

SC290/SC390 Extra Software Programming Guide

Custom Property

3. 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; }
```

3. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_QUEUE_BUFFER_SIZE (216)

The property allow 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 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 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 );
```

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_VERTICAL_MIRROR (244)

5. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_HORIZONTAL_MIRROR (245)

This property is 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#01: ENABLE THE VERTICAL MIRROR FUNCTION ON DISPLAY WINDOW

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

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

EXAMPLE#02: 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 );
```

1. **KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION (940)**

1. **KSPROPERTY_CUSTOM_XET_GPIO_DATA (941)**

1. **KSPROPERTY_CUSTOM_XET_GPIO_SUPPORT (942) (READ ONLY)**

The properties allow you to access AH840'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, 0x000000FF );
```

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

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 940, 0x0000FFFF );
```

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 941, 0x0000FFFF );
```

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

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 940, 0x00000000 );
```

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

NOTE!! For DirectShow developer, please use IKsPropertySet to access the two properties. The property size is sizeof(ULONG) always.

- 2. KSPROPERTY_CUSTOM_SET_OSD_LINE (920) (WRITE ONLY)
- 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 AH840's OSD context. The property KSPROPERTY_CUSTOM_SET_OSD_COLOR allows you to change string's color. Here, there are 16 kinds of colors can be selected by you. Also, you can modify it dynamically during recording.

SUPPORT VALUE: 0 ~ 15 - COLOR#0 ~ COLOR#15

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 ~ 2 - LINE#0 ~ LINE#2

SUPPORT LINE#0 STRING: 32 CHARACTERS

SUPPORT LINE#1 STRING: 15 CHARACTERS

SUPPORT LINE#2 STRING: 15 CHARACTERS

EXAMPLE#01: TO CHANGE OSD COLOR TO COLOR#1.

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

EXAMPLE#02: TO CHANGE LINE#0'S STRING.

```
CHAR string[ 32 + 1 ] = "0000.00.00 00:00:00:";
```

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

```
AMESDK_SET_CUSTOM_PROPERTY_EX( hDev, 921, (BYTE *) (string), strlen(string) + 1 );
```

EXAMPLE#03: TO CHANGE LINE#1'S STRING.

```
CHAR string[ 15 + 1 ] = "CH00";
```

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 920, 0x00000001 );
```

```
AMESDK_SET_CUSTOM_PROPERTY_EX( hDev, 921, (BYTE *) (string), strlen(string) + 1 );
```

7. Application Note for AMESDK_GET_LOCK()

Customer to use AMESDK_GET_LOCK, please notes it. AH8400 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 SC390N4 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 SC390N8 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;
```


8. Access Video Encoder Property

Developer can use the AMESDK_G/SET_VIDEOCOMPRESSION_PROPERTY function to access all AH840'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, 2
VideoCompression_BitRate	128,000 ~ 12,000,000
VideoCompression_QPStep	0 ~ 255
VideoCompression_PeakBitRate	128,000 ~ 12,000,000
VideoCompression_TroughQuality	0 ~ 10,000
VideoCompression_PostResolution	SEE SDK
VideoCompression_PostSkipFrameRate	0 ~ 255

9. Access Custom Property for DirectShow Developer

Customer uses DirectShow to develop surveillance software can bypass our SDK to access AH840 directly. The interface can be queried from our capture source filter.

At Section 5.1, 5.2, and 5.3, you can use IKsPropertySet to access all.

9.1 Device Serial Number Property:

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

9.2 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)
```

9.3 OSD Property:

```
#define KSPROPERTY_CUSTOM_SET_OSD_LINE 920 (WRITE ONLY) (ULONG)
```

```
#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_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.

To disable the default time OSD on LINE#0, you can follow these steps as below:

```
ULONG nValue = 0x00000000; // LINE#0
CHAR psz[ 16 ] = " "; // SPACE
if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 920, NULL, 0, &nValue, sizeof(ULONG) ) ) {
    return FALSE;
}
if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 921, NULL, 0, (BYTE *) (psz), 16 ) ) {
    return FALSE;
}
```

9.4 Video Encoder Property:

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

```

static const GUID GUID_KPS_AH8400 = { 0xD1E5209F, 0x68FD, 0x4529, 0xBE, 0xE0, 0x5E, 0x7A, 0x1F, 0x47, 0x92, 0x16 };

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_AH8400, 407, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 403, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x00000005 ) { // QP.STEP

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 408, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x00000006 ) { // PEAK.BIT.RATE

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 409, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x00000007 ) { // TROUGH.QUALITY

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 410, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 401, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_AH8400, 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_AH8400, 407, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000004 ) { // BIT.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 403, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000005 ) { // QP.STEP
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 408, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000006 ) { // PEAK.BIT.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 409, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000007 ) { // TROUGH.QUALITY
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 410, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000008 ) { // POST.RESOLUTION
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 401, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000009 ) { // POST.SKIP.FRAME.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_AH8400, 402, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    return TRUE;
}

```

10. 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.

11. Application Note for SC390N8 or SC390N16 Developer

SC390's preview path is controlled by 2nd AH840 chipset. All 16 channels' preview streams are outputted from one multiplexer, AM8816. When you change the video standard from NTSC to PAL or otherwise, you also need change channel 5's video standard at the same time. Please reference the SC290 sample code as below:

```
// [PATCH PROGRAM] [2009.07.15] [HUENGPEI@YUAN.COM.TW] FIX HARDWARE STANDARD MISS-MATCHING ISSUE

// DEAR CUSTOMER,

// BECAUSE SC390 HAS SOME HARDWARE CIRCUITS ARE CONTROLLED BY 2ND CHIPSET (05CH ~ 08CH),

// THE 2ND CHIPSET NEED BE CONTROLLED AT THE SAME TIME.

// YOU CAN MODIFY THIS SOURCE CODE TO SUPPORT 16CH, TOO.

// BEST REGARDS,

// H.P.

{    DEVICE_HANDLE hDev = AMESDK_CREATE( "AH8400 PCI", 4, 0, NULL, on_process_video_buffer_CH05, this ); // CH#05

    if( hDev & 0x80000000 ) { hDev = 0xFFFFFFFF; }

    AMESDK_SET_STANDARD( hDev, standard ); // STANDARD

    AMESDK_SET_FORMAT( hDev, MAKEFOURCC('U', 'Y', 'V', 'Y'), 352, 240, 16, 29.97 ); } // RESOLUTION

    AMESDK_SET_DEINTERLACE( hDev, 0x00000000 ); // NOTE!! 352 ;Ñ 240 IS PROGRESSIVE FIELD FORMAT

    AMESDK_RUN( hDev );

    if( hDev != 0xFFFFFFFF ) { AMESDK_DESTROY( hDev ); hDev = 0xFFFFFFFF; }

}
```

For Directshow developer, the same method should be used for capture source filter.