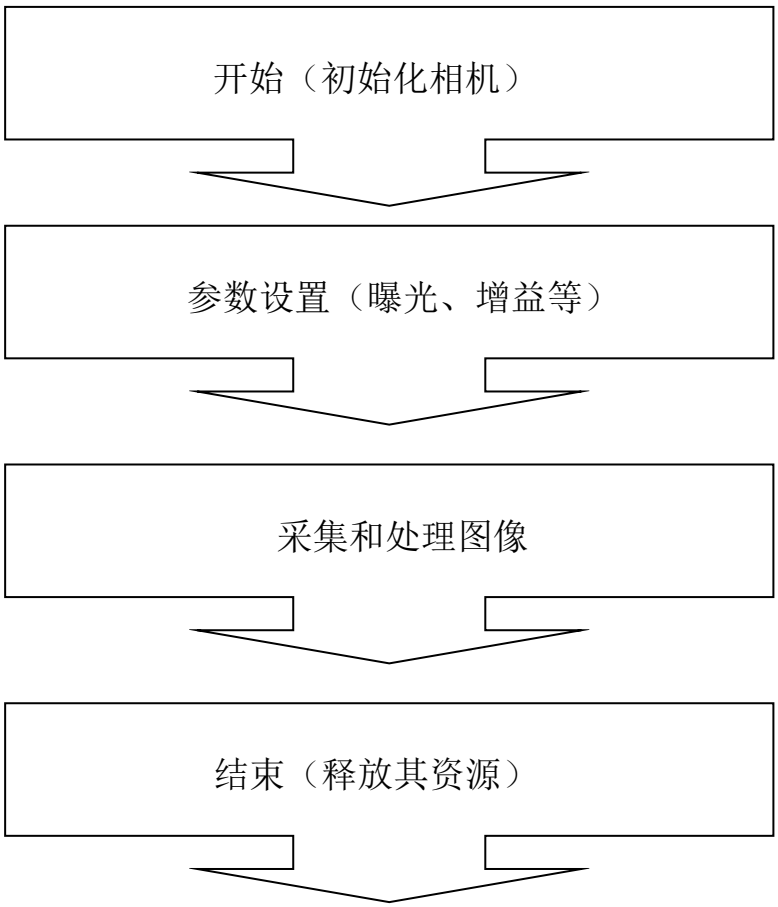
SDK 开发说明书

目录

1 概述

在Microsoft的32位或者64位Windows XP/Windows 7/Windows 10操作系统中，JH系列工业相机二次开发包(JHCap2 SDK)提供直接操作工业相机的编程接口。用户在编制自己的应用程序时，可以直接调用这些库函数来实现指定的功能。

工业相机二次开发工作流程如下



一般情况下，工业相机的初始化及其参数的初始化设置，最好在用户应用程序的初始化中完成，工业相机的结束操作应该在应用程序退出前执行。

1.1 SDK 组成

JHCap SDK 采用 C 语言实现，库文件包含:

头文件: JHCap.h

动态库文件: JHCap2.dll

链接符号: JHCap2.lib

JHCap SDK 支持 32 位和 64 位 Windows 平台，需要使用不同版本的 lib 和 dll，32 位 SDK 位于 sdk/SDK, 64 位 SDK 位于 sdk/SDK64。

SDK 提供丰富的例程，演示相机和 SDK 的各种功能。

1.2 开发工具

Borland C ++ / Delphi，QT 和 Microsoft Visual Stdio 等主流开发工具可用于开发应用程序。支持的编程语言，包括但不限于 C / C ++ / Visual Basic / C＃/ VB。net / Delphi / Python。

使用 C/C++编程工具用户应在程序中调用相关的包含文件（.h），并将链接库（.lib）文件加入到工程文件中，供编译程序在链接（Link）时使用。

2 API 接口函数

2.1 相机信息

本组函数不需要初始化相机即可调用。

CameraGetVersion

获取 API 版本。

函数原型

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

参数

[out] **major** 主要版本号，经常以年命名，如 2016。

[out] **minor** 次要版本号，经常以月命名，如 9。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

在遇到无法解决的问题需要将版本号一并反馈。

CameraGetFirmVersion

获取相机的固件版本。

函数原型

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

参数

[in] **index** 相机序号。

[out] **ver** 固件版本号，一个 16 进制数，如 0x4501。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

