



SDK Reference



Contents

Contents	2
1 Introduction	1
1.1 SDK Components	2
1.2 Development Tools	2
2 API List	3
2.1 Camera Information	3
CameraGetVersion	3
CameraGetFirmVersion	3
CameraGetCount	4
CameraGetName	4
CameraGetID	5
CameraGetLastError	5
2.2 Camera Control	7
CameraInit	8
CameraFree	8
CameraReset	8
CameraReconnect	9
CameraSetTimeout	9
CameraGetTimeout	10
2.3 Camera Setting	10
2.3.1 Resolution and ROI	10
CameraGetResolutionCount	10
CameraGetResolutionMax	11
CameraGetResolutionMode	11
CameraSetResolutionMode	12
CameraGetResolution	12
CameraSetResolution	13
CameraSetROI	14
2.3.2 Exposure and Gain	15
CameraSetGain	15
CameraGetGain	16
CameraSetAdvancedGain	16
CameraGetAdvancedGain	16
CameraSetAGC	17
CameraGetAGC	17
CameraSetExposure	18
CameraGetExposure	18



CameraGetExposureUnit	19
CameraGetExposureTime	19
CameraSetAEC	19
CameraGetAEC	20
CameraSetAETarget	20
CameraGetAETarget	21
CameraSetAntiFlicker	21
CameraGetAntiFlicker	21
2.3.3 Image Setting	22
CameraGetAvg	22
CameraSetGamma	22
CameraGetGamma	23
CameraSetContrast	23
CameraGetContrast	23
CameraSetSaturation	24
CameraGetSaturation	24
CameraSetBlackLevel	25
CameraGetBlackLevel	25
CameraSetEnhancement	25
CameraGetEnhancement	26
CameraSetGlobalReset	26
CameraGetGlobalReset	27
CameraSetDenoise	27
CameraGetDenoise	28
CameraSetFPN	28
CameraGetFPN	29
CameraEnableLSC	29
CameraGetLSC	30
CameraEnableLDC	30
CameraGetLDC	31
CameraSetLUT	31
CameraGetLUT	32
2.3.4 Frame Control	33
CameraSetHighspeed	33
CameraGetHighspeed	33
CameraSetDelay	34
CameraGetDelay	34
CameraSetFrameRate	35
CameraGetFrameRate	35
CameraSetMirrorX	36
CameraGetMirrorX	37
CameraSetMirrorY	37
CameraGetMirrorY	37



CameraSetRotate	38
2.3.5 White Balance	38
CameraSetAWB	38
CameraGetAWB	39
CameraSetWBGain	39
CameraGetWBGain	39
CameraOnePushWB	40
2.3.6 Line Scan	40
CameraSetLineHeight	41
CameraGetLineHeight	41
CameraSetLineTrigger	41
CameraGetLineTrigger	42
2.3.7 Storage	42
CameraWriteUserData	42
CameraReadUserData	43
CameraReadSerialNumber	44
CameraSaveParameter	44
CameraLoadParameter	45
2.4 External I/O Control	45
CameraEnableStrobe	45
CameraGetStrobe	46
CameraSetStrobePolarity	46
CameraGetStrobePolarity	46
CameraSetTriggerPolarity	47
CameraGetTriggerPolarity	47
CameraSetTriggerDelay	48
CameraGetTriggerDelay	48
CameraSetStrobeDelay	49
CameraGetStrobeDelay	49
CameraSetStrobeDuration	50
CameraGetStrobeDuration	50
CameraSetDebouncerTime	51
CameraGetDebouncerTime	51
CameraSetSnapMode	52
CameraGetSnapMode	52
CameraSetSnapNum	53
CameraGetSnapNum	53
CameraTriggerShot	53
CameraGetGPIO	54
CameraSetGPIO	54
2.5 Capture Image	55
2.5.1 Query Image	55
CameraGetImageSize	55



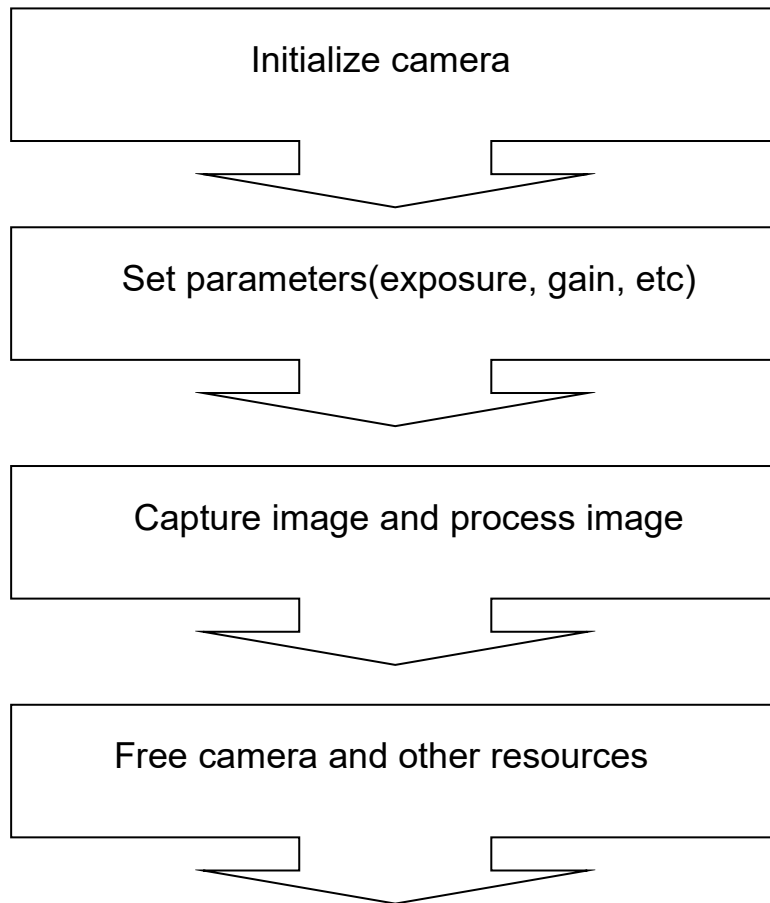
CameraGetImageDepth	55
CameraGetImageBufferSize	56
CameraGetISPIImageBufferSize	56
CameraClearHostBuffer	57
CameraQueryImage	57
CameraISP	60
2.5.2 Callback Function	61
CameraSetOption	61
CameraGetOption	62
CameraPlay	62
CameraPlayEx	63
CameraPlayWithoutCallback	64
CameraStop	64
2.5.3 Shortcut Function	65
CameraShowImage	65
CameraSaveBMPB	66
CameraSaveJpegB	66
CameraSaveHBITMAP	66
CameraSaveImage	67
2.5.4 Utility Function	67
CameraSaveBMP	68
CameraSaveBMP8	68
CameraSaveJpeg	69
CameraShowBufferImage	69
CameraSaveBufferImage	70
3 Revisions	72



1 Introduction

JH series camera can run on Microsoft Windows XP/Windows 7/Windows 10 32 bit or 64 bit OS. The JHCap2 SDK provides programming interface to the camera. User can implement their own application based on the API provided by the SDK.

The typical flow of development based on SDK is illustrated below.



Usually, the camera initialization and parameter setting is done when application start up and freeing camera when exiting the application.



1.1 SDK Components

JHCap SDK is implemented by C++, the SDK including:

Header file: JHCap.h

Dynamic link library: JHCap2.dll

Link symbols: JHCap2.lib

JHCap SDK supports windows 32/64 bit systems. Lib for 32 bit system is in sdk/SDK, lib for 64 bit system is in sdk/SDK64.

Samples is provided along with the SDK, demonstrating the camera function and SDK capabilities.

1.2 Development Tools

The mainstream development tools such as Borland C++/Delphi, QT and Microsoft Visual Stdio can be used to develop the application. Supported programming language including but not limited to C/C++/Visual Basic/C#/VB.net/Delphi/Python.

In the development tool, including the header files(.h)first, then link the library(.lib). When the binary generated, you have to copy the corresponding dynamic library(.dll) to the same location as the binary or on the system PATH before running the program.



2 API List

2.1 Camera Information

This group of function can be called before initializing the camera.

CameraGetVersion

Get the sdk version.

Syntax

```
API_STATUS __stdcall CameraGetVersion(int *major, int *minor)
```

Parameters

[out] **major** major version, usually named by year such as 2016.

[out] **minor** minor version, usually named by month such as 9.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Report the version information when requesting technique support.

CameraGetFirmVersion

Get the camera's firmware version.

