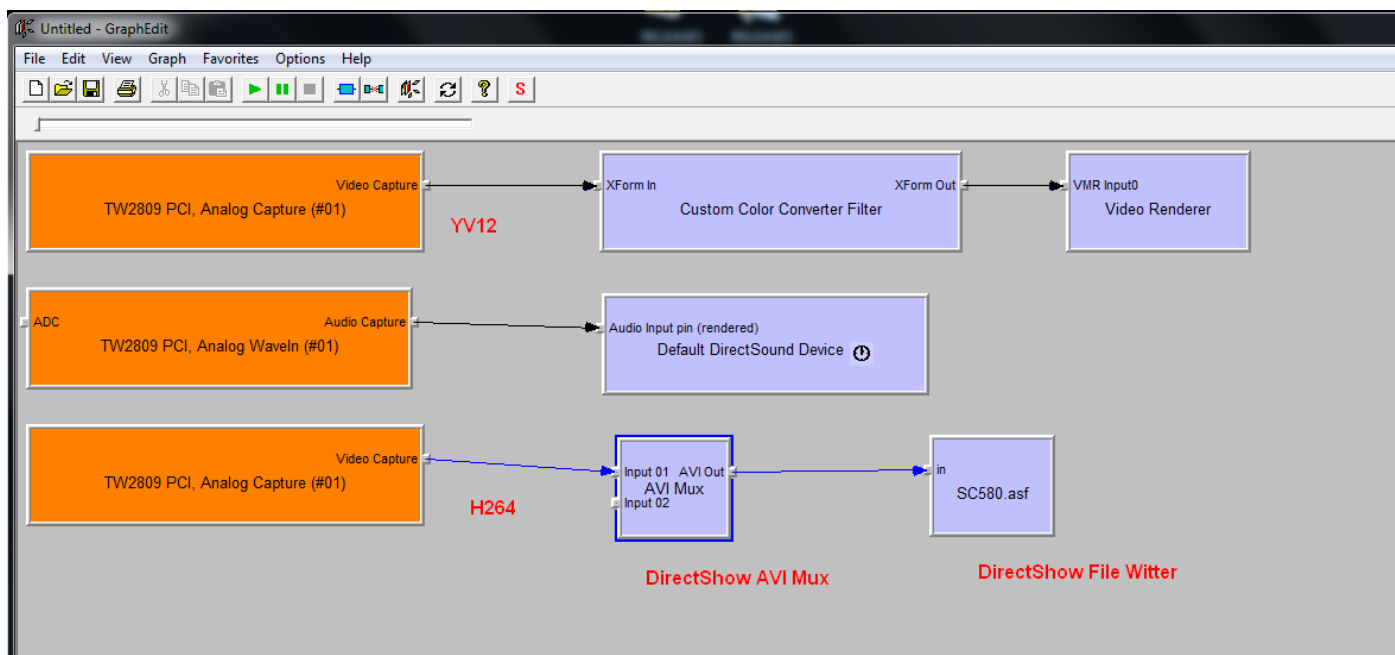


Sub stream connection is as:



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, YV12 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 1920X1080 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', 'V', '1', '2');
((VIDEOINFOHEADER *) (pmt->pbFormat))->bmiHeader.biHeight = 1920;
((VIDEOINFOHEADER *) (pmt->pbFormat))->bmiHeader.biWidth = 1080;
((VIDEOINFOHEADER *) (pmt->pbFormat))->bmiHeader.biBitCount = 12;
((VIDEOINFOHEADER *) (pmt->pbFormat))->bmiHeader.biSizeImage = 1920 * 1080 * 12 / 8;
((VIDEOINFOHEADER *) (pmt->pbFormat))->AvgTimePerFrame = (ULONG) (INT) (10000000.0 / 30.000);
((VIDEOINFOHEADER *) (pmt->pbFormat))->dwBitRate = (ULONG) (INT) (1920 * 1080 * 12 * 30.000);
m_pAMStreamConfig->SetFormat( pmt );
DeleteMediaType( pmt );
```

EXAMPLE#02: SET AUDIO OUTPUT FORAMT TO MONO, 16BITS, AND 48000HZ.

```
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) (48000);
((WAVEFORMATEX *) (pmt->pbFormat))->nBlockAlign = (USHORT) (1 * 16 / 8);
((WAVEFORMATEX *) (pmt->pbFormat))->nAvgBytesPerSec = (ULONG) (1 * 16 * 48000 / 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, 0x1E };
```

All custom properties are defined as below:

```
typedef enum {  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_FRAME_RATE           = 208,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_RESOLUTION           = 210,  
    KSPROPERTY_CUSTOM_XET_ANALOG_AUDIO_SAMPLE_FREQUENCY      = 253,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_QUEUE_BUFFER_SIZE     = 216,  
    KSPROPERTY_CUSTOM_XET_ANALOG_AUDIO_INPUT                  = 255,  
    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_XET_ANALOG_AUDIO_INPUT (255)

The property allows you to get/change current audio input source. You can select audio from embedded audio data or from extra line-in cable.

SUPPORT VALUE: 0: Embedded Audio
1: Line In

Note!! The property is enabled only by SDI input mode.

EXAMPLE#01: CHANGE TO EMBEDDED AUDIO INPUT.

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

EXAMPLE#02: CHANGE TO LINE-IN INPUT.

```
ULONG input = 1;
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,
                        KSPROPERTY_CUSTOM_XET_ANALOG_AUDIO_INPUT,
                        NULL, 0,
                        &input, sizeof(ULONG) );
```

EXAMPLE#03: GET CURRENT AUDIO INPUT SOURCE.

