

# MVSDK API

## 模块

这里列出了所有模块:

[详情级别 1 2 3]

▼API	
▼基本功能	
枚举相机	
打开相机	
播放控制	
相机设置页	
图像抓取	
抓图回调	
图像处理	
图像显示	
关闭相机	
曝光增益	
ROI	
触发功能	
GPIO	
高级设置	
色彩调节	
图像增强	
实用功能	
镜像旋转	
图片保存	
录像功能	
用户数据	
参数加载	

掉线重连

彩点修正

镜头校正

多目相关

缩放显示

▼采集器

创建

控制

取图回调

抓图

销毁

**Image**

类型定义

枚举类型

宏定义

# MVSDK API

## API

模块

### 模块

基本功能

曝光增益

ROI

触发功能

GPIO

高级设置

色彩调节

图像增强

实用功能

镜像旋转

图片保存

录像功能

用户数据

参数加载

掉线重连

彩点修正

镜头校正

多目相关

缩放显示

## 详细描述

---

# MVSDK API

## 基本功能 API

模块 | 函数

### 模块

枚举相机

打开相机

播放控制

相机设置页

图像抓取

抓图回调

图像处理

图像显示

关闭相机

### 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSdkInit** (int iLanguageSel)

初始化SDK语言。该函数在整个进程运行期间只需要调用一次。[更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetSysOption** (char const \*optionName, char const \*value)

配置系统选项（通常需要在 CameraInit 打开相机之前配置好）[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall [CameraGetInformation](#)  
(**CameraHandle** hCamera,  
char \*\*pbuffer)  
获得相机的描述信息 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall [CameraGetCapability](#)  
(**CameraHandle** hCamera,  
**tSdkCameraCapbility**  
\*pCameraInfo)  
获得相机的特性描述结构体。  
该结构体中包含了相机可设置的各种参数的范围信息。决定了相关函数的参数返回，也可用于动态创建相机的配置界面。[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall [CameraGetFrameStatistic](#)  
(**CameraHandle** hCamera,  
**tSdkFrameStatistic**  
\*psFrameStatistic)  
获得相机接收帧率的统计信息，包括错误帧和丢帧的情况。[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall [CameraSdkGetVersionString](#)  
(char \*pVersionString)  
读取SDK版本号 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall [CameraGetFirmwareVersion](#)  
(**CameraHandle** hCamera,  
char \*pVersion)  
获得固件版本的字符串 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall [CameraGetEnumInfo](#)  
(**CameraHandle** hCamera,

**tSdkCameraDevInfo**

**\*pCameraInfo)**

获得指定设备的枚举信息 [更多...](#)

MVSDK\_API **CameraSdkStatus \_\_stdcall CameraGetInterfaceVersion**

**(CameraHandle hCamera,**

**char \*pVersion)**

获得指定设备接口的版本 [更多...](#)

MVSDK\_API **CameraSdkStatus \_\_stdcall CameraGetCapabilityEx2**

**(CameraHandle hCamera, int**

**\*pMaxWidth, int \*pMaxHeight,**

**int \*pbColorCamera)**

获得该相机的一些特性。 [更多...](#)

## 详细描述

## 函数说明

### ◆ CameraGetCapability()

```
MVSDK_API  
CameraSdkStatus __stdcall  
CameraGetCapability( CameraHandle hCamera,  
                      tSdkCameraCapbility * pCameraInfo  
)
```

获得相机的特性描述结构体。该结构体中包含了相机可设置的各种参数的范围信息。决定了相关函数的参数返回，也可用于动态创建相机的配置界面。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **pCameraInfo** 指针，返回该相机特性描述的结构体。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

#### 示例：

[grab.cpp](#), [grab\\_ex.cpp](#), [roi.cpp](#), [soft\\_trigger.cpp](#),  
[win\\_basic.cpp](#), [win\\_callback.cpp](#) , 以及 [win\\_grabber.cpp](#).

### ◆ CameraGetCapabilityEx2()

```
MVSDK_API CameraSdkStatus ( CameraHandle hCamera,
```

```
__stdcall CameraGetCapabilityEx2
{
    int *          pMaxWidth,
    int *          pMaxHeight,
    int *          pbColorCamera
}
```

获得该相机的一些特性。

#### 参数

[in] <b>hCamera</b>	相机的句柄。
[out] <b>pMaxWidth</b>	返回该相机最大分辨率的宽度
[out] <b>pMaxHeight</b>	返回该相机最大分辨率的高度
[out] <b>pbColorCamera</b>	返回该相机是否是彩色相机。1表示彩色相机，0表示黑白相机

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetEnumInfo()

MVSDK\_API **CameraSdkStatus**

```
__stdcall CameraGetEnumInfo ( CameraHandle      hCamera,
                             tSdkCameraDevInfo * pCameraInfo
)
```

获得指定设备的枚举信息

#### 参数

[in] <b>hCamera</b>	相机的句柄。
[out] <b>pCameraInfo</b>	指针，返回设备的枚举信息。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

#### 示例:

## grab\_ex.cpp.

### ◆ CameraGetFirmwareVersion()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetFirmwareVersion( CameraHandle hCamera,  
                           char * pVersion  
                         )
```

获得固件版本的字符串

#### 参数

- [in] **hCamera** 相机的句柄。
- [out] **pVersion** 必须指向一个大于32字节的缓冲区，返回固件的版本字符串。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetFrameStatistic()

```
MVSDK_API  
CameraSdkStatus __stdcall  
CameraGetFrameStatistic( CameraHandle hCamera,  
                          tSdkFrameStatistic * psFrameStatistic  
                        )
```

获得相机接收帧率的统计信息，包括错误帧和丢帧的情况。

#### 参数

- [in] **hCamera** 相机的句柄。
- [out] **psFrameStatistic** 指针，返回统计信息。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraGetInterfaceVersion()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetInterfaceVersion  
    ( CameraHandle hCamera,  
      char *          pVersion  
    )
```

获得指定设备接口的版本

### 参数

[in] **hCamera** 相机的句柄。  
[out] **pVersion** 指向一个大于32字节的缓冲区, 返回接口版本字符串。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraGetInformation()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetInformation  
    ( CameraHandle hCamera,  
      char **        pBuffer  
    )
```

获得相机的描述信息

### 参数

[in] **hCamera** 相机的句柄。  
[out] **pBuffer** 指向相机描述信息指针的指针。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraSdkGetVersionString()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSdkGetVersionString ( char \* pVersionString )

读取SDK版本号

#### 参数

[out] **pVersionString** 指针, 返回SDK版本字符串。该指针指向的缓冲区大小必须大于32个字节

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraSdkInit()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSdkInit ( int iLanguageSel )

初始化SDK语言。该函数在整个进程运行期间只需要调用一次。

#### 参数

[in] **iLanguageSel** 用于选择SDK内部提示信息和界面的语种,0:表示英文,1:表示中文。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraSetSysOption()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetSysOption( char const * optionName,  
                    char const * value  
)
```

配置系统选项（通常需要在CameraInit打开相机之前配置好）

### 参数

[in] **optionName** 选项("NumBuffers", "3")  
[in] **value** 值

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

# MVSDK API

## 枚举相机

[API](#) » [基本功能](#)

函数

### 函数

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraEnumerateDevice** (**tSdkCameraDevInfo** \*pCameraList, INT \*piNums)  
枚举设备，并建立设备列表 [更多...](#)

MVSDK\_API INT \_\_stdcall **CameraEnumerateDeviceEx** ()  
枚举设备，并建立设备列表。  
在调用**CameraInitEx**之前，必须调用该函数枚举设备。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGigeEnumerateDevice** (char const \*\*ppIpList, int numIp, **tSdkCameraDevInfo** \*pCameraList, int \*piNums)  
从指定IP枚举GIGE设备，并建立设备列表（适用于相机和电脑不在同一网段的情况） [更多...](#)

## 详细描述

## 函数说明

### ◆ CameraEnumerateDevice()

```
MVSDK_API CameraSdkStatus  
__stdcall CameraEnumerateDevice ( tSdkCameraDevInfo * pCameraList,  
                                     INT * piNums  
                                   )
```

枚举设备，并建立设备列表

#### 参数

[out] <b>pCameraList</b>	设备列表数组指针
[in,out] <b>piNums</b>	设备的个数指针，调用时传入pCameraList数组的元素个数，函数返回时，保存实际找到的设备个数

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义

#### 警告

piNums指向的值必须初始化，且不超过pCameraList数组元素个数，否则有可能造成内存溢出

#### 注解

返回的相机信息列表，会根据acFriendlyName排序的。例如可以将两个相机分别改为“Camera1”和“Camera2”的名字后，名字为“Camera1”的相机会排前面，名为“Camera2”的相机排后面。

#### 示例：

[grab.cpp](#), [roi.cpp](#), [soft\\_trigger.cpp](#), [win\\_basic.cpp](#),  
[win\\_callback.cpp](#), 以及 [win\\_grabber.cpp](#).

## ◆ CameraEnumerateDeviceEx()

MVSDK\_API INT \_\_stdcall CameraEnumerateDeviceEx( )

枚举设备，并建立设备列表。在调用[CameraInitEx](#)之前，必须调用该函数枚举设备。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

### 示例：

[grab\\_ex.cpp](#).

## ◆ CameraGigeEnumerateDevice()

```
MVSDK_API CameraSdkStatus  
__stdcall  
CameraGigeEnumerateDevice ( char const **          pplpList,  
                           int                  numIp,  
                           tSdkCameraDevInfo * pCameraList,  
                           int *                piNums  
                         )
```

从指定IP枚举GIGE设备，并建立设备列表（适用于相机和电脑不在同一网段的情况）

### 参数

[in]	<b>pplpList</b>	目标IP
[in]	<b>numIp</b>	目标IP个数
[out]	<b>pCameraList</b>	设备列表数组指针
[in,out]	<b>piNums</b>	设备的个数指针，调用时传入pCameraList数组的元素个数，函数返回时，保存实际找

到的设备个数

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义

### 警告

piNums指向的值必须初始化, 且不超过pCameraList数组元素个数, 否则有可能造成内存溢出

### 注解

返回的相机信息列表, 会根据acFriendlyName排序的。例如可以将两个相机分别改为“Camera1”和“Camera2”的名字后, 名字为“Camera1”的相机会排前面, 名为“Camera2”的相机排后面。

# MVSDK API

## 打开相机

[API](#) » [基本功能](#)

函数

### 函数

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameralOpened</b> ( <b>tSdkCameraDevInfo</b> *pCameraInfo, BOOL *pOpened) 检测设备是否已经被打开 <a href="#">更多...</a>
--	--

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameralInit</b> ( <b>tSdkCameraDevInfo</b> *pCameraInfo, int emParamLoadMode, int emTeam, <b>CameraHandle</b> *pCameraHandle) 相机初始化。初始化成功后， 才能调用其他相机相关的操作 接口。 <a href="#">更多...</a>
--	--

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameralInitEx</b> (int iDeviceIndex, int emParamLoadMode, int emTeam, <b>CameraHandle</b> *pCameraHandle) 相机初始化。初始化成功后， 才能调用其他相机相关的操作 接口。 <a href="#">更多...</a>
--	---

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameralInitEx2</b> (char *CameraName, <b>CameraHandle</b> *pCameraHandle) 相机初始化。初始化成功后， 才能调用其他相机相关的操作
--	---

接口。更多...

