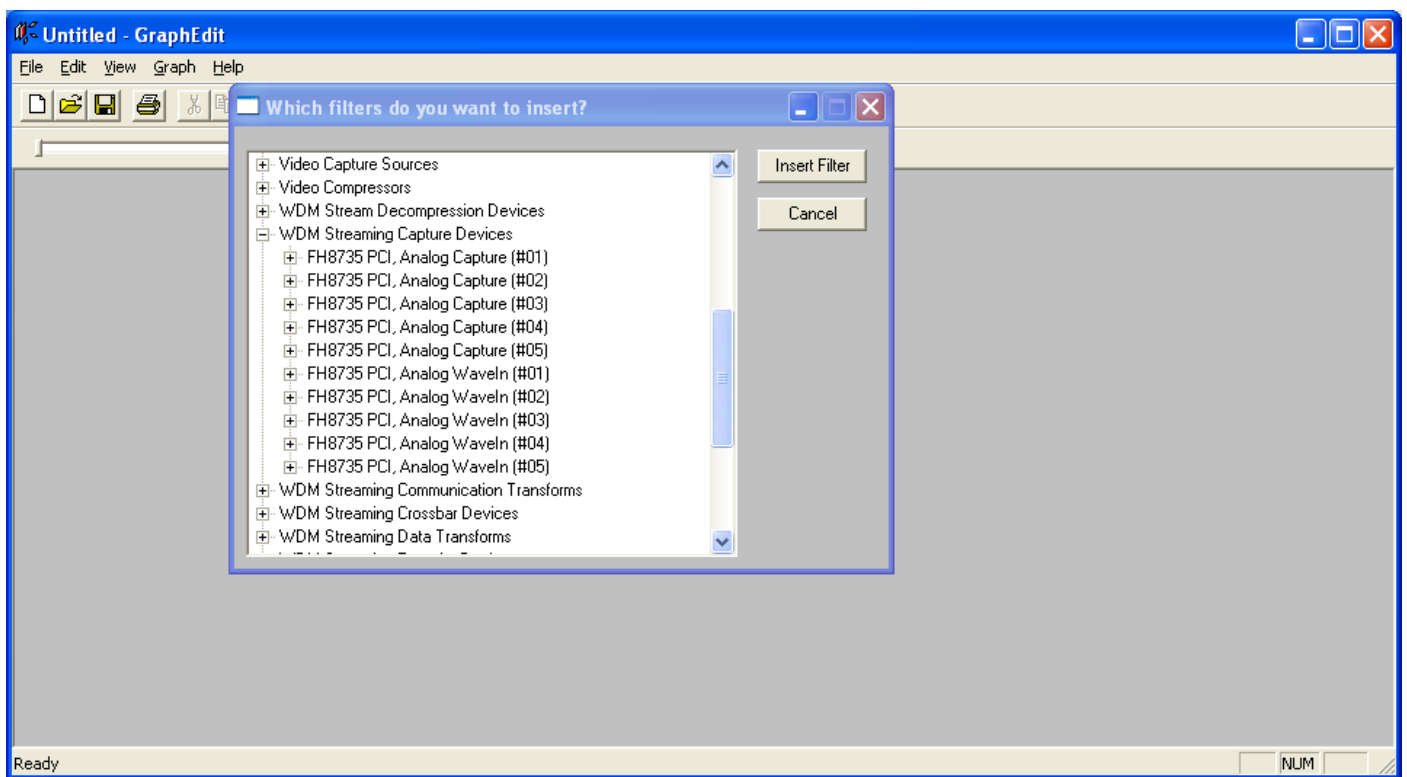


SC3A0 DirectShow Software Programming Guide

Customer uses DirectShow to develop software can bypass our SDK to access FH8735 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.