Syntax

```
API_STATUS __stdcall CameraGetFirmVersion(int index, int *ver)
```

Parameters

[in] **index** camera index.

[out] **ver** firmware version, a hex number such as 0x4501.

Return Value

success: API_OK, fail: API_ERROR



Remarks

- 1 Report the version information when requesting technique support.
- 2 Camera index is ZERO based.

CameraGetCount

Get to total camera count connected to the computer.

Syntax

```
API_STATUS __stdcall CameraGetCount(int *count)
```

Parameters

[out] **count** total camera count connected.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Valid camera index is 0 to count-1.

CameraGetName

Get the camera name and model name.

Syntax

```
API_STATUS __stdcall CameraGetName(int index, char *name, char *model)
```

Parameters

[in] **index** camera index.

[out] **name** camera name, readable string.

[out] **model** camera model name readable string.

Return Value

success: API_OK, fail: API_ERROR

Remarks



Examples

```
int count=0;

char *name=new char[255];

char *model=new char[255];

CameraGetCount(&count);

for(int i=0; i<count; i++)

    CameraGetName(i, name, model);
```

CameraGetID

Get camera model id and product id.

Syntax

```
API_STATUS __stdcall CameraGetID (int index,int *modelID, int *productID)
```

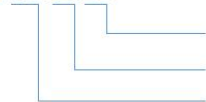
Parameters

[in] **index** camera index.

[out] **modelID** camera model id, internal use only.

[out] **productID** numeric model id of the product.

0x 0 30 0



Chroma 0 Color B Mono
Pixels(10 times of Mega pixels)
Interface 0 USB2.0 3 USB3.0

Value 0x0300 represents a 3M pixel color camera, 0x013B represents a 1.3M pixel Mono camera, etc,

Return Value

success: API_OK, fail: API_ERROR

CameraGetLastError

Get the last error code reported by API. It's a thread local variable. Every thread have its own error code. Windows XP does not support thread local variable, the error



code is application based.use DLL under sdk/SDK/WindowsXP to avoid application crash.

Syntax

```
API_STATUS __stdcall CameraGetLastError(int *last_error)
```

Parameters

[in] **last_error** error code.

Return Value

success: API_OK,fail: API_ERROR

Remarks

When API call returns API_ERROR, call CameraGetLastError to retrieve the error code.



Code	Constant	Description
1	CAMERA_ERROR_INDEX_OUT_OF_RANGE	Camera index out of range.
2	CAMERA_ERROR_PARAMETER_UNREASONABLE	Parameter unreasonable.
3	CAMERA_ERROR_NOT_INITIALIZED	Camera have not been initialized.
4	CAMERA_ERROR_USB_DISCONNECTED	USB is disconnected from computer.
5	CAMERA_ERROR_NO_CAMERA_FOUND	No camera.
6	CAMERA_ERROR_INITIALIZED_ALREADY	Camera is initialized already.
7	CAMERA_ERROR_RECONNECT_FAILURE	Reconnect fail.
8	CAMERA_ERROR_NOT_SUPPORTED	Feature not supported.
9	CAMERA_ERROR_CONTROL_PIPE	Command channel invalid.
10	CAMERA_ERROR_READ_PIPE	Read command data error.
11	CAMERA_ERROR_WRITE_PIPE	Write command data error.
12	CAMERA_ERROR_IMAGE_SIZE	Image size is not correct.
13	CAMERA_ERROR_DATA_PIPE	Data channel invalid.
14	CAMERA_ERROR_BAD_FRAME	Bad image frame.
15	CAMERA_ERROR_USER_ABORT	User abort query image by free camera
16	CAMERA_ERROR_DATA_XFER	Data transfer error.
17	CAMERA_ERROR_DATA_TIMEOUT	Timeout to query image.
18	CAMERA_ERROR_OUT_OF_MEMORY	Out of memory
19	CAMERA_ERROR_RESET	Reinitialize usb connection.
0	API_OK	No error

2.2 Camera Control

This group of function must be called after initializing the camera.



CameraInit

Initialize the camera.

Syntax

```
API_STATUS __stdcall CameraInit(int index)
```

Parameters

[in] **index** camera index.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Index is 0 based. Call CameraInit more than one time will take no further effect. Usually user have to call CameraFree before call CameraInit again.

Examples

```
CameraInit(n); //n=0,1,2,...,count-1
```

CameraFree

Release the camera resource.

Syntax

```
API_STATUS __stdcall CameraFree(int device_id)
```

Parameters

[in] **device_id** camera index.

Return Value

success: API_OK, fail: API_ERROR

CameraReset

Reset the camera to recover from the data channel transfer error.

Syntax



API_STATUS __stdcall CameraReset(int device_id)

Parameters

[in] **device_id** camera index.

Return Value

success: API_OK, fail: API_ERROR

Remarks

CameraReset is called automatically in continuous mode when encountering transfer error. If CameraQueryImage returns API_ERROR in trigger mode, if a trigger is issued and timeout error happened, a CameraReset call is needed.

CameraReconnect

Simulate a USB reconnect from the computer, plug out/plug in.

Syntax

API_STATUS __stdcall CameraReconnect (int device_id)

Parameters

[in] **device_id** camera index.

Return Value

success: API_OK, fail: API_ERROR

Remarks

CameraReconnect will make the windows driver release the device temporary. Application can receive the windows message WM_DEVICECHANGE. Application can reinitialize(CameraFree/CameraInit) the camera based on the message.

CameraSetTimeout

Set the read image timeout.

Syntax



```
API_STATUS __stdcall CameraSetTimeout (int device_id, int timeout)
```

Parameters

[in] **device_id** camera index.

[in] **timeout** maximum time in milliseconds.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Default and minimum timeout is 500ms. If timeout happens, a CameraReset will be called automatically in continuous mode. If the exposure time is bigger than 500ms, timeout should be set to a bigger value.

CameraGetTimeout

Get the read image timeout.

Syntax

```
API_STATUS __stdcall CameraGetTimeout (int device_id, int *timeout)
```

Parameters

[in] **device_id** camera index.

[out] **timeout** maximum time in milliseconds.

Return Value

success: API_OK, fail: API_ERROR

2.3 Camera Setting

This group of function must be called after initializing the camera.

2.3.1 Resolution and ROI

CameraGetResolutionCount

Get the preset resolution count.



Syntax

```
API_STATUS __stdcall CameraGetResolutionCount(int device_id, int *count)
```

Parameters

[in] **device_id** camera index.

[out] **count** preset resolution count.

Return Value

success: API_OK, fail: API_ERROR

CameraGetResolutionMax

Get the maximum resolution of the camera, it is the maximum image size.

Syntax

```
API_STATUS __stdcall CameraGetResolutionMax(int device_id, int *width, int *height)
```

Parameters

[in] **device_id** camera index.

[out] **width** maximum width.

[out] **height** maximum height.

Return Value

success: API_OK, fail: API_ERROR

CameraGetResolutionMode

Get resolution mode on change to a small preset resolution.

Syntax

```
API_STATUS __stdcall CameraGetResolutionMode(int device_id, int *mode)
```

Parameters

[in] **device_id** camera index.

[out] **mode** resolution mode.

Return Value



success: API_OK, fail: API_ERROR

Remarks

Resolution mode will take effect only if you set a resolution less(equal) than half of the maximum resolution or line scan mode:

CAMERA_RESOLUTION_CROPPING: crop a certain region.

CAMERA_RESOLUTION_SKIPPING: skip 1 row/column pixels.

CAMERA_RESOLUTION_BINNING: merge neighbouring pixels.

CAMERA_RESOLUTION_LINESCAN: line scan mode.

CameraSetResolutionMode

Set resolution mode on change to a small preset resolution.

Syntax

```
API_STATUS __stdcall CameraSetResolutionMode(int device_id, int mode)
```

Parameters

[in] **device_id** camera index.

[in] **mode** resolution mode.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Refer to **CameraGetResolutionMode**.

CameraGetResolution

Get width and height of a certain preset resolution.

Syntax

```
API_STATUS __stdcall CameraGetResolution(int device_id, int index, int *width,
int *height)
```

Parameters

[in] **device_id** camera index.



[in] **index** preset resolution's index.

[out] **width** resolution width.

[out] **height** resolution height.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Index is 0 based, range from 0, count-1, where count is the preset resolution count, which can be get by **CameraGetResolutionCount**.

Examples

```
int count=0;
CameraGetResolutionCount(m_index,&count);
for(int j=0;j<count;j++)
{
    CameraGetResolution(m_index, j,&width, &height);
    ...
}
```

CameraSetResolution

Set the camera's resolution to a preset resolution.