---

## 详细描述

## 函数说明

### ◆ CameraInit()

```
MVSDK_API  
CameraSdkStatus  
__stdcall CameraInit( tSdkCameraDevInfo * pCameraInfo,  
                      int          emParamLoadMode,  
                      int          emTeam,  
                      CameraHandle * pCameraHandle  
)
```

相机初始化。初始化成功后，才能调用其他相机相关的操作接口。

#### 参数

[in] <b>pCameraInfo</b>	设备的枚举信息结构体指针，由 <b>CameraEnumerateDevice</b> 获得。
[in] <b>emParamLoadMode</b>	相机初始化时使用的参数加载方式。-1表示使用上次退出时的参数加载方式。其它取值参考 <b>emSdkParameterMode</b> 定义。
[in] <b>emTeam</b>	初始化时使用的参数组。-1表示加载上次退出时的参数组。
[out] <b>pCameraHandle</b>	相机的句柄指针，初始化成功后，该指针返回该相机的有效句柄。

#### 返回

成功返回**CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考**CameraStatus.h**中错误码的定义。

#### 示例:

**grab.cpp**, **roi.cpp**, **soft\_trigger.cpp**, **win\_basic.cpp**, 以及  
**win\_callback.cpp**.

## ◆ CameraInitEx()

### MVSDK\_API CameraSdkStatus

```
__stdcall CameraInitEx( int iDeviceIndex,  
                        int emParamLoadMode,  
                        int emTeam,  
                        CameraHandle * pCameraHandle  
)
```

相机初始化。初始化成功后，才能调用其他相机相关的操作接口。

#### 参数

[in] <b>iDeviceIndex</b>	相机的索引号， <b>CameraEnumerateDeviceEx</b> 返回相机个数。
[in] <b>emParamLoadMode</b>	相机初始化时使用的参数加载方式。-1表示使用上次退出时的参数加载方式。其它取值参考 <b>emSdkParameterMode</b> 定义。
[in] <b>emTeam</b>	初始化时使用的参数组。-1表示加载上次退出时的参数组。
[out] <b>pCameraHandle</b>	相机的句柄指针，初始化成功后，该指针返回该相机的有效句柄。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

#### 示例:

**grab\_ex.cpp**.

## ◆ CameraInitEx2()

```
MVSDK_API CameraSdkStatus __stdcall CameraInitEx2( char * CameraName,  
                                                    CameraHandle * pCameraHandle  
                                                    )
```

相机初始化。初始化成功后，才能调用其他相机相关的操作接口。

### 参数

- [in] **CameraName** 相机昵称。  
**tSdkCameraDevInfo.acFriendlyName**
- [out] **pCameraHandle** 相机的句柄指针，初始化成功后，该指针返回该相机的有效句柄。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CamerasOpened()

```
MVSDK_API CameraSdkStatus __stdcall CamerasOpened( tSdkCameraDevInfo * pCameraInfo,  
                                                    BOOL * pOpened  
                                                    )
```

检测设备是否已经被打开

### 参数

- [in] **pCameraInfo** 设备的枚举信息结构体指针，由 **CameraEnumerateDevice** 获得。
- [out] **pOpened** 设备的状态指针，返回设备是否被打开的状态，TRUE为打开，FALSE为空闲。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

# MVSDK API

## 播放控制

[API](#) » 基本功能

函数

## 函数

<b>MVSDK_API CameraSdkStatus __stdcall CameraPlay (CameraHandle hCamera)</b>	让相机进入工作模式，开始接收来自相机发送的图像数据。 <a href="#">更多...</a>
--	--

<b>MVSDK_API CameraSdkStatus __stdcall CameraPause (CameraHandle hCamera)</b>	让相机进入暂停模式，不接收来自相机的图像数据，同时也会发送命令让相机暂停输出，释放传输带宽。暂停模式下，可以对相机的参数进行配置，并立即生效。 <a href="#">更多...</a>
---	---

<b>MVSDK_API CameraSdkStatus __stdcall CameraStop (CameraHandle hCamera)</b>	让相机进入停止状态，一般是反初始化时调用该函数，该函数被调用，不能再对相机的参数进行配置。 <a href="#">更多...</a>
--	---

## 详细描述

## 函数说明

### ◆ CameraPause()

MVSDK\_API **CameraSdkStatus**  
\_\_stdcall CameraPause ( **CameraHandle hCamera** )

让相机进入暂停模式，不接收来自相机的图像数据，同时也会发送命令让相机暂停输出，释放传输带宽。暂停模式下，可以对相机的参数进行配置，并立即生效。

#### 参数

[in] **hCamera** 相机的句柄。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraPlay()

MVSDK\_API **CameraSdkStatus**  
\_\_stdcall CameraPlay ( **CameraHandle hCamera** )

让相机进入工作模式，开始接收来自相机发送的图像数据。

#### 参数

[in] **hCamera** 相机的句柄。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## 示例:

**grab.cpp**, **grab\_ex.cpp**, **roi.cpp**, **soft\_trigger.cpp**,  
**win\_basic.cpp**, 以及 **win\_callback.cpp**.

## ◆ CameraStop()

MVSDK\_API **CameraSdkStatus**  
\_\_stdcall CameraStop ( **CameraHandle hCamera** )

让相机进入停止状态, 一般是反初始化时调用该函数, 该函数被调用, 不能再对相机的参数进行配置。

## 参数

[in] **hCamera** 相机的句柄。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

# MVSDK API

类型定义 | 函数

## 相机设置页

API » 基本功能

### 类型定义

```
typedef void(WINAPI * CAMERA_PAGE_MSG_PROC) (CameraHandle  
hCamera, UINT MSG, UINT uParam, PVOID  
pContext)
```

### 函数

<b>MVSDK_API CameraSdkStatus __stdcall</b>	<b>CameraShowSettingPage</b> ( <b>CameraHandle</b> hCamera, <b>BOOL</b> bShow) 设置相机属性配置窗口显示状态。 <a href="#">更多...</a>
--	--

<b>MVSDK_API CameraSdkStatus __stdcall</b>	<b>CameraCreateSettingPage</b> ( <b>CameraHandle</b> hCamera, <b>HWND</b> hParent, <b>char</b> *pWinText, <b>CAMERA_PAGE_MSG_PROC</b> pCallbackFunc, <b>PVOID</b> pCallbackCtx, <b>UINT</b> uReserved) 创建该相机的属性配置窗口。调用 该函数，SDK内部会帮您创建好相 机的配置窗口，省去了您重新开发 相机配置界面的时间。强烈建议使 用您使用该函数让SDK为您创建好 配置窗口。 <a href="#">更多...</a>
--	---

<b>MVSDK_API CameraSdkStatus __stdcall</b>	<b>CameraCreateSettingPageEx</b> ( <b>CameraHandle</b> hCamera) 使用默认参数创建该相机的属性配 置窗口。 <a href="#">更多...</a>
--	---

<b>MVSDK_API CameraSdkStatus __stdcall</b>	<b>CameraSetActiveSettingSubPage</b> ( <b>CameraHandle</b> hCamera, <b>INT</b> index)
--	---

设置相机配置窗口的激活页面。相机配置窗口有多个子页面构成，该函数可以设定当前哪一个子页面为激活状态，显示在最前端。[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraSetSettingPageParent**  
**(CameraHandle hCamera, HWND hParentWnd, DWORD Flags)**  
把相机配置页设置为子窗口风格，  
并且指定它的父窗口。[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetSettingPageHWnd**  
**(CameraHandle hCamera, HWND \*hWnd)**  
获取相机配置页的窗口句柄。[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraUpdateSettingPage**  
**(CameraHandle hCamera)**  
刷新相机配置页 [更多...](#)

## 详细描述

---

### 类型定义说明

---

#### ◆ CAMERA\_PAGE\_MSG\_PROC

```
typedef void(WINAPI* CAMERA_PAGE_MSG_PROC) (CameraHandle  
hCamera, UINT MSG, UINT uParam, PVOID pContext)
```

相机配置页面的消息回调函数定义

## 函数说明

### ◆ CameraCreateSettingPage()

MVSDK\_API

**CameraSdkStatus**

stdcall

CameraCreateSettingPage ( **CameraHandle**

**HWND**

**char \***

**CAMERA\_PAGE\_MSG\_PROC**

**PVOID**

**UINT**

**hCamera**,

**hParent**,

**pWinText**,

**pCallbackFunc**,

**pCallbackCtx**,

**uReserved**

)

创建该相机的属性配置窗口。调用该函数，SDK内部会帮您创建好相机的配置窗口，省去了您重新开发相机配置界面的时间。强烈建议使用您使用该函数让SDK为您创建好配置窗口。

#### 参数

[in] <b>hCamera</b>	相机的句柄。
[in] <b>hParent</b>	应用程序主窗口的句柄。可以为NULL。
[in] <b>pWinText</b>	字符串指针，窗口显示的标题栏。
[in] <b>pCallbackFunc</b>	窗口消息的回调函数，当相应的事件发生时， pCallbackFunc所指向的函数会被调用
[in] <b>pCallbackCtx</b>	回调函数的附加参数。可以为NULL。
[in] <b>uReserved</b>	预留。必须设置为0。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，  
请参考 **CameraStatus.h** 中错误码的定义。

#### 示例:

[win\\_basic.cpp](#), [win\\_callback.cpp](#) , 以及 [win\\_grabber.cpp](#).

