

# SC2A0/SC3A0 Extra Software Programming Guide

## Custom Property

1. **KSPROPERTY\_CUSTOM\_GET\_DEVICE\_SERIAL\_NUMBER\_INFO** (0) (READ ONLY)

1. **KSPROPERTY\_CUSTOM\_GET\_DEVICE\_BUS\_NUMBER\_INFO** (2) (READ ONLY)

The property **KSPROPERTY\_CUSTOM\_GET\_DEVICE\_SERIAL\_NUMBER\_INFO** allows you to get Vendor ID (VID) and Product ID (PID) for the capture card. Vendor ID and product ID are 16-bit numbers used to identify PCI devices to a computer. The VID and PID are embedded in the capture card and communicated to the computer.

EXAMPLE#01: TO GET THE VENDER ID AND PRODUCT ID FROM THE CAPTURE CARD.

```
ULONG dwSerialNumber = 0x00000000;  
AMESDK_GET_CUSTOM_PROPERTY( hDevice, 0, &dwSerialNumber);
```

The property **KSPROPERTY\_CUSTOM\_GET\_DEVICE\_BUS\_NUMBER\_INFO** allows you to get current PCI bus number on the capture card. For example, the capture card on the first PCI slot, on the second PCI slot, or on the third PCI slot, etc.

EXAMPLE#02: TO GET THE BUS NUMBER ON THE CAPTURE CARD.

```
ULONG dwBusNumber = 0x00000000;  
AMESDK_GET_CUSTOM_PROPERTY( hDevice, 2, &dwBusNumber);
```

2. KSPROPERTY\_CUSTOM\_GET\_ANALOG\_VIDEO\_SIGNAL\_LOCK\_STATUS (230) (READ ONLY)

2. KSPROPERTY\_CUSTOM\_GET\_ANALOG\_VIDEO\_MACROVISION (202) (READ ONLY)

The property (230) is used to determine whether the signal is locked.

SUPPORT VALUE: 0 ~ 1 - UNLOCK ~ LOCK

EXAMPLE#01: TO GET THE CURRENT SIGNAL STATUS.

```
LONG nLock = 0x00;
```

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 230, &nLock );
```

The property (202) 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#06: GET HDCP PROTECT.

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 202, &HDCP );
```

```
if( HDCP == 1 ) { RECORD_FUNCTION = DISABLE; }
```

```
if( HDCP == 0 ) { RECORD_FUNCTION = ENABLE; }
```

- 3. **KSPROPERTY\_CUSTOM\_GET\_ANALOG\_VIDEO\_SINGAL\_DEBUG\_INFO** (271) (READ ONLY)
- 3. **KSPROPERTY\_CUSTOM\_GET\_PREVIEW\_VIDEO\_STARAM\_FRAME\_NUMBER\_INFO** (351) (READ ONLY)
- 3. **KSPROPERTY\_CUSTOM\_GET\_PREVIEW\_AUDIO\_STARAM\_FRAME\_NUMBER\_INFO** (361) (READ ONLY)
- 3. **KSPROPERTY\_CUSTOM\_GET\_ENCODER\_VIDEO\_DEFAULT\_FRAME\_NUMBER\_INFO** (430) (READ ONLY)

The property **KSPROPERTY\_CUSTOM\_GET\_ANALOG\_VIDEO\_SINGAL\_DEBUG\_INFO** is used to get the debug information in capture card running state. The output information is 32-bit error numbers. If the number is 0, the device is working properly. You can call it in timer function to get current signal status regularly.

SUPPORT VALUE: 0: GOOD

OTHERS: ERROR BITS

EXAMPLE#01: TO GET CURRENT SINGAL DEBUG STATUS.

```
ULONG dwSingalDebugInfo = 0x00000000;  
AMESDK_GET_CUSTOM_PROPERTY( hDevice, 271, &dwSingalDebugInfo);
```

The property **KSPROPERTY\_CUSTOM\_GET\_PREVIEW\_VIDEO\_STARAM\_FRAME\_NUMBER\_INFO** allows you to get the total number of frames in preview video. The property reads frame number information from video stream. You can call it in timer function to get current frame number regularly.