Syntax

```
API_STATUS __stdcall CameraSetResolution(int device_id, int index, int *width,
int *height)
```

Parameters

[in] **device_id** camera index.

[in] **index** preset resolution's index.

[out] **width** resolution width.

[out] **height** resolution height.

Return Value



success: API_OK,fail: API_ERROR

Remarks

Refer to **CameraGetResolution**.

CameraSetROI

Set Region of interest(ROI).

Syntax

```
API_STATUS __stdcall CameraSetROI(int device_id, int offset_width, int offset_height,
int width, int height);
```

Parameters

[in] **device_id** camera index.

[in] **offset_width** horizontal start at the full image.

[in] **offset_height** vertical start at the full image.

[in] **width** ROI width.

[in] **height** ROI height.

Return Value

success: API_OK,fail: API_ERROR

Remarks

Value of **offset_width** and **offset_height** need to be smaller than the maximum resolution and satisfy

offset_width + width <= max horizontal resolution

offset_height + height <= max vertical resolution

offset_height,height need to be divisible by 4. **offset_width, width** need to be divisible by 4, and even need to satisfy additional criteria on the following specific camera models:

offset_width, width need to be divisible by 32	
JHUM501/JHUM501B	JHUM1800
JHUM202B	JHUM502B



offset_width, width need to be divisible by 16	
JHUM32B	JHUM132B
JHUM131/B	
offset_width, width need to be divisible by 8	
JHUM201/B	JHUM600/B
JHUM1200/B	JHUM2000/B
JHUM804/B	JHUM1204/B
JHSM400/B	
offset_width , offset_height need to be divisible by 8, width, height need to be divisible by 16	
JHUM306/B	JHUM506/B
JHUM806/B	

2.3.2 Exposure and Gain

CameraSetGain

Set gain.

Syntax

```
API_STATUS __stdcall CameraSetGain(int device_id, int gain)
```

Parameters

[in] **device_id** camera index.

[in] **gain** camera gain.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Range from 1-255, noise may be introduced by gain. The value below 32 belongs to analog gain, which have little noise. When gain bigger than 32, the digital gain will introduce more noise.



CameraGetGain

Get gain.

Syntax

```
API_STATUS __stdcall CameraGetGain(int device_id, int *gain)
```

Parameters

[in] **device_id** camera index.

[out] **gain** camera gain. Range from 1-255.

Return Value

success: API_OK, fail: API_ERROR

CameraSetAdvancedGain

Set advanced gain. Only valid for 1.2M/2M pixel camera.

Syntax

```
API_STATUS __stdcall CameraSetAdvancedGain(int device_id, int advanced_gain)
```

Parameters

[in] **device_id** camera index.

[in] **advanced_gain** camera's advanced gain.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Advanced gain's default value 0, and have range from 0-5, it is digital gain, which will introduce noise.

CameraGetAdvancedGain

Get advanced gain. Only valid for 1.2M/2M pixel camera.

Syntax



```
API_STATUS __stdcall CameraGetAdvancedGain(int device_id, int *advanced_gain)
```

Parameters

[in] **device_id** camera index.

[out] **advanced_gain** camera's advanced gain. Range from 0-5.

Return Value

success: API_OK, fail: API_ERROR

CameraSetAGC

Enable/disable auto gain control.

Syntax

```
API_STATUS __stdcall CameraSetAGC(int device_id, bool agc)
```

Parameters

[in] **device_id** camera index.

[in] **agc** enable auto gain control. True for enable, false for disable.

Return Value

success: API_OK, fail: API_ERROR

CameraGetAGC

Get auto gain control status.

Syntax

```
API_STATUS __stdcall CameraGetAGC(int device_id, bool *agc)
```

Parameter

[in] **device_id** camera index.

[out] **agc** auto gain control.

Return Value

success: API_OK, fail: API_ERROR



CameraSetExposure

Set exposure.

Syntax

```
API_STATUS __stdcall CameraSetExposure(int device_id, int exposure)
```

Parameters

[in] **device_id** camera index.

[in] **exposure** camera's exposure value.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Exposure ranges from 1-32768. Exposure time can be get by **CameraGetExposureTime** after set exposure value. If the exposure time bigger than 500ms, **CameraSetTimeout** must be called to set a larger timeout.

CameraGetExposure

Get exposure.

Syntax

```
API_STATUS __stdcall CameraGetExposure(int device_id, int *exposure)
```

Parameters

[in] **device_id** camera index.

[out] **exposure** camera's exposure value.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Refer to **CameraGetExposure**.



CameraGetExposureUnit

Get the exposure time represents by exposure value 1.

Syntax

```
API_STATUS __stdcall CameraGetExposureUnit(int device_id, double *exposure_unit)
```

Parameters

[in] **device_id** camera index.

[out] **exposure_unit** exposure time in milliseconds.

Return Value

success: API_OK, fail: API_ERROR

CameraGetExposureTime

Get the exposure time.

Syntax

```
API_STATUS __stdcall CameraGetExposureTime(int device_id, double *exposure_time)
```

Parameters

[in] **device_id** camera index.

[out] **exposure_time** current exposure time in milliseconds.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Exposure time equals exposure_value*exposure_unit.

CameraSetAEC

Enable/disable auto exposure control.

Syntax

```
API_STATUS __stdcall CameraSetAEC(int device_id, bool aec)
```



Parameters

[in] **device_id** camera index.

[in] **agc** enable auto exposure control. True for enable, false for disable.

Return Value

success: API_OK, fail: API_ERROR

CameraGetAEC

Get auto exposure control status.

Syntax

```
API_STATUS __stdcall CameraGetAEC(int device_id, bool *aec)
```

Parameters

[in] **device_id** camera index.

[out] **aec** auto exposure control status.

Return Value

success: API_OK, fail: API_ERROR

CameraSetAETarget

Set the target illuminance of auto exposure. If AEC is enabled, the image's illuminance will coverage to the target value.

Syntax

```
API_STATUS __stdcall CameraSetAETarget(int device_id, int target)
```

Parameters

[in] **device_id** camera index.

[in] **target** illuminance target.

Return Value

success: API_OK, fail: API_ERROR



CameraGetAETarget

Get the target illuminance of auto exposure.

Syntax

```
API_STATUS __stdcall CameraGetAETarget(int device_id, int *target)
```

Parameters

[in] **device_id** camera index.

[out] **target** illuminance target.

Return Value

success: API_OK, fail: API_ERROR

CameraSetAntiFlicker

Set anti-flicker mode to eliminate stripes caused by AC light source.

Syntax

```
API_STATUS __stdcall CameraSetAntiFlicker (int device_id, int flicker)
```

Parameters

[in] **device_id** camera index.

[in] **flicker** flicker frequency, 1 for 50HZ , 2 for 60HZ

Return Value

success: API_OK, fail: API_ERROR

Remarks

Anti-flicker is implemented by set a proper exposure value, so it will only take effect when auto exposure is enabled.

CameraGetAntiFlicker

Get anti-flicker mode.

Syntax



*API_STATUS __stdcall CameraGetAntiFlicker (int device_id, int *flicker)*

Parameters:

[in] **device_id** camera index.

[out] **flicker** flicker frequency, 1 for 50HZ , 2 for 60HZ

Return Value

success: API_OK, fail: API_ERROR

Remarks

Refer to CameraSetAntiFlicker.

2.3.3 Image Setting

CameraGetAvg

Get all pixels' average of the last image, a.k.a. illuminance.

Syntax

*API_STATUS __stdcall CameraGetAvg(int device_id, int *avg)*

Parameters

[in] **device_id** camera index.

[out] **avg** pixels' average value.

Return Value

success: API_OK, fail: API_ERROR

CameraSetGamma

Set image gamma.

Syntax

API_STATUS __stdcall CameraSetGamma(int device_id, double gamma)

Parameters

[in] **device_id** camera index.

[in] **gamma** image gamma. Ranges from 0.3-3.0.



Return Value

success: API_OK, fail: API_ERROR

CameraGetGamma

Get image gamma.

Syntax

```
API_STATUS __stdcall CameraGetGamma(int device_id, double *gamma)
```

Parameters

[in] **device_id** camera index.

[out] **gamma** image gamma. Ranges from 0.3-3.0.

Return Value

success: API_OK, fail: API_ERROR

CameraSetContrast

Set image contrast.

Syntax

```
API_STATUS __stdcall CameraSetContrast(int device_id, double contrast)
```

Parameters

[in] **device_id** camera index.

