

# SC2B0/SC3B0 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 properties allow 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 2 OUTPUT PINS [0:1] AND 2 INPUT PINS [2:3].

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 940, 0x00000003 );
```

EXAMPLE#02: TO DEFINE GPIO AS 4 OUTPUT PINS [0:3] THEN PULL HIGH FOR ALL.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 940, 0x0000000F );
```

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 941, 0x0000000F );
```

EXAMPLE#03: TO DEFINE GPIO AS 4 INPUT PINS [0:3] 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)**

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.

```
CHAR string[] = "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#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\_GET\_MOTION\_MASK\_STATUS (900) (READ ONLY)
- 3. KSPROPERTY\_CUSTOM\_XET\_MOTION\_TEMPORAL\_SENSITIVITY (910)
- 3. KSPROPERTY\_CUSTOM\_XET\_MOTION\_SPATIAL\_SENSITIVITY (911)
- 3. KSPROPERTY\_CUSTOM\_XET\_MOTION\_LEVEL\_SENSITIVITY (912)
- 3. 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 SENSITIVE

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

The property \*GET\_MOTION\_MASK\_STATUS returns the result of motion detection. 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.

```
BYTE status[ 24 ];
```

```
AMESDK_SET_CUSTOM_PROPERTY_EX( hDev, 900, status, sizeof(status) );
```

#### 4. KSPROPERTY\_CUSTOM\_XET\_ANALOG\_VIDEO\_DEINTERLACE\_TYPE (200)

TW5864 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.

SUPPORT VALUE: 0 ~ 1 - OFF ~ ON

EXAMPLE#01: TO TURN OFF HARDWARE DEINTERLACE FUNCTION.

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

EXAMPLE#02: TO TURN ON HARDWARE DEINTERLACE FUNCTION.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 200, 1 );
```

Note!! The function, AMESDK\_SET\_DEINTERLACE, is used for software-based deinterlacer only. If you enable the hardware-based deinterlacer function, you don't need call AMESDK\_SET\_DEINTERLACE again.

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

Customer to use AMESDK\_GET\_LOCK, please notes it. If your product is N series, TW5864 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 SC3B0N4 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 SC3B0N8 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;
```

If your product is D series, TW5864 is one 8CH integrated SOC. We will group 8 channels' status into 8bits return value.

EXAMPLE#03: GET SC2B0D16 SIGNAL STATUS.

```
AMESDK_GET_LOCK( hDev[ 0 ], &status ); // GET CH01 ~ CH08 STATUS
ULONG status_ch01 = (status >> 0) & 0x01;
ULONG status_ch02 = (status >> 1) & 0x01;
...
ULONG status_ch08 = (status >> 7) & 0x01;
```

```
AMESDK_GET_LOCK( hDev[ 8 ], &status ); // GET CH09 ~ CH16 STATUS
ULONG status_ch09 = (status >> 0) & 0x01;
ULONG status_ch10 = (status >> 1) & 0x01;
...
ULONG status_ch16 = (status >> 7) & 0x01;
```

## 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 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

## 8. Access Custom Property for DirectShow Developer

Customer uses DirectShow to develop surveillance software can bypass our SDK to access TW5864 directly. The interface can be queried from our capture source filter.

At Section 8.1, 8.2, and 8.3, 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 (READ ONLY) (ULONG)
```

### 8.3 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)
```

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_TW5864, 920, NULL, 0, &nValue, sizeof(ULONG) ) ) {
    return FALSE;
}
if( S_OK != m_pKsPropertySet->Set( GUID_KPS_TW5864, 921, NULL, 0, (BYTE *) (psz), 16 ) ) {
    return FALSE;
}
```

### 8.4 Video Encoder Property:

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



```

static const GUID GUID_KPS_TW5864 = { 0xD1E5209F, 0x68FD, 0x4529, 0xBE, 0xE0, 0x5E, 0x7A, 0x1F, 0x47, 0x92, 0x1D };

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;
}

```

## **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.