

SC3C0 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 MZ0380'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_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_GET_ANALOG_VIDEO_RESOLUTION (210) (READ ONLY)
- 3. KSPROPERTY_CUSTOM_GET_ANALOG_VIDEO_FRAME_RATE (208) (READ ONLY)
- 3. KSPROPERTY_CUSTOM_GET_ANALOG_VIDEO_FRACTION_1000_1001 (241) (READ ONLY)

Our driver can auto detect video format and can report the current input format to your software. These properties can help to obtain current video 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
960×480i@60fps	0x03C001E0	60
960×576i@50fps	0x03C00240	50
720×480i@60fps	0x02D000F0	60
720×576i@50fps	0x02D00120	50
720×240P@60fps	0x05A001E0	60
720×288P@50fps	0x05A00240	50

*₁ THE FORMAT IS USED BY SONY PS1/PS2 GAME MACHINE.

*₂ THE FORMAT IS USED BY MICROSOFT XBOX360 GAME MACHINE (640×480p@60fps).

*₃ THE FORMAT IS USED BY NEC IPC MACHINE (640×400p@56.4fps).

Here, the resolution property can be described as below:

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

Note!! Developer should design one polling operation in one background thread to obtain/update current input format.

EXAMPLE#01: GET CURRENT VIDEO FORMAT.

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 210, &RESOLUTION );
AMESDK_GET_CUSTOM_PROPERTY( hDev, 208, &FRAMERATE );
```

6. 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 );
```

11 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 );
```

12. Access Encoder Property

Developer can use the AMESDK_G/SET_VIDEOCOMPRESSION_PROPERTY function to access all MZ0380'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 ~ 64,000,000
VideoCompression_PostResolution	(cx << 12) + (cy << 0)
VideoCompression_PostSkipFrameRate	0 ~ 255
VideoCompression_PostAvgFrameRate	0 ~ 60
VideoCompression_BFrames	0, 1, 2
VideoCompression_Profile	0 (HIGH DEFAULT), 1 (BASELINE), 2 (MAIN), 3 (HIGH)
VideoCompression_AspectRatio	(cx << 12) + (cy << 0)
VideoCompression_Level	A UNSIGNED INTEGER VALUE
VideoCompression_Entropy	0 (CABAC), 1 (CAVLC), 2 (CABAC)

13. Access Custom Property for DirectShow Developer

Customer uses DirectShow to develop software can bypass our SDK to access MZ0380 directly. The interface can be queried from our capture source filter.

At Section 13.1 and 13.2, you can use IKsPropertySet to access all.

13.1 Device Serial Number Property:

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

13.2 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)
```

13.3 Video & Audio Property:

```
#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)
```

13.4 Video Encoder Property:

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

```
static const GUID GUID_KPS_MZ0380 = { 0xD1E5209F, 0x68FD, 0x4529, 0xBE, 0xE0, 0x5E, 0x7A, 0x1F, 0x47, 0x92, 0x22 };
```

```
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_MZ0380, 407, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_MZ0380, 403, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_MZ0380, 401, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_MZ0380, 402, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x0000000D ) { // POST.AVG.FRAME.RATE

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_MZ0380, 422, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x0000000A ) { // B.FRAME

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_MZ0380, 411, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x0000000B ) { // PROFILE

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_MZ0380, 412, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x0000000C ) { // ASPECT.RATIO

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_MZ0380, 413, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
    if( nProperty == 0x0000000E ) { // LEVEL

        if( S_OK != m_pKsPropertySet->Get( GUID_KPS_MZ0380, 414, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

            return FALSE;
        }
    }
}
```



```
if( nProperty == 0x0000000F ) { // ENTROPY
    if( S_OK != m_pKsPropertySet->Get( GUID_KPS_MZ0380, 415, 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_MZ0380, 407, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000004 ) { // BIT.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_MZ0380, 403, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000008 ) { // POST.RESOLUTION
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_MZ0380, 401, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x00000009 ) { // POST.SKIP.FRAME.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_MZ0380, 402, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000D ) { // POST.AVG.FRAME.RATE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_MZ0380, 422, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000A ) { // B.FRAME
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_MZ0380, 411, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000B ) { // PROFILE
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_MZ0380, 412, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000C ) { // ASPECT.RATIO
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_MZ0380, 413, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000E ) { // LEVEL
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_MZ0380, 414, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    if( nProperty == 0x0000000F ) { // ENTROPY
        if( S_OK != m_pKsPropertySet->Set( GUID_KPS_MZ0380, 415, NULL, 0, &nValue, sizeof(ULONG) ) ) {
            return FALSE;
        }
    }
    return TRUE;
}

```

4. 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.