[in] **contrast** image contrast. Ranges from 0.3-2.0.

Return Value

success: API_OK, fail: API_ERROR

CameraGetContrast

Get image contrast.

Syntax

```
API_STATUS __stdcall CameraGetContrast(int device_id, double *contrast)
```



Parameters

[in] **device_id** camera index.

[out] **contrast** image contrast. Ranges from 0.3-2.0.

Return Value

success: API_OK, fail: API_ERROR

CameraSetSaturation

Set color saturation.

Syntax

```
API_STATUS __stdcall CameraSetSaturation(int device_id, double saturation)
```

Parameters

[in] **device_id** camera index.

[out] **saturation** image saturation. Ranges from 0.0-2.0.

Return Value

success: API_OK, fail: API_ERROR

CameraGetSaturation

Get image saturation.

Syntax

```
API_STATUS __stdcall CameraGetSaturation(int device_id, double *saturation)
```

Parameters

[in] **device_id** camera index.

[out] **saturation** image saturation. Ranges from 0.0-2.0.

Return Value

success: API_OK, fail: API_ERROR



CameraSetBlackLevel

Set image black level, all raw pixel value will subtract black level. Image will became dark with high black level.

Syntax

```
API_STATUS __stdcall CameraSetBlackLevel(int device_id, int black)
```

Parameters

[in] **device_id** camera index.

[in] **black** black level value. Ranges from 0-255.

Return Value

success: API_OK, fail: API_ERROR

CameraGetBlackLevel

Get image black level.

Syntax

```
API_STATUS __stdcall CameraGetBlackLevel(int device_id, int *black)
```

Parameters

[in] **device_id** camera index.

[out] **black** black level value. Ranges from 0-255.

Return Value

success: API_OK, fail: API_ERROR

CameraSetEnhancement

Enable/disable color enhancement.

Syntax

```
API_STATUS __stdcall CameraSetEnhancement (int device_id, bool enhance)
```

Parameters



[in] **device_id** camera index.

[in] **enhance** color enhance, true for enable and false for disable.

Return Value

success: API_OK, fail: API_ERROR

Remarks

usbVideo can preset a CCM matrix to camera. If enhancement is enabled, the CCM will apply to every pixel. In general, CCM is used to enhance the color saturation and improve the color accuracy.

CameraGetEnhancement

Get color enhancement status.

Syntax

```
API_STATUS __stdcall CameraGetEnhancement (int device_id, bool *enhance)
```

Parameters

[in] **device_id** camera index.

[out] **enhance** color enhance, true for enable and false for disable.

Return Value

success: API_OK, fail: API_ERROR

CameraSetGlobalReset

Enable/disable global reset shutter.

Syntax

```
API_STATUS __stdcall CameraSetGlobalReset (int device_id, bool en)
```

Parameters

[in] **device_id** camera index.

[in] **en** enable global reset, true for enable and false for disable.

Return Value



success: API_OK, fail: API_ERROR

Remarks

Only part of rolling camera support global reset feature. Together with global reset and mechanical shutter or strobe light, the camera can achieve the similar result of global shutter camera, capturing motion object with no distortion.

CameraGetGlobalReset

Get global reset shutter status.

Syntax

```
API_STATUS __stdcall CameraGetGlobalReset (int device_id, bool *en)
```

Parameters

[in] **device_id** camera index.

[out] **en** enable global reset, true for enable and false for disable or not support.

Return Value

success: API_OK, fail: API_ERROR

CameraSetDenoise

Enable/disable software noise reduction function. Enable this function can let software fix the a white spot by comparing the pixel value with its surrounding ones, if it's too large, this pixel value is replaced by the average value of its surrounding pixels.

Syntax

```
API_STATUS __stdcall CameraSetDenoise (int device_id, bool denoise)
```

Parameters

[in] **device_id** camera index.

[in] **denoise** enable/disable noise reduction function, true for enable and false



for disable.

Return Value

success: API_OK, fail: API_ERROR

Remarks

This function need extra computational power, which may slow down the fps.

CameraGetDenoise

Get noise reduction function status.

Syntax

```
API_STATUS __stdcall CameraGetDenoise (int device_id, bool *denoise)
```

Parameters

[in] **device_id** camera index.

[out] **denoise** return noise reduction function status, true for enable and false for disable.

Return Value

success: API_OK, fail: API_ERROR

CameraSetFPN

Enable/disable FPN fix function. The Fix Pattern Noise(FPN) is caused by the defect of image sensor process. It's a kind of stripe noise on the image, neither vertical or horizontal. By enable the FPN fix function, it can use the FPN data calibrated on usbVideo (or pre-loaded by factory) to compensate the noise.

Syntax

```
API_STATUS __stdcall CameraSetFPN (int device_id, bool fpn)
```

Parameters

[in] **device_id** camera index.



[in] **fpn** enable/disable fpn fix function, true for enable and false for disable.

Return Value

success: API_OK, fail: API_ERROR

Remarks

This function need extra computational power, which may slow down the fps.

CameraGetFPN

Get FPN fix function status.

Syntax

```
API_STATUS __stdcall CameraGetFPN (int device_id, bool *fpn)
```

Parameters

[in] **device_id** camera index.

[out] **fpn** return fpn function status, true for enable and false for disable.

Return Value

success: API_OK, fail: API_ERROR

CameraEnableLSC

Enable/disable Lens Shading Correction(LSC). LSC is a technology to flatten the nonuniform intensity of the image caused by lens.

Syntax

```
API_STATUS __stdcall CameraEnableLSC (int device_id, bool en)
```

Parameters

[in] **device_id** camera index.

[in] **en** enable/disable LSC, true for enable and false for disable.

Return Value



success: API_OK, fail: API_ERROR

Remarks

This function need extra LSC data parameter to take effect. It need more computation which may slow down the fps.

Set parameters LSC needed before enable LSC correction. The algorithm using a two-order functions to fit the nonuniform. For a pixel located distance r to the optical center on the image, the intensity is present as

$$q * r^2 + p * r + c$$

The optical center is usually located at the image center, but it may offset a little, which is present by dx0 and dy0;

These parameters may obtain the a calibrate image. Capture a uniform white scene. Different lenses need to calibrate their own parameters.

CameraGetLSC

Get LSC function status.

Syntax

```
API_STATUS __stdcall CameraGetLSC(int device_id, bool *en)
```

Parameters

[in] **device_id** camera index.

[out] **en** return LSC function status, true for enable and false for disable.

Return Value

success: API_OK, fail: API_ERROR

CameraEnableLDC

Enable/disable Lens Distortion Correction(LDC). LDC is a technology to flatten the distorted image caused by lens' distortion.

Syntax



API_STATUS __stdcall CameraEnableLDC(int device_id, bool en)

Parameters

[in] **device_id** camera index.

[in] **en** enable/disable LDC, true for enable and false for disable.

Return Value

success: API_OK, fail: API_ERROR

Remarks

This function need extra LDC data parameter to take effect. It need more computation which may slow down the fps.

Set parameters before enable LDC correction. The distortion parameters of the camera can be calibrated using using the method of Zhang Zhengyou calibration. Opencv and Matlab provided some standard procedures and tools. LDC using the intrinsic parameter $[f_x, f_y, c_x, c_y]$ and lens distortion parameter $[k_1, k_2, p_1, p_2, k_3]$ to compensate the distortion.

CameraGetLDC

Get LDC function status.

Syntax

*API_STATUS __stdcall CameraGetLDC(int device_id, bool *en)*

Parameters

[in] **device_id** camera index.

[out] **en** return LDC function status, true for enable and false for disable.

Return Value

success: API_OK, fail: API_ERROR

CameraSetLUT

Set the Look Up Table(LUT). The LUT is used to convert 12bit pixel data to 8 bit



pixel data.

Syntax

```
API_STATUS __stdcall CameraSetLUT(int device_id, unsigned char *lut, int len)
```

Parameters

[in] **device_id** camera index.

[in] **lut** look up table, a 12bit to 8bit data map.

[in] **len** lut data length.

Return Value

success: API_OK, fail: API_ERROR

Remarks

This function need extra computational power, which may slow down the fps.

CameraGetLUT

Get the Look Up Table(LUT). The LUT is used to convert 12bit pixel data to 8 bit pixel data.

Syntax

```
API_STATUS __stdcall CameraGetLUT(int device_id, unsigned char *lut, int *len)
```

Parameters

[in] **device_id** camera index.

[out] **lut** look up table, a 12bit to 8bit data map.

