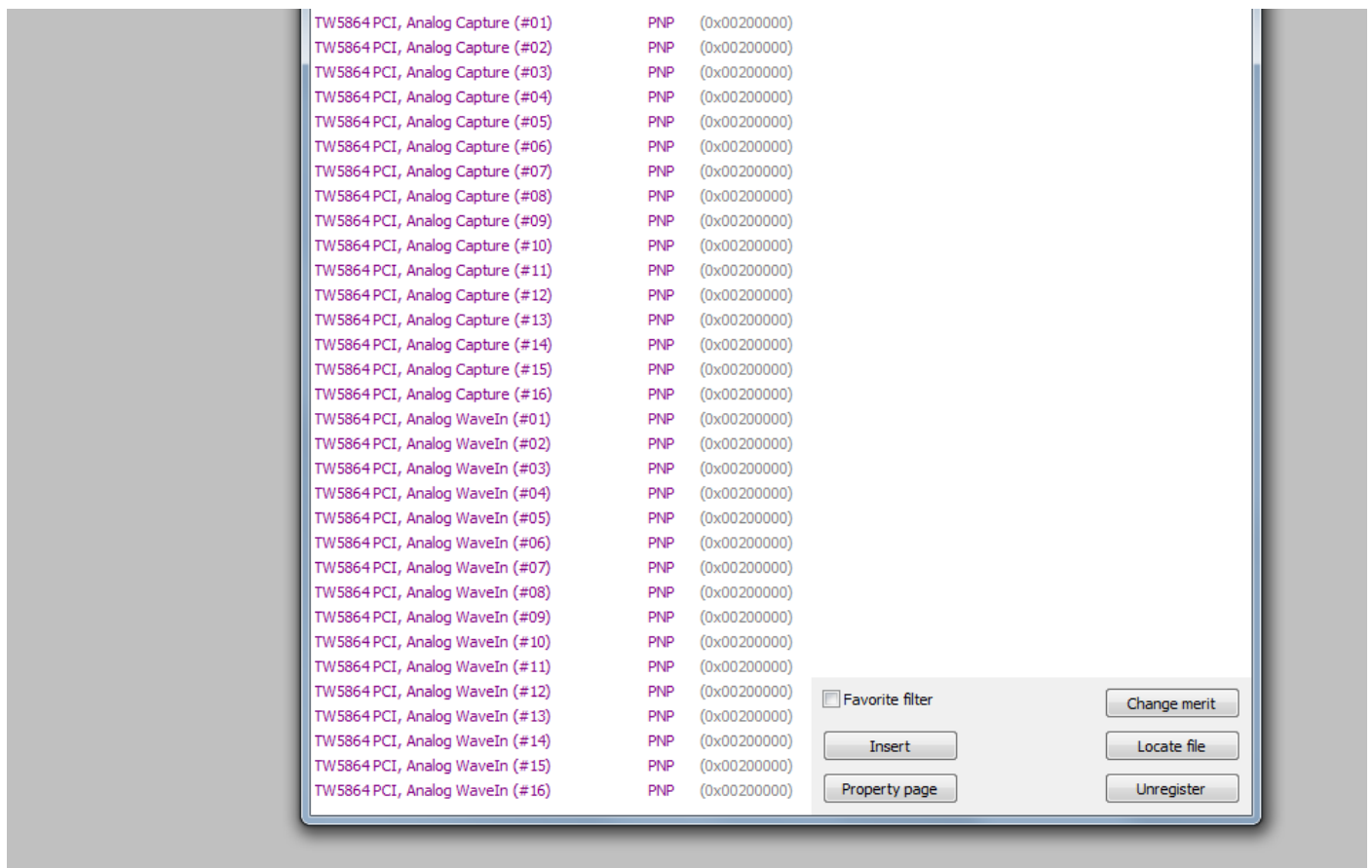


# SC3B0 DirectShow Software Programming Guide

Customer uses DirectShow to develop software can bypass our SDK to access TW5864 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.