- ◆ CameraCreateSettingPageEx()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraCreateSettingPageEx ( CameraHandle hCamera )
```

使用默认参数创建该相机的属性配置窗口。

## 参数

[in] **hCamera** 相机的句柄。

返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

- ◆ CameraGetSettingPageHWnd()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetSettingPageHWnd  
    ( CameraHandle hCamera,  
      HWND * hWnd  
    )
```

获取相机配置页的窗口句柄。

参数

[in] **hCamera** 相机的句柄。

[out] hWnd 返回配置页的窗口句柄。

返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

- ◆ CameraSetActiveSettingSubPage()

MVSDK\_API CameraSdkStatus \_\_stdcall  
CameraSetActiveSettingSubPage ( CameraHandle hCamera,  
INT index

)

设置相机配置窗口的激活页面。相机配置窗口有多个子页面构成，该函数可以设定当前哪一个子页面为激活状态，显示在最前端。

### 参数

[in] **hCamera** 相机的句柄。

[in] **index** 子页面的索引号。参考[emSdkPropSheetMask](#)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetSettingPageParent()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetSettingPageParent

( **CameraHandle** hCamera,  
  **HWND**                 hParentWnd,  
  **DWORD**              Flags  
)

把相机配置页设置为子窗口风格，并且指定它的父窗口。

### 参数

[in] **hCamera** 相机的句柄。

[in] **hParentWnd** 父窗口句柄，为NULL(0)则恢复配置页为弹出窗口。

[in] **Flags** 功能标志位，bit0: 隐藏标题栏，bit1-31: 保留(必须为0)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraShowSettingPage()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraShowSettingPage

( **CameraHandle** hCamera,  
  **BOOL**              bShow

)

设置相机属性配置窗口显示状态。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **bShow** TRUE, 显示;FALSE, 隐藏。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码,  
请参考 **CameraStatus.h** 中错误码的定义。

### 注解

必须先调用**CameraCreateSettingPage**成功创建相机属性配置窗口后,  
才能调用本函数进行显示。

### 示例:

[win\\_basic.cpp](#), [win\\_callback.cpp](#), 以及 [win\\_grabber.cpp](#).

## ◆ CameraUpdateSettingPage()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraUpdateSettingPage ( **CameraHandle hCamera** )

刷新相机配置页

### 参数

- [in] **hCamera** 相机的句柄。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码,  
请参考 **CameraStatus.h** 中错误码的定义。

# MVSDK API

## 图像抓取

[API](#) » [基本功能](#)

函数

### 函数

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameraGetImageBuffer</b> ( <b>CameraHandle</b> hCamera, <b>tSdkFrameHead</b> *pFrameInfo, BYTE **pbyBuffer, UINT wTimes) 获得一帧图像数据。为了提高效率, SDK在图像抓取时采用了零拷贝机 制, 本函数实际获得是内核中的一个 缓冲区地址。 <a href="#">更多...</a>
MVSDK_API unsigned char * __stdcall	<b>CameraGetImageBufferEx</b> ( <b>CameraHandle</b> hCamera, INT *piWidth, INT *piHeight, UINT wTimes) 获得一帧图像数据。该接口获得的图 像数据是已经经过图像处理的数据。 <a href="#">更多...</a>
MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameraSnapToBuffer</b> ( <b>CameraHandle</b> hCamera, <b>tSdkFrameHead</b> *pFrameInfo, BYTE **pbyBuffer, UINT wTimes) 抓拍一张图像到缓冲区中。相机会进 入抓拍模式, 并且自动切换到抓拍模 式的分辨率进行图像捕获。 <a href="#">更多...</a>
MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameraSnapJpegToFile</b> ( <b>CameraHandle</b> hCamera, char const *lpszFileName, BYTE byQuality, UINT wTimes) 抓拍一张JPEG格式图像到文件中。 (仅部分相机硬件支持此功能) <a href="#">更多...</a>
MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameraReleaseImageBuffer</b> ( <b>CameraHandle</b> hCamera, BYTE

\*pbyBuffer)  
释放由**CameraGetImageBuffer**获得的缓冲区。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetImageBufferEx2**  
(**CameraHandle** hCamera, BYTE  
\*pImageData, UINT uOutFormat, int  
\*piWidth, int \*piHeight, UINT  
wTimes)  
获得一帧图像数据。该接口获得的图  
像是经过处理后的RGB格式。 [更  
多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetImageBufferEx3**  
(**CameraHandle** hCamera, BYTE  
\*pImageData, UINT uOutFormat, int  
\*piWidth, int \*piHeight, UINT  
\*puTimeStamp, UINT wTimes)  
获得一帧图像数据。该接口获得的图  
像是经过处理后的RGB格式。 [更  
多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetImageBufferPriority**  
(**CameraHandle** hCamera,  
**tSdkFrameHead** \*pFrameInfo,  
BYTE \*\*pbyBuffer, UINT wTimes,  
UINT Priority)  
获得一帧图像数据。 [更多...](#)

MVSDK\_API unsigned char \* \_\_stdcall **CameraGetImageBufferPriorityEx**  
(**CameraHandle** hCamera, INT  
\*piWidth, INT \*piHeight, UINT  
wTimes, UINT Priority)  
获得一帧图像数据。该接口获得的图  
像是经过处理后的RGB格式。 [更  
多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetImageBufferPriorityEx2**  
(**CameraHandle** hCamera, BYTE  
\*pImageData, UINT uOutFormat, int  
\*piWidth, int \*piHeight, UINT  
wTimes, UINT Priority)  
获得一帧图像数据。该接口获得的图  
像是经过处理后的RGB格式。 [更](#)

多...

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetImageBufferPriorityEx3**  
(**CameraHandle** hCamera, BYTE \*pImageData, UINT uOutFormat, int \*piWidth, int \*piHeight, UINT \*puTimeStamp, UINT wTimes, UINT Priority)

获得一帧图像数据。该接口获得的图像是经过处理后的RGB格式。[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraClearBuffer**  
(**CameraHandle** hCamera)  
清空相机内已缓存的所有帧 [更多...](#)

---

## 详细描述

### 函数说明

#### ◆ CameraClearBuffer()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraClearBuffer( CameraHandle hCamera )
```

清空相机内已缓存的所有帧

#### 参数

[in] **hCamera** 相机的句柄。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### 示例:

[soft\\_trigger.cpp](#).

#### ◆ CameraGetImageBuffer()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetImageBuffer( CameraHandle hCamera,  
                      tSdkFrameHead * pFrameInfo,  
                      BYTE ** pbyBuffer,  
                      UINT wTimes  
                    )
```

获得一帧图像数据。为了提高效率, SDK在图像抓取时采用了零拷贝机制, 本函数实际获得是内核中的一个缓冲区地址。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **pFrameInfo** 图像的帧头信息指针。  
[out] **pbyBuffer** 返回图像数据的缓冲区指针。  
[in] **wTimes** 抓取图像的超时时间，单位毫秒。在wTimes时间内还未获得图像，则该函数会返回超时错误。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## 注解

该函数成功调用后，必须调用**CameraReleaseImageBuffer**释放缓冲区，以便让内核继续使用该缓冲区。

## 示例：

[grab.cpp](#), [roi.cpp](#)，以及 [soft\\_trigger.cpp](#).

## ◆ CameraGetImageBufferEx()

```
MVSDK_API unsigned char* __stdcall  
CameraGetImageBufferEx( CameraHandle hCamera,  
                        INT * piWidth,  
                        INT * piHeight,  
                        UINT wTimes  
)
```

获得一帧图像数据。该接口获得的图像数据是已经经过图像处理的数据。

## 参数

[in] **hCamera** 相机的句柄。  
[out] **piWidth** 整形指针，返回图像的宽度。  
[out] **piHeight** 整形指针，返回图像的高度。  
[in] **wTimes** 抓取图像的超时时间，单位毫秒。在wTimes时间内还未获得图像，则该函数会返回超时错误。

## 返回

成功时，返回帧数据缓冲区的首地址，否则返回0。

## 注解

本函数不需要调用**CameraReleaseImageBuffer**释放缓冲区。

示例：  
grab\_ex.cpp.

- ◆ CameraGetImageBufferEx2()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetImageBufferEx2(   
    _In_     CameraHandle hCamera,  
    _Out_    BYTE*          pImageData,  
    _In_     UINT           uOutFormat,  
    _Out_    int*           piWidth,  
    _Out_    int*           piHeight,  
    _In_     UINT           wTimes  
)
```

获得一帧图像数据。该接口获得的图像是经过处理后的RGB格式。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **pImageData** 接收图像数据的缓冲区，大小必须和uOutFormat指定的格式相匹配，否则数据会溢出。  
[in] **uOutFormat** 输出格式 0:Mono8 1:rgb24 2:rgba32 3:bgr24  
4:bgra32  
[out] **piWidth** 整形指针，返回图像的宽度  
[out] **piHeight** 整形指针，返回图像的高度  
[in] **wTimes** 抓取图像的超时时间。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

不需要调用 [CameraReleaseImageBuffer](#)

- ◆ CameraGetImageBufferEx3()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetImageBufferEx3(   
    _In_     CameraHandle hCamera,  
    _Out_    BYTE*          pImageData,
```

```
    UINT          uOutFormat,  
    int *         piWidth,  
    int *         piHeight,  
    UINT *        puTimeStamp,  
    UINT          wTimes  
)
```

获得一帧图像数据。该接口获得的图像是经过处理后的RGB格式。

## 参数

[in] <b>hCamera</b>	相机的句柄。
[out] <b>plImageData</b>	接收图像数据的缓冲区，大小必须和uOutFormat指定的格式相匹配，否则数据会溢出
[in] <b>uOutFormat</b>	输出格式 0:Mono8 1:rgb24 2:rgba32 3:bgr24 4:bgra32
[out] <b>piWidth</b>	整形指针，返回图像的宽度
[out] <b>piHeight</b>	整形指针，返回图像的高度
[out] <b>puTimeStamp</b>	返回图像时间戳
[in] <b>wTimes</b>	抓取图像的超时时间。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## 注解

不需要调用 [CameraReleaseImageBuffer](#)

## ◆ CameraGetImageBufferPriority()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetImageBufferPriority
```

```
( CameraHandle     hCamera,  
  tSdkFrameHead * pFrameInfo,  
  BYTE **          pbyBuffer,  
  UINT             wTimes,  
  UINT             Priority  
)
```

获得一帧图像数据。

## 参数

- [in] **hCamera** 相机的句柄。
- [out] **pFrameInfo** 图像的帧头信息指针。
- [out] **pbyBuffer** 指向图像的数据的缓冲区指针。
- [in] **wTimes** 抓取图像的超时时间。
- [in] **Priority** 取图优先级 详见：[emCameraGetImagePriority](#)

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## 注解

除了多一个优先级参数外与[CameraGetImageBuffer](#)相同

## 示例:

[win\\_basic.cpp](#).

## ◆ CameraGetImageBufferPriorityEx()