SUPPORT VALUE: FRAME NUMBER

EXAMPLE#02: TO GET VIDEO PREVIEW STREAM'S FRAME NUMBER.

```
ULONG dwPreviewVideoFrameNumber = 0;  
AMESDK_GET_CUSTOM_PROPERTY( hDev, 351, &dwPreviewVideoFrameNumber );
```

The property **KSPROPERTY\_CUSTOM\_GET\_PREVIEW\_AUDIO\_STARAM\_FRAME\_NUMBER\_INFO** allows you to get the total number of frames in preview audio. The property reads frame number information from audio stream. You can call it in timer function to get current frame number regularly.

SUPPORT VALUE: FRAME NUMBER

EXAMPLE#03: TO GET AUDIO PREVIEW STREAM'S FRAME NUMBER.

```
ULONG dwPreviewAudioFrameNumber = 0;  
AMESDK_GET_CUSTOM_PROPERTY( hDev, 361, &dwPreviewAudioFrameNumber);
```

The property **KSPROPERTY\_CUSTOM\_GET\_ENCODER\_VIDEO\_DEFAULT\_FRAME\_NUMBER\_INFO** allows you to get the total number of frames in video encoder. The property reads frame number information from compressed video stream. You can call it

in timer function to get current frame number regularly.

SUPPORT VALUE: FRAME NUMBER

EXAMPLE#04: TO GET VIDEO ENCODER STREAM STREAM'S FRAME NUMBER.

```
ULONG dwEncoderVideoFrameNumber = 0;
```

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 430, &dwEncoderVideoFrameNumber);
```

#### 4. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_INPUT\_AUTO\_SCAN (232)

This property allows you to enable or disable the automatic scan video input signal source. If this function detects the actual video input source and format on capture card, it will automatically set the correct video input source and format.

SUPPORT VALUE: 0 ~ 1 - DISABLE ~ ENABLE

EXAMPLE#01: ENABLE THE AUTO INPUT SCAN FUNCTION

```
LONG enable = 0x01;
```

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 232, enable );
```

EXAMPLE#02: DISENABLE THE AUTO INPUT SCAN FUNCTION

```
LONG disable = 0x00;
```

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 232, disable );
```

- 5. **KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_DEINTERLACE\_TYPE (200)**
- 5. **KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_VERTICAL\_MIRROR (244)**
- 5. **KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_HORIZONTAL\_MIRROR (245)**

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.

The two properties (244/245) are used to set mirror function. When mirror function is enabled, the vertical or horizontal video frame is inverted on display window. Same as deinterlacing, the property is used for display engine only.

SUPPORT VALUE: 0 ~ 1 - DISABLE ~ ENABLE

EXAMPLE#03: ENABLE THE VERTICAL MIRROR FUNCTION ON DISPLAY WINDOW

```
LONG enable = 0x01;
```

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 244, enable);
```

EXAMPLE#04: ENABLE THE HORIZONTAL MIRROR FUNCTION ON DISPLAY WINDOW

```
LONG enable = 0x01;
```

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 245, enable);
```

## 6. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_AUDIO\_VOLUME (251)

The property is used to control the current audio ADC's volume on the capture card.

SUPPORT VALUE: 0 (Mute): ~ 255 (Full)

**Note!! The property is enabled only by HDMI, DVI-D, and SDI input mode.**

EXAMPLE#01: TO SET THE AUDIO VOLUME AMPLITUDE.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 251, 128 );
```

EXAMPLE#02: TO GET THE AUDIO VOLUME AMPLITUDE.

