

SC580 Extra Software Programming Guide

Custom Property

1. KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION (940)

1. KSPROPERTY_CUSTOM_XET_GPIO_DATA (941)

1. KSPROPERTY_CUSTOM_XET_GPIO_SUPPORT (942) (READ ONLY)

The property allows you to access FH8735'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].

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 940, 0x00FF );
```

EXAMPLE#02: TO DEFINE GPIO AS 16 OUTPUT PINS [0:15] THEN PULL HIGH FOR ALL.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 940, 0xFFFF );
```

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 941, 0xFFFF );
```

EXAMPLE#03: TO DEFINE GPIO AS 16 INPUT PINS [0:15] THEN READ DATA FROM IT.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 940, 0x0000 );
```

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 941, &GPIO );
```

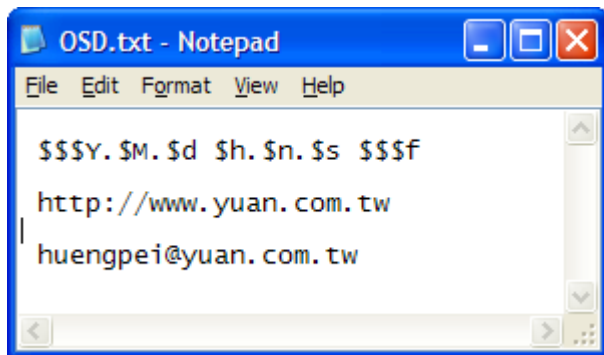
2. KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING (921) (WRITE ONLY)

2. KSPROPERTY_CUSTOM_SET_OSD_COLOR (929) (WRITE ONLY)

The properties allow you to change FH8735's test 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.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 929, 0x00000001 );
```

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

```
CHAR path[] = "C:\\WINDOWS\\FH8735\\OSD.TXT";
```

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 920, 0x00000000 );
```

```
AMESDK_SET_CUSTOM_PROPERTY_EX( hDev, 921, (BYTE *) (path), strlen(path) );
```

2. KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PARAMS (970)

2. KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PATH (971) (WRITE ONLY)

The properties allow you to change FH8735's graphic OSD context. The property KSPROPERTY_CUSTOM_SET_PICTURE_PARAMS allows you to set 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 OSD 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";  
AMESDK_SET_CUSTOM_PROPERTY_EX( hDev, 970, params, sizeof(params) );  
AMESDK_SET_CUSTOM_PROPERTY_EX( hDev, 971, (BYTE *) (path), strlen(path) );
```

4. 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 );
```

5. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_INPUT (201)

The property allows you to get/change current input source. We can support total 5 kinds of video input sources, HDMI, DVI-D, Components, DVI-A, and SDI.

SUPPORT VALUE: 0: HDMI
1: DVI-Digital
2: Components (YCbCr)
3: DVI-Analog (RGB) (VGA)
4: SDI

EXAMPLE#01: CHANGE TO HDMI INPUT.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 201, 0 );
```

EXAMPLE#02: CHANGE TO SDI INPUT.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 201, 4 );
```

EXAMPLE#03: GET CURRENT INPUT SOURCE.

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 201, &INPUT );
```

6. 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 HDMI, DVI-D, and SDI input mode.

EXAMPLE#01: CHANGE TO EMBEDDED AUDIO INPUT.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 255, 0 );
```

EXAMPLE#02: CHANGE TO LINE-IN INPUT.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 255, 1 );
```

EXAMPLE#03: GET CURRENT AUDIO INPUT SOURCE.

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 255, &INPUT );
```

7. 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.

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 202, &HDCP );  
IF( HDCP == 1 ) { RECORD_FUNCTION = DISABLE; }  
IF( HDCP == 0 ) { RECORD_FUNCTION = ENABLE; }
```


8. KSPROPERTY_CUSTOM_GET_ANALOG_VIDEO_RESOLUTION (210) (READ ONLY)

8. 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 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	* ₁
1920×1080p@50fps	0x07800438	50	* ₁
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	* ₂
720×288P@50fps	0x05A00240	50	* ₂
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	* ₃
640×400p@60fps	0x02800190	60	* ₄
640×384p@60fps	0x02800180	60	* ₄

*₁ THE FORMAT WILL BE DOWN SPEED TO 1080P@30FPS/1080P@25FPS.

*₂ THE FORMAT IS USED BY SONY PS1/PS2 GAME MACHINE.

*₃ THE FORMAT IS USED BY MICROSOFT XBOX360 GAME MACHINE (640×480p@60fps).

*₄ THE FORMAT IS USED BY NEC IPC MACHINE (640×400p@56.4fps).

Here, the resolution property can be described as below:

$$\text{RESOLUTION} = (\text{WIDTH} \ll 16) \mid (\text{HEIGHT} \ll 0)$$

EXAMPLE#01: GET CURRENT VIDEO FORMAT.

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 210, &RESOLUTION );
```

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 208, &FRAMERATE );
```

9. 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 stereo and 16bits quality. The only difference is their sample frequency, so you can use the property to obtain the input's sample frequency.

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

EXAMPLE#01: GET CURRENT AUDIO SAMPLE FREQUENCY.