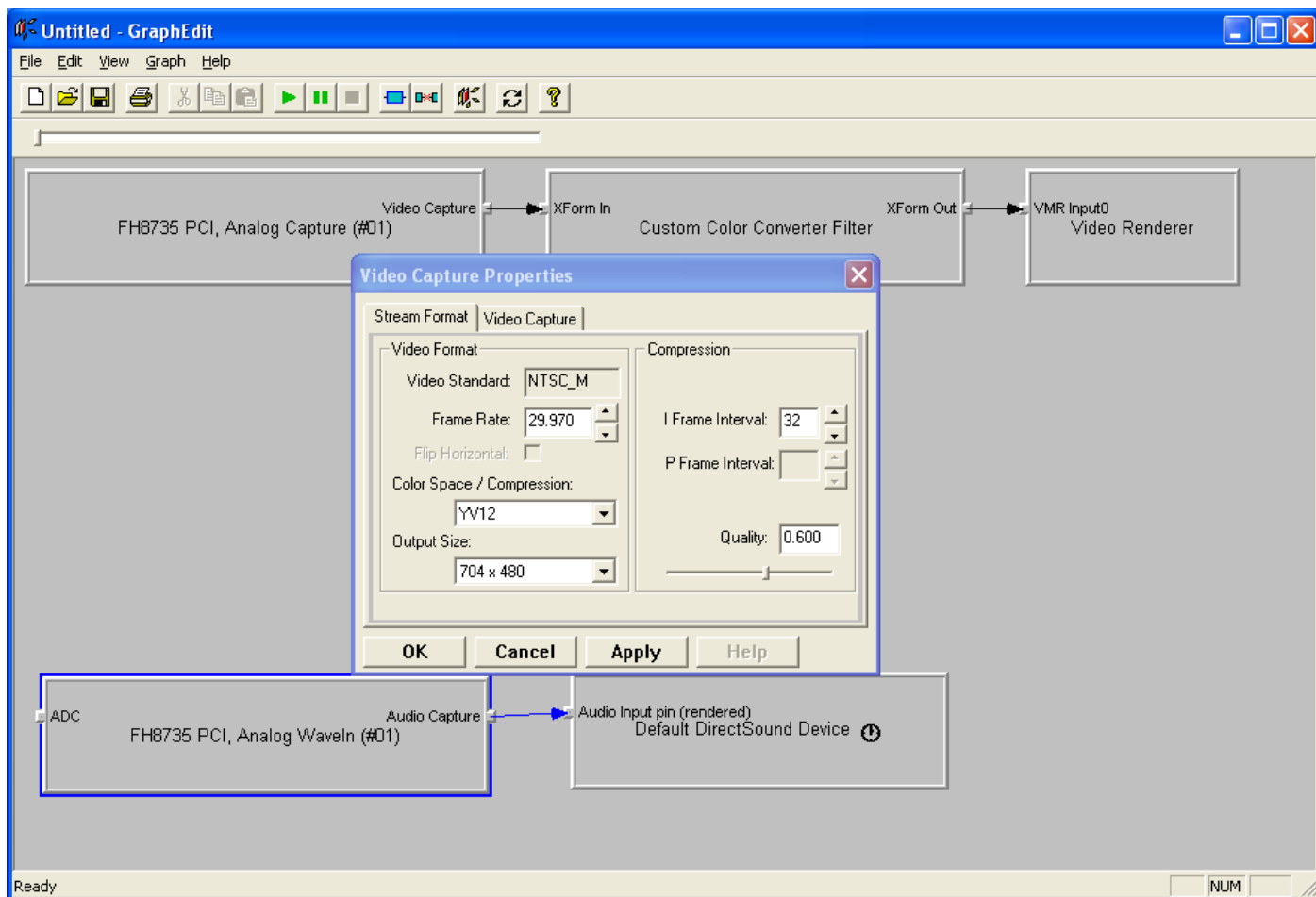
All filter names are "FH8735 PCI, Analog Capture (#XX)" for video, and "FH8735 PCI, Analog WaveIn (#XX)" for audio. They are registered at "WDM Streaming Captures Devices" category.