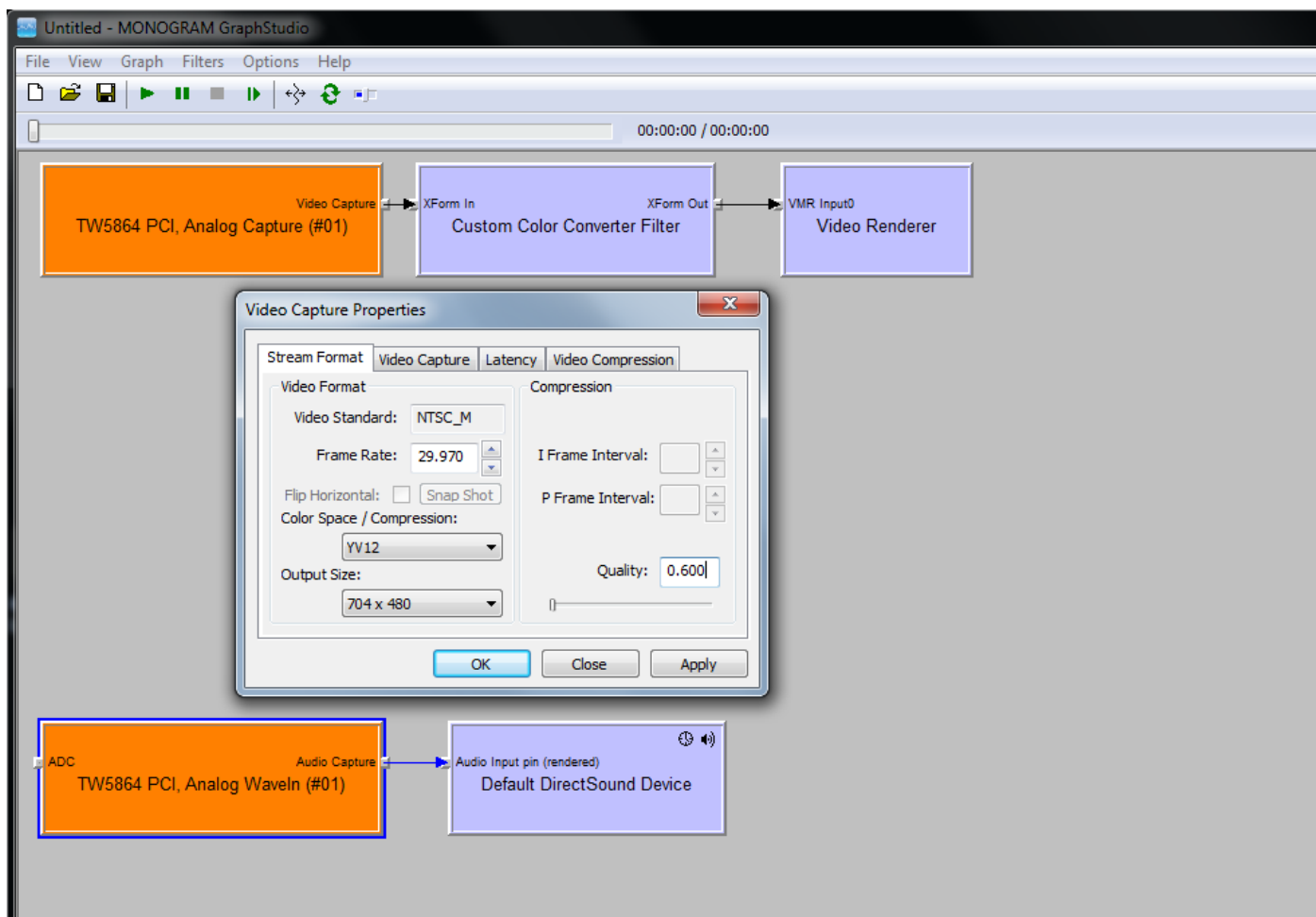
All filter names are "TW5864 PCI, Analog Capture (#XX)" for video, and "TW5864 PCI, Analog WaveIn (#XX)" for audio. They are registered at "WDM Streaming Captures Devices" category.