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 251, &VOLUME );
```

## 7. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_IS\_720H\_PIXELS\_OUTPUT (214)

The property allows you to check whether the output video height is 720 pixels. If the value is 0, the output video height is 704 pixels. If the value is 1, the output video height is 720 pixels.

SUPPORT VALUE: 0: 704 PIXELS  
                  1: 720 PIXELS

EXAMPLE#01: TO GET HEIGHT OUTPUT.

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 214, &HEIGHT_PIXELS );
```



## 8. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_QUEUE\_BUFFER\_SIZE (216)

The property allows 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

```
LONG nBfferSize = 10;  
AMESDK_SET_CUSTOM_PROPERTY( hPreviewDevice, 216, nBfferSize );
```

EXAMPLE#02: TO SET THE HARDWARE-ENCODED QUEUE(MAIN) SIZE TO 16 FRAMES

```
LONG nBfferSize = 16;  
AMESDK_SET_CUSTOM_PROPERTY( hMainDevice, 216, nBfferSize );
```

EXAMPLE#03: TO SET THE HARDWARE-ENCODED QUEUE(SUB) SIZE TO 16 FRAMES

```
LONG nBfferSize = 16;  
AMESDK_SET_CUSTOM_PROPERTY( hSubDevice, 216, nBfferSize );
```

## 9. 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 );
```

## 10. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_FLEXIBLE\_RESOLUTION\_PATCH (220)

The property allows you to adjust the video's resolution from hardware board. If it is disabled, the output resolution is equal to input signal's resolution. If it is enabled, we will enable one auto scalar to output customized format. For example, input resolution is 704x480 and capture output pin's resolution is 352x240.

SUPPORT VALUE: 0 ~ 1 - DISABLE ~ ENABLE

EXAMPLE#01: TO ENABLE RESOLUTION SCALER.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 220, 1 );
```

## 11. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_FLEXIBLE\_FPS\_PATCH (218)

The property allows you to control the output format from one video capture filter. It allows you to adjust the video's frame rate from driver side. If it is disabled, the output frame rate is equal to input signal's frame rate.

SUPPORT VALUE: 0 ~ 1 - DISABLE ~ ENABLE

EXAMPLE#01: TO ENABLE FRAMERATE SCALER.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 218, 1 );
```

## 12. KSPROPERTY\_CUSTOM\_XET\_PREVIEW\_VIDEO\_STERAM\_POST\_RESOLUTION (350)

The property allows you to adjust current video resolution dynamically. The driver will re-allocate memory during changing video format on capture card running state.

SUPPORT VALUE:    RESOLUTION = (WIDTH << 16) | (HEIGHT << 0)

EXAMPLE#01: TO SET PREVIEW VIDEO RESOLUTION DYNAMICALLY.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 350, &RESOLUTION );
```

**13. KSPROPERTY\_CUSTOM\_XET\_PREVIEW\_VIDEO\_STREAM\_POST\_SKIP\_FRAMERATE (246)**

**13. KSPROPERTY\_CUSTOM\_XET\_PREVIEW\_VIDEO\_STARAM\_POST\_AVG\_FRAMERATE (247)**

The property (246) allows you to adjust current video skip frame rate dynamically. The range of the property is from 1 to 255. It is identical to the skip number of frame. For example, the value 1 will generate the preview frame rate, 15.000fps.

SUPPORT VALUE:    0: DISABLE  
                  1, 2, 3, 4, ... SKIP

EXAMPLE#01: TO SET PREVIEW VIDEO SKIP FRAMERATE DYNAMICALLY.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 246, &FRAMERATE );
```

The property (247) allows you to adjust current video average frame rate dynamically. The range of the property is from 1 to 85. To enable it, our driver will follow the setting value to output one average fps. For example, 9 mean 9.00fps.

SUPPORT VALUE:    0: DISABLE  
                  1 ~ 85 FPS

EXAMPLE#02: TO SET PREVIEW VIDEO AVERAGE FRAMERATE DYNAMICALLY.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 247, &FRAMERATE );
```