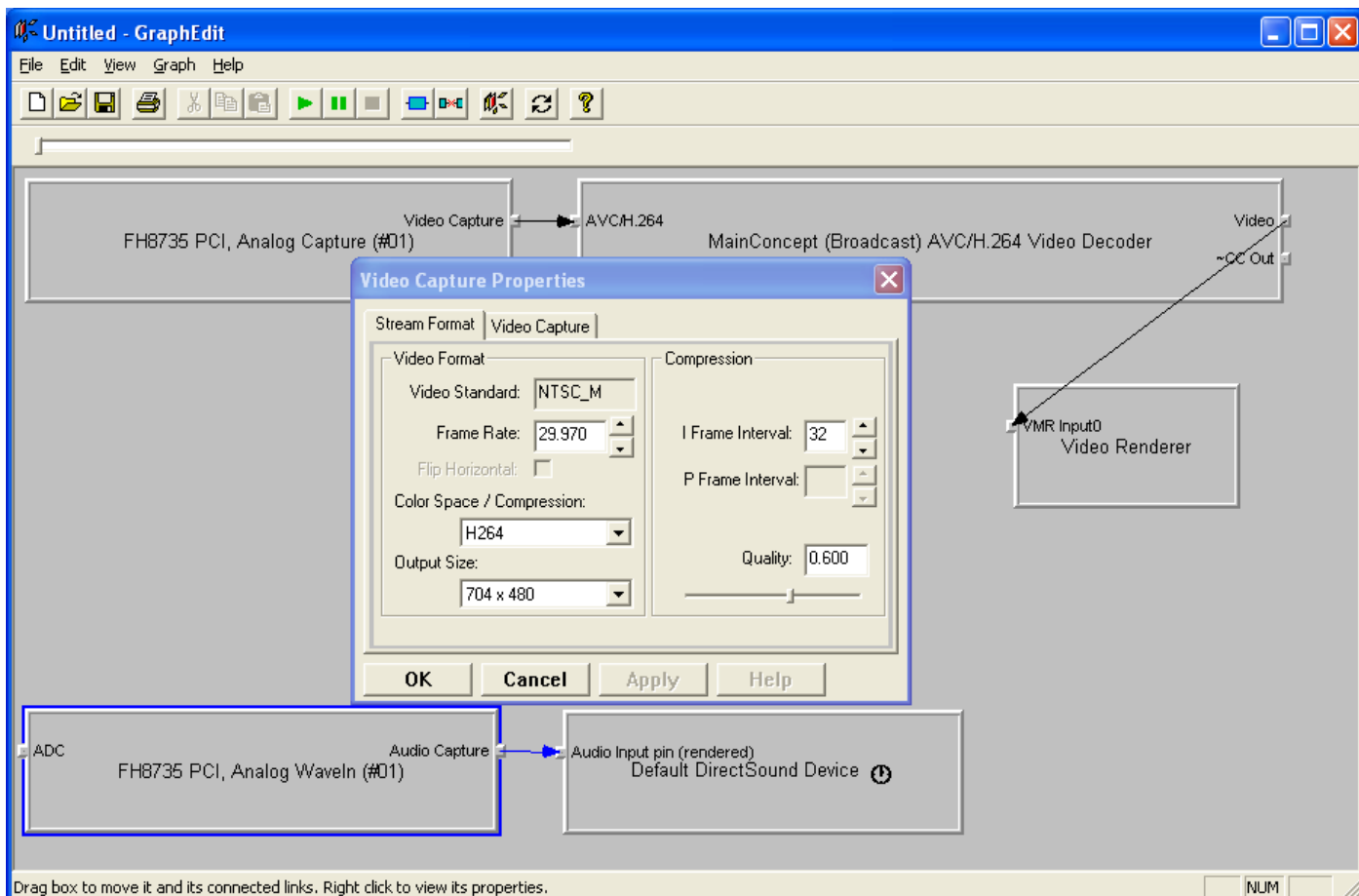
FH8735 is a hardware compression chip, it can output YV12, and two H.264 streams at the same time. We use H.264 to stand for main stream, X.264 for sub stream.

```
#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
```