[inout] **len** return the LUT size if lut is NULL, or indicate the valid data length of lut otherwise.

Return Value

success: API_OK, fail: API_ERROR



2.3.4 Frame Control

CameraSetHighspeed

Enable/disable high speed image transfer.

Syntax

```
API_STATUS __stdcall CameraSetHighspeed (int device_id, bool high)
```

Parameters

[in] **device_id** camera index.

[in] **high** speed, true for high speed, false for low speed.

Return Value

success: API_OK, fail: API_ERROR

Remarks

High speed has double data rate of low speed. The nominal speed of USB2.0 is 480Mbps, when the actually transfer speed is about 40MBps. The camera run at low speed by default.

Computer usually has two high speed USB channel on the CPU/motherboard. If two camera plugged to separate channel, both of them can run at high speed, otherwise they can both run at low speed.

CameraGetHighspeed

Get high speed transfer status.

Syntax

```
API_STATUS __stdcall CameraGetHighspeed (int device_id, bool *high)
```

Parameters

[in] **device_id** camera index.

[out] **high** speed, true for high speed, false for low speed.

Return Value



success: API_OK, fail: API_ERROR

Remarks

Refer to CameraSetHighspeed.

CameraSetDelay

Set delay, delay is insert into image line interval and image frame interval. It can be used to adjust the frame rate, higher delay result in lower frame rate.

Syntax

```
API_STATUS __stdcall CameraSetDelay(int device_id, int delay)
```

Parameters

[in] **device_id** camera index.

[in] **delay** delay value.

Return Value

success: API_OK, fail: API_ERROR

Remarks

If computer's performance is poor, a delay may help to reduce the frame rate rate, then reduce the process load of the CPU. The default value is 0.

CameraGetDelay

Get delay.

Syntax

```
API_STATUS __stdcall CameraGetDelay(int device_id, int *delay)
```

Parameters

[in] **device_id** camera index.

[out] **delay** delay value.

Return Value

success: API_OK, fail: API_ERROR



Remarks

Refer to `CameraSetDelay`.

CameraSetFrameRate

Set fps constrain status, and set constrain fps.

Syntax

```
API_STATUS __stdcall CameraSetFrameRate(int device_id, bool en, double fps)
```

Parameters

[in] **device_id** camera index.

[in] **en** Enable or disable fps constrain.

[in] **fps** fps expected.

Return Value

success: `API_OK`, fail: `API_ERROR`

Remarks

Constrains the frame rate can reduce the data rate, save CPU's computational power. It's recommended to reduce the frame rate to match the actual needs.

Only parts of models support this feature. Camera's max frame rate is limited by the sensor output. Set a value bigger than the max frame rate will just be ignored. Set **en** to false will return to the max frame rate.

CameraGetFrameRate

Check if fps constrain enabled, and get the constrain fps.

Syntax

```
API_STATUS __stdcall CameraGetFrameRate(int device_id, bool *en, double *fps)
```

Parameters



[in] **device_id** camera index.

[out] **en** return fps constrain enable status.

[out] **fps** return the fps expected.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Refer to **CameraSetFrameRate**.

CameraSetMirrorX

Set horizontal mirror of the image.

Syntax

```
API_STATUS __stdcall CameraSetMirrorX(int device_id, bool mx)
```

Parameters

[in] **device_id** camera index.

[in] **mx** horizontal mirror, true for mirror, false for none action.

Return Value

success: API_OK, fail: API_ERROR

Examples

```
//flip the image horizontally.  
bool flag=false;  
CameraGetMirrorX(m_index,&flag);  
if(flag)  
    CameraSetMirrorX(m_index,false);  
else  
    CameraSetMirrorX(m_index,true);
```



CameraGetMirrorX

Get horizontal mirror status of the image.

Syntax

```
API_STATUS __stdcall CameraGetMirrorX(int device_id, bool *mx)
```

Parameters

[in] **device_id** camera index.

[out] **mx** horizontal mirror, true for mirror, false for none action.

Return Value

success: API_OK, fail: API_ERROR

CameraSetMirrorY

Set vertical mirror status of the image.

Syntax

```
API_STATUS __stdcall CameraSetMirrorY(int device_id, bool my)
```

Parameters

[in] **device_id** camera index.

[in] **my** vertical mirror, true for mirror, false for none action.

Return Value

success: API_OK, fail: API_ERROR

CameraGetMirrorY

Get vertical mirror of the image.

Syntax

```
API_STATUS __stdcall CameraGetMirrorY(int device_id, bool *my)
```

Parameters:

[in] **device_id** camera index.



[out] **my** vertical mirror, true for mirror, false for none action.

Return Value

success: API_OK, fail: API_ERROR

CameraSetRotate

Set image rotation.

Syntax

```
API_STATUS __stdcall CameraSetRotate(int device_id, int rotate)
```

Parameters

[in] **device_id** camera index.

[in] **rotate** image rotation, options include[90, 180, 270]

Return Value

success: API_OK, fail: API_ERROR

2.3.5 White Balance

CameraSetAWB

Enable/disable auto white balance.

Syntax

```
API_STATUS __stdcall CameraSetAWB (int device_id, bool awb)
```

Parameters

[in] **device_id** camera index.

[in] **awb** true for enable and false for disable.

Return Value

success: API_OK, fail: API_ERROR



CameraGetAWB

Get auto white balance status.

Syntax

```
API_STATUS __stdcall CameraGetAWB (int device_id, bool *awb)
```

Parameters

[in] **device_id** camera index.

[out] **awb** true for enable and false for disable.

Return Value

success: API_OK, fail: API_ERROR

CameraSetWBGain

Set white balance R/G/B gain.

Syntax

```
API_STATUS __stdcall CameraSetWBGain(int device_id, double rg, double gg, double bg)
```

Parameters

[in] **device_id** camera index.

[in] **rg** red gain, ranges from 0.0-3.0.

[in] **gg** green gain, ranges from 0.0-3.0, usually fixed to 1.0.

[in] **bg** blue gain, ranges from 0.0-3.0.

Return Value

success: API_OK, fail: API_ERROR

CameraGetWBGain

Get white balance R/G/B gain.

Syntax

```
API_STATUS __stdcall CameraGetWBGain(int device_id, double *rg, double *gg, double *bg)
```



Parameters

- [in] **device_id** camera index.
- [out] **rg** red gain, ranges from 0.0-3.0.
- [out] **gg** green gain, ranges from 0.0-3.0.
- [out] **bg** blue gain, ranges from 0.0-3.0.

Return Value

success: API_OK, fail: API_ERROR

CameraOnePushWB

Achieve white balance by one or more calls.

Syntax

```
API_STATUS __stdcall CameraOnePushWB(int device_id)
```

Parameters

- [in] **device_id** camera index.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Before call CameraOnePushWB, you have to make the camera shooting on a gray (white) object, and not overexposure. Then call this function one or more time until the color close to the actually object.

2.3.6 Line Scan

Some model support line scan mode. The line scan mode on area image sensor is implemented by utilizing the center *line_height* lines pixels of the sensor. *Lines_height* can be set to 2-32. A image is composed by multiple exposures of these lines. Exposure can be automatic or by external trigger, which can be set by *line_trigger*. In automatic exposure mode, *line_time* is used to tune the scan speed to match the scanning speed.



CameraSetLineHeight

Set line height, line count in one exposure.

Syntax

```
API_STATUS __stdcall CameraSetLineHeight(int device_id, int line_height)
```

Parameters

[in] **device_id** camera index.

[in] **line_height** line height.

Return Value

success: API_OK, fail: API_ERROR

Remarks

The acquisition frame rate will increase as the line_height increase. Line scan mode usually need small line height, 2 is recommended.

CameraGetLineHeight

Get line height in line scan.

Syntax

```
API_STATUS __stdcall CameraGetLineHeight(int device_id, int *line_height)
```

Parameters

[in] **device_id** camera index.

[out] **line_height** return the line height.

Return Value

success: API_OK, fail: API_ERROR

CameraSetLineTrigger

Set line trigger mode.

Syntax



```
API_STATUS __stdcall CameraSetLineTrigger(int device_id, int line_trigger)
```

Parameters

[in] **device_id** camera index.

[in] **line_trigger** line trigger mode, possible value:

CAMERA_LINE_TRIGGER_AUTO trigger as camera maximum speed.

CAMERA_LINE_TRIGGER_EXTERNAL trigger by external signal.

Return Value

success: API_OK, fail: API_ERROR

CameraGetLineTrigger

Get line trigger mode.