- 15. **KSPROPERTY\_CUSTOM\_XET\_GPIO\_DIRECTION (940)**
- 15. **KSPROPERTY\_CUSTOM\_XET\_GPIO\_DATA (941)**
- 15. **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 );`

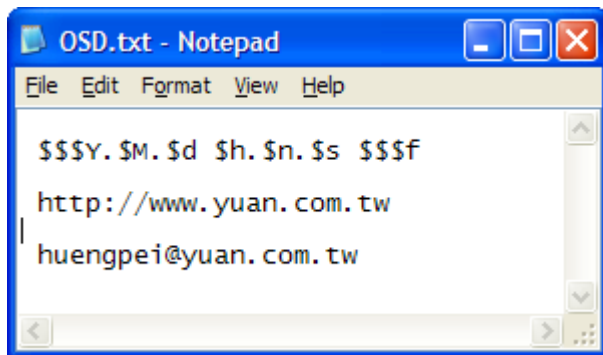
**16. KSPROPERTY\_CUSTOM\_SET\_OSD\_TEXT\_STRING (921) (WRITE ONLY)**

**16. KSPROPERTY\_CUSTOM\_SET\_OSD\_COLOR (929) (WRITE ONLY)**

The properties allow you to change FH8735's 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) );
```

## 17. 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 );
```

- 18. **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_MODE** (407)
- 18. **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_QUALITY** (404)
- 18. **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_BITRATE** (403)
- 18. **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_MAX\_BITRATE** (409)
- 18. **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_MIN\_BITRATE** (410)

The property **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_MODE** allows you to get/set record mode on hardware-compressed capture device. There are 3 kinds of encoder mode: variable bitrate (VBR), constant bitrate (CBR) and average bitrate (ABR).

SUPPORT VALUE: 0: VBR ( FQP )  
1: CBR  
2: ABR

In the VBR mode, you choose the desired quality going from 0 (lowest quality) to 1000 (highest quality). The encoder tries to maintain the given quality for your video file. The main advantage is that you are able to specify the quality level that you want to reach, but the disadvantage is that the video size is unpredictable.

The property **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_QUALITY** allows you to set a suitable quality in VBR mode.

SUPPORT VALUE: 0 ~ 10000

EXAMPLE#01: TO SET VIDEO ENCODER QUALITY.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 407, 0 );  
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 404, 8000 );
```

In the CBR mode, the bitrate will be the same for the whole video file. The quality of your video is variable. The main advantage is that final video size can be accurately predicted, but the disadvantage is that the complex video parts will be a lower quality.

The property **KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_BITRATE** allows you to set a suitable bitrate in CBR mode.

SUPPORT VALUE: 0 ~ 60000000 BPS

EXAMPLE#02: TO SET VIDEO ENCODER BITRATE.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 407, 1 );
```

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 403, 12 * 1024 * 1024 );
```

In the ABR mode, you choose a target bitrate and the encoder will try to constantly maintain an average bitrate while using higher bitrate for the parts of your video that need more bits. The result will be of higher quality than CBR encoding while the average file size will remain predictable.

SUPPORT VALUE: 0 ~ 60000000 BPS

EXAMPLE#03: TO SET VIDEO ENCODER BITRATE.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 407, 2 );
```

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 403, 12 * 1024 * 1024 );
```

The two properties

**KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_MAX\_BITRATE /**  
**KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_RECORD\_MIN\_BITRATE**

allow you to set a suitable through/peak bitrate in ABR mode.

SUPPORT VALUE: 0 ~ 60000000 BPS

EXAMPLE#04: TO SET VIDEO ENCODER PEAK BITRATE.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 407, 2 );
```

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 409, 12 * 1024 * 1024 );
```

EXAMPLE#05: TO SET VIDEO ENCODER THROUGH BITRATE.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 407, 2 );
```