1 在遇到无法解决的问题需要将固件版本号一并反馈。

2 相机序号以 0 为开始。

CameraGetCount

获取连接 PC 机的相机数量。

函数原型

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

参数

[out] **count** 已连接的相机数量。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

有效的相机序为 0 到 count-1。

CameraGetName

获取指定序号 index 的相机的名称及型号名称。

函数原型

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

参数

[in] **index** 相机序号。

[out] **name** 相机名称。

[out] **model** 相机型号序号。

返回值

调用成功: API_OK，调用失败: API_ERROR。

例子

```
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

获取指定序号 index 的相机的产品 ID 及模板 ID。

函数原型

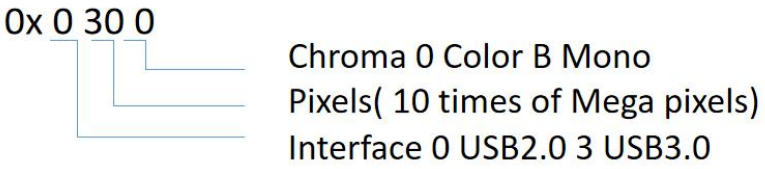
*API_STATUS __stdcall CameraGetID (int index, int *modeID, int *productID)*

参数

[in] **index** 相机序号。

[out] **modeID** 相机型号 ID, 限内部使用。

[out] **productID** 产品的数字模型 ID。



比如:0x0300 表示 300 万的彩色相机, 0x013B 表示 130 万的黑白相机

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetLastError

通过 API 获取最后一个错误提示。它是一个线程局部变量，每个线程都有自己的错误提示。由于 Windows XP 不支持线程局部变量，错误提示是基于应用程序的，所以应该使用 sdk / SDK / WindowsXP 下的 DLL 来避免应用程序崩溃。

函数原型

*API_STATUS __stdcall CameraGetLastError(int *last_error)*

参数

[in] **last_errror** 错误提示。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

当调用其他 SDK 函数返回 API_ERROR 时，可再调用 CameraGetLastError 获取错误的原因。

编号	对应值	详细
1	CAMERA_ERROR_INDEX_OUT_OF_RANGE	函数调用相机序号越界
2	CAMERA_ERROR_PARAMETER_UNREASONABLE	参数设置不合理
3	CAMERA_ERROR_NOT_INITIALIZED	相机未初始化
4	CAMERA_ERROR_USB_DISCONNECTED	USB 断开连接
5	CAMERA_ERROR_NO_CAMERA_FOUND	找不到相机
6	CAMERA_ERROR_INITIALIZED_ALREADY	相机已初始化
7	CAMERA_ERROR_RECONNECT_FAILURE	相机重连错误
8	CAMERA_ERROR_NOT_SUPPORTED	该相机不支持这个功能
9	CAMERA_ERROR_CONTROL_PIPE	命令通道无效
10	CAMERA_ERROR_READ_PIPE	读数据错误
11	CAMERA_ERROR_WRITE_PIPE	写数据错误
12	CAMERA_ERROR_IMAGE_SIZE	图像大小错误
13	CAMERA_ERROR_DATA_PIPE	数据通道无效
14	CAMERA_ERROR_BAD_FRAME	坏帧
15	CAMERA_ERROR_USER_ABORT	用户释放相机停止取图
16	CAMERA_ERROR_DATA_XFER	数据传输中断
17	CAMERA_ERROR_DATA_TIMEOUT	取图超时
18	CAMERA_ERROR_OUT_OF_MEMORY	内存不足
19	CAMERA_ERROR_RESET	相机重连中
0	API_OK	调用成功

2.2 相机控制

调用改组函数前必须先对相机初始化（CameraInit）。

CameraInit

相机初始化。

函数原型

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

参数

[in] **index** 相机序号。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

1 序号从 0 开始计数。

2 可以多次调用 CameraInit，对相机多次初始化不会产生问题。

3 通常用户必须在再次调用 CameraInit 前需先调用 CameraFree。

例子

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

CameraFree

释放相机资源，一般在程序结束前调用 。

函数原型

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

参数

[in] **device_id** 相机序号。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraReset

重置相机以修复数据传输错误的问题。

函数原型

```
API_STATUS __stdcall CameraReset(int device_id)
```

参数

[in] **device_id** 相机序号。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

遇到传输错误时，CameraReset 会在连续模式下自动调用。在触发模式下，如果发出信号，CameraQueryImage 超时并返回 API_ERROR，则需要调用 CameraReset。

CameraReconnect

模拟从计算机 USB 重新连接，相当于拔插 USB 接口。

函数原型