Syntax

```
API_STATUS __stdcall CameraGetLineTrigger(int device_id, int *line_trigger)
```

Parameters

[in] **device_id** camera index.

[out] **line_trigger** return the line trigger mode.

Return Value

success: API_OK, fail: API_ERROR

2.3.7 Storage

CameraWriteUserData

Write user defined data, length within 64 byte.

Syntax

```
API_STATUS __stdcall CameraWriteUserData (int device_id, char data[], int length)
```

Parameters

[in] **device_id** camera index.

[in] **data[]** data array contains user defined data.



[in] **length** data array length, less than 64.

Return Value

success: API_OK, fail: API_ERROR

Examples

```
char data[20]={10,21,5,0,30,1,2,3,4,5,5,4,3,2,1,10,20,30,40,50};  
CameraWriteUserData(m_index,data,20);
```

CameraReadUserData

Write user defined data, length within 64 byte.

Syntax

```
API_STATUS __stdcall CameraReadUserData (int device_id, char data[], int length)
```

Parameters

[in] **device_id** camera index.

[out] **data[]** data array.

[in] **length** data array length, less than 64.

Return Value

success: API_OK, fail: API_ERROR

Examples

```
char data[20];  
CameraReadUserData(m_index,data,20);  
CString temp;  
CString str;  
for(int i=0;i<20;i++)  
{  
    temp.Format("%d",data[i]);  
    str+=temp;  
}  
AfxMessageBox(str);
```



CameraReadSerialNumber

Read serial number(Each camera has a unique serial number).

Syntax

```
API_STATUS __stdcall CameraReadSerialNumber(int device_id, char id[], int length)
```

Parameters

[in] **device_id** camera index.

[out] **id[]** serial number.

[in] **length** array length, must be 12.

Return Value

success: API_OK, fail: API_ERROR

Examples

```
char id[12];  
CameraReadSerialNumber(m_index, id, 12);  
//result: id[12]=['a','1','b','2','5','6','7','8','9','c','2','3'];
```

CameraSaveParameter

Save parameter in camera.

Syntax

```
API_STATUS __stdcall CameraSaveParameter(int device_id, int group_no)
```

Parameters

[in] **device_id** camera index.

[in] **group_no** parameters group, 0 or 1.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Every camera can save two group of independent parameters with group



number 0 or 1. The parameters can be restored when calling CameraInit, or call CameraLoadParameter. CameraInit will load parameters in group 0.

CameraLoadParameter

Load saved parameter in camera.

Syntax

```
API_STATUS __stdcall CameraLoadParameter(int device_id, int group_no)
```

Parameters

[in] **device_id** camera index.

[in] **group_no** parameters group, 0 or 1.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Refer to CameraSaveParameter.

2.4 External I/O Control

This group of function must be called after initializing the camera..

CameraEnableStrobe

Enable/disable strobe/flash output.

Syntax

```
API_STATUS __stdcall CameraEnableStrobe(int device_id, bool en)
```

Parameters

[in] **device_id** camera index.

[in] **en** strobe output, true for enable, false for disable.

Return Value

success: API_OK, fail: API_ERROR



CameraGetStrobe

Get strobe output enable status.

Syntax

```
API_STATUS __stdcall CameraGetStrobe(int device_id, bool *en)
```

Parameters

[in] **device_id** camera index.

[out] **en** return strobe output status, true for enable, false for disable.

Return Value

success: API_OK, fail: API_ERROR

CameraSetStrobePolarity

Set strobe output's polarity.

Syntax

```
API_STATUS __stdcall CameraSetStrobePolarity(int device_id, bool high)
```

Parameters

[in] **device_id** camera index.

[in] **high** strobe polarity, true for high, false for low level.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Only some models support this function.

CameraGetStrobePolarity

Get strobe output's polarity.



Syntax

```
API_STATUS __stdcall CameraGetStrobePolarity(int device_id, bool *high)
```

Parameters

[in] **device_id** camera index.

[out] **high** return strobe polarity, true for high, false for low level.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Only some models support this function.

CameraSetTriggerPolarity

Set trigger signal polarity, some camera support rising edge and falling edge. If not support both edge, the default trigger polarity is rising edge.

Syntax

```
API_STATUS __stdcall CameraSetTriggerPolarity(int device_id, bool high)
```

Parameters

[in] **device_id** camera index.

[in] **high** trigger polarity, true for falling edge, false for rising edge.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Only some models support this function.

CameraGetTriggerPolarity

Get trigger input polarity, rising or falling edge.

Syntax

```
API_STATUS __stdcall CameraGetTriggerPolarity(int device_id, bool *high)
```



Parameters

[in] **device_id** camera index.

[out] **high** return trigger polarity, true for falling edge, false for rising edge.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Only some models support this function.

CameraSetTriggerDelay

Set trigger delay, start from trigger signal.

Syntax

```
API_STATUS __stdcall CameraSetTriggerDelay(int device_id, int delay_us)
```

Parameters

[in] **device_id** camera index.

[in] **delay_us** trigger delay in us.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Only some models support this function.

CameraGetTriggerDelay

Get trigger delay, starting from trigger signal.

Syntax

```
API_STATUS __stdcall CameraGetTriggerDelay(int device_id, int*delay_us)
```

Parameters

[in] **device_id** camera index.

[out] **delay_us** return trigger delay, in us.



Return Value

success: API_OK, fail: API_ERROR

Remarks

Only some models support this function.

CameraSetStrobeDelay

Set strobe delay, start from trigger signal.

Syntax

```
API_STATUS __stdcall CameraSetStrobeDelay(int device_id, int delay_us)
```

Parameters

[in] **device_id** camera index.

[in] **delay_us** strobe delay in us.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Only some models support this function.

CameraGetStrobeDelay

Get strobe delay, starting from trigger signal.

Syntax

```
API_STATUS __stdcall CameraGetStrobeDelay(int device_id, int*delay_us)
```

Parameters

[in] **device_id** camera index.

[out] **delay_us** return strobe delay, in us.

Return Value

success: API_OK, fail: API_ERROR

Remarks



Only some models support this function.

CameraSetStrobeDuration

Set strobe output duration. Strobe duration equals to exposure time if strobe duration set to 0.

Syntax

```
API_STATUS __stdcall CameraSetStrobDuration(int device_id, int duration_us)
```

Parameters

[in] **device_id** camera index.

[in] **duration_us** strobe duration in us.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Only some models support this function.

CameraGetStrobeDuration

Get strobe duration, strobe signal length.

Syntax

```
API_STATUS __stdcall CameraGetStrobeDuration(int device_id, int*duration_us)
```

Parameters

[in] **device_id** camera index.

[out] **duration_us** return strobe duration in us.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Only some models support this function.



CameraSetDebouncerTime

Set debouncer time for trigger signal. The trigger signal will be ignored if the width of the signal less than debouncer time.

Syntax

```
API_STATUS __stdcall CameraSetDebouncerTime(int device_id, int debouncer_us)
```

Parameters

[in] **device_id** camera index.

[in] **debouncer_us** debouncer time for trigger signal in us.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Only some models support this function.

CameraGetDebouncerTime

Get debouncer time for trigger signal.

Syntax

```
API_STATUS __stdcall CameraGetDebouncerTime(int device_id, int*debouncer_us)
```

Parameters

[in] **device_id** camera index.

[out] **debouncer_us** return debouncer time in us.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Only some models support this function.



CameraSetSnapMode

Set image snap mode.

Syntax

```
API_STATUS __stdcall CameraSetSnapMode(int device_id, int snap_mode)
```

Parameters

[in] **device_id** camera index.

[in] **snap_mode** image snap mode including CAMERA_SNAP_TRIGGER
or CAMERA_SNAP_CONTINUATION.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Default mode is CAMERA_SNAP_CONTINUATION, you have set snap mode to CAMERA_SNAP_TRIGGER explicitly if you want to use software or external trigger.

CameraGetSnapMode

Get image snap mode.

Syntax

```
API_STATUS __stdcall CameraGetSnapMode(int device_id, int *snap_mode)
```

Parameters

[in] **device_id** camera index.

[out] **snap_mode** image snap mode including CAMERA_SNAP_TRIGGER
or CAMERA_SNAP_CONTINUATION.

Return Value

success: API_OK, fail: API_ERROR



CameraSetSnapNum

Set number of images to be captured by a trigger signal.

Syntax

```
API_STATUS __stdcall CameraSetSnapNum(int device_id, int num)
```

Parameters

[in] **device_id** camera index.

[in] **num** number of images to be captured by a trigger signal.

Return Value

success: API_OK, fail: API_ERROR

Remarks

CameraGetSnapNum

Get number of images to be captured by a trigger signal.

