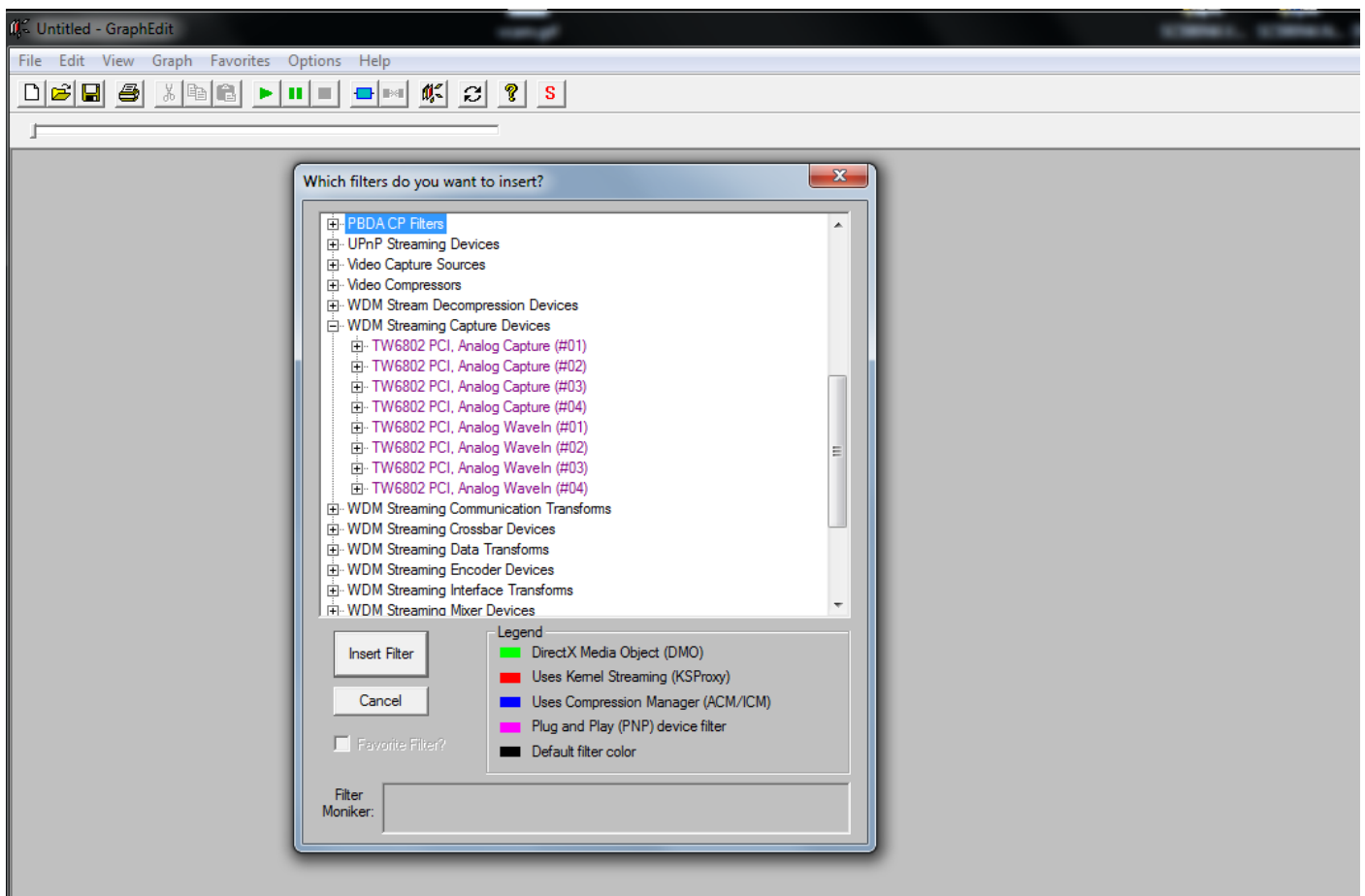


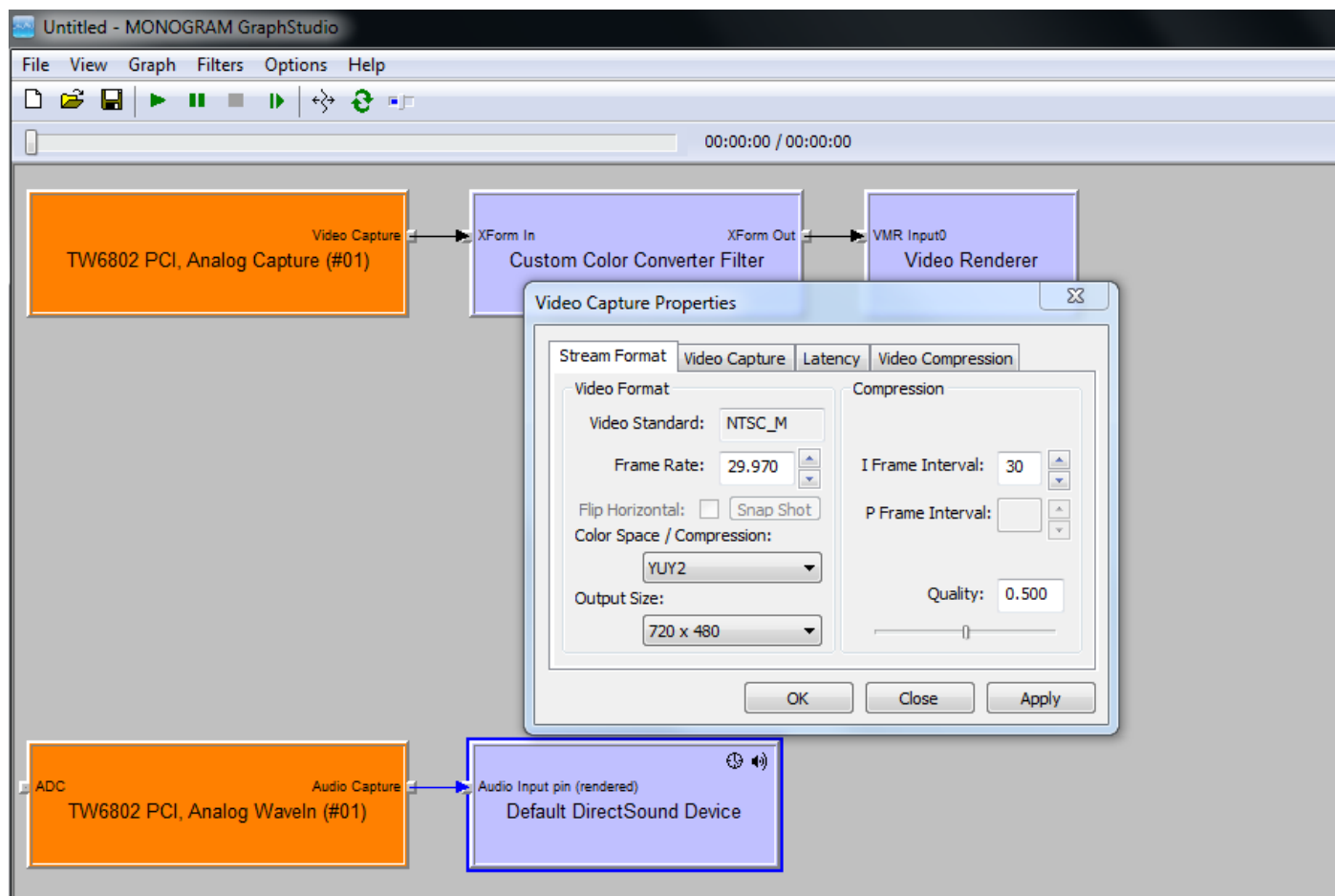
SC230 DirectShow Software Programming Guide

Customer uses DirectShow to develop software can bypass our SDK to access TW6802 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.