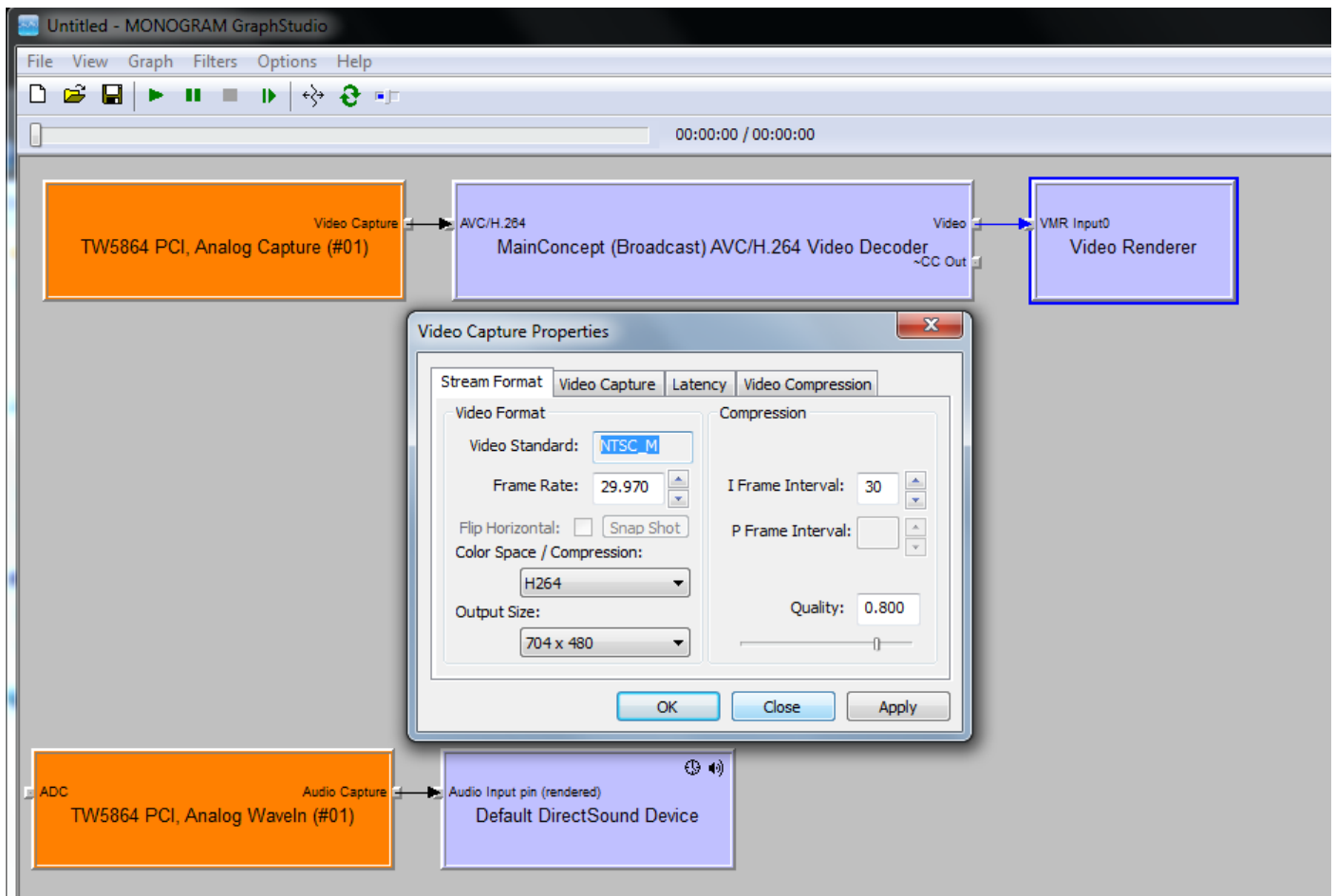
TW5864 is a hardware compression chip, it can output YV12, and two H.264 streams at the same time. We use H.264 to stand for main stream, X.264 for sub stream.

```
#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
```