```
API_STATUS __stdcall CameraReconnect (int device_id)
```

参数

[in] **device_id** 相机序数。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

CameraReconnect 将使 Windows 驱动程序临时释放设备。应用程序可以接收 Windows 发送的消息 WM_DEVICECHANGE，根据消息应用重新初始化（CameraFree / CameraInit）相机。

CameraSetTimeout

设置相机读取一帧图像的最长时间。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **timeout** 最长时间（以毫秒为单位）。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

默认值和最小值为 500ms。如果取图超时，在连续模式下，会自动调用 CameraReset 。如果相机的曝光时间超过 500ms，那么应该设置超时时间大于曝光时间。

CameraGetTimeout

获取相机读取一帧图像的最大时间。

函数原型

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

参数

[in] **device_id** 相机序数。

[out] **timeout** 最长时间（以毫秒为单位）。

返回值

调用成功: API_OK，调用失败: API_ERROR。

2.3 相机参数设置

调用改组函数前必须先对相机初始化（CameraInit）。

2.3.1 分辨率和 ROI

CameraGetResolutionCount

获取预设的分辨率数。

函数原型

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

参数

[in] **device_id** 相机序数。

[out] **count** 预设分辨率数。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetResolutionMax

获得相机的最大分辨率，它是最大的图像尺寸。

函数原型

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

参数

[in] **device_id** 相机序数。

[out] **width** 最大宽。

[out] **height** 最大高。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetResolutionMode

获取当前设置的相机小分辨率模式。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **mode** resolution mode。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

仅当您将分辨率设置为小于（等于）最大分辨率的一半,或者使用线扫模式，分辨率模式才会生效:

CAMERA_RESOLUTION_CROPPING: 截取一部分像素。

CAMERA_RESOLUTION_SKIPPING: 隔行抽取像素。

CAMERA_RESOLUTION_BINNING: 合并相邻像素。

CAMERA_RESOLUTION_LINESCAN: 线扫模式。

CameraSetResolutionMode

设置相机小分辨率模式。

函数原型

API_STATUS __stdcall CameraSetResolutionMode(int device_id, int mode)

参数

[in] **device_id** 相机序数。

[in] **mode** 分辨率模式。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

参考 **CameraGetResolutionMode。**

CameraGetResolution

获取特定预设分辨率的宽度和高度。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **index** 预设分辨率的序号。

[out] **width** 分辨率宽度。

[out] **height** 分辨率高度。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

Index 从 0 开始，范围从 0 到 count-1，其中 count 是预设的分辨率计数，可以通过 **CameraGetResolutionCount** 获得。

例子

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

CameraSetResolution

将相机的分辨率设置为预设分辨率。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **index** 预设分辨率序号。

[out] **width** 分辨率宽。

[out] **height** 分辨率高。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

参考 **CameraGetResolution**。

CameraSetROI

设置感兴趣区域(ROI)。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **offset_width** ROI 的起始水平位置。

[in] **offset_height** ROI 的起始垂直位置。

[in] **width** ROI 宽。

[in] **height** ROI 高。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

offset_width 和 **offset_height** 的值需要小于最大分辨率并满足以下：

offset_width + **width** <= 最大水平分辨率

offset_height + **height** <= 最大垂直分辨率

offset_height 和 **height** 值须为 2 的倍数，**offset_width , width** 必须为 4 的倍数。 另外需要满足：

以下相机型号 **offset_width , width** 必须为 32 的倍数	
JHUM501/B	JHUM1800
JHUM202B	JHUM502B
以下相机型号 **offset_width , width** 必须为 16 的倍数	
JHUM32B	JHUM132B
JHUM131/B	
以下相机型号 **offset_width , width** 必须为 8 的倍数	
JHUM201/B	JHUM600/B
JHUM1200/B	JHUM2000/B
JHUM804/B	JHUM1204/B
JHSM400/B	
以下相机型号 **offset_width , offset_height** 必须为 8 的倍数, **width, height** 必须为 16 的倍数	
JHUM306/B	JHUM506/B
JHUM806/B	

2.3.2 曝光和增益

CameraSetGain

设置增益。

函数原型

API_STATUS __stdcall CameraSetGain(int device_id, int gain)

参数

[in] **device_id** 相机序号。

[in] **gain** 相机增益。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

参数范围从 1 到 255，噪声可能由增益引入。低于 32 的值属于模拟增益，噪声很小。当增益大于 32 时，数字增益会引入更多噪声。

CameraGetGain

获得增益值。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **gain** 相机增益，参数范围为 1 到 255。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetAdvancedGain

设置高级增益，仅仅对 120/200/400/44/204/304/504/804/306/506/806/600/2000 等相机有效。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **advanced_gain** 相机高级增益。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

高级增益的默认值为 0，范围为 0-5，当使用高级增益时会产生噪声。

CameraGetAdvancedGain

获取高级增益值，仅仅对 120/200/400/44/204/304/504/804/306/506/806/600/2000 等相机有效。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **advanced_gain** 相机高级增益，参数范围从 0 到 5。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetAGC

开启/关闭自动增益。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **agc** 使能自动增益，true 是开启, false 是关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetAGC

获取是否开启自动增益状态。

函数原型

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

Parameter

[in] **device_id** 相机序号。

[out] **agc** 自动增益状态。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetExposure

设置曝光。

函数原型

API_STATUS __stdcall CameraSetExposure(int device_id, int exposure)

参数

[in] **device_id** 相机序号。

[in] **exposure** 相机曝光值。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

1 曝光值范围为 1 到 32768。

2 曝光时间可以在设置曝光值后通过调用 **CameraGetExposureTime** 获得。

3 如果曝光时间超过 500ms，需要调用 **CameraSetTimeout** 来设置更大的超时时间。

CameraGetExposure

获取曝光值。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **exposure** 相机曝光值。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

参考 **CameraGetExposure**。

CameraGetExposureUnit

获取 1 个单位的曝光值表示的曝光时间。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **exposure_unit** 曝光时间，以毫秒为单位。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetExposureTime

获取曝光时间。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **exposure_time** 当前曝光时间，以毫秒为单位。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

曝光时间=曝光值*单位曝光时间。

CameraSetAEC

开启/关闭自动曝光。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **agc** 使能自动曝光，true 是开启, false 是关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetAEC

获取是否开启自动曝光状态。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **aec** 自动曝光状态。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetAETarget

设置自动曝光的目标值。如果开启了 AEC，图像的像素平均值将趋向目标值。

函数原型

API_STATUS __stdcall CameraSetAETarget(int device_id, int target)

参数

[in] **device_id** 相机序号。

[in] **target** 亮度目标值。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetAETarget

获取自动曝光的目标值。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **target** 目标值。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetAntiFlicker

设置防闪烁模式以消除 AC 光源引起的条纹。

函数原型

API_STATUS __stdcall CameraSetAntiFlicker (int device_id, int flicker)

参数

[in] **device_id** 相机序号。

[in] **flicker** 频闪值, 1 表示 50HZ , 2 表示 60HZ

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

防闪烁是通过设置适当的曝光值来实现的，因此只有在启用自动曝光时才会生效。

CameraGetAntiFlicker

获取设置的防闪烁模式。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **flicker** 频闪，1 表示 50HZ，2 表示 60HZ

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

参考 **CameraSetAntiFlicker**。

2.3.3 图像调节

CameraGetAvg

获取最后一张图像像素平均值。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **avg** 像素平均值。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetGamma

设置伽马值。

函数原型