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

3.2. KSPROPERTY_CUSTOM_GET_ANALOG_VIDEO_RESOLUTION (210) (READ ONLY)

3.3. KSPROPERTY_CUSTOM_GET_ANALOG_VIDEO_FRAME_RATE (208) (READ ONLY)

Our driver can auto detect video format and can report the current input format to your software. The both properties can help to obtain current video format's resolution and frame rate. Some supported formats are described in the table. The format table keeps on increasing into the new driver. Please check our sales to obtain the latest one.

FORMAT	RESOLUTION	FRAME RATE
1920×1080p@60fps	0x07800438	60
1920×1080p@50fps	0x07800438	50
1920×1080p@30fps	0x07800438	30
1920×1080p@25fps	0x07800438	25
1920×1080p@24fps	0x07800438	24
1920×1080i@60fps	0x0780021C	60
1920×1080i@50fps	0x0780021C	50
1280×720P@60fps	0x050002D0	60
1280×720P@50fps	0x050002D0	50
1280×720P@30fps	0x050002D0	30
1280×720P@25fps	0x050002D0	25
1280×720P@24fps	0x050002D0	24
720×480P@60fps	0x02D001E0	60
720×576P@50fps	0x02D00240	50
720×480i@60fps	0x02D000F0	60
720×576i@50fps	0x02D00120	50
720×240P@60fps	0x05A001E0	60
720×288P@50fps	0x05A00240	50
1440×900p@60fps	0x05A00384	60
1280×1024p@60fps	0x05000400	60
1280×960p@60fps	0x050003C0	60
1280×800p@60fps	0x05000320	60
1280×768p@60fps	0x05000300	60
1024×768p@60fps	0x04000300	60
800×600p@60fps	0x03200258	60
640×480p@60fps	0x028001E0	60
640×400p@60fps	0x02800190	60
640×384p@60fps	0x02800180	60

*₁ THE FORMAT IS USED BY SONY PS1/PS2 GAME MACHINE.

*₂ THE FORMAT IS USED BY MICROSOFT XBOX360 GAME MACHINE (640×480p@60fps).

*₃ THE FORMAT IS USED BY NEC IPC MACHINE (640×400p@56.4fps).

Here, the resolution property can be described as below:

RESOLUTION = (WIDTH << 16) | (HEIGHT << 0)

Note!! Developer should design one polling operation in one background thread to obtain/update current input format.

EXAMPLE#01: GET CURRENT VIDEO FORMAT.

```
ULONG resolution = 0;
```

```
ULONG framerate = 0;
```

```
m_pKsPropertySet->Get( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_GET_ANALOG_VIDEO_RESOLUTION,  
                        NULL, 0,  
                        &resolution, sizeof(ULONG), &temp );  
  
m_pKsPropertySet->Get( PROPSETID_AMEBDAD_CUSTOM_PROP,  
                        KSPROPERTY_CUSTOM_GET_ANALOG_VIDEO_FRAME_RATE,  
                        NULL, 0,  
                        &framerate, sizeof(ULONG), &temp );
```

3.4. KSPROPERTY_CUSTOM_XET_ANALOG_AUDIO_SAMPLE_FREQUENCY (253) (READ ONLY)

The driver also can auto detect current audio format and can report it to upper software. Currently, all audio formats are mono and 16bits quality. The only difference is their sample frequency, so you can use the property to obtain the input's sample frequency.

Note!! Developer should design one polling operation in one background thread to obtain/update current input format.

SUPPORT VALUE: 48000 - MONO / 16BITS / 48000HZ
44100 - MONO / 16BITS / 44100HZ
32000 - MONO / 16BITS / 32000HZ

EXAMPLE#01: GET CURRENT AUDIO SAMPLE FREQUENCY.

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

3.5. KSPROPERTY_CUSTOM_GET_ANALOG_VIDEO_MACROVISION (202) (READ ONLY)

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

Note!! To protect the content license, all behaviors in software porting should be complied with HDCP rules. Detect in any registered content of HDCP or MarcoVision, please disable the recording function in software.

SUPPORT VALUE: 0, 1 - NO ~ YES

EXAMPLE#01: GET HDCP PROTECT STATUS.

```
ULONG HDCP = 0;
```

[illegible]

3.6. KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION (940)

3.6. KSPROPERTY_CUSTOM_XET_GPIO_DATA (941)

3.6. KSPROPERTY_CUSTOM_GET_GPIO_SUPPORT (942) (READ ONLY)

The property allows you to access SAA7160'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) );
```


3.7. KSPROPERTY_CUSTOM_SET_OSD_LINE (920) (WRITE ONLY)
3.7. KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_1 (921) (WRITE ONLY)
3.7. KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_2 (922) (WRITE ONLY)
3.7. KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_3 (923) (WRITE ONLY)
3.7. KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_4 (924) (WRITE ONLY)

The properties allow you to change TW2809's OSD context. 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. The max string length is 16 characters.

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 )
```

3.8. 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]

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 )
```


3.9 Video Encoder Property:

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

```
static const GUID GUID_KPS_TW2809 = { 0xD1E5209F, 0x68FD, 0x4529, 0xBE, 0xE0, 0x5E, 0x7A, 0x1F, 0x47, 0x92, 0x1E };
```

```
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_TW2809, 407, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_TW2809, 403, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_TW2809, 401, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_TW2809, 402, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_TW2809, 422, 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_TW2809, 407, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000004 ) { // BIT.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_TW2809, 403, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000008 ) { // POST.RESOLUTION
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_TW2809, 401, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000009 ) { // POST.SKIP.FRAME.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_TW2809, 402, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000D ) { // POST.AVG.FRAME.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_TW2809, 422, 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.