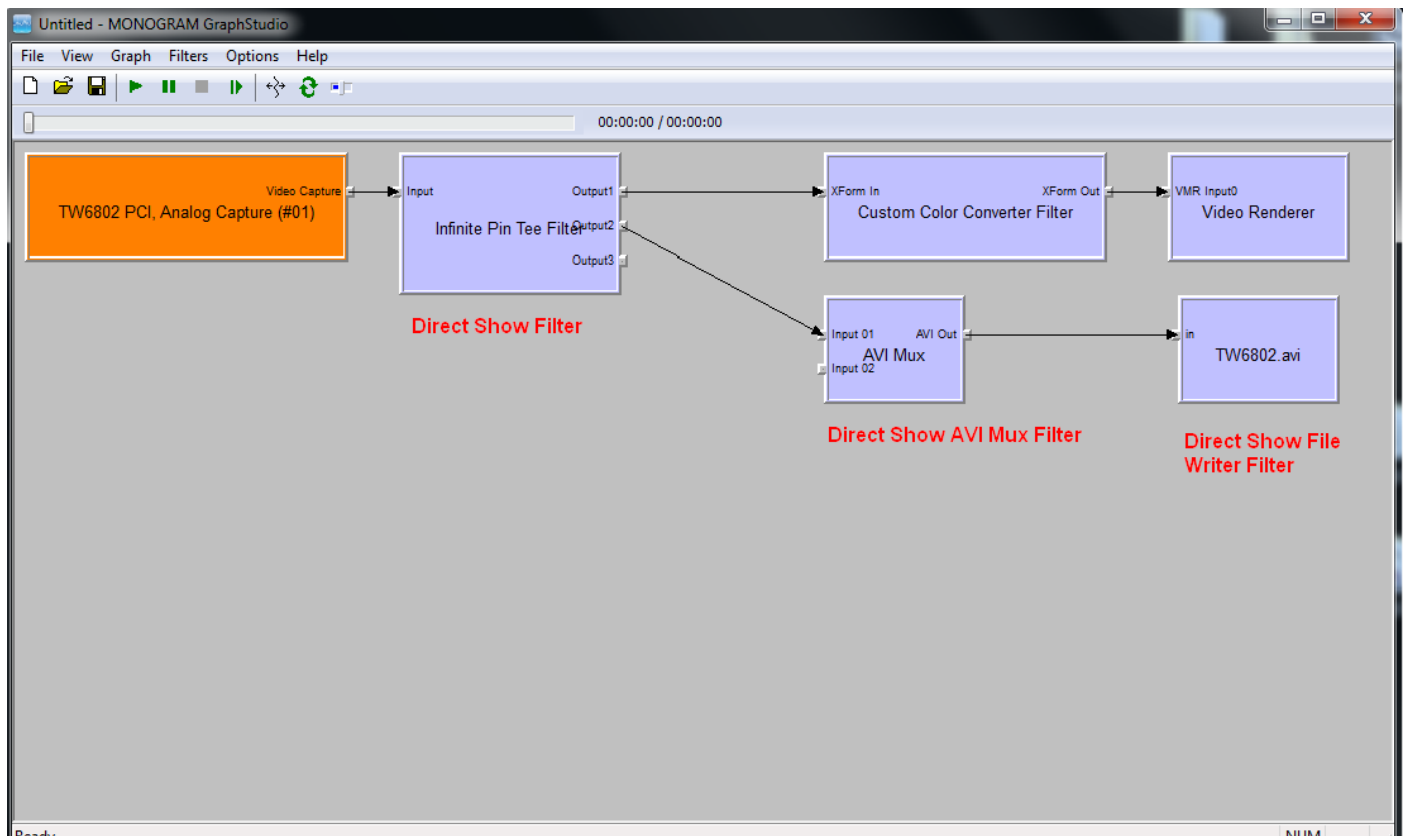
All filter names are "TW6802 PCI, Analog Capture (#XX)" for video, and "TW6802 PCI, Analog WaveIn (#XX)" for audio. They are registered at "WDM Streaming Captures Devices" category.



For the preview output, here, the video format is YUY2 and audio format is PCM. The connection of filters is as:



Moreover, customer wants to use graphedit to save raw video stream into AVI can reference as below:



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 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, 0x12 };
```

All custom properties are defined as below:

```
typedef enum {  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_VIDEO_AGC = 204,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_VIDEO_SWITCH_SPEED = 205,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_VIDEO_SWITCH_CHANNEL_TABLE = 206,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_VIDEO_SWITCH_RESOLUTION_TABLE = 207,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_VIDEO_POST_FRAME_RATE = 208,  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_AUDIO_VOLUME = 251,  
    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_VIDEO_AGC (204)

The property allows you to enable or disable TW6802's AGC loop circuit. When the AGC loop function is disabled, the AGC gain can be decided by your software manually. The range of AGC gain is from 0 to 511. If MSB bit.31 is set, the AGC loop function will be enabled.

SUPPORT VALUE: 0x00000000 ~ 0x000001FF - DISABLE AGC LOOP & SET AGC GAIN

SUPPORT VALUE: 0x80000000 - ENABLE AGC LOOP

EXAMPLE#01:

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 204, 0x80000000 );
```

EXAMPLE#02:

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 204, 0x00000100 );
```

3.2. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_SWITCH_SPEED (205)

Software programmer can use this property to control the switching speed under switching mode^(*). Currently, there are 3 level speeds that can be controlled by you. Please reference this table as below: Here, the total fps means the total output frames per second for one chip under switching mode. For example, the total fps is 20fps. If you split one chip into four sub-channels, the every sub-channel's fps will be 5fps.

(*) Here, the switching mode means that one TW6802 chip is spitted to 2, 3 or 4 channels. We call these channels as sub-channel. SC300Q16 owns 4 chips and max 16 sub-channels.