```
API_STATUS __stdcall CameraSetGamma(int device_id, double gamma)
```

参数

[in] **device_id** 相机序号。

[in] **gamma** 伽马值，数值范围为 0.3-3.0。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetGamma

获取伽马值。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **gamma** 伽马值，数值范围为 0.3-2.0。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetContrast

设置图像对比度。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **contrast** 对比度，范围为 0.3-2.0。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetContrast

获取图像对比度。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **contrast** 对比度，范围为 0.3-2.0。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetSaturation

设置颜色饱和度。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **saturation** 饱和度，范围为 0.0-2.0。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetSaturation

获取颜色饱和度。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **saturation** 饱和度，范围为 0.0-2.0。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetBlackLevel

设置图像黑电平，所有原始像素值将减去黑电平，图像将变黑并具有高黑电平。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **black** 黑电平，范围为 0-255。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetBlackLevel

获取图像黑电平。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **black** 黑电平，范围为 0-255。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetEnhancement

开启/关闭颜色增强。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **enhance** 是否颜色增强，true 表示开启，false 表示关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

usbVideo 将 CCM 矩阵预设到摄像机，如果启用了增强功能，则 CCM 将应用于每个像素。一般来说，CCM 用于增强色彩饱和度并提高色彩准确度。

本功能需要额外的计算资源，可能降低帧率。

CameraGetEnhancement

获取是否开启颜色增强的状态。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **enhance** 是否颜色增强，true 表示开启，false 表示关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetGlobalReset

开启/关闭全局复位快门。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **en** 是否使能全局复位快门，true 表示开启，false 表示关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

只有部分卷帘曝光的相机支持此功能，全局复位配合机械快门或者闪光灯可以达到全局曝光类似的效果，实现拍摄运动物体无扭曲变形。

CameraGetGlobalReset

获取是否开启全局复位快门。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **en** 是否使能全局复位快门，true 表示开启，false 表示关闭或不支持。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetDenoise

设置软件降噪。在图像处理过程中，降噪可以将图像中亮度明显大于周围像素的

那个像素（白点）使用周围像素的均值替代。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **denoise** 是否开启降噪，true 表示开启，false 表示关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

本功能需要额外的计算资源，可能降低帧率。

CameraGetDenoise

获取软件降噪设置。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] *denoise* 返回是否开启降噪，true 表示开启，false 表示关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetFPN

设置 FPN 修正功能是否开启。 固定噪声(FPN)是传感器技术缺陷引起的在图像中有固定横条纹或竖条纹，开启 FPN 功能可以使用在 usbVideo 上预先校准（或出厂校准)的 FPN 数据来修复图像上的条纹瑕疵。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **fpn** 是否开启 FPN 修正功能，true 表示开启，false 表示关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

本功能需要额外的计算资源，可能降低帧率。

CameraGetFPN

获取 FPN 设置。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **fpn** 返回是否开启 FPN 修正功能，true 表示开启，false 表示关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraEnableLSC

使能/关闭镜头阴影矫正(LSC)。 LSC 是一项用来矫正由于镜头导致的画面亮度不均匀的技术。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **en** 是否开启 LSC 修正功能，true 表示开启，false 表示关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

这个功能开启需要参数才能生效。这个功能需要一些计算量，开启本功能可能降低帧率。

计算 LSC 需要的参数。算法使用距离光学中心的二阶函数来拟合亮度。对一个距离光学中心为 *r* 的像素，它的强度表示为

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

光学中心通常在图像的中心，如果有偏差，可以通过设置 *dx0*，*dy0* 来调整。

参数需要通过拍摄一张矫正图像来的到，通常是拍摄一个纯白的均匀物体。不同镜头需要分别矫正自己的参数。

CameraGetLSC

获取是否开启 LSC 的状态。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **en** 返回是否开启 LSC 修正功能，true 表示开启，false 表示关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraEnableLDC

使能/关闭镜头畸变矫正(LDC)。 LDC 是一项用来矫正由于镜头畸变导致的画面扭曲变形的技术。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **en** 是否开启 LDC 修正功能，true 表示开启，false 表示关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

这个功能开启需要参数才能生效。这个功能需要一些计算量，开启本功能可能降低帧率。

计算 LDC 需要的参数。用张正友标定法得到相机的内参[f_x, f_y, c_x, c_y]和镜头的畸变参数[k_1, k_2, p_1, p_2, k_3]，可以使用 Opencv 的方法或者 Matlab 提供的工具和标定板进行畸变参数的标定。

CameraGetLDC

获取是否开启 LSC 的状态。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **en** 返回是否开启 LDC 修正功能，true 表示开启，false 表示关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetLUT

设置查找表。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **lut** 查找表，一般位 4096 长度，12bit 转 8bit 数据的映射表。

[in] **len** 查找表长度。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

本功能需要额外的计算资源，可能降低帧率。

CameraGetLUT

获取查找表。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **lut** 查找表，一般位 4096 长度，12bit 转 8bit 数据的映射表。

[inout] **len** lut 为空指针是获取所需分配的查找表大小。lut 不空时为 lut 有效数据长度。