Syntax

```
API_STATUS __stdcall CameraGetSnapNum(int device_id, int *num)
```

Parameters

[in] **device_id** camera index.

[out] **num** return the number;

Return Value

success: API_OK, fail: API_ERROR

CameraTriggerShot

Send a software trigger signal to camera. Ignored if camera is not in trigger mode.

Syntax

```
API_STATUS __stdcall CameraTriggerShot(int device_id)
```

Parameters



[in] **device_id** camera index.

Return Value

success: API_OK, fail: API_ERROR

Remarks

It's the same as the CAMERA_IMAGE_TRIG option for CameraQueryImage, except that **CameraTriggerShot** helps to break the query image and trigger to two part, which can be run in two difference threads.

CameraGetGPIO

Get general purpose IO status. Only valid with camera with external IO.

Syntax

```
API_STATUS __stdcall CameraGetGPIO(int device_id, int *val);
```

Parameters

[in] **device_id** camera index.

[out] **val** IO status, do a bit and operation with CAMERA_IO_IN to determine the input status.

Return Value

success: API_OK, fail: API_ERROR

CameraSetGPIO

Set general purpose IO status. Only valid with camera with external IO.

Syntax

```
API_STATUS __stdcall CameraSetGPIO(int device_id, int mask, int val);
```

Parameters

[in] **device_id** camera index.

[in] **mask** always set constant CAMERA_IO_OUT.

[in] **val** output status, CAMERA_IO_OUT for high, 0 for low.



Return Value

success: API_OK, fail: API_ERROR

2.5 Capture Image

This group of function have to call after initializing the camera.

Two ways to capture image, 1) query image at any time when need a image. 2) call back function, a thread will launch at background grabbing image, a call back function will be called on every captured image.

2.5.1 Query Image

CameraGetImageSize

Get the current setting's image size.

Syntax

```
API_STATUS __stdcall CameraGetImageSize(int device_id, int *width, int *height)
```

Parameters

[in] **device_id** camera index.

[out] **width** image width.

[out] **height** image height.

Return Value

success: API_OK, fail: API_ERROR

Remarks

The image size is set by CameraSetResolution or CameraSetROI.

CameraGetImageDepth

Get the image pixel bit depth.

Syntax



```
API_STATUS __stdcall CameraGetImageDepth(int device_id,int *depth)
```

Parameters

[in] **device_id** camera index.

[out] **depth** pixel bit depth.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Only 8bit and 12bit are supported.

CameraGetImageBufferSize

Get the current image's image data size.

Syntax

```
API_STATUS __stdcall CameraGetImageBufferSize(int device_id, int *size, int option)
```

Parameters

[in] **device_id** camera index.

[out] **size** image data size.

[in] **option** image format. The format can be CAMERA_IMAGE_RAW8, CAMERA_IMAGE_GRAY8, or CAMERA_IMAGE_RGB24

Return Value

success: API_OK, fail: API_ERROR

CameraGetISPImageBufferSize

Get isp result's image data size. Capture image can be break to two step, first query the raw image, and then call CameraIsp to convert to normal image.

Syntax

```
API_STATUS __stdcall CameraGetISPImageBufferSize(int device_id,int *size, int width, int height, int option)
```



Parameters

[in] **device_id** camera index.

[out] **size** image data size.

[in] **width** image width.

[in] **height** image height.

[in] **option** image format.

Return Value

success: API_OK, fail: API_ERROR

CameraClearHostBuffer

Clear the buffer in host, avoid access to an old image in continuous mode and only query image periodically .

Syntax

```
API_STATUS __stdcall CameraClearHostBuffer(int device_id)
```

Parameters

[in] **device_id** camera index.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Avoid to use this function in trigger mode. User should never get old images in software or hardware trigger mode.

CameraQueryImage

Query a frame of image from camera.

Syntax

```
API_STATUS __stdcall CameraQueryImage(int device_id, unsigned char *imgbuf, int *length,
```



int option)

Parameters

[in] **device_id** camera index.

[out] **imgbuf** image data.

[out] **length** image data length.

[in] **option** image format and other options.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Get the needed space by CameraGetImageBufferSize, and allocate enough space before call CameraQueryImage.

Options:

Constant	Value	Description
CAMERA_IMAGE_RAW8 CAMERA_IMAGE_RAW	0x1	Capture raw image data. (*1)
CAMERA_IMAGE_GRAY8	0x2	Capture GRAY8 image data.
CAMERA_IMAGE_RGB24	0x4	Capture RGB24 image data.
CAMERA_IMAGE_BGR	0x100	Capture BGR image data instead of RGB along with CAMERA_IMAGE_RGB24.
CAMERA_IMAGE_QUAD	0x200	Capture 32 bit image data. Set along with CAMERA_IMAGE_RGB24
CAMERA_IMAGE_SYNC	0x10000	Drop old images, capture a new image.
CAMERA_IMAGE_TRIG	0x20000	Trigger and then capture the triggered image.
CAMERA_IMAGE_BMP	0x104	Equals to (CAMERA_IMAGE_RGB24



		CAMERA_IMAGE_BGR)
CAMERA_IMAGE_STRETCH	0x1000000	CameraPlay will display image by maintain the images' aspect, set this option by CameraSetOption can change it to stretch mode.

1. In the context of CameraQueryImage, this value indicate get the raw data(RAW8 or RAW12 depends on camera's output) from the camera, two bytes represents a RAW12 pixel. In the context of CameraISP, this value indicates a displayable image format, RAW8. Therefore, only RAW12 data can feed into CameraISP with CAMERA_IMAGE_RAW option.

Examples

1: Capture a image.

```
CameraInit(0);
CameraSetExposure(0, 1000);
int len=0;
CameraGetImageBufferSize(0,&len, CAMERA_IMAGE_RGB24);
unsigned char *inBuf = new unsigned char[len];
if(CameraQueryImage(0,inBuf,&len,CAMERA_IMAGE_BMP)==API_OK)
{
    //Process & display.
}
CameraFree(0);
```

2: Software trigger.

```
CameraInit(0);
CameraSetGain(0, 32);
CameraSetExposure(0, 1000);
```



```
//Enable trigger mode.
CameraSetSnapMode(0, CAMERA_SNAP_TRIGGER);

int width=0, height=0, len=0;
CameraGetImageSize(0,&width, &height);
CameraGetImageBufferSize(0,&len, CAMERA_IMAGE_BMP);
unsigned char *imageBuf = new unsigned char[len];
if(CameraQueryImage(0,imageBuf, &len,
    CAMERA_IMAGE_BMP |CAMERA_IMAGE_TRIG)==API_OK)
{
    TRACE("GRABING DONE\n");
}
else
{
    TRACE("ERROR\n");
    delete []imageBuf;
    CameraReset(0); //Recover from error.
}
CameraFree(0);
```

CameraISP

Covert raw image data to normal RGB/Gray image data.

Syntax

```
API_STATUS __stdcall CameraISP(int device_id, unsigned char *pdata,
unsigned char *imgbuf, int width, int height, int option)
```

Parameters

[in] **device_id** camera index.



[out] **pdata** raw image data.

[out] **imgbuf** normal image data.

[in] **width** image width.

[in] **height** image height.

[in] **option** image format and other options.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Capture raw image data in one thread and convert to normal image data in other threads can increase the concurrency of the program.

2.5.2 Callback Function

CameraSetOption

Set options for CameraPlay. The options are the same as CameraQueryImage.

Syntax

```
API_STATUS __stdcall CameraSetOption(int device_id , int format)
```

Parameters

[in] **device_id** camera index.

[in] **format** options, refer to CameraQueryImage.

Return Value

success: API_OK, fail: API_ERROR

Remarks

The default format for CameraPlay is CAMERA_IMAGE_BMP, if other format, such as CAMERA_IMAGE_GRAY8, is needed, user have to call CameraSetOption to set it. CameraSetOption should execute before CameraPlay.

Examples

```
CameraSetOption(CAMERA_IMAGE_GRAY8); //set option to gray8
```



```
CameraPlay(m_index,hWnd,SnapThreadCallback);
```

CameraGetOption

Get options for CameraPlay.

Syntax

```
API_STATUS __stdcall CameraGetOption(int device_id, int *Format)
```

Parameters

[in] **device_id** camera index.

[out] format options, refer to **CameraQueryImage**.

Return Value

success: API_OK, fail: API_ERROR

Remarks

Refer to CameraSetOption.

CameraPlay

Play camera video on a windows control, and execute callback function on every frame.