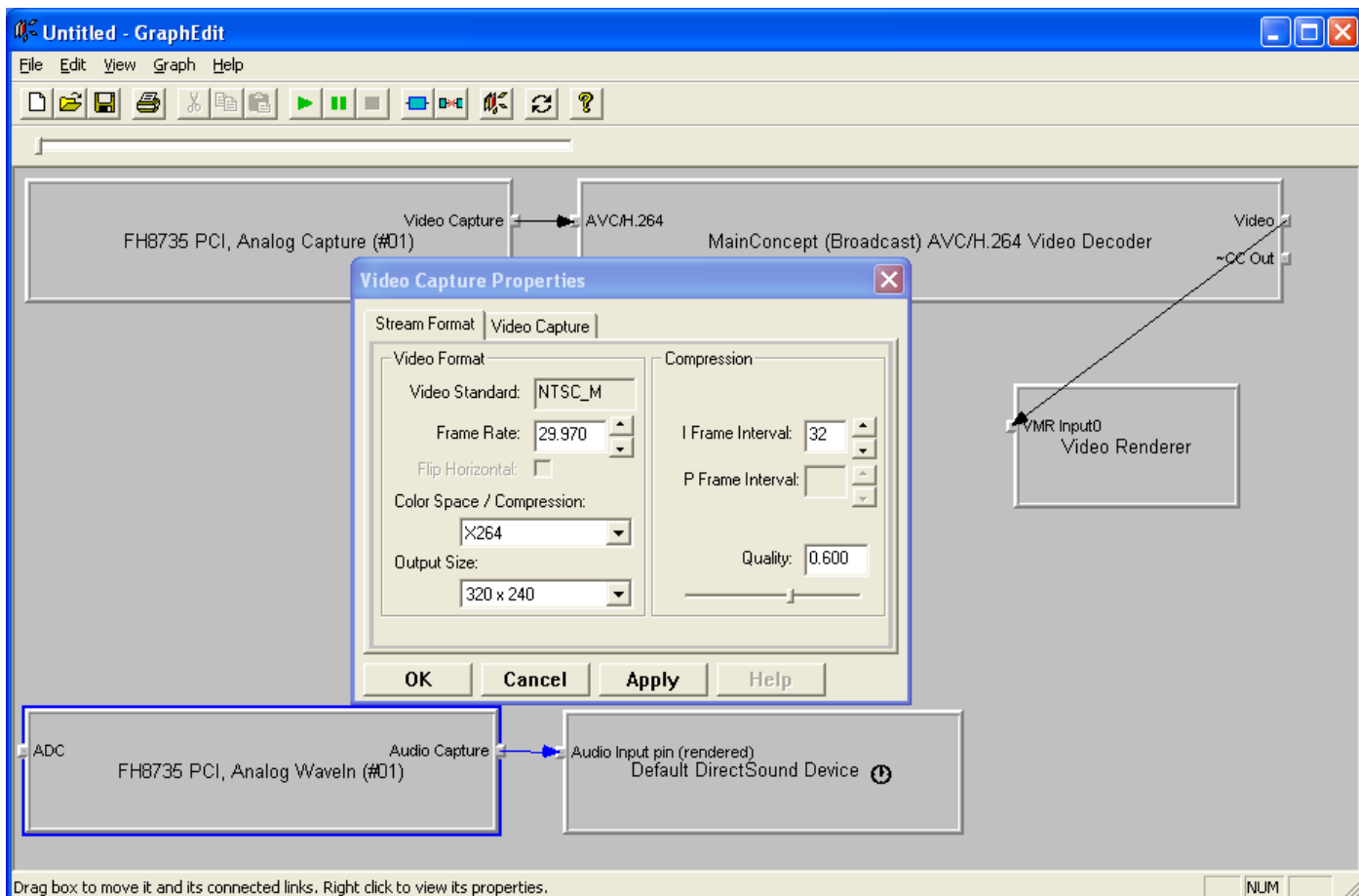
For the preview output, here, the video format is YV12 and audio format is PCM. The connection of filters is as:



Main stream connection is as:

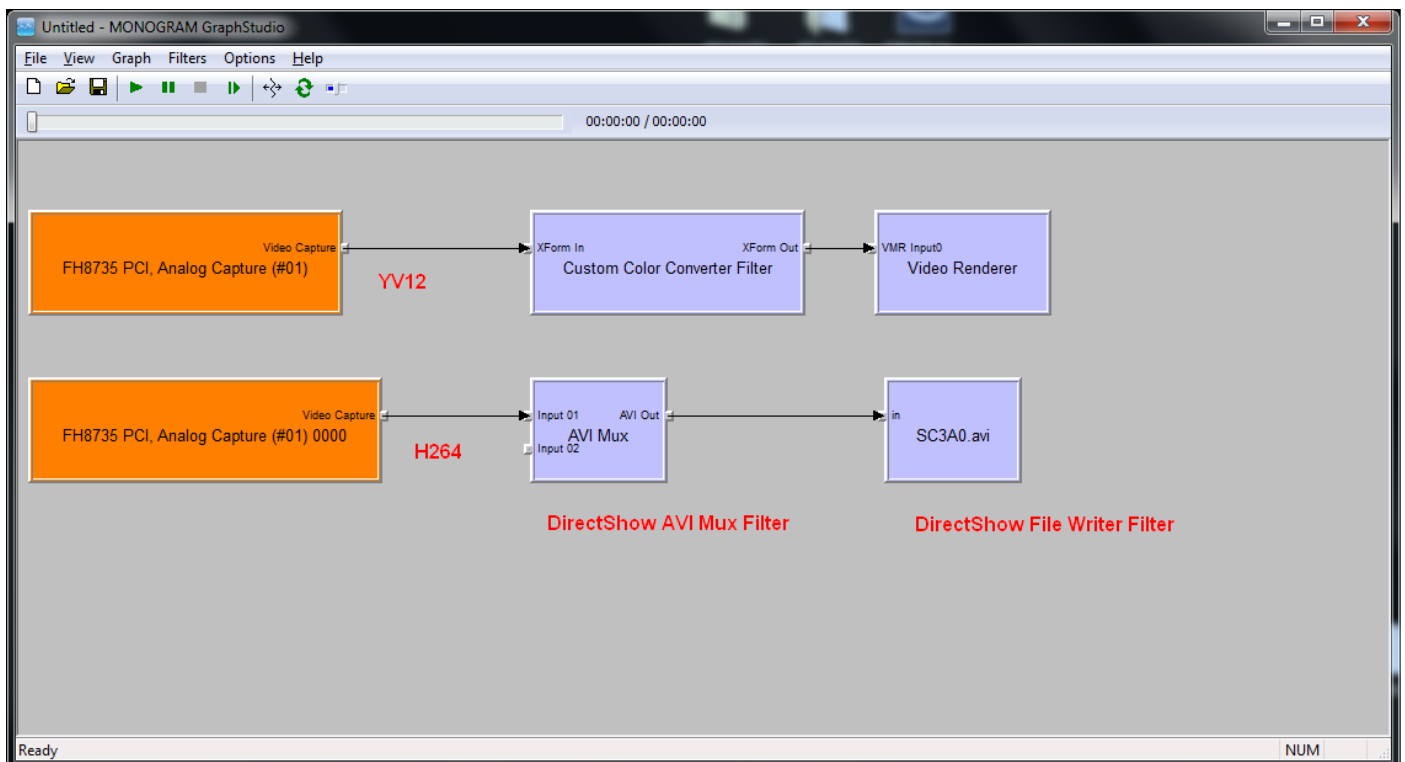


Sub stream connection is as:



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 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', 'V', '1', '2');
((VIDEOINFOHEADER *) (pmt->pbFormat))->bmiHeader.biHeight = 704;
((VIDEOINFOHEADER *) (pmt->pbFormat))->bmiHeader.biWidth = 480;
((VIDEOINFOHEADER *) (pmt->pbFormat))->bmiHeader.biBitCount = 12;
((VIDEOINFOHEADER *) (pmt->pbFormat))->bmiHeader.biSizeImage = 704 * 480 * 12 / 8;
((VIDEOINFOHEADER *) (pmt->pbFormat))->AvgTimePerFrame = (ULONG)(INT)(10000000.0 / 30.000);
((VIDEOINFOHEADER *) (pmt->pbFormat))->dwBitRate = (ULONG)(INT)(704 * 480 * 12 * 30.000);
m_pAMStreamConfig->SetFormat( pmt );
DeleteMediaType( pmt );
```

EXAMPLE#02: SET AUDIO OUTPUT FORAMT TO MONO, 16BITS, AND 16000HZ.

```
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)(16000);
((WAVEFORMATEX *) (pmt->pbFormat))->nBlockAlign = (USHORT)(1 * 16 / 8);
((WAVEFORMATEX *) (pmt->pbFormat))->nAvgBytesPerSec = (ULONG)(1 * 16 * 16000 / 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, 0x1A };
```

All custom properties are defined as below:

```
typedef enum {  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_DEINTERLACE_TYPE    = 200,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_DENOISE_TYPE        = 217,  
    KSPROPERTY_CUSTOM_XET_REGION_MASK_PARAMS                = 890,  
    KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_1                = 921,  
    KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_2                = 922,  
    KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_3                = 923,  
    KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_4                = 924,  
    KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PARAMS               = 970,  
    KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PATH_1               = 971,  
    KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PATH_2               = 972,  
    KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PATH_3               = 973,  
    KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PATH_4               = 974,  
    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_VIDEO_DEINTERLACE_TYPE (200)

FH8735 offers one hardware-based deinterlacer on chip. The property will allow you to enable/disable this function. Currently, we offer 5 levels deinterlace methods to your application. The value 0 will turn off it.