返回值

调用成功: API_OK，调用失败: API_ERROR。

2.3.4 帧控制

CameraSetHighspeed

开启/关闭相机高速模式。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **high** 是否开启高速模式，true 表示开启，false 表示关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

高速模式的帧率是低速的两倍。USB2.0 带宽标称值为 480Mbps，实际上是 40MBps。相机默认以低速模式运行。

计算机通常在 CPU /主板上有两个高速 USB 通道。如果两个摄像头插入单独的通道，它们都可以高速运行，否则它们都以低速运行。

CameraGetHighspeed

获取是否开启高速模式的状态。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **high** 是否开启高速模式，true 表示开启，false 表示关闭。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

参考 **CameraSetHighspeed**。

CameraSetDelay

设置延迟，延迟插入图像行间隔和图像帧间隔。它可用于调整帧速率，较高的延迟导致较低的帧速率。

函数原型

API_STATUS __stdcall CameraSetDelay(int device_id, int delay)

参数

[in] **device_id** 相机序号。

[in] **delay** 延迟的值。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

如果计算机的性能很差，延迟可能有助于降低帧速率，然后减少 CPU 的进程负载。默认值为 0。

CameraGetDelay

获取延迟的值。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **delay** 延迟的值。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

参考 **CameraSetDelay**。

CameraSetFrameRate

设置限制帧率，调整相机的输出帧率为最大值以下的某个值。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **en** 限制帧率使能。

[in] **fps** 期望的帧率。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

降低帧率降低采集的数据量，节省计算资源，可以调节到与实际需要相匹配的帧率。

只有部分相机支持此功能。相机的最高帧率受传感器输出限制，如果设置的 fps 值大于最大帧率将被忽略。设置使能 en 为 false 将返回最大帧率。

CameraGetFrameRate

获取帧率设置。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **en** 返回限制帧率使能。

[out] **fps** 返回设置的帧率

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

参考 **CameraSetFrameRate**。

CameraSetMirrorX

设置图像水平镜像。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **mx** 是否水平镜像，true 表示是，false 表示否。

返回值

调用成功: API_OK，调用失败: API_ERROR。

例子

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

CameraGetMirrorX

获取是否水平镜像的状态。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **mx** 是否水平镜像，true 表示是，false 表示否。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetMirrorY

设置图像垂直镜像。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **my** 是否垂直镜像，true 表示是，false 表示否。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetMirrorY

获取是否垂直镜像的状态。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **my** 是否垂直镜像，true 表示是，false 表示否。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetRotate

设置图像旋转。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **rotate** 图像旋转角度，仅包含 90, 180, 270。

返回值

调用成功: API_OK，调用失败: API_ERROR。

2.3.5 白平衡

CameraSetAWB

开启/关闭自动白平衡

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **awb** 是否开启自动白平衡，true 表示是，false 表示否。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetAWB

获取是否开启自动白平衡状态。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **awb** 是否开启自动白平衡，true 表示是，false 表示否。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetWBGain

设置白平衡 R/G/B 增益。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **rg** 红色增益，范围为0.0-3.0。

[in] **gg** 绿色增益，范围为0.0-3.0，通常固定为1.0。

[in] **bg** 蓝色增益，范围为0.0-3.0。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetWBGain

获得白平衡 R/G/B 增益的数值。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **rg** 红色增益，范围为0.0-3.0。

[out] **gg** 绿色增益，范围为0.0-3.0。

[out] **bg** 蓝色增益，范围为0.0-3.0。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraOnePushWB

调用一次或者更多来实现白平衡。

函数原型

API_STATUS __stdcall CameraOnePushWB(int device_id)

参数

[in] **device_id** 相机序号。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

在调用 CameraOnePushWB 之前，须将相机对着灰色（白色）物体上拍摄，而不是过度曝光。然后调用此函数一次或多次，直到颜色接近实际颜色。

2.3.6 线扫参数

只有部分相机型号具有线扫功能。对面阵相机的线扫功能，是通过取传感器中心的 line_height 行，可以设置为 2-32 行，曝光多次，组合为一张图像。曝光可以自动进行或者通过外部触发同步，通过 line_trigger 设定。 自动进行是可以通过 line_time 来调节每一行的时间来匹配需要扫描的速度。

CameraSetLineHeight

设置线扫高度。

函数原型

API_STATUS __stdcall CameraSetLineHeight(int device_id, int line_height)

参数

[in] **device_id** 相机序号。

[in] **line_height** 线扫高度，一次曝光扫描的行数。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

扫描高度越大，图像采集帧率越高，线扫描的特性越弱。高度 2 为推荐参数。

CameraGetLineHeight

获取线扫高度。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **line_height** 返回线扫高度。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetLineTrigger

设置行触发方式。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **line_trigger** 行触发方式，取值可以为一下两种：

CAMERA_LINE_TRIGGER_AUTO 相机以最大速度触发

CAMERA_LINE_TRIGGER_EXTERNAL 外部信号用于行触发

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetLineTrigger

获取行触发方式。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **line_trigger** 返回行触发方式。

返回值

调用成功: API_OK，调用失败: API_ERROR。

2.3.7 存储数据

CameraWriteUserData

写入用户自定义的数据，数据长度小于个 64 字节。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **data[]** 用户自定义的数据数组。

[in] **length** 数组长度，小于 64。

返回值

调用成功: API_OK，调用失败: API_ERROR。

例子