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 410, 1 * 1024 * 1024 );
```

**19. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STREAM\_POST\_SKIP\_FRAMERATE (402)**

**19. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STARAM\_POST\_AVG\_FRAMERATE (422)**

The property (402) allows you to adjust encoding skip frame rate dynamically. The range of the property is from 1 to 255. It is identical to the skip number of frame. For example, the value 1 will generate the encoding frame rate, 15.000fps.

SUPPORT VALUE:    0: DISABLE  
                  1, 2, 3, 4, ... SKIP

EXAMPLE#01: TO SET VIDEO ENCODER SKIP FRAMERATE DYNAMICALLY.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 402, 1 );
```

The property (422) allows you to adjust encoding average frame rate dynamically. The range of the property is from 1 to 85. To enable it, our driver will follow the setting value to output one average fps. For example, 9 mean 9.00fps.

SUPPORT VALUE:    0: DISABLE  
                  1 ~ 85 FPS

EXAMPLE#02: TO SET VIDEO ENCODER AVERAGE FRAMERATE DYNAMICALLY.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 422, 9 );
```

## 20. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_POST\_RESOLUTION (401)

The property allows you to adjust video encoding resolution dynamically. The driver will re-allocate memory during changing video format on capture card running state.

SUPPORT VALUE:    RESOLUTION = (WIDTH << 16) | (HEIGHT << 0)

EXAMPLE#01: TO SET VIDEO ENCODER RESOLUTION DYNAMICALLY.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 401, &RESOLUTION );
```

## 21. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_PROFILE (412)

The property allows you to adjust video encoder profile on hardware-compressed capture device. There are 2 kinds of profile value.

SUPPORT VALUE:    0: DEFAULT (MAIN)  
                  1: BASELINE  
                  2: MAIN

EXAMPLE#01: TO SET VIDEO ENCODER PROFILE.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 412, 2 );
```

## 22. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_LEVEL (414)

The property allows you to adjust video encoder level on hardware-compressed capture device. The range of the property is from 1 to 42.

SUPPORT VALUE: 1 ~ 42

EXAMPLE#01: TO SET VIDEO ENCODER LEVEL.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 414, 41 );
```



### 23. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_ENTROPY (415)

The property allows you to adjust video encoder entropy on hardware-compressed capture device. There are 2 kinds of entropy value.

SUPPORT VALUE:    0: DEFAULT (CABAC)  
                  1: CAVLC  
                  2: CABAC

EXAMPLE#01: TO SET VIDEO ENCODER ENTROPY.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 412, 2 );
```

## 24. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_GOP (405)

The property allows you to set the maximum number of frames between each key frame on hardware-compressed capture device. For example, a value of 100 will create a key frame every 100 frames. A smaller GOP value will increase the size of your video file, but it will allow more precise playback in most players. The GOP set to higher value can increase the compression ratio, but that would not be free to jump to any time point in playback.

SUPPORT VALUE: 0 ~ 255

EXAMPLE#01: TO SET VIDEO ENCODER GOP.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 405, 30 );
```

## 25. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_ASPECT\_RATIO (413)

The property allows you to set video encoder aspect ratio on hardware-compressed capture device. The property can maintain aspect ratio when resizing your video. Aspect ratio is expresses as the relation of the width and height.

SUPPORT VALUE: ASPECT\_RATIO = (WIDTH << 16) | (HEIGHT << 0)

EXAMPLE#01: TO SET VIDEO ENCODER ASPECT RATIO.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 413, &ASPECT_RATIO );
```

## 26. KSPROPERTY\_CUSTOM\_SET\_ENCODER\_VIDEO\_STERAM\_FORCE\_KEY\_FRAME (406)

The property allows you to set video encoder force key frame on hardware-compressed capture device. The property puts key frame on the next frame or forcing a key frame at specified timestamp.

SUPPORT VALUE: 1: FROCE KEYFRAME

EXAMPLE#01: TO SET VIDEO ENCODER FORCE KEYFRAME.

