

# SC590 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 TW2809'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-SUPPOR

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\_LINE (920) (WRITE ONLY)**

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

The properties allow you to change TW2809'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. The max string length is 16 characters.

SUPPORT VALUE: 0 ~ 2 - LINE#0 ~ LINE#2

EXAMPLE#01: TO CHANGE LINE#0'S STRING.

```
CHAR string[] = "1234567890ABCDEF";
```

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

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

EXAMPLE#02: TO CHANGE LINE#1'S STRING.

```
CHAR string[] = "CH00";
```

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

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

### 3. 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

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 );
```

4. KSPROPERTY\_CUSTOM\_GET\_ANALOG\_VIDEO\_RESOLUTION (210) (READ ONLY)
4. 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. All supported formats are described in the table:

FORMAT	RESOLUTION	FRAME RATE	
1920×1080p@60fps	0x07800438	60	* <sub>1</sub>
1920×1080p@50fps	0x07800438	50	* <sub>1</sub>
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	

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

Here, the resolution property can be described as below:

RESOLUTION = (WIDTH << 16) | (HEIGHT << 0)

```
EXAMPLE#01: GET CURRENT VIDEO FORMAT.  
AMESDK_GET_CUSTOM_PROPERTY( hDev, 210, &RESOLUTION );  
AMESDK_GET_CUSTOM_PROPERTY( hDev, 208, &FRAMERATE );
```

## 5. Dynamically Change of Preview Resolution

Our driver can allow you to dynamically change preview resolution size. Because SC590N4 is designed by PCI-Ex1 bandwidth, the max preview configuration can output 1CH 1920×1080@30fps and 3CHs x 960x540@30fps. The helper property can help you obtain one good quality preview video on full-screen mode of your software:

EXAMPLE#01: TO SET VIDEO RESOLUTION FORM 960X540 TO 1920X1080.

```
hDev = AMESDK_CREATE( hDev, ... );
AMESDK_SET_FORMAT( hDev, MAKEFOURCC('Y', 'U', 'Y', '2'), 960, 540, 29.970 );
AMESDK_RUN( hDev );
...
AMESDK_STOP( hDev );
AMESDK_SET_FORMAT( hDev, MAKEFOURCC('Y', 'U', 'Y', '2'), 1920, 1080, 29.970 );
AMESDK_RUN( hDev );
```

## 6 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 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 );
```

## 7. Access Encoder Property

Developer can use the AMESDK\_G/SET\_VIDEOCOMPRESSION\_PROPERTY function to access all TW2809'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
VideoCompression_BitRate	128,000 ~ 12,000,000
VideoCompression_PostResolution	(cx << 12) + (cy << 0)
VideoCompression_PostSkipFrameRate	0 ~ 255
VideoCompression_PostAvgFrameRate	0 ~ 60

## 8. Access Custom Property for DirectShow Developer

Customer uses DirectShow to develop software can bypass our SDK to access TW2809 directly. The interface can be queried from our capture source filter.

At Section 8.1, 8.2, 8.3 and 8.4, you can use IKsPropertySet to access all.

### 8.1 Device Serial Number Property:

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

### 8.2 GPIO Property:

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

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

```
#define KSPROPERTY_CUSTOM_XET_GPIO_SUPPORT 942 (ULONG)
```

### 8.3 GPIO 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.

### 8.4 Video & Audio Property:

```
#define KSPROPERTY_CUSTOM_XET_ANALOG_AUDIO_INPUT 255 (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_PREVIEW_VIDEO_RESOLUTION 350 (ULONG)
```



## 8.5 Video Encoder Property:

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

```
static const GUID GUID_KPS_TW2809 = { 0xD1E5209F, 0x68FD, 0x4529, 0xBE, 0xE0, 0x5E, 0x7A, 0x1F, 0x47, 0x92, 0x1E };
```

```
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_TW2809, 407, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000004 ) { // BIT.RATE
        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_TW2809, 403, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000008 ) { // POST.RESOLUTION
        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_TW2809, 401, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000009 ) { // POST.SKIP.FRAME.RATE
        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_TW2809, 402, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000D ) { // POST.AVG.FRAME.RATE
        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_TW2809, 422, 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_TW2809, 407, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000004 ) { // BIT.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_TW2809, 403, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000008 ) { // POST.RESOLUTION
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_TW2809, 401, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000009 ) { // POST.SKIP.FRAME.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_TW2809, 402, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000D ) { // POST.AVG.FRAME.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_TW2809, 422, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    return TRUE;
}

```

## **9. 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.