```
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

读取用户自定义的数据，数据长度小于 64 个字节。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **data[]** 数据。

[in] **length** 数组长度，小于 64。

返回值

调用成功: API_OK，调用失败: API_ERROR。

例子

```
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

读取相机序列号(每台相机有唯一的相机序列号)。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **id[]** 序列号。

[in] **length** 序列号长度，固定为 12。

返回值

调用成功: API_OK，调用失败: API_ERROR。

例子

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

CameraSaveParameter

保存相机参数到相机内。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **group_no** 参数组，0 表示第 0 组，1 表示第 1 组。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

每个摄像机可以保存两组独立参数，组号为 0 或 1。调用 CameraInit 或调用 CameraLoadParameter 时，可以恢复参数。调用 CameraInit 默认加载第 0 组的参数。

CameraLoadParameter

加载相机参数。

函数原型

API_STATUS __stdcall CameraLoadParameter(int device_id, int group_no)

参数

[in] **device_id** 相机序号。

[in] **group_no** 参数组，0 表示第 0 组，1 表示第 1 组。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

参考 **CameraSaveParameter**。

2.4 外部 I/O 控制

调用改组函数前必须先对相机初始化（CameraInit）。

CameraEnableStrobe

开启/关闭闪光输出。

函数原型

API_STATUS __stdcall CameraEnableStrobe(int device_id, bool en)

参数

[in] **device_id** 相机序号。

[in] **en** 是否开启频闪输出，true 表示是，false 表示否。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetStrobe

获取闪光输出状态。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **en** 返回是否开启频闪输出，true 表示是，false 表示否。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetStrobePolarity

设置闪光输出的极性。

函数原型

API_STATUS __stdcall CameraSetStrobePolarity(int device_id, bool high)

参数

[in] **device_id** 相机序号。

[in] **high** 频闪极性，true 表示高，false 表示低。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

只有部分相机型号支持该功能。

CameraGetStrobePolarity

获取闪光输出的极性。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **high** 返回闪光输出极性，true 表示高，false 表示低。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

只有部分相机型号支持该功能。

CameraSetTriggerPolarity

设置触发的极性。上升沿或者下降沿触发。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **high** 频闪极性，true 表示高，false 表示低。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

只有部分相机型号支持该功能。

CameraGetTriggerPolarity

获取触发的极性。上升沿或者下降沿触发。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **high** 返回触发极性，true 表示上升沿，false 表示下降沿。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

只有部分相机型号支持该功能。

CameraSetTriggerDelay

设置触发延时，从触发信号生效开始计算，us 为单位。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **delay_us** 触发延时，us为单位。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

只有部分相机型号支持该功能。

CameraGetTriggerDelay

获取触发延迟。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **delay_us** 返回触发延时，us为单位。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

只有部分相机型号支持该功能。

CameraSetStrobeDelay

设置闪光延迟，从触发信号生效开始计算，us 为单位。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **delay_us** 闪光延时，us为单位。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

只有部分相机型号支持该功能。

CameraGetStrobeDelay

获取闪光延迟。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **delay_us** 返回闪光延迟，us为单位。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

只有部分相机型号支持该功能。

CameraSetStrobeDuration

设置闪光时间，设置为 0 时，闪光时间等于曝光时间长度。在卷帘曝光的相机里，闪光时间需要覆盖整个画面每一行的曝光时间。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **duration_us** 闪光时间，us为单位。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

只有部分相机型号支持该功能。

CameraGetStrobeDuration

获取闪光时间。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **duration_us** 返回闪光时间，us为单位。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

只有部分相机型号支持该功能。

CameraSetDebouncerTime

设置触发信号的消抖时间，脉宽小于消抖时间的触发信号将被忽略。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **debouncer_us** 消抖时间，us为单位。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

只有部分相机型号支持该功能。

CameraGetDebouncerTime

获取消抖时间。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **debouncer_us** 返回消抖时间，us为单位。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

只有部分相机型号支持该功能。

CameraSetSnapMode

设置相机拍照模式。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **snap_mode** 相机拍照模式，仅包括CAMERA_SNAP_TRIGGER和CAMERA_SNAP_CONTINUATION。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

默认拍照模式为 CAMERA_SNAP_CONTINUATION，如果你想要软件触发或者外部触发拍照时，你需要设置拍照模式 CAMERA_SNAP_TRIGGER。

CameraGetSnapMode

获取当前相机的拍照模式。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **snap_mode** 相机拍照模式，仅包括CAMERA_SNAP_TRIGGER和CAMERA_SNAP_CONTINUATION。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetSnapNum

设置一次触发信号触发的帧数。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **num** 帧数，1-65534。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

CameraGetSnapNum

获取一次触发信号的触发的帧数。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **num** 触发帧数。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraTriggerShot

发送一个软件触发信号到相机，如果不是相机不是触发模式下该信号会被忽略。

函数原型

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

参数

[in] **device_id** 相机序号。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

该函数等价于 CameraQueryImage 的 CAMERA_IMAGE_TRIG 选型，只是 **CameraTriggerShot** 可以使取图和触发这两个操作在两个不同的线程中执行。

CameraGetGPIO

获取通用 IO 状态。仅适用于带有外部 IO 的摄像头。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **val** IO 状态，用 CAMERA_IO_IN 做一点操作来确定输入状态。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSetGPIO

设置通用 IO 状态。仅适用于带有外部 IO 的摄像头。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **mask** 固定写为常量 CAMERA_IO_OUT。

[in] **val** 输出状态，CAMERA_IO_OUT 为高，0 为低。

返回值

调用成功: API_OK，调用失败: API_ERROR。

2.5 读取图像

调用改组函数前必须先对相机初始化（CameraInit）。

两种方法读取图像, 1) 查询图像，在需要时随时读取图像。2) 回调函数，相机每采集到一张图像都会调用回调函数一次。

2.5.1 查询图像

CameraGetImageSize

获取当前图片的图片大小。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **width** 图像的宽。

[out] **height** 图像的高。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

图像大小可以通过 CameraSetResolution 或者 CameraSetROI 来设置。

CameraGetImageDepth

获取当前图像位深度。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **depth** 图像的像素位数。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

支持 8 位和 12 位两种像素位深度。

CameraGetImageBufferSize

获取当前图像数据需要分配的内存大小。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **size** 图像数据大小。

[in] **option** 图像格式。 该格式为 CAMERA_IMAGE_RAW8, CAMERA_IMAGE_GRAY8 或者 CAMERA_IMAGE_RGB24

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraGetISPImageBufferSize

获取 ISP 图像需要分配的内存大小。捕获图像可以分为两步，首先查询原始图像，然后调用 CameraISP 转换为普通图像。

函数原型

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

参数

[in]**device_id** 相机序号。

[out] **size** 图像数据大小。

[in] **width** 图像的宽。

[in] **height** 图像的高。

[in] **option** 图像格式。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraClearHostBuffer

清除主机端的接收缓存，避免 CameraQueryImage 访问到过时的图片，适用于间歇性调用 CameraQueryImage 的情况。

函数原型

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

参数

[in] **device_id** 相机序号。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

应当避免在触发模式下调用改函数，因为在触发模式下不会读取到过时的图片。

CameraQueryImage

从相机读取一帧图片

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **imgbuf** 图像数据。

[inout] **length** 传入 imgbuf 大小，返回数据长度。

[in] **option** 图像格式或者其它功能选项。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

可以调用 CameraGetImageBufferSize 来得知需要分配的内存大小,在调用 CameraQueryImage 前需分配好足够的内存空间。

Options:

常量	对应值	详细描述
CAMERA_IMAGE_RAW CAMERA_IMAGE_RAW8	0x1	读取相机的 RAW 数据 (*1)
CAMERA_IMAGE_GRAY8	0x2	读取 GRAY8 图像数据
CAMERA_IMAGE_RGB24	0x4	读取 RGB24 图像数据
CAMERA_IMAGE_RAW12	0x8	读取 RAW12 图像数据
CAMERA_IMAGE_BGR	0x100	读取 BGR 图像数据而不是 RGB 和 CAMERA_IMAGE_RGB24。
CAMERA_IMAGE_QUAD	0x200	读取 32 位图像数据。一般同时与 CAMERA_IMAGE_RGB24 一起
CAMERA_IMAGE_SYNC	0x10000	删除旧图像，捕获新图像。已废弃，使用 CameraClearHostBuffer 代替。
CAMERA_IMAGE_TRIG	0x20000	触发然后捕获触发的图像。已废弃，使用 CameraTriggerShot 代替。
CAMERA_IMAGE_BMP	0x104	等价于 (CAMERA_IMAGE_RGB24 | CAMERA_IMAGE_BGR)

CAMERA_IMAGE_STRETCH	0x1000000	CameraPlay 保持比例显示图像，可以通过 CameraSetOption 来使图像拉伸显示。

*1，这个选项在和 CameraQueryImage 中使用时，将自动匹配相机输出格式为 RAW8 或者 RAW12 图像数据，RAW12 数据使用 2 个字节存储。这个选项在 CameraISP 中使用时，只能提供 RAW12 数据转换成 RAW8 数据。

例子

1: 读取图像