```
AMESDK_SET_CUSTOM_PROPERTY( hEncoderDev, 406, 1 );
```

## 27. KSPROPERTY\_CUSTOM\_XET\_ENCODER\_VIDEO\_STERAM\_FRAME\_QUEUE\_LENGTH (424)

The property allows you to specify the number of the rendered video frame in the queue buffer for video encoded stream. By the default, the queue size of the corresponding video encoded stream is set 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 if the total buffer sizes you want to set exceed the system capabilities.

Note: Setting queue buffer size will involve in dynamically allocated memory.

EXAMPLE#01: TO SET THE VIDEO ENCODER QUEUE SIZE TO 16 FRAMES

```
LONG nBfferSize = 16;
```

```
AMESDK_SET_CUSTOM_PROPERTY(hEncoderDev, 424, nBfferSize );
```

## 28. Application Note for AMESDK\_GET\_LOCK()

Customer to use AMESDK\_GET\_LOCK, please notes it. FH8735 is one 4CH integrated SOC. In order to reducing your software loading, we can group 4 channels' status into 4bits return value. You can call AMESDK\_GET\_LOCK to obtain 4CHs' status at the same time.

EXAMPLE#01: GET SC3A0N4 SIGNAL STATUS.

```
AMESDK_GET_LOCK( hDev[ 0 ], &status ); // GET CH01 ~ CH04 STATUS
ULONG status_ch01 = (status >> 0) & 0x01;
ULONG status_ch02 = (status >> 1) & 0x01;
ULONG status_ch03 = (status >> 2) & 0x01;
ULONG status_ch04 = (status >> 3) & 0x01;
```

EXAMPLE#02: GET SC3A0N8 SIGNAL STATUS.

```
AMESDK_GET_LOCK( hDev[ 0 ], &status ); // GET CH01 ~ CH04 STATUS
ULONG status_ch01 = (status >> 0) & 0x01;
ULONG status_ch02 = (status >> 1) & 0x01;
ULONG status_ch03 = (status >> 2) & 0x01;
ULONG status_ch04 = (status >> 3) & 0x01;
```

```
AMESDK_GET_LOCK( hDev[ 4 ], &status ); // GET CH05 ~ CH08 STATUS
ULONG status_ch05 = (status >> 0) & 0x01;
ULONG status_ch06 = (status >> 1) & 0x01;
ULONG status_ch07 = (status >> 2) & 0x01;
ULONG status_ch08 = (status >> 3) & 0x01;
```

## 29. 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_PostResolution	(cx << 12) + (cy << 0)
VideoCompression_PostSkipFrameRate	0 ~ 255
VideoCompression_PostAvgFrameRate	0 ~ 85
VideoCompression_Profile	0 (MAIN DEFAULT), 1 (BASELINE), 2 (MAIN)
VideoCompression_Level	1 ~ 42
VideoCompression_Entropy	0 (CABAC DEFAULT), 1 (CAVLC), 2 (CABAC)
VideoCompression_RecordMode	0 (VBR), 1 (CBR), 2 (ABR)
VideoCompression_RecordQuality	0 ~ 10000
VideoCompression_BitRate	0 ~ 60000000 BPS
VideoCompression_MaxBitRate	0 ~ 60000000 BPS
VideoCompression_MinBitRate	0 ~ 60000000 BPS
VideoCompression_KeyFrameRate	0 ~ 255
VideoCompression_AspectRatio	(cx << 12) + (cy << 0)
VideoCompression_OverrideKeyFrame	1 (WRITE ONLY)

## 30. 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.

You can use IKsPropertySet to access all.

### 30.1 Device Serial Number Property:

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

### 30.2 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)
```

### 30.3 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.

### 30.4 Region Mask Property:

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



## 30.5 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 ) { // GOP ( KEY.FRAME.RATE )
        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;
}

```

### **31. 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.