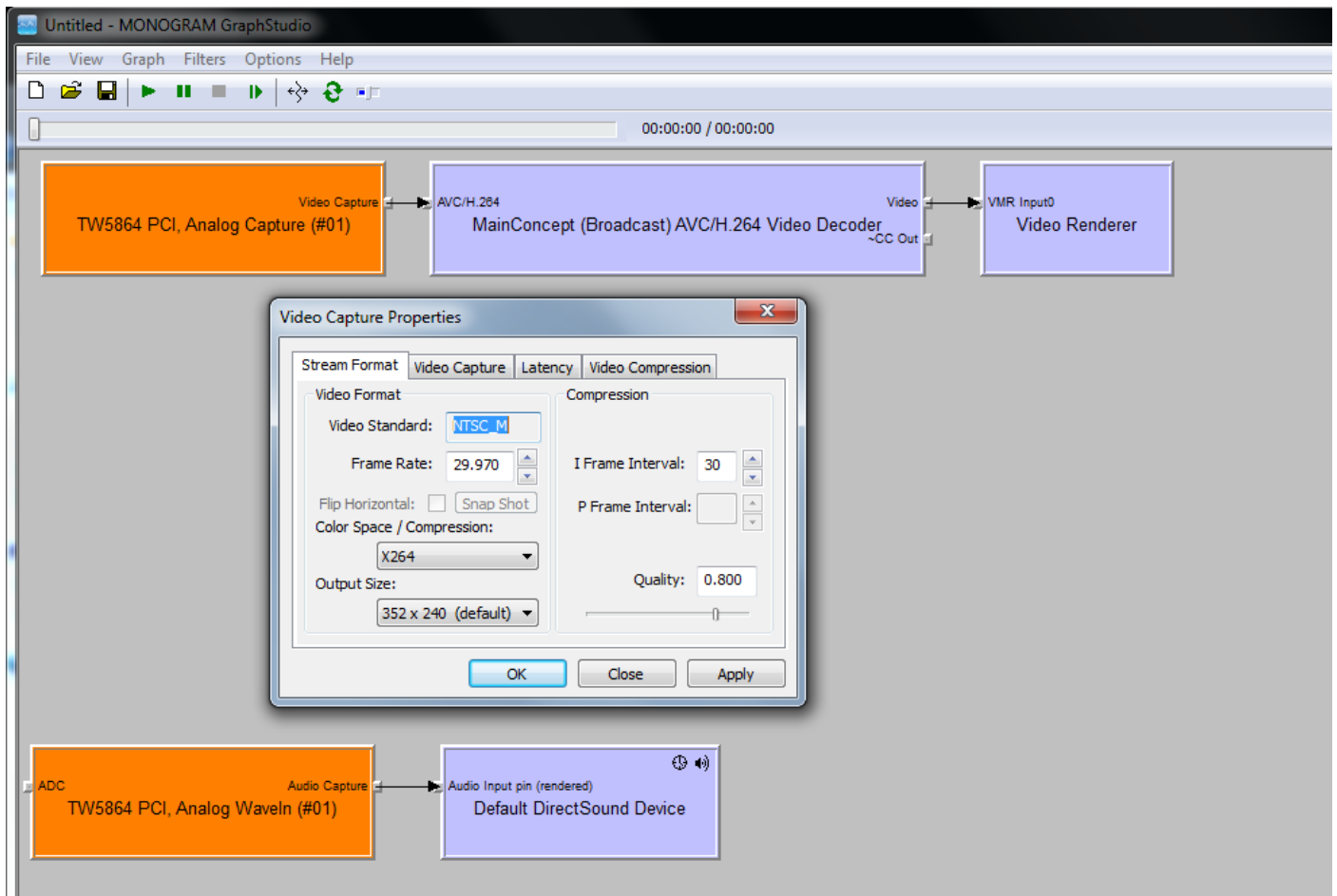
For the preview output, here, the video format is YV12 and audio format is PCM. The connection of filters is as:



Main stream connection is as:

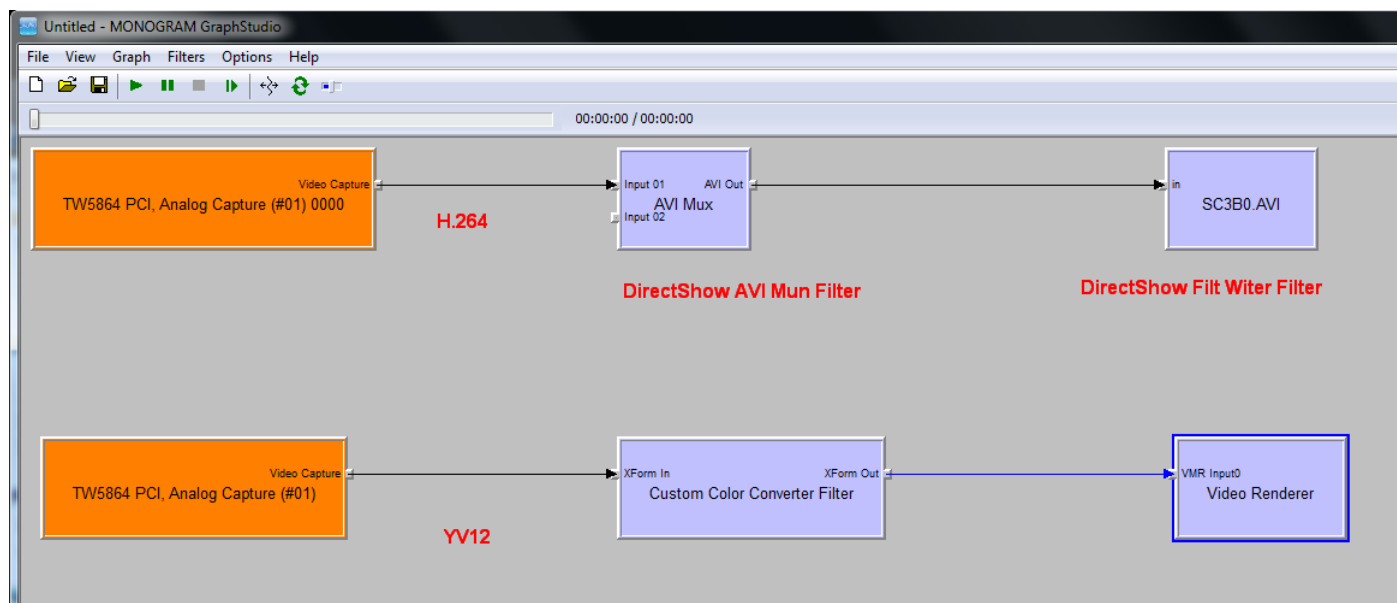


Sub stream connection is as:



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 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, 0x1D };
```

All custom properties are defined as below:

```
typedef enum {  
    KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_DEINTERLACE_TYPE      = 200  
    KSPROPERTY_CUSTOM_GET_MOTION_MASK_STATUS1                 = 900  
    KSPROPERTY_CUSTOM_GET_MOTION_MASK_STATUS2                 = 901  
    KSPROPERTY_CUSTOM_GET_MOTION_MASK_STATUS3                 = 902  
    KSPROPERTY_CUSTOM_GET_MOTION_MASK_STATUS4                 = 903  
    KSPROPERTY_CUSTOM_GET_MOTION_MASK_STATUS5                 = 904  
    KSPROPERTY_CUSTOM_GET_MOTION_MASK_STATUS6                 = 905  
    KSPROPERTY_CUSTOM_XET_MOTION_TEMPORAL_SENSITIVITY         = 910  
    KSPROPERTY_CUSTOM_XET_MOTION_SPATIAL_SENSITIVITY           = 911  
    KSPROPERTY_CUSTOM_XET_MOTION_LEVEL_SENSITIVITY            = 912  
    KSPROPERTY_CUSTOM_XET_MOTION_SPEED                         = 913  
    KSPROPERTY_CUSTOM_SET_OSD_LINE                             = 920  
    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_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)

TW5864 offers one hardware-based de-interlace on chip. you can choose to enable or disable video de-interlace function.

SUPPORT VALUE: 0 ~ 1 - OFF ~ ON

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.

```
ULONG input = 1;
```

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

Note!! If you enable the hardware-based de-interlace function, you don't need call the software-based de-interlace function again.



3.2 KSPROPERTY\_CUSTOM\_GET\_MOTION\_MASK\_STATUS1 (900) (READ ONLY)  
 3.2 KSPROPERTY\_CUSTOM\_GET\_MOTION\_MASK\_STATUS2 (901) (READ ONLY)  
 3.2 KSPROPERTY\_CUSTOM\_GET\_MOTION\_MASK\_STATUS3 (902) (READ ONLY)  
 3.2 KSPROPERTY\_CUSTOM\_GET\_MOTION\_MASK\_STATUS4 (903) (READ ONLY)  
 3.2 KSPROPERTY\_CUSTOM\_GET\_MOTION\_MASK\_STATUS5 (904) (READ ONLY)  
 3.2 KSPROPERTY\_CUSTOM\_GET\_MOTION\_MASK\_STATUS6 (905) (READ ONLY)  
 3.2 KSPROPERTY\_CUSTOM\_XET\_MOTION\_TEMPORAL\_SENSITIVITY (910)  
 3.2 KSPROPERTY\_CUSTOM\_XET\_MOTION\_SPATIAL\_SENSITIVITY (911)  
 3.2 KSPROPERTY\_CUSTOM\_XET\_MOTION\_LEVEL\_SENSITIVITY (912)  
 3.2 KSPROPERTY\_CUSTOM\_XET\_MOTION\_SPEED (913)

TW5864 can offer hardware-based motion detection function. The property set allows you to access it. The properties \*XET\_MOTION\_\*\_SENSITIVITY and \*XET\_MOTION\_SPEED help you to control the motion detection's quality in different application.