```
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: 软件触发

```
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];
CameraTriggerShot(0);
if(CameraQueryImage(0,imageBuf, &len,
            CAMERA_IMAGE_BMP)==API_OK)
{
      TRACE("GRABING DONE\n");
}
else
{
     TRACE("ERROR\n");
     delete []imageBuf;
}
CameraFree(0);
```

CameraISP

将原始图像数据转换为普通 RGB /灰度图像数据。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **pdata** 原始图像数据，一通过 CameraQueryImage 获取的数据。

[out] **imgbuf** 正常图像数据，通过 ISP 处理后的图像数据。

[in] **width** 图像宽。

[in] **height** 图像高。

[in] **option** 图像格式或者其它功能选项。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

在一个线程中捕获原始图像数据并在其他线程中转换为普通图像数据可以增加程序的并发性。

2.5.2 回调函数

CameraSetOption

设置 CameraPlay 的相关格式或功能，这些选项的功能等于 CameraQueryImage。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **format** 图像格式或功能选项，参考 **CameraQueryImage**。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

CameraPlay 的默认图片格式是 CAMERA_IMAGE_BMP, 如果需要选择其它格式如 CAMERA_IMAGE_GRAY8，可以通过调用 CameraSetOption 来设置。CameraSetOption 应该在 CameraPlay 执行前调用。

例子

```
CameraSetOption(CAMERA_IMAGE_GRAY8);   //set option to gray8
CameraPlay(m_index,hWnd,SnapThreadCallback);
```

CameraGetOption

获取 CameraPlay 的格式或者功能选项。

函数原型

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

参数

[in] **device_id** 相机序号。

[out] **Format** 格式或者功能选项，参考 **CameraQueryImage**。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

参考 **CameraSetOption**。

CameraPlay

在 Windows 控件上显示相机图片，并在每个帧上执行回调功能。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **hWnd** 显示图片控件的句柄，可为 NULL。

[in] **proc** 回调函数，可为 NULL。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

回调函数原型:

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

参数

pImageBuffer 图像数据。

width 图像宽。

height 图像高。

format 图像格式或者功能选项，参考 **CameraSetOption**。

CameraPlayEx

在 Windows 控件上显示相机图片，并在每个帧上执行回调功能。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **hWnd** 显示图片控件的句柄，可为 NULL。

[in] **proc** 回调函数，可为 NULL。

[in] **param** 传递到回调函数的参数，可为 NULL。

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

回调函数原型:

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

参数

pImageBuffer 图像数据。

width 图像宽。

height 图像高。

format 图像格式或者功能选项，参考 **CameraSetOption**。

param 从 CameraPlayEx 传递来的参数。

例子

```
//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

在 Windows 控件上显示相机图片。

函数原型

API_STATUS __stdcall CameraPlayWithoutCallback(int device_id, HWND hWnd)

参数

[in] **device_id** 相机序号。

[in] **hWnd** 显示图片控件的句柄。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraStop

停止显示图片，并释放资源，与 CameraPlay 相对。

函数原型

API_STATUS __stdcall CameraStop(int device_id)

参数

[in] **device_id** 相机序号。

返回值

调用成功: API_OK，调用失败: API_ERROR。

2.5.3 快捷功能

该组函数通过整合 2 个及以上的函数简化开发过程。

CameraShowImage

读取并显示图像。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **hdc** 显示图像控件的句柄。

[in] **x** 控件上的水平起始位置。

[in] **y** 控件上的垂直起始位置。

[in] **cx** 控件上的宽度。

[in] **cy** 控件上的高度。

[in] **proc** 回调函数。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSaveBMPB

读取一帧图像并另存为 BMP 文件。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **fileName** 图片文件名。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSaveJpegB

读取一帧图像并另存为 JPEG 文件。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **fileName** 图片文件名。

[in] **color** 未使用的参数，保存图像色彩以当前采集的图像格式决定。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSaveHBITMAP

读取一帧图像作为 HBITMAP 对象，可以在其他 Windows API 调用中使用。

函数原型

HBITMAP __stdcall CameraSaveHBITMAP(int device_id)

参数

[in] **device_id** 相机序号。

返回值

返回一个 windows 的 HBITMAP 句柄

说明

使用 HBITMAP 后须调用 DeleteObject 释放对象，否则可能会出现内存泄漏。

CameraSaveImage

读取一帧图像并保存为图片文件。

函数原型

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

参数

[in] **device_id** 相机序号。

[in] **fileName** 图片文件名。

[in] **color** 是否为彩色图片，true 表示彩色图，false 表示黑白图片。

[in] **option** 文件格式，支持的格式为:

CAMERA_FILE_BMP24　24 位 BMP 格式

CAMERA_FILE_BMP8　8 位 BMP 格式

CAMERA_FILE_JPEG　JPEG 格式

返回值

调用成功: API_OK，调用失败: API_ERROR。

2.5.4 实用功能

该组函数是一组用于保存或显示图像的实用函数。

CameraSaveBMP

将图像数据保存为一个 BMP 图片。

函数原型

```
API_STATUS __stdcall CameraSaveBMP(char *fileName,BYTE * buf,
```

UINT width,UINT height)

参数

[in] **fileName** 图片的路径及名字。

[in] **buf** CameraQueryImage 获取的 CAMERA_IMAGE_RGB24 格式的图像数据或其它自定义数据。

[in] **width** 图像宽。

[in] **height** 图像高。

返回值

调用成功: API_OK，调用失败: API_ERROR。

例子 CameraSaveBMP(“temp。bmp”,buf,width,height)；

CameraSaveBMP8

将图像数据保存为一个 8 位 BMP 图片。

函数原型

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

参数

[in] **fileName** 图片的路径及名字。

[in] **buf** CameraQueryImage 获取的 CAMERA_IMAGE_GRAY8 格式的图像数据或其它自定义数据。

[in] **width** 图像宽。

[in] **height** 图像高。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraSaveJpeg

将图像数据保存为一个 JPEG 图片。

函数原型

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

参数

[in] **fileName** 图片的路径及名字。

[in] **dataBuf** CameraQueryImage 获取的图像数据或其它自定义数据，CAMERA_IMAGE_GRAY8 格式时，***color*** 设置为 false，CAMERA_IMAGE_RGB24 格式时，***color*** 设置为 true。

[in] **width** 图像宽。

[in] **height** 图像高。

[in] **color** 是否为彩色图片，true 表示彩色图，false 表示黑白图片。

[in] **quality** 图像质量，范围为 0 到 100，值越高质量越好。

返回值

调用成功: API_OK，调用失败: API_ERROR。

CameraShowBufferImage

使用图像数据显示图像。

函数原型

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

参数

[in] **hWnd** 显示图像控件的句柄。

[in] **buf** 图像数据。

[in] **width** 图像宽。

[in] **height** 图像高。

[in] **color** 是否为彩色图片，true 表示彩色图，false 表示黑白图片。

[in] **showStretchMode** 是否拉伸图像。

返回值

调用成功: API_OK，调用失败: API_ERROR。

例子

```
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

将图像数据保存为图片文件。

函数原型

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

参数

[in] **fileName** 图片的路径及名字。

[in] **dataBuf** 图像数据。

[in] **width** 图像宽。

[in] **height** 图像高。

[in] **color** 是否为彩色图片，true 表示彩色图，false 表示黑白图片。

[in] **quality** 图像质量，范围为 0 到 100，值越高质量越好。

[in] **option** 文件格式，支持的格式:

CAMERA_FILE_BMP24　24 位 BMP 格式

CAMERA_FILE_BMP8　8 位 BMP 格式

CAMERA_FILE_JPEG　　JPEG 格式

返回值

调用成功: API_OK，调用失败: API_ERROR。

说明

1 **quality** 只在 **option** 是 CAMERA_FILE_JPEG 时生效。

2 如果 **option** 是 CAMERA_FILE_JPEG，图像数据在 **dataBuf** 是 RGB 格式而不是 BGR。

3 修订

2023/05/19	修改 Gamma 的范围为 0.3-3	
2021/07/02	增加 LSC, LDC 功能 API	
2020/09/27	增加外部 IO 极性时延相关 API	
2020/04/27	增加 12bit 支持相关 API，增加线扫描相关 API	
2019/11/26	增加新相机型号，修改 SetROI 说明	
2012/11/28	初始发行	