RESOLUTION		SPEED	TOTAL FPS	COMMENT
D1	720×480	2	20FPS	
	704×480			
	640×480	1	20FPS	
	720×576			
	704×576	0	12FPS	
	640×576			
HALF D1	720×240	2	20FPS	
	704×240			
	640×240	1	30FPS	TO OBTAIN PERFECT OUTPUT RESULT, BUT TO CAUSE ONE LEFT-RIGHT SHIFTING SIDE EFFECT.
	720×288			
	704×288	0	15FPS	
	640×288			
CIF	360×240	2	20FPS	
	352×240			
	320×240	1	30FPS	TO OBTAIN PERFECT OUTPUT RESULT, BUT TO CAUSE ONE LEFT-RIGHT SHIFTING SIDE EFFECT.
	360×288			
	352×288	0	15FPS	
	320×288			

SUPPORT VALUE: 0, 1, 2

EXAMPLE#01: SET SWITCHING SPEED.

```
ULONG speed = 0;
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 205, NULL, 0, &speed,
sizeof(ULONG) );
```


3.3 KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_SWITCH_CHANNEL_TABLE (206)

In default setting, our switching algorithm uses one averaged channel table to control the channel switching sequence. The table size is 12 items length. Every item can be 0, 1, 2 or 3 to correspond to its sub-channels. For example, the split number is 4. The default switching channel table will be as { 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3 }.

Now, you can control the switching table dynamically by our SDK. For example, the table can be updated to { 0, 0, 1, 2, 0, 0, 1, 2, 0, 0, 1, 2 }. The total 20fps for every sub-channel will be changed as below:

CH#01: 10fps,
CH#02: 5fps,
CH#03: 5fps, and
CH#04: 0fps.

For another example, the table is { 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3 }. The result simulates one channel jumping effect.

Moreover, the table also can support single channel switching such as { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }. When the table is set, the switching mode will auto be returned to real-time mode. So, by this table, the CH#02's fps will be up to 30fps.

EXAMPLE#01: DISABLE CH#03.

```
BYTE TABLE[ 12 ] = { 0, 1, 3, 0, 1, 3, 0, 1, 3, 0, 1, 3 };  
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 205, NULL, 0, TABLE, 12 );
```

EXAMPLE#02: CHANNEL JUMPING.

```
BYTE TABLE[ 12 ] = { 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3 };  
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 205, NULL, 0, TABLE, 12 );
```

EXAMPLE#03: GET CURRENT SWITCH CHANNEL TABLE.

```
BYTE TABLE[ 12 ];  
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 205, NULL, 0, TABLE, 12 );
```

EXAMPLE#04: SINGLE CHANNEL OUTPUT.

```
BYTE TABLE[ 12 ] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };  
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 205, NULL, 0, TABLE, 12 );
```

3.4. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_SWITCH_RESOLUTION_TABLE (207)

The custom property allows the developer to change default resolution at any time. For example, at four channel's switching mode, developer need obtain this configuration as below:

CH#01: 704x480

CH#02: 704x240

CH#03: 352x240

CH#04: 352x240

The property programming is similar to **ANALOG_VIDEO_SWITCH_CHANNEL_TABLE**. It also uses 12 bytes to setup the kernel driver's resolution table. Every item is corresponded to its channel item in the switching channel table. The parameter range is from 0 to 2.

0x00: D1

0x01: HALF.D1

0x02: CIF.

EXAMPLE#01: SETUP RESOLUTION TABLE AS BELOW:

CH#01: 704x480

CH#02: 704x240

CH#03: 352x240

CH#04: 352x240

```
BYTE CHANNEL_TABLE[ 12 ] = { 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3 };
```

```
BYTE RESOLUTION_TABLE[ 12 ] = { 0, 1, 2, 2, 0, 1, 2, 2, 0, 1, 2, 2 };
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 206, NULL, 0, CHANNEL_TABLE, 12 );
```

```
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 207, NULL, 0, RESOLUTION_TABLE, 12 );
```

3.5. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_POST_FRAME_RATE (208)

Generally, the software developer can use AMESDK_SET_FORMAT to control video frame rate output. For some special applications, developer could adjust the frame rate dynamically during **recording**. The post frame rate is dynamically used to adjust **current stream output**, which is set by AMESDK_SET_FORMAT function at initialize stage. The range of the post frame rate property is from 0 to 255. It is identical to the skip number of frame (or field). The value 1 means **the recording frame rate** is 15.000fps in NTSC.

EXAMPLE#01: SET FRAMERATE TO 30.000FPS:

```
ULONG fps = 0;          // 30FPS / (0 + 1) = 30FPS
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 208, NULL, 0, &fps, sizeof(ULONG) );
```

EXAMPLE#02: SET FRAMERATE TO 15.000FPS:

```
ULONG fps = 1;          // 30FPS / (1 + 1) = 15FPS
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 207, NULL, 0, &fps, sizeof(ULONG) );
```

EXAMPLE#03: SET FRAMERATE TO 10.000FPS:

```
ULONG fps = 2;          // 30FPS / (2 + 1) = 10FPS
m_pKsPropertySet->Set( PROPSETID_AMEBDAD_CUSTOM_PROP, 207, NULL, 0, &fps, sizeof(ULONG) );
```

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) );
```


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.