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 253, &FREQUENCY );
```

10. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_DEINTERLACE_TYPE (200)

FH8735 offers one hardware-based deinterlacer on chip. The property will allow you to access it. You can call the function, `AMESDK_SET_CUSTOM_PROPERTY`, 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.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 200, 0 );
```

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

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 200, 5 );
```

Note!! The function, `AMESDK_SET_DEINTERLACE`, is used for software-based deinterlacer only. If you enable the hardware-based deinterlace function, you don't need call `AMESDK_SET_DEINTERLACE` again.

11. 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 call the function, `AMESDK_SET_CUSTOM_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.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 217, 0 );
```

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

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 217, 3 );
```

12. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_VGA_YCBCR_AUTO_PHASE (219)

12. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_VGA_YCBCR_OFFSET_X (221)

12. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_VGA_YCBCR_OFFSET_Y (222)

If input is in VGA or YCbCr, these properties allow you to adjust the hardware receiver's property.

The auto phase property can be set as below:

SUPPORT VALUE: 0 ~ 63 - MANUAL PHASE DEGREE

SUPPORT VALUE: 0x80000000 - AUTO PHASE

The offset property allows you to adjust the horizontal and vertical offset for signal. Moreover, our driver will do auto memorize for setting value in next detection.

SUPPORT VALUE: -127 ~ +128

EXAMPLE#01: TO SET HORIZONTAL OFFSET FOR VGA.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 221, -8 );
```

13. Access Encoder Property

Developer can use the AMESDK_G/SET_VIDEOCOMPRESSION_PROPERTY function to access all FH8735's video encoder properties. These properties as describe as the table below:

PROPERTY	RANGE
VideoCompression_KeyFrameRate	0 ~ 255
VideoCompression_OverrideKeyFrame	1 (WRITE ONLY)
VideoCompression_Quality	0 ~ 10,000
VideoCompression_BitRateMode	0, 1, 2
VideoCompression_BitRate	128,000 ~ 12,000,000
VideoCompression_PostResolution	(cx << 12) + (cy << 0)
VideoCompression_PostSkipFrameRate	0 ~ 255
VideoCompression_PostAvgFrameRate	0 ~ 60
VideoCompression_BFrames	0, 1, 2
VideoCompression_Profile	0 (MAINPROFILE), 1 (BASELINE)
VideoCompression_AspectRatio	(cx << 12) + (cy << 0)

14. Access Custom Property for DirectShow Developer

Customer uses DirectShow to develop software can bypass our SDK to access FH8735 directly. The interface can be queried from our capture source filter.

At Section 14.1, 14.2, 14.3, 14.4 and 14.5, you can use IKsPropertySet to access all.

14.1 Device Serial Number Property:

```
#define KSPROPERTY_CUSTOM_GET_DEVICE_SERIAL_NUMBER 0 (READ ONLY) (ULONG)
```

14.2 GPIO Property:

```
#define KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION 940 (ULONG)
```

```
#define KSPROPERTY_CUSTOM_XET_GPIO_DATA 941 (ULONG)
```

```
#define KSPROPERTY_CUSTOM_GET_GPIO_SUPPORT 942 (READ ONLY) (ULONG)
```

14.3 Text OSD Property:

```
#define KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_1 921 (WRITE ONLY) (16 BYTES)
```

```
#define KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_2 922 (WRITE ONLY) (16 BYTES)
```

```
#define KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_3 923 (WRITE ONLY) (16 BYTES)
```

```
#define KSPROPERTY_CUSTOM_SET_OSD_TEXT_STRING_4 924 (WRITE ONLY) (16 BYTES)
```

```
#define KSPROPERTY_CUSTOM_SET_OSD_COLOR 929 (WRITE ONLY) (ULONG)
```

The property *SET_OSD_TEXT_STRING_1 accesses the first 16 characters.

The property *SET_OSD_TEXT_STRING_2 accesses the 17 ~ 32 characters.

The property *SET_OSD_TEXT_STRING_3 accesses the 33 ~ 48 characters.

The property *SET_OSD_TEXT_STRING_4 accesses the 48 ~ 64 characters.

14.4 Graphic OSD Property:

```
#define KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PARAMS 970 (WRITE ONLY) (4 ULONGS)
#define KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PATH_1 971 (WRITE ONLY) (16 BYTES)
#define KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PATH_2 972 (WRITE ONLY) (16 BYTES)
#define KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PATH_3 973 (WRITE ONLY) (16 BYTES)
#define KSPROPERTY_CUSTOM_SET_OSD_PICTURE_PATH_4 974 (WRITE ONLY) (16 BYTES)
```

The property *SET_OSD_PICTURE_PATH_1 accesses the first 16 characters.

The property *SET_OSD_PICTURE_PATH_2 accesses the 17 ~ 32 characters.

The property *SET_OSD_PICTURE_PATH_3 accesses the 33 ~ 48 characters.

The property *SET_OSD_PICTURE_PATH_4 accesses the 48 ~ 64 characters.

14.5 Region Mask Property:

```
#define KSPROPERTY_CUSTOM_XET_REGION_MASK_PARAMS 890 (16 BYTES)
```

14.6 Video & Audio Property:

```
#define KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_INPUT 201 (ULONG)
#define KSPROPERTY_CUSTOM_GET_ANALOG_VIDEO_MACROVISION 202 (READ ONLY) (ULONG)
#define KSPROPERTY_CUSTOM_GET_ANALOG_VIDEO_RESOLUTION 210 (READ ONLY) (ULONG)
#define KSPROPERTY_CUSTOM_GET_ANALOG_VIDEO_FRAME_RATE 208 (READ ONLY) (ULONG)
#define KSPROPERTY_CUSTOM_XET_ANALOG_AUDIO_SAMPLE_FREQUENCY 253 (READ ONLY) (ULONG)
```

14.7 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;
}

```

15. 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.