SUPPORT VALUE: 0 ~ 8 - OFF ~ LEVEL 8

EXAMPLE#01: TO TURN OFF HARDWARE DEINTERLACE FUNCTION.

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

EXAMPLE#02: TO TURN ON HARDWARE DEINTERLACE FUNCTION AT LEVEL 5.

```
ULONG input = 5;
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,
                        KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_DEINTERLACE_TYPE,
                        NULL, 0,
                        &input, sizeof(ULONG) );
```


3.2. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_DENOISE_TYPE (217)

FH8735 offers one hardware-based de-noise function on chip. The property will allow you to access it. You can use this property to enable/disable this function. Currently, we offer 3 levels de-noise methods to your application. The value 0 will turn off it.

SUPPORT VALUE: 0 ~ 3 - OFF ~ LEVEL 3

EXAMPLE#01: TO TURN OFF HARDWARE DENOISE FUNCTION.

```
ULONG input = 0;
```

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

EXAMPLE#02: TO TURN ON HARDWARE DENOISE FUNCTION AT LEVEL 3.

```
ULONG input = 3;
```

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

3.4. KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION (940)

3.4. KSPROPERTY_CUSTOM_XET_GPIO_DATA (941)

3.4. 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.

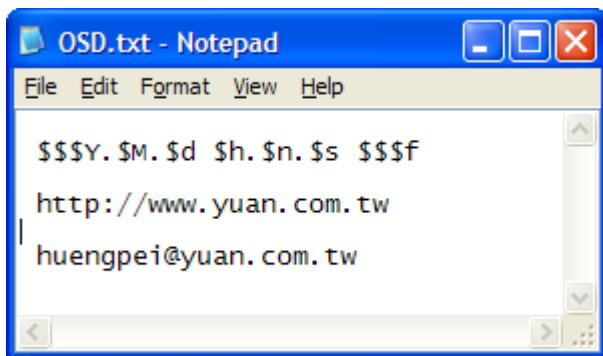
```
ULONG input = 0xFFFF;
ULONG data = 0xFFFF;
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,
                        KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION, NULL, 0,
                        &input, sizeof(ULONG) );
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP,
                        KSPROPERTY_CUSTOM_XET_GPIO_DATA,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


3.5. KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_1 (921) (WRITE ONLY)
3.5. KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_2 (922) (WRITE ONLY)
3.5. KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_3 (923) (WRITE ONLY)
3.5. KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_4 (924) (WRITE ONLY)
3.5. KSPROPERTY_CUSTOM_SET_OSD_COLOR (929) (WRITE ONLY)

The properties allow you to change FH8735's text OSD context. The property KSPROPERTY_CUSTOM_SET_OSD_COLOR allows you to change string's color. Here, there are 6 kinds of colors can be selected by you. Also, you can modify it dynamically during recording.

SUPPORT VALUE: 0 ~ 5 - COLOR#0 ~ COLOR#5

The property KSPROPERTY_SET_OSD_TEXT_STRING helps you to change string context on screen. FH8735 allows software to load one text file into its board memory. The format of test file is described as this example below:



SUPPORT FORMAT:

\$\$\$Y: YEAR

\$M: MONTH

\$d: DAY

\$h: HOUR

\$n: MINUTE

\$s: SECOND

\$X: WEEKDAY

\$W: WEEK

\$\$\$f: FRAME NUMBER

Note!! When you set the custom string into device, our driver will auto disable default time OSD.

Note!! The length of file path cannot over 64 bytes, so the max characters length is 63 only.

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) );
```

EXAMPLE#02: TO LOAD TEXT STRING FROM FILE, OSD.TXT.

```
CHAR path[] = "C:\\WINDOWS\\FH8735\\OSD.TXT";
CHAR psz[ 256 ];
memset( psz, 0x00, 64 );
sprintf( psz, "%s", path );
psz[ 63 ] = 0x00;
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 )
```