Syntax

```
API_STATUS __stdcall CameraPlay(int device_id,HWND hWnd,
CAPTURE_FRAME_PROC proc)
```

Parameters

[in] **device_id** camera index.

[in] **hWnd** handle to the window for displaying image, which can be NULL.

[in] **proc** Callback function, which can be NULL.

Return Value

success: API_OK, fail: API_ERROR

Remarks



Callback function prototype:

```
int __stdcall proc(unsigned char *pImageBuffer, int width, int height, int format)
```

Parameters:

pImageBuffer image data.

width image width.

height image height.

format image format. Refer to CameraSetOption.

CameraPlayEx

Play camera video on a windows control, and execute callback function with parameter on every frame.

Syntax

```
API_STATUS __stdcall CameraPlayEx(int device_id,HWND hWnd,  
CAPTURE_FRAME_PROC_EX proc, , void *param)
```

Parameters

[in] **device_id** camera index.

[in] **hWnd** handle to the window for displaying image, which can be NULL.

[in] **proc** Callback function, witch can be NULL.

[in] **param** Parameter pass to callback function, witch can be NULL.

Return Value

success: API_OK,fail: API_ERROR

Remarks

Callback function prototype:

```
int __stdcall proc_ex(unsigned char *pImageBuffer, int width, int height, int format,  
void *param)
```

Parameters:

pImageBuffer image data.

width image width.



height image height.

format image format. Refer to CameraSetOption.

param parameter passed from CameraPlay.

Examples

```
//Stretch show on the display control.  
CameraInit(0);  
CameraSetOption(0, CAMERA_IMAGE_BMP|CAMERA_IMAGE_STRETCH);  
CWnd *p=GetDlgItem(IDC_IMAGE);  
HWND hWnd=p->GetSafeHwnd();  
CameraPlayEx(0,hWnd,0,0);  
  
.....  
CameraStop(0);  
CameraFree(0);
```

CameraPlayWithoutCallback

Play camera video on a windows control.

Syntax

```
API_STATUS __stdcall CameraPlayWithoutCallback(int device_id,  HWND hWnd)
```

Parameters

[in] **device_id** camera index.

[in] **hWnd** handle to the window for displaying image.

Return Value

success: API_OK, fail: API_ERROR

CameraStop

Stop video capture, corresponding to CameraPlay, free resources.

Syntax



```
API_STATUS __stdcall CameraStop(int device_id)
```

Parameters

[in] **device_id** camera index.

Return Value

success: API_OK, fail: API_ERROR

2.5.3 Shortcut Function

This group of function combines two or more functions together to simplify the development.

CameraShowImage

Capture a frame of image and display on the control.

Syntax

```
API_STATUS __stdcall CameraShowImage(int device_id, HDC hdc, int x, int y, int cx, int cy, CAPTURE_FRAME_PROC proc)
```

Parameters

[in] **device_id** camera index.

[in] **hdc** window's HDC handle to draw the image.

[in] **x** horizontal start position on the control.

[in] **y** vertical start position on the control.

[in] **cx** width on the control.

[in] **cy** height on the control.

[in] **proc** callback function.

Return Value

success: API_OK, fail: API_ERROR



CameraSaveBMPB

Capture a frame of image and save as a BMP file.

Syntax

```
API_STATUS __stdcall CameraSaveBMPB(int device_id, char *fileName)
```

Parameters

[in] **device_id** camera index.

[in] **fileName** file path and name.

Return Value

success: API_OK, fail: API_ERROR

CameraSaveJpegB

Capture a frame of image and save as a JPEG file.

Syntax

```
API_STATUS __stdcall CameraSaveJpegB(int device_id, char *fileName, BOOL color)
```

Parameters

[in] **device_id** camera index.

[in] **fileName** file path and name.

[in] **color** true for color and false for mono image.

Return Value

success: API_OK, fail: API_ERROR

CameraSaveHBITMAP

Capture a frame of image as a HBITMAP object, which can be used in other Windows API calls.

Syntax

```
HBITMAP __stdcall CameraSaveHBITMAP(int device_id)
```



Parameters

[in] **device_id** camera index.

Return Value

Return a windows HBITMAP handle.

Remarks

Call DeleteObject to release the object after used the HBITMAP, or there may will be memory leak.

CameraSaveImage

Capture a frame of image and save as a image file.

Syntax

```
API_STATUS __stdcall CameraSaveImage(int device_id,char *fileName,  
bool color,int option)
```

Parameters

[in] **device_id** camera index.

[in] **fileName** file path and name.

[in] **color** true for color and false for mono image.

[in] **option** file format, supported format:

CAMERA_FILE_BMP24 24 bit bitmap file

CAMERA_FILE_BMP8 8 bit bitmap file

CAMERA_FILE_JPEG JPEG file

Return Value

success: API_OK,fail: API_ERROR

2.5.4 Utility Function

This group of function is a set of utilities for save or display image.



CameraSaveBMP

Save image data buffer to BMP file.

Syntax

```
API_STATUS __stdcall CameraSaveBMP(char *fileName, BYTE * buf,  
UINT width, UINT height)
```

Parameters

[in] **fileName** file path and name.

[in] **buf** image data.

[in] **width** image width.

[in] **height** image height.

Return Value

success: API_OK, fail: API_ERROR

Examples CameraSaveBMP("temp.bmp", buf, width, height);

CameraSaveBMP8

Save image data buffer to 8 bit BMP file.

Syntax

```
API_STATUS __stdcall CameraSaveBMP8(char *fileName, BYTE * buf,  
UINT width, UINT height)
```

Parameters

[in] **fileName** file path and name.

[in] **buf** image data.

[in] **width** image width.

[in] **height** image height.

Return Value

success: API_OK, fail: API_ERROR



CameraSaveJpeg

Save image data buffer to JPEG file.

Syntax

```
API_STATUS __stdcall CameraSaveJpeg(char *fileName, //path
    BYTE *dataBuf,    //RGB Buffer
    UINT width,UINT height, BOOL color,int quality)
```

Parameters

- [in] **fileName** file path and name.
- [in] **dataBuf** image data.
- [in] **width** image width.
- [in] **height** image height.
- [in] **color** true for color and false for mono image.
- [in] **quality** image quality, range from 0 to 100, the higher the better.

Return Value

success: API_OK,fail: API_ERROR

CameraShowBufferImage

Display image with buffered image data.

Syntax

```
API_STATUS __stdcall CameraShowBufferImage(HWND hWnd,
    unsigned char *buf, int width, int height, bool color, bool showStretchMode)
```

Parameters

- [in] **hWnd** handle to the window for displaying image.
- [in] **buf** image data.
- [in] **width** image width.
- [in] **height** image height.
- [in] **color** true for color and false for mono image.



[in] **showStretchMode** display in stretch mode or not.

Return Value

success: API_OK, fail: API_ERROR

Examples

```
int len = 0, width = 0, height = 0;
CameraGetImageSize(g_index, &width, &height);
CameraGetImageBufferSize(g_index, &len, CAMERA_IMAGE_BMP);
unsigned char *buf = new unsigned char[len];
if (CameraQueryImage(g_index, buf, &len, CAMERA_IMAGE_BMP) == 0)
{
    CameraShowBufferImage(m_handle, buf, width, height, true, true);
}
```

CameraSaveBufferImage

Save image data buffer to image file.

Syntax

```
API_STATUS __stdcall CameraSaveBufferImage(char *fileName, BYTE *dataBuf,
UINT width, UINT height, BOOL color, int quality, int option)
```

Parameters

[in] **fileName** file path and name.

[in] **dataBuf** image data.

[in] **width** image width.

[in] **height** image height.

[in] **color** true for color and false for mono image.

[in] **quality** image quality, range from 0 to 100.

[in] **option** file format, supported format:

CAMERA_FILE_BMP24 24 bit bitmap file

CAMERA_FILE_BMP8 8 bit bitmap file



CAMERA_FILE_JPEG JPEG file

Return Value

success: API_OK, fail: API_ERROR

Remarks

1 **quality** only valid when **option** is CAMERA_FILE_JPEG.

2 If **option** is CAMERA_FILE_JPEG, the pixel image data in **dataBuf** is RGB, otherwise BGR.



3 Revisions

2023/05/19	Change Gamma range 0.3-3	
2021/07/02	Add LSC/LDC related API	
2020/09/27	Add IO polarity and delay related API	
2020.04.27	Add line scan mode API, Add 12bit related API	
2019.11.26	Add new model, update CameraSetROI remarks	
2012.11.28	Initial Release	