SENSITIVITY SUPPORT VALUE: 0 ~ 15 - MORE ~ LESS SENITIVE

SPEED SUPPORT VALUE: 0 ~ 63 - 1 FIELD ~ 64 FIELD INTERVALS

The property \*GET\_MOTION\_MASK\_STATUS1~6 returns the result of motion detection. Each of status obtained is for 4-byte value (32 cell), it represent where motion detection is set on or off. TW5864 uses 192 (16x12) detection cells in full screen for motion detection. Each detection cell is composed of 44 pixels and 20 lines for NTSC and 24 lines for PAL. The property \*GET\_MOTION\_MASK\_STATUS offer 24 bytes ( 192 bits ) to describe all 192 cells' result.

001	002	003	004	005	006	007	008	009	010	011	012	013	014	015	016
017	018	019	020	021	022	023	024	025	026	027	028	029	030	031	032
033	034	035	036	037	038	039	040	041	042	043	044	045	046	047	048
049	050	051	052	053	054	055	056	057	058	059	060	061	062	063	064
065	066	067	068	069	070	071	072	073	074	075	076	077	078	079	080
081	082	083	084	085	086	087	088	089	090	091	092	093	094	095	096
097	098	099	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192

EXAMPLE#01: TO GET MOTION DETECTION RESULT.

```
ULONG p_motion_status[ 6 ];      // 24 bytes
memset( p_motion_status, 0xFF, sizeof( p_motion_status ) );
ULONG nProperty = KSPROPERTY_CUSTOM_GET_MOTION_MASK_STATUS1;
BYTE *pValue = (BYTE *)p_motion_status;
for( ULONG c = 0 ; c < 24 ; c += 4 )
{
    m_pKsPropertySet->Get( PROPSETID_AMEBDAD_CUSTOM_PROP,
                          nProperty + (c / 4),
                          NULL, 0, pValue + c, sizeof(ULONG), &cbBytes )

    printf("pValue = 0x%02x%02x%02x%02x",
          *(pValue + c + 0),
          *(pValue + c + 1),
          *(pValue + c + 2),
          *(pValue + c + 3) );
}
```

3.3. KSPROPERTY\_CUSTOM\_SET\_OSD\_LINE (920) (WRITE ONLY)  
3.3. KSPROPERTY\_CUSTOM\_SET\_OSD\_TEXT\_STRING\_1 (921) (WRITE ONLY)  
3.3. KSPROPERTY\_CUSTOM\_SET\_OSD\_TEXT\_STRING\_2 (922) (WRITE ONLY)  
3.3. KSPROPERTY\_CUSTOM\_SET\_OSD\_TEXT\_STRING\_3 (923) (WRITE ONLY)  
3.3. KSPROPERTY\_CUSTOM\_SET\_OSD\_TEXT\_STRING\_4 (924) (WRITE ONLY)

The properties allow you to change TW5864'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.

SUPPORT VALUE: 0 ~ 7 - LINE#0 ~ LINE#7

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 )
```

### 3.4. 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 TW5864'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. Access Video Encoder Property

Developer can use the AMESDK\_G/SET\_VIDEOCOMPRESSION\_PROPERTY function to access all TW5864's video encoder properties. These properties as describe as the table below:

PROPERTY	RANGE
VideoCompression_KeyFrameRate	0 ~ 255
VideoCompression_Quality	0 ~ 10,000
VideoCompression_OverrideKeyFrame	1 (WRITE ONLY)
VideoCompression_BitRateMode	0, 1
VideoCompression_BitRate	128,000 ~ 12,000,000
VideoCompression_PostResolution	SEE SDK
VideoCompression_PostSkipFrameRate	0 ~ 255

### 4.1 Video Encoder Property:

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

```

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_TW5864, 407, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) )
        {
            return FALSE;
        }
    }
    if( nProperty == 0x00000004 ) { // BIT.RATE

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_TW5864, 403, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) )
        {
            return FALSE;
        }
    }
    if( nProperty == 0x00000008 ) { // POST.RESOLUTION

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_TW5864, 401, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) )
        {
            return FALSE;
        }
    }
    if( nProperty == 0x00000009 ) { // POST.SKIP.FRAME.RATE

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_TW5864, 402, 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_TW5864, 407, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000004 ) { // BIT.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_TW5864, 403, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000008 ) { // POST.RESOLUTION
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_TW5864, 401, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000009 ) { // POST.SKIP.FRAME.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_TW5864, 402, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    return TRUE;
}

```

## **5. 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.