3.6. KSPROPERTY_CUSTOM_SET_PICTURE_PARAMS (970) (WRITE ONLY)
3.6. KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PATH_1 (971) (WRITE ONLY)
3.6. KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PATH_2 (972) (WRITE ONLY)
3.6. KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PATH_3 (973) (WRITE ONLY)
3.6. KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PATH_4 (974) (WRITE ONLY)

The properties allow you to change FH8735's graphic OSD context. The property KSPROPERTY_CUSTOM_SET_PICTURE_PARAMS allows you to change picture's position and transparent. Here, the parameters are described by one structure as below:

```
ULONG params[ 4 ] = {  
    (PICTURE.INDEX),  
    (PICTURE.START.X),  
    (PICTURE.START.Y),  
    (PICTURE.TRANSPARENT)  
};
```

Here, PICTURE.INDEX is 0 or 1. FH8735 supports 2 pictures at the same time. The PICTURE.START.X and PICTURE.START.Y is the left and the top position. The PICTURE.TRANSPARENT is from 0 to 255.

The property KSPROPERTY_SET_OSD_PICTURE_PATH describes the path of file. FH8735 allows software to load one 8 bits (256 colors) BMP file into its board memory.

Note!! The length of file path cannot over 64 bytes, so the max characters length is 63 only.

EXAMPLE#01: TO LOAD ONE BITMAP.

```
ULONG params[ 4 ] = { 0, 0, 0, 255 };  
CHAR path[] = "C:\\\\WINDOWS\\\\FH8735\\\\LOGO.BMP";  
CHAR psz[ 256 ];  
memset( psz, 0x00, 64 );  
sprintf( psz, "%s", path );  
psz[ 63 ] = 0x00;  
  
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 970, NULL, 0, params, 16 )  
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 971, NULL, 0, psz + 0, 16 )  
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 972, NULL, 0, psz + 16, 16 )  
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 973, NULL, 0, psz + 32, 16 )  
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 974, NULL, 0, psz + 48, 16 )
```

3.7. KSPROPERTY_CUSTOM_XET_REGION_MASK_PARAMS (890)

The properties allow you to enable FH8735's region mask function. FH8735 owns 2 region masks per channel. Every mask can be described by 4 ULONG params.

```
ULONG params[ 4 ] = {  
    (REGION.INDEX),  
    (REGION.START.X << 16) |  
    (REGION.START.Y << 0),  
    (REGION.WIDTH),  
    (REGION.HEIGHT),  
};
```

Here, REGION.INDEX is 0 or 1. In FH8735, the image is divided into many small blocks. Every block is 16 x 16 pixels. For example, REGION.START.X = 1 and REGION.WIDTH = 10. The start position is pixel 16 and the width is 160 pixels.

EXAMPLE#01: SET REGION MASK#0.

```
ULONG params[ 4 ] = { 0, (0 << 16) | (0 << 0), 10, 10 };
```

```
AMESDK_SET_CUSTOM_PROPERTY_EX( hDev, 890, params );
```

EXAMPLE#01: SET REGION MASK#1.

```
ULONG params[ 4 ] = { 1, (5 << 16) | (5 << 0), 20, 20 };
```

```
AMESDK_SET_CUSTOM_PROPERTY_EX( hDev, 890, params );
```

3.8 Video Encoder Property:

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

```
static const GUID GUID_KPS_FH8735 = { 0xD1E5209F, 0x68FD, 0x4529, 0xBE, 0xE0, 0x5E, 0x7A, 0x1F, 0x47, 0x92, 0x1A };
```

```
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_FH8735, 407, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_FH8735, 403, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_FH8735, 401, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_FH8735, 402, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_FH8735, 422, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_FH8735, 411, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_FH8735, 412, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_FH8735, 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_FH8735, 407, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000004 ) { // BIT.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_FH8735, 403, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000008 ) { // POST.RESOLUTION
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_FH8735, 401, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000009 ) { // POST.SKIP.FRAME.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_FH8735, 402, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000D ) { // POST.AVG.FRAME.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_FH8735, 422, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000A ) { // B.FRAME
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_FH8735, 411, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000B ) { // PROFILE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_FH8735, 412, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000C ) { // ASPECT.RATIO
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_FH8735, 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.