SC2A0/SC3A0 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 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.

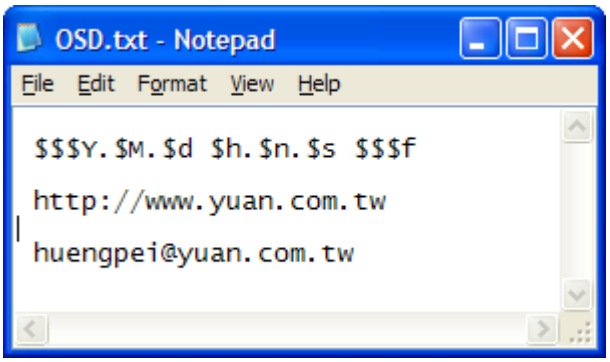
```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 940, 0x0000 );
AMESDK_GET_CUSTOM_PROPERTY( hDev, 941, &GPIO );
```

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 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) );
```

3. 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 );
```

4. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_DEINTERLACE_TYPE (200)

FH8735 offers one hardware-based deinteralcer 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 deinteralce 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 deinteralcer only. If you enable the hardware-based deinteralce function, you don’t need call AMESDK_SET_DEINTERLACE again.

5. 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-nosie 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 );
```

6. 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;
```

7. 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_KeyFrameRate	0 ~ 255
VideoCompression_OverrideKeyFrame	1 (WRITE ONLY)
VideoCompression_Quality	0 ~ 10,000
VideoCompression_BitRateMode	0, 1, 2
VideoCompression_BitRate	128,000 ~ 12,000,000
VideoCompression_PostResolution	(cx << 12) + (cy << 0)
VideoCompression_PostSkipFrameRate	0 ~ 255
VideoCompression_PostAvgFrameRate	0 ~ 60
VideoCompression_BFrames	0, 1, 2
VideoCompression_Profile	0 (MAINPROFILE), 1 (BASELINE)
VideoCompression_AspectRatio	(cx << 12) + (cy << 0)

8. 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.

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_GET_GPIO_SUPPORT 942 (READ ONLY) (ULONG)
```

8.3 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.

8.4 Region Mask Property:

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

8.5 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.FRAMERATE

            if( S_OK != m_pKsPropertySet->Get( GUID_KPS_FH8735, 402, NULL, 0, pValue, sizeof(ULONG), &cbBytes ) ) {

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

            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.FRAMERATE

		if( S_OK != m_pKsPropertySet->Set( GUID_KPS_FH8735, 402, NULL, 0, &nValue, sizeof(ULONG) ) ) {

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

		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;
}
```

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.