```
MVSDK_API unsigned char* __stdcall  
CameraGetImageBufferPriorityEx ( CameraHandle hCamera,  
                                INT * piWidth,  
                                INT * piHeight,  
                                UINT wTimes,  
                                UINT Priority  
                                )
```

获得一帧图像数据。该接口获得的图像是经过处理后的RGB格式。

## 参数

- [in] **hCamera** 相机的句柄。
- [out] **piWidth** 整形指针, 返回图像的宽度
- [out] **piHeight** 整形指针, 返回图像的高度
- [in] **wTimes** 抓取图像的超时时间。单位毫秒。
- [in] **Priority** 取图优先级 详见：[emCameraGetImagePriority](#)

## 返回

成功时, 返回RGB数据缓冲区的首地址;否则返回0。

## 注解

除了多一个优先级参数外与[CameraGetImageBufferEx](#)相同

## ◆ CameraGetImageBufferPriorityEx2()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetImageBufferPriorityEx2

( **CameraHandle** hCamera,  
BYTE \* pImageData,  
UINT uOutFormat,  
int \* piWidth,  
int \* piHeight,  
UINT wTimes,  
UINT Priority  
)

获得一帧图像数据。该接口获得的图像是经过处理后的RGB格式。

### 参数

[in] **hCamera** 相机的句柄。  
[out] **pImageData** 接收图像数据的缓冲区，大小必须和uOutFormat指定的格式相匹配，否则数据会溢出  
[in] **uOutFormat** 输出格式 0:Mono8 1:rgb24 2:rgba32 3:bgr24  
4:bgra32  
[out] **piWidth** 整形指针，返回图像的宽度  
[out] **piHeight** 整形指针，返回图像的高度  
[in] **wTimes** 抓取图像的超时时间。单位毫秒。  
[in] **Priority** 取图优先级 详见：[emCameraGetImagePriority](#)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

### 注解

除了多一个优先级参数外与[CameraGetImageBufferEx2](#)相同

## ◆ CameraGetImageBufferPriorityEx3()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetImageBufferPriorityEx3

( **CameraHandle** hCamera,  
BYTE \* pImageData,

```

        UINT          uOutFormat,
        int *         piWidth,
        int *         piHeight,
        UINT *        puTimeStamp,
        UINT          wTimes,
        UINT          Priority
    )

```

获得一帧图像数据。该接口获得的图像是经过处理后的RGB格式。

## 参数

[in] <b>hCamera</b>	相机的句柄。
[out] <b>pImageData</b>	接收图像数据的缓冲区，大小必须和uOutFormat指定的格式相匹配，否则数据会溢出
[in] <b>uOutFormat</b>	输出格式 0:Mono8 1:rgb24 2:rgba32 3:bgr24 4:bgra32
[out] <b>piWidth</b>	整形指针，返回图像的宽度
[out] <b>piHeight</b>	整形指针，返回图像的高度
[out] <b>puTimeStamp</b>	返回图像时间戳
[in] <b>wTimes</b>	抓取图像的超时时间。
[in] <b>Priority</b>	取图优先级 详见： <a href="#">emCameraGetImagePriority</a>

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## 注解

除了多一个优先级参数外与[CameraGetImageBufferEx3](#)相同

## ◆ CameraReleaseImageBuffer()

```

MVSdk_API CameraSdkStatus __stdcall
CameraReleaseImageBuffer(
    ( CameraHandle hCamera,
    BYTE *       pbyBuffer
)

```

释放由[CameraGetImageBuffer](#)获得的缓冲区。

## 参数

[in] <b>hCamera</b>	相机的句柄。
---------------------	--------

[in] **pbyBuffer** 帧缓冲区地址。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### 示例:

[grab.cpp](#), [roi.cpp](#), [soft\\_trigger.cpp](#), 以及 [win\\_basic.cpp](#).

### ◆ CameraSnapJpegToFile()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSnapJpegToFile

( **CameraHandle** hCamera,  
char const \* lpszFileName,  
BYTE byQuality,  
UINT wTimes  
)

抓拍一张JPEG格式图像到文件中。(仅部分相机硬件支持此功能)

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **lpszFileName** 图片保存文件完整路径。  
[in] **byQuality** 图像保存的质量因子, 范围1到100。  
[in] **wTimes** 抓取图像的超时时间, 单位毫秒。在wTimes时间内还未获得图像, 则该函数会返回超时错误。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraSnapToBuffer()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSnapToBuffer

( **CameraHandle** hCamera,  
**tSdkFrameHead** \* pFrameInfo,  
BYTE \*\* pbyBuffer,  
UINT wTimes

)

抓拍一张图像到缓冲区中。相机会进入抓拍模式，并且自动切换到抓拍模式的分辨率进行图像捕获。

## 参数

- [in] **hCamera** 相机的句柄。
- [out] **pFrameInfo** 图像的帧头信息指针。
- [out] **pbyBuffer** 返回图像数据的缓冲区指针。
- [in] **wTimes** 抓取图像的超时时间，单位毫秒。在wTimes时间内还未获得图像，则该函数会返回超时错误。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## 注解

该函数成功调用后，必须调用[CameraReleaseImageBuffer](#)释放缓冲区，以便让内核继续使用该缓冲区。

## 警告

本函数可能会进行分辨率切换，因此效率会比[CameraGetImageBuffer](#)低。如果没有切换分辨率抓拍的需求，请使用[CameraGetImageBuffer](#)。

# MVSDK API

类型定义 | 函数

## 抓图回调

API » 基本功能

### 类型定义

```
typedef void(WINAPI * CAMERA_SNAP_PROC) (CameraHandle  
hCamera, BYTE *pFrameBuffer,  
tSdkFrameHead *pFrameHead, PVOID  
pContext)
```

### 函数

**MVSDK\_API CameraSdkStatus \_\_stdcall CameraSetCallbackFunction**  
(**CameraHandle** hCamera,  
**CAMERA\_SNAP\_PROC**  
pCallBack, **PVOID** pContext,  
**CAMERA\_SNAP\_PROC**  
\*pCallbackOld)

设置图像捕获的回调函数。当  
捕获到新的图像数据帧时，  
pCallBack所指向的回调函数  
就会被调用。[更多...](#)

## 详细描述

---

### 类型定义说明

---

#### ◆ CAMERA\_SNAP\_PROC

```
typedef void(WINAPI* CAMERA_SNAP_PROC) (CameraHandle  
hCamera, BYTE *pFrameBuffer, tSdkFrameHead *pFrameHead, PVOID  
pContext)
```

图像捕获的回调函数定义

## 函数说明

### ◆ CameraSetCallbackFunction()

```
MVSDK_API  
CameraSdkStatus __stdcall  
CameraSetCallbackFunction ( CameraHandle hCamera,  
                            CAMERA_SNAP_PROC pCallBack,  
                            PVOID pContext,  
                            CAMERA_SNAP_PROC * pCallbackOld  
                           )
```

设置图像捕获的回调函数。当捕获到新的图像数据帧时，pCallBack所指向的回调函数就会被调用。

#### 参数

[in]	<b>hCamera</b>	相机的句柄。
[in]	<b>pCallBack</b>	回调函数指针。
[in]	<b>pContext</b>	回调函数的附加参数，在回调函数被调用时该附加参数会被传入，可以为NULL。多用于多个相机时携带附加信息。
[out]	<b>pCallbackOld</b>	用于返回之前设置的回调函数。可以为NULL。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

#### 示例：

[win\\_callback.cpp](#).

# MVSDK API

## 图像处理

[API](#) » [基本功能](#)

函数

### 函数

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameralmageProcess</b> ( <b>CameraHandle</b> hCamera, BYTE *pbyIn, BYTE *pbyOut, <b>tSdkFrameHead</b> *pFrInfo)	将获得的相机原始输出图像数据 进行处理，叠加饱和度、颜色增 益和校正、降噪等处理效果，最 后得到RGB888格式的图像数 据。 <a href="#">更多...</a>
--	---	---

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameralmageProcessEx</b> ( <b>CameraHandle</b> hCamera, BYTE *pbyIn, BYTE *pbyOut, <b>tSdkFrameHead</b> *pFrInfo, <b>UINT</b> <b>uOutFormat</b> , <b>UINT</b> <b>uReserved</b> )	将获得的相机原始输出图像数据 进行处理，叠加饱和度、颜色增 益和校正、降噪等处理效果，最 后得到RGB888格式的图像数 据。 <a href="#">更多...</a>
--	---	---

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameralmageOverlay</b> ( <b>CameraHandle</b> hCamera, BYTE *pRgbBuffer, <b>tSdkFrameHead</b> *pFrInfo)	将输入的图像数据上叠加十字 线、白平衡参考窗口、自动曝光 参考窗口等图形。只有设置为可 见状态的十字线和参考窗口才能 被叠加上。 <a href="#">更多...</a>
--	--	--

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameraSetBayerDecAlgorithm</b>
--	-----------------------------------

([CameraHandle](#) hCamera, INT iLspProcessor, INT iAlgorithmSel)  
设置Bayer数据转彩色的算法。  
[更多...](#)

MVSDK\_API [CameraSdkStatus](#) \_\_stdcall [CameraGetBayerDecAlgorithm](#)  
([CameraHandle](#) hCamera, INT iLspProcessor, INT \*piAlgorithmSel)  
获得Bayer数据转彩色所选择的算法。[更多...](#)

MVSDK\_API [CameraSdkStatus](#) \_\_stdcall [CameraSetBlackLevel](#)  
([CameraHandle](#) hCamera, INT iBlackLevel)  
设置图像的黑电平基准， 默认值为0 [更多...](#)

MVSDK\_API [CameraSdkStatus](#) \_\_stdcall [CameraGetBlackLevel](#)  
([CameraHandle](#) hCamera, INT \*piBlackLevel)  
获得图像的黑电平基准， 默认值为0 [更多...](#)

MVSDK\_API [CameraSdkStatus](#) \_\_stdcall [CameraSetWhiteLevel](#)  
([CameraHandle](#) hCamera, INT iWhiteLevel)  
设置图像的白电平基准， 默认值为255 [更多...](#)

MVSDK\_API [CameraSdkStatus](#) \_\_stdcall [CameraGetWhiteLevel](#)  
([CameraHandle](#) hCamera, INT \*piWhiteLevel)  
获得图像的白电平基准， 默认值为255 [更多...](#)

MVSDK\_API [CameraSdkStatus](#) \_\_stdcall [CameraSetIspOutFormat](#)  
([CameraHandle](#) hCamera, UINT uFormat)  
设置[CameralImageProcess](#)函数的图像处理的输出格式。 [更](#)

多...

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetIspOutFormat**  
(**CameraHandle** hCamera, **UINT**  
\*puFormat)  
获取输出格式 [更多...](#)

---

## 详细描述

## 函数说明

### ◆ CameraGetBayerDecAlgorithm()

```
MVSDK_API CameraSdkStatus  
__stdcall CameraGetBayerDecAlgorithm ( CameraHandle hCamera,  
                                         INT             ilspProcessor,  
                                         INT *           piAlgorithmSel  
                                       )
```

获得Bayer数据转彩色所选择的算法。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **ilspProcessor** 选择执行该算法的对象，参考  
[emSdkIspProcessor](#)  
[in] **piAlgorithmSel** 返回当前选择的算法编号。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetBlackLevel()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetBlackLevel ( CameraHandle hCamera,  
                      INT *           piBlackLevel  
                    )
```

获得图像的黑电平基准， 默认值为0

### 参数

[in] **hCamera** 相机的句柄。

[out] **piBlackLevel** 返回当前的黑电平值。范围为0到255。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetIspOutFormat()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetIspOutFormat ( CameraHandle hCamera,  
                        UINT * puFormat  
                      )
```

获取输出格式

### 参数

[in] **hCamera** 相机的句柄。

[out] **puFormat** 返回当前输出格式

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

### 参见

[CameraSetIspOutFormat](#)

## ◆ CameraGetWhiteLevel()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetWhiteLevel ( CameraHandle hCamera,  
                      INT * piWhiteLevel  
                    )
```

获得图像的白电平基准， 默认值为255

### 参数

[in] **hCamera** 相机的句柄。

[out] **piWhiteLevel** 返回当前的白电平值。范围为0到255。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameralImageOverlay()

MVSDK\_API **CameraSdkStatus**

\_\_stdcall CameralImageOverlay

( **CameraHandle** hCamera,  
BYTE \* pRgbBuffer,  
**tSdkFrameHead** \* pFrInfo  
)

将输入的图像数据上叠加十字线、白平衡参考窗口、自动曝光参考窗口等图形。只有设置为可见状态的十字线和参考窗口才能被叠加上。

### 参数

[in] **hCamera** 相机的句柄。

[in] **pRgbBuffer** 图像数据缓冲区。

[in] **pFrInfo** 图像的帧头信息。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### 示例:

[win\\_basic.cpp](#) , 以及 [win\\_callback.cpp](#).

## ◆ CameralImageProcess()

MVSDK\_API **CameraSdkStatus** \_\_stdcall ( **CameraHandle** hCamera,

## CameralImageProcess

```
    BYTE *          pbyIn,
    BYTE *          pbyOut,
    tSdkFrameHead * pFrInfo
)
```

将获得的相机原始输出图像数据进行处理，叠加饱和度、颜色增益和校正、降噪等处理效果，最后得到RGB888格式的图像数据。

### 参数

[in]	<b>hCamera</b>	相机的句柄。
[in]	<b>pbyIn</b>	输入图像数据的缓冲区地址，不能为NULL。
[out]	<b>pbyOut</b>	处理后图像输出的缓冲区地址，不能为NULL。
[in,out]	<b>pFrInfo</b>	输入图像的帧头信息，处理完成后，帧头信息中的图像格式uiMediaType会随之改变。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

### 示例：

[grab.cpp](#), [roi.cpp](#), [soft\\_trigger.cpp](#), [win\\_basic.cpp](#) , 以及 [win\\_callback.cpp](#).

## ◆ CameralImageProcessEx()

MVSDK\_API **CameraSdkStatus**  
\_\_stdcall CameralImageProcessEx

```
( CameraHandle     hCamera,
    BYTE *          pbyIn,
    BYTE *          pbyOut,
    tSdkFrameHead * pFrInfo,
    UINT            uOutFormat,
    UINT            uReserved
)
```

将获得的相机原始输出图像数据进行处理，叠加饱和度、颜色增益和校正、降噪等处理效果，最后得到RGB888格式的图像数据。

## 参数

[in]	<b>hCamera</b>	相机的句柄。
[in]	<b>pbyIn</b>	输入图像数据的缓冲区地址，不能为NULL。
[out]	<b>pbyOut</b>	处理后图像输出的缓冲区地址，不能为NULL。
[in,out]	<b>pFrInfo</b>	输入图像的帧头信息，处理完成后，帧头信息中的图像格式uiMediaType会随之改变。
[in]	<b>uOutFormat</b>	处理完后图像的输出格式。可以是CAMERA_MEDIA_TYPE_MONO8、CAMERA_MEDIA_TYPE_RGB、CAMERA_MEDIA_TYPE_RGBA8的其中一种。
[in]	<b>uReserved</b>	预留参数，必须设置为0。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetBayerDecAlgorithm()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetBayerDecAlgorithm ( CameraHandle hCamera,  
                           INT             ilspProcessor,  
                           INT             iAlgorithmSel  
                         )
```

设置Bayer数据转彩色的算法。

## 参数

[in]	<b>hCamera</b>	相机的句柄。
[in]	<b>ilspProcessor</b>	选择执行该算法的对象，参考 <a href="#">emSdkIlspProcessor</a>
[in]	<b>iAlgorithmSel</b>	要选择的算法编号。从0开始，最大值由tSdkCameraCapbility.iBayerDecAlmSwDesc和tSdkCameraCapbility.iBayerDecAlmHdDesc决定。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetBlackLevel()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetBlackLevel( CameraHandle hCamera,  
                      INT iBlackLevel  
)
```

设置图像的黑电平基准, 默认值为0

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iBlackLevel** 要设定的电平值。范围为0到255。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetIspOutFormat()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetIspOutFormat( CameraHandle hCamera,  
                        UINT uFormat  
)
```

设置**CameralImageProcess**函数的图像处理的输出格式。

### 参数

[in] **hCamera** 相机的句柄。  
[in] **uFormat** 输出格式。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## 注解

支持的格式：CAMERA\_MEDIA\_TYPE\_MONO8、  
CAMERA\_MEDIA\_TYPE\_MONO16、  
CAMERA\_MEDIA\_TYPE\_RGB8、CAMERA\_MEDIA\_TYPE\_RGBA8  
、CAMERA\_MEDIA\_TYPE\_BGR8、  
CAMERA\_MEDIA\_TYPE\_BGRA8

## 示例：

[grab.cpp](#), [grab\\_ex.cpp](#), [roi.cpp](#), [soft\\_trigger.cpp](#), [win\\_basic.cpp](#),  
[win\\_callback.cpp](#), 以及 [win\\_grabber.cpp](#).

## ◆ CameraSetWhiteLevel()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetWhiteLevel( CameraHandle hCamera,  
                      INT iWhiteLevel  
)
```

设置图像的白电平基准， 默认值为255

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iWhiteLevel** 要设定的电平值。范围为0到255。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

# MVSDK API

## 图像显示

[API](#) » 基本功能

函数

### 函数

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraDisplayInit</a> (CameraHandle hCamera, HWND hWndDisplay) 初始化SDK内部的显示模 块。在调用 <a href="#">CameraDisplayRGB24</a> 前必 须先调用该函数初始化。如 果您在二次开发中，使 用自己的方式进行图像显示(不调 用CameraDisplayRGB24)则 不需要调用本函数。 <a href="#">更多...</a>
-------------------------------------	---

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraDisplayRGB24</a> (CameraHandle hCamera, BYTE *pFrameBuffer, tSdkFrameHead *pFrInfo) 显示图像。必须调用过 <a href="#">CameraDisplayInit</a> 进行初 始化才能调用本函数。 <a href="#">更 多...</a>
-------------------------------------	--

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraSetDisplayMode</a> (CameraHandle hCamera, INT iMode) 设置显示的模式。 <a href="#">更 多...</a>
-------------------------------------	---

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraSetDisplayOffset</a> (CameraHandle hCamera, int iOffsetX, int iOffsetY) 设置显示的起始偏移值。仅
-------------------------------------	---

当显示模式为  
DISPLAYMODE\_REAL时有  
效。例如显示控件的大小为  
320X240，而图像的尺寸  
为640X480，那么当iOffsetX  
= 160,iOffsetY = 120时显示  
的区域就是图像的居中  
320X240的位置。[更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraSetDisplaySize](#)  
(CameraHandle hCamera,  
INT iWidth, INT iHeight)  
设置显示控件的尺寸。[更多...](#)

## 详细描述

## 函数说明

### ◆ CameraDisplayInit()

```
MVSDK_API CameraSdkStatus  
__stdcall CameraDisplayInit( CameraHandle hCamera,  
                           HWND           hWndDisplay  
                         )
```

初始化SDK内部的显示模块。在调用**CameraDisplayRGB24**前必须先调用该函数初始化。如果您在二次开发中，使用自己的方式进行图像显示(不调用**CameraDisplayRGB24**)则不需要调用本函数。

#### 参数

[in] **hCamera** 相机的句柄。

[in] **hWndDisplay** 显示窗口的句柄，一般为窗口的m\_hWnd成员。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

#### 示例：

[win\\_basic.cpp](#) , 以及 [win\\_callback.cpp](#).

### ◆ CameraDisplayRGB24()

```
MVSDK_API CameraSdkStatus ( CameraHandle hCamera,
```

```
__stdcall CameraDisplayRGB24  
    ( BYTE * pFrameBuffer,  
      tSdkFrameHead * pFrInfo  
    )
```

显示图像。必须调用过**CameraDisplayInit**进行初始化才能调用本函数。

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **pFrameBuffer** 图像的帧缓冲区
- [in] **pFrInfo** 图像的帧头信息

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### 示例:

[win\\_basic.cpp](#), 以及 [win\\_callback.cpp](#).

### ◆ CameraSetDisplayMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetDisplayMode ( CameraHandle hCamera,  
                      INT iMode  
                    )
```

设置显示的模式。

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iMode** 显示模式, 参见**emSdkDisplayMode**的定义。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetDisplayOffset()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetDisplayOffset( CameraHandle hCamera,  
                        int           iOffsetX,  
                        int           iOffsetY  
                      )
```

设置显示的起始偏移值。仅当显示模式为DISPLAYMODE\_REAL时有效。例如显示控件的大小为320X240，而图像的尺寸为640X480，那么当iOffsetX = 160,iOffsetY = 120时显示的区域就是图像的居中320X240的位置。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iOffsetX** 偏移的X坐标。
- [in] **iOffsetY** 偏移的Y坐标。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetDisplaySize()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetDisplaySize( CameraHandle hCamera,  
                      INT           iWidth,  
                      INT           iHeight  
                    )
```

设置显示控件的尺寸。

### 参数

- [in] **hCamera** 相机的句柄。

[in] **iWidth** 宽度  
[in] **iHeight** 高度

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### 示例:

[win\\_basic.cpp](#), 以及 [win\\_callback.cpp](#).

# MVSDK API

## 关闭相机

[API](#) » [基本功能](#)

函数

## 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraUnInit**  
**(CameraHandle**  
hCamera)  
相机反初始化。释放资源。[更多...](#)

## 详细描述

---

## 函数说明

---

### ◆ CameraUnInit()

```
MVSDK_API CameraSdkStatus  
__stdcall CameraUnInit( CameraHandle hCamera )
```

相机反初始化。释放资源。

#### 参数

[in] **hCamera** 相机的句柄。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### 示例:

[grab.cpp](#), [grab\\_ex.cpp](#), [roi.cpp](#), [soft\\_trigger.cpp](#),  
[win\\_basic.cpp](#), 以及 [win\\_callback.cpp](#).

# MVSDK API

## 曝光增益

函数

API

### 函数

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraSetAeState \(CameraHandle hCamera, BOOL bAeState\)](#)  
设置相机曝光的模式。自动或者手动。[更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraGetAeState \(CameraHandle hCamera, BOOL \\*pAeState\)](#)  
获得相机当前的曝光模式。[更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraSetAeTarget \(CameraHandle hCamera, int iAeTarget\)](#)  
设定自动曝光的亮度目标值。设定范围[\[tSdkExpose.uiTargetMin, tSdkExpose.uiTargetMax\]](#) [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraGetAeTarget \(CameraHandle hCamera, int \\*piAeTarget\)](#)  
获得自动曝光的亮度目标值。[更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraSetAeExposureRange \(CameraHandle hCamera, double fMinExposureTime, double fMaxExposureTime\)](#)  
设定自动曝光模式的曝光时间调节范围 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraGetAeExposureRange \(CameraHandle hCamera, double \\*fMinExposureTime, double \\*fMaxExposureTime\)](#)  
获得自动曝光模式的曝光时间调节范围 [更多...](#)

MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetAeAnalogGainRange</b> (CameraHandle hCamera, int iMinAnalogGain, int iMaxAnalogGain) 设定自动曝光模式的增益调节范围 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetAeAnalogGainRange</b> (CameraHandle hCamera, int *iMinAnalogGain, int *iMaxAnalogGain) 获得自动曝光模式的增益调节范围 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetAeThreshold</b> (CameraHandle hCamera, int iThreshold) 设置自动曝光模式的调节阈值 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetAeThreshold</b> (CameraHandle hCamera, int *iThreshold) 获取自动曝光模式的调节阈值 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetExposureTime</b> (CameraHandle hCamera, double fExposureTime) 设置曝光时间。单位为微秒。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetExposureLineTime</b> (CameraHandle hCamera, double *pfLineTime) 获得一行的曝光时间。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetExposureTime</b> (CameraHandle hCamera, double *pfExposureTime) 获得相机的曝光时间。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetExposureTimeRange</b> (CameraHandle hCamera, double *pfMin, double *pfMax, double *pfStep) 获得相机的曝光时间范围 <a href="#">更多...</a>

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetMultiExposureTime**  
(**CameraHandle** hCamera, int index,  
double fExposureTime)  
设置多重曝光时间。单位为微秒。(此  
功能仅线阵相机支持) [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetMultiExposureTime**  
(**CameraHandle** hCamera, int index,  
double \*fExposureTime)  
获取多重曝光时间。单位为微秒。(此  
功能仅线阵相机支持) [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetMultiExposureCount**  
(**CameraHandle** hCamera, int count)  
设置多重曝光使能个数。(此功能仅线  
阵相机支持) [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetMultiExposureCount**  
(**CameraHandle** hCamera, int \*count)  
获取多重曝光使能个数。(此功能仅线  
阵相机支持) [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetMultiExposureMaxCount**  
(**CameraHandle** hCamera, int  
\*max\_count)  
获取多重曝光的最大曝光个数。(此功  
能仅线阵相机支持) [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetAnalogGain**  
(**CameraHandle** hCamera, INT  
iAnalogGain)  
设置相机的图像模拟增益值。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetAnalogGain**  
(**CameraHandle** hCamera, INT  
\*piAnalogGain)  
获得图像信号的模拟增益值。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetAntiFlick** (**CameraHandle**  
hCamera, BOOL bEnable)  
设置自动曝光时抗频闪功能的使能状  
态。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetAntiFlick**

**(CameraHandle hCamera, BOOL \*pbEnable)**  
获得自动曝光时抗频闪功能的使能状态。[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetLightFrequency**  
**(CameraHandle hCamera, int \*piFrequencySel)**  
获得自动曝光时，消频闪的频率选择。[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraSetLightFrequency**  
**(CameraHandle hCamera, int iFrequencySel)**  
设置自动曝光时消频闪的频率。[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CamerasAeWinVisible**  
**(CameraHandle hCamera, BOOL \*pbIsVisible)**  
获得自动曝光参考窗口的显示状态。[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraSetAeWinVisible**  
**(CameraHandle hCamera, BOOL bIsVisible)**  
设置自动曝光参考窗口的显示状态。[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetAeWindow**  
**(CameraHandle hCamera, INT \*piHOFF, INT \*piVOFF, INT \*piWidth, INT \*piHeight)**  
获得自动曝光参考窗口的位置。[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraSetAeWindow**  
**(CameraHandle hCamera, int iHOFF, int iVOFF, int iWidth, int iHeight)**  
设置自动曝光的参考窗口。[更多...](#)

## 详细描述

## 函数说明

### ◆ CameraGetAeAnalogGainRange()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetAeAnalogGainRange ( CameraHandle hCamera,  
                           int *          iMinAnalogGain,  
                           int *          iMaxAnalogGain  
                         )
```

获得自动曝光模式的增益调节范围

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **iMinAnalogGain** 最小增益  
[out] **iMaxAnalogGain** 最大增益

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetAeExposureRange()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetAeExposureRange ( CameraHandle hCamera,  
                           double *        fMinExposureTime,  
                           double *        fMaxExposureTime  
                         )
```

获得自动曝光模式的曝光时间调节范围

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **fMinExposureTime** 最小曝光时间 (微秒)  
[out] **fMaxExposureTime** 最大曝光时间 (微秒)

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetAeState()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetAeState  
( **CameraHandle** hCamera,  
  **BOOL** \* pAeState  
)

获得相机当前的曝光模式。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **pAeState** 返回自动曝光的使能状态。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetAeTarget()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetAeTarget  
( **CameraHandle** hCamera,  
  **int** \* piAeTarget  
)

获得自动曝光的亮度目标值。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **piAeTarget** 指针, 返回目标值。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetAeThreshold()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetAeThreshold( CameraHandle hCamera,  
                      int * iThreshold  
)
```

获取自动曝光模式的调节阈值

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **iThreshold** 读取到的调节阈值

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetAeWindow()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetAeWindow( CameraHandle hCamera,  
                    INT * piHOFF, INT * piVOFF,  
                    INT * piWidth, INT * piHeight  
)
```

获得自动曝光参考窗口的位置。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **piHOFF** 指针, 返回窗口位置左上角横坐标值。  
[out] **piVOFF** 指针, 返回窗口位置左上角纵坐标值。  
[out] **piWidth** 指针, 返回窗口的宽度。  
[out] **piHeight** 指针, 返回窗口的高度。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetAnalogGain()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetAnalogGain ( **CameraHandle** hCamera,  
                          **INT** \*               piAnalogGain  
                          )

获得图像信号的模拟增益值。

### 参数

[in] **hCamera** 相机的句柄。  
[out] **piAnalogGain** 指针, 返回当前的模拟增益值。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### 参见

[CameraSetAnalogGain](#)

## ◆ CameraGetAntiFlick()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetAntiFlick ( **CameraHandle** hCamera,  
                          **BOOL** \*              pbEnable  
                          )

获得自动曝光时抗频闪功能的使能状态。

### 参数

[in] **hCamera** 相机的句柄。  
[out] **pbEnable** 指针, 返回该功能的使能状态。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetExposureLineTime()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetExposureLineTime( CameraHandle hCamera,  
                           double * pfLineTime  
                         )
```

获得一行的曝光时间。

### 参数

[in] **hCamera** 相机的句柄。  
[out] **pfLineTime** 指针，返回一行的曝光时间，单位为微秒。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

### 注解

对于CMOS传感器，其曝光的单位是按照行来计算的，因此，曝光时间并不能在微秒级别连续可调。而是会按照整行来取舍。这个函数的作用就是返回CMOS相机曝光一行对应的时间。

## ◆ CameraGetExposureTime()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetExposureTime( CameraHandle hCamera,  
                       double * pfExposureTime  
                     )
```

获得相机的曝光时间。

### 参数

[in] **hCamera** 相机的句柄。  
[out] **pfExposureTime** 指针，返回当前的曝光时间，单位微秒。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

参见

[CameraSetExposureTime](#)

◆ CameraGetExposureTimeRange()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetExposureTimeRange

( **CameraHandle** hCamera,  
double \* pfMin,  
double \* pfMax,  
double \* pfStep  
)

获得相机的曝光时间范围

参数

- [in] **hCamera** 相机的句柄。
- [out] **pfMin** 指针，返回曝光时间的最小值，单位微秒。
- [out] **pfMax** 指针，返回曝光时间的最大值，单位微秒。
- [out] **pfStep** 指针，返回曝光时间的步进值，单位微秒。

返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

◆ CameraGetLightFrequency()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetLightFrequency

( **CameraHandle** hCamera,  
int \* piFrequencySel  
)

获得自动曝光时，消频闪的频率选择。

参数

- [in] **hCamera** 相机的句柄。
- [out] **piFrequencySel** 指针，返回选择的索引号。0:50HZ 1:60HZ

返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

- ◆ CameraGetMultiExposureCount()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetMultiExposureCount ( CameraHandle hCamera,  
                           int * count  
                           )
```

获取多重曝光使能个数。(此功能仅线阵相机支持)

**参数**

[in] **hCamera** 相机的句柄。  
[out] **count** 使能个数。

**返回**

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

- ◆ CameraGetMultiExposureMaxCount()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetMultiExposureMaxCount ( CameraHandle hCamera,  
                               int * max_count  
                               )
```

获取多重曝光的最大曝光个数。(此功能仅线阵相机支持)

**参数**

[in] **hCamera** 相机的句柄。  
[out] **max\_count** 支持的最大曝光个数。

**返回**

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetMultiExposureTime()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetMultiExposureTime( CameraHandle hCamera,  
                           int index,  
                           double * fExposureTime  
                         )
```

获取多重曝光时间。单位为微秒。(此功能仅线阵相机支持)

### 参数

[in] **hCamera** 相机的句柄。  
[in] **index** 曝光索引。  
[out] **fExposureTime** 返回曝光时间，单位微秒。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraIsAeWinVisible()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraIsAeWinVisible( CameraHandle hCamera,  
                      BOOL * pbIsVisible  
                    )
```

获得自动曝光参考窗口的显示状态。

### 参数

[in] **hCamera** 相机的句柄。  
[out] **pbIsVisible** 指针，返回TRUE，则表示当前窗口会被叠加在图像内容上。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetAeAnalogGainRange()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetAeAnalogGainRange ( CameraHandle hCamera,  
                           int          iMinAnalogGain,  
                           int          iMaxAnalogGain  
                           )
```

设定自动曝光模式的增益调节范围

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iMinAnalogGain** 最小增益  
[in] **iMaxAnalogGain** 最大增益

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetAeExposureRange()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetAeExposureRange ( CameraHandle hCamera,  
                           double       fMinExposureTime,  
                           double       fMaxExposureTime  
                           )
```

设定自动曝光模式的曝光时间调节范围

### 参数

[in] **hCamera** 相机的句柄。  
[in] **fMinExposureTime** 最小曝光时间 (微秒)  
[in] **fMaxExposureTime** 最大曝光时间 (微秒)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetAeState()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetAeState( CameraHandle hCamera,  
                  BOOL bAeState  
                )
```

设置相机曝光的模式。自动或者手动。

### 参数

[in] **hCamera** 相机的句柄。  
[in] **bAeState** TRUE:自动曝光； FALSE:手动曝光。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### 示例:

[grab.cpp](#), [grab\\_ex.cpp](#), [roi.cpp](#), [soft\\_trigger.cpp](#), [win\\_basic.cpp](#),  
[win\\_callback.cpp](#) , 以及 [win\\_grabber.cpp](#).

## ◆ CameraSetAeTarget()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetAeTarget( CameraHandle hCamera,  
                   int iAeTarget  
                 )
```

设定自动曝光的亮度目标值。设定范围[**tSdkExpose.uiTargetMin**,  
**tSdkExpose.uiTargetMax**]

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iAeTarget** 亮度目标值。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetAeThreshold()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetAeThreshold

( **CameraHandle** hCamera,  
int iThreshold  
)

设置自动曝光模式的调节阈值

### 参数

[in] **hCamera** 相机的句柄。

[in] **iThreshold** 如果  $\text{abs}(\text{目标亮度}-\text{图像亮度}) < \text{iThreshold}$  则停止自动调节

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetAeWindow()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetAeWindow

( **CameraHandle** hCamera,  
int iHOff, int iVOff,  
int iWidth, int iHeight  
)

设置自动曝光的参考窗口。

### 参数

[in] **hCamera** 相机的句柄。

[in] **iHOff** 窗口左上角的横坐标

[in] **iVOff** 窗口左上角的纵坐标

[in] **iWidth** 窗口的宽度

[in] **iHeight** 窗口的高度

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

如果iHOFF、iVOFF、iWidth、iHeight全部为0, 则窗口设置为每个分辨率下的居中1/2大小。可以随着分辨率的变化而跟随变化。

如果iHOFF、iVOFF、iWidth、iHeight所决定的窗口位置范围超出了当前分辨率范围内, 则自动使用居中1/2大小窗口。

### ◆ CameraSetAeWinVisible()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetAeWinVisible

( **CameraHandle** hCamera,  
  **BOOL**                blsVisible  
  )

设置自动曝光参考窗口的显示状态。

#### 参数

[in] **hCamera** 相机的句柄。

[in] **blsVisible** TRUE, 设置为显示; FALSE, 不显示。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

当设置窗口状态为显示, 调用[CameralImageOverlay](#)后, 能够将窗口位置以矩形的方式叠加在图像上。

### ◆ CameraSetAnalogGain()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetAnalogGain

( **CameraHandle** hCamera,  
  **INT**                iAnalogGain  
  )

设置相机的图像模拟增益值。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iAnalogGain** 设定的模拟增益值。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

该值乘以**tSdkExpose.fAnalogGainStep**, 就得到实际的图像信号放大倍数。

### ◆ CameraSetAntiFlick()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetAntiFlick ( **CameraHandle** hCamera,  
                          **BOOL**              bEnable  
                          )

设置自动曝光时抗频闪功能的使能状态。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **bEnable** TRUE, 开启抗频闪功能;FALSE, 关闭该功能。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

对于手动曝光模式下无效。

### ◆ CameraSetExposureTime()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetExposureTime ( **CameraHandle** hCamera,  
                          **double**            fExposureTime  
                          )

设置曝光时间。单位为微秒。

## 参数

[in] **hCamera** 相机的句柄。  
[in] **fExposureTime** 曝光时间，单位微秒。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## 注解

对于CMOS传感器，其曝光的单位是按照行来计算的，因此，曝光时间并不能在微秒级别连续可调。而是会按照整行来取舍。在调用本函数设定曝光时间后，建议再调用[CameraGetExposureTime](#)来获得实际设定的值。

## 示例：

[grab.cpp](#), [grab\\_ex.cpp](#), [roi.cpp](#), [soft\\_trigger.cpp](#), [win\\_basic.cpp](#),  
[win\\_callback.cpp](#)，以及 [win\\_grabber.cpp](#).

## ◆ CameraSetLightFrequency()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetLightFrequency( CameraHandle hCamera,  
                        int iFrequencySel  
                      )
```

设置自动曝光时消频闪的频率。

## 参数

[in] **hCamera** 相机的句柄。  
[in] **iFrequencySel** 0:50HZ , 1:60HZ

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetMultiExposureCount()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetMultiExposureCount( CameraHandle hCamera,  
                            int count  
                          )
```

设置多重曝光使能个数。(此功能仅线阵相机支持)

#### 参数

[in] **hCamera** 相机的句柄。

[in] **count** 使能个数。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraSetMultiExposureTime()

MVSDK\_API **CameraSdkStatus** \_\_stdcall

CameraSetMultiExposureTime

( **CameraHandle** hCamera,  
int index,  
double fExposureTime  
)

设置多重曝光时间。单位为微秒。(此功能仅线阵相机支持)

#### 参数

[in] **hCamera** 相机的句柄。

[in] **index** 曝光索引。

[in] **fExposureTime** 曝光时间, 单位微秒。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

对于CMOS传感器, 其曝光的单位是按照行来计算的, 因此, 曝光时间并不能在微秒级别连续可调。而是会按照整行来取舍。在调用本函数设定曝光时间后, 建议再调用[CameraGetMultiExposureTime](#)来获得实际设定的值。

# MVSDK API

## ROI

API

函数

### 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetImageResolution**  
(**CameraHandle** hCamera,  
**tSdkImageResolution**  
\*psCurVideoSize)  
获得当前预览的分辨率。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetImageResolutionEx**  
(**CameraHandle** hCamera, int  
\*iIndex, char acDescription[32],  
int \*Mode, UINT \*ModeSize, int  
\*x, int \*y, int \*width, int \*height,  
int \*ZoomWidth, int  
\*ZoomHeight)  
获得当前预览的分辨率。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetImageResolution**  
(**CameraHandle** hCamera,  
**tSdkImageResolution**  
\*pImageResolution)  
设置预览的分辨率。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetImageResolutionEx**  
(**CameraHandle** hCamera, int  
iIndex, int Mode, UINT  
ModeSize, int x, int y, int width,  
int height, int ZoomWidth, int  
ZoomHeight)  
获得当前预览的分辨率。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetResolutionForSnap**  
(**CameraHandle** hCamera,

**tSdkImageResolution**

**\*pImageResolution)**

获得抓拍模式下的分辨率选择索引号。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraSetResolutionForSnap**  
(**CameraHandle** hCamera,  
**tSdkImageResolution**  
**\*pImageResolution)**  
设置抓拍模式下相机输出图像的  
分辨率。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraCustomizeResolution**  
(**CameraHandle** hCamera,  
**tSdkImageResolution**  
**\*pImageCustom)**  
打开分辨率自定义面板，并通过  
可视化的方式来配置一个自定义  
分辨率。 [更多...](#)

## 详细描述

## 函数说明

### ◆ CameraCustomizeResolution()

```
MVSDK_API CameraSdkStatus  
__stdcall  
CameraCustomizeResolution ( CameraHandle          hCamera,  
                           tSdkImageResolution * plImageCustom  
                         )
```

打开分辨率自定义面板，并通过可视化的方式来配置一个自定义分辨率。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **plImageCustom** 指针，返回自定义的分辨率。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetImageResolution()

```
MVSDK_API  
CameraSdkStatus __stdcall  
CameraGetImageResolution ( CameraHandle          hCamera,  
                           tSdkImageResolution * psCurVideoSize  
                         )
```

获得当前预览的分辨率。

## 参数

- [in] **hCamera** 相机的句柄。
- [out] **psCurVideoSize** 返回当前的分辨率。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetImageResolutionEx()

MVSDK\_API **CameraSdkStatus**

\_\_stdcall

CameraGetImageResolutionEx

```
( CameraHandle hCamera,
    int *          iIndex,
    char          acDescription[32],
    int *          Mode,
    UINT *         ModeSize,
    int *          x,
    int *          y,
    int *          width,
    int *          height,
    int *          ZoomWidth,
    int *          ZoomHeight
)
```

获得当前预览的分辨率。

## 参数

- [in] **hCamera** 相机的句柄。
- [out] **iIndex** 索引号, [0,N]表示预设的分辨率(N 为预设分辨率的最大个数, 一般不超过20),0xFFFF 表示自定义分辨率(ROI)
- [out] **acDescription** 该分辨率的描述信息。仅预设分辨率时该信息有效。自定义分辨率可忽略该信息
- [out] **Mode** 0: 普通模式 1: Sum 2: Average 3: Skip 4: Resample
- [out] **ModeSize** 普通模式下忽略, 第1位表示2X2 第二位表示3X3 ...

[out] <b>x</b>	水平偏移
[out] <b>y</b>	垂直偏移
[out] <b>width</b>	宽
[out] <b>height</b>	高
[out] <b>ZoomWidth</b>	最终输出时缩放宽度, 0表示不缩放
[out] <b>ZoomHeight</b>	最终输出时缩放高度, 0表示不缩放

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetResolutionForSnap()

MVSDK\_API

```
CameraSdkStatus __stdcall  
CameraGetResolutionForSnap ( CameraHandle           hCamera,  
                           tSdkImageResolution * plImageResolution  
                         )
```

获得抓拍模式下的分辨率选择索引号。

#### 参数

[in] <b>hCamera</b>	相机的句柄。
[out] <b>plImageResolution</b>	指针, 返回抓拍模式的分辨率。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraSetImageResolution()

MVSDK\_API

```
CameraSdkStatus __stdcall  
CameraSetImageResolution ( CameraHandle           hCamera,  
                           tSdkImageResolution * plImageResolution  
                         )
```

设置预览的分辨率。

## 参数

- [in] **hCamera** 相机的句柄。
- [in] **pImageResolution** 新分辨率。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## 示例:

[roi.cpp.](#)

## ◆ CameraSetImageResolutionEx()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetImageResolutionEx( CameraHandle hCamera,  
                           int iIndex,  
                           int Mode,  
                           UINT ModeSize,  
                           int x,  
                           int y,  
                           int width,  
                           int height,  
                           int ZoomWidth,  
                           int ZoomHeight  
                           )
```

获得当前预览的分辨率。

## 参数

- [in] **hCamera** 相机的句柄。
- [in] **iIndex** 索引号, [0,N]表示预设的分辨率(N 为预设分辨率的最大个数, 一般不超过20),0xFFFF 表示自定义分辨率(ROI)
- [in] **Mode** 0: 普通模式 1: Sum 2: Average 3: Skip 4: Resample
- [in] **ModeSize** 普通模式下忽略, 第1位表示2X2 第二位表示3X3

...  
[in] **x** 水平偏移  
[in] **y** 垂直偏移  
[in] **width** 宽  
[in] **height** 高  
[in] **ZoomWidth** 最终输出时缩放宽度, 0表示不缩放  
[in] **ZoomHeight** 最终输出时缩放高度, 0表示不缩放

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetResolutionForSnap()

MVSDK\_API

```
CameraSdkStatus __stdcall  
CameraSetResolutionForSnap ( CameraHandle          hCamera,  
                           tSdkImageResolution * pImageResolution  
                         )
```

设置抓拍模式下相机输出图像的分辨率。

## 参数

[in] **hCamera** 相机的句柄。  
[in] **pImageResolution** 分辨率

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## 注解

如果pImageResolution->iWidth = pImageResolution->iHeight = 0, 则表示设定为跟随当前预览分辨率。抓拍到的图像的分辨率会和当前设定的预览分辨率一样。

# MVSDK API

## 触发功能

API

函数

### 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetTriggerDelayTime**  
(**CameraHandle** hCamera, **UINT**  
**uDelayTimeUs**)  
设置硬件触发模式下的触发延时时间，  
单位微秒。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetTriggerDelayTime**  
(**CameraHandle** hCamera, **UINT**  
**\*puDelayTimeUs**)  
获得当前设定的硬触发延时时间。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetTriggerCount**  
(**CameraHandle** hCamera, **INT** iCount)  
设置触发模式下的触发帧数。对软件触  
发和硬件触发模式都有效。默认为1  
帧，即一次触发信号采集一帧图像。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetTriggerCount**  
(**CameraHandle** hCamera, **INT**  
**\*piCount**)  
获得一次触发的帧数。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSoftTrigger** (**CameraHandle**  
hCamera)  
执行一次软触发。执行后，会触发由  
**CameraSetTriggerCount**指定的帧  
数。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetTriggerMode**  
(**CameraHandle** hCamera, **int**  
**iModeSel**)  
设置相机的触发模式。 [更多...](#)

MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetTriggerMode</b> ( <b>CameraHandle</b> hCamera, INT *piModeSel) 获得相机的触发模式。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetStrobeMode</b> ( <b>CameraHandle</b> hCamera, INT iMode) 设置IO引脚端子上的STROBE信号。该信号可以做闪光灯控制，也可以做外部机械快门控制。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetStrobeMode</b> ( <b>CameraHandle</b> hCamera, INT *piMode) 获取当前STROBE信号设置的模式。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetStrobeDelayTime</b> ( <b>CameraHandle</b> hCamera, UINT uDelayTimeUs) 当STROBE信号处于 STROBE_SYNC_WITH_TRIGGER时，通过该函数设置其相对触发信号延时时间。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetStrobeDelayTime</b> ( <b>CameraHandle</b> hCamera, UINT *upDelayTimeUs) 当STROBE信号处于 STROBE_SYNC_WITH_TRIGGER时，通过该函数获得其相对触发信号延时时间。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetStrobePulseWidth</b> ( <b>CameraHandle</b> hCamera, UINT uTimeUs) 当STROBE信号处于 STROBE_SYNC_WITH_TRIGGER时，通过该函数设置其脉冲宽度。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetStrobePulseWidth</b> ( <b>CameraHandle</b> hCamera, UINT *upTimeUs) 当STROBE信号处于 STROBE_SYNC_WITH_TRIGGER时，通

过该函数获得其脉冲宽度。 [更多...](#)

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameraSetStrobePolarity</b> ( <b>CameraHandle</b> hCamera, INT uPolarity) 当STROBE信号处于 STROBE_SYNC_WITH_TRIG时，通 过该函数设置其有效电平的极性。默认 为高有效，当触发信号到来时， STROBE信号被拉高。 <a href="#">更多...</a>
--	---

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameraGetStrobePolarity</b> ( <b>CameraHandle</b> hCamera, INT *upPolarity) 获得相机当前STROBE信号的有效极 性。默认为高电平有效。 <a href="#">更多...</a>
--	--

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameraSetExtTrigSignalType</b> ( <b>CameraHandle</b> hCamera, INT iType) 设置相机外触发信号的种类。上边沿、 下边沿、或者高、低电平方式。 <a href="#">更多...</a>
--	---

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameraGetExtTrigSignalType</b> ( <b>CameraHandle</b> hCamera, INT *ipType) 获得相机当前外触发信号的种类。 <a href="#">更多...</a>
--	---

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameraSetExtTrigShutterType</b> ( <b>CameraHandle</b> hCamera, INT iType) 设置外触发模式下，相机快门的方式， 默认为标准快门方式。部分滚动快门的 CMOS相机支持GRR方式。 <a href="#">更多...</a>
--	---

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameraGetExtTrigShutterType</b> ( <b>CameraHandle</b> hCamera, INT *ipType) 获得外触发模式下，相机快门的方式。 <a href="#">更多...</a>
--	--

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameraSetExtTrigDelayTime</b> ( <b>CameraHandle</b> hCamera, UINT uDelayTimeUs) 设置外触发信号延时时间， 默认为0， 单位为微秒。 <a href="#">更多...</a>
--	---

MVSDK_API <b>CameraSdkStatus</b> __stdcall	<b>CameraGetExtTrigDelayTime</b>
--	----------------------------------

**(CameraHandle hCamera, UINT \*upDelayTimeUs)**  
获得设置的外触发信号延时时间， 默认为0， 单位为微秒。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraSetExtTrigBufferedDelayTime** (**CameraHandle** hCamera, **UINT uDelayTimeUs**)  
设置外触发信号延时激活时间， 默认为0， 单位为微秒。当设置的值uDelayTimeUs不为0时， 相机接收到外触发信号后， 将延时uDelayTimeUs个微秒后再进行图像捕获。并且会把延时期间收到的触发信号缓存起来， 被缓存的信号也将延时uDelayTimeUs个微秒后生效（最大缓存个数128）。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetExtTrigBufferedDelayTime** (**CameraHandle** hCamera, **UINT \*puDelayTimeUs**)  
获得设置的外触发信号延时激活时间， 默认为0， 单位为微秒。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraSetExtTrigIntervalTime** (**CameraHandle** hCamera, **UINT uTimeUs**)  
设置外触发信号间隔时间， 默认为0， 单位为微秒。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetExtTrigIntervalTime** (**CameraHandle** hCamera, **UINT \*upTimeUs**)  
获得设置的外触发信号间隔时间， 默认为0， 单位为微秒。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraSetExtTrigJitterTime** (**CameraHandle** hCamera, **UINT uTimeUs**)  
设置相机外触发信号的消抖时间。默认为0， 单位为微秒。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetExtTrigJitterTime** (**CameraHandle** hCamera, **UINT \*upTimeUs**)  
获得设置的相机外触发消抖时间， 默认

为0.单位为微秒。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall [CameraGetExtTrigCapability](#)  
(**CameraHandle** hCamera, **UINT**  
**\*puCapabilityMask**)  
获得相机外触发的属性掩码 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall [CameraPauseLevelTrigger](#)  
(**CameraHandle** hCamera)  
当外触发信号为电平模式时，暂时停止  
触发相机，直到电平信号跳变后继续触  
发。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall [CameraSoftTriggerEx](#) (**CameraHandle**  
**hCamera**, **UINT** uFlags)  
执行软触发。 [更多...](#)

## 详细描述

### 函数说明

- ◆ CameraGetExtTrigBufferedDelayTime()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetExtTrigBufferedDelayTime ( CameraHandle hCamera,  
                                     UINT *          puDelayTimeUs  
                                   )
```

获得设置的外触发信号延时激活时间， 默认为0， 单位为微秒。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **puDelayTimeUs** 触发延时

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

- ◆ CameraGetExtTrigCapability()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetExtTrigCapability ( CameraHandle hCamera,  
                           UINT *          puCapabilityMask  
                         )
```

获得相机外触发的属性掩码

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **puCapabilityMask** 指针, 返回该相机外触发特性掩码, 掩码参考 [CameraDefine.h](#) 中EXT\_TRIG\_MASK\_ 开头的宏定义。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetExtTrigDelayTime()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetExtTrigDelayTime ( CameraHandle hCamera,  
                           UINT *          upDelayTimeUs  
                         )
```

获得设置的外触发信号延时时间, 默认为0, 单位为微秒。

## 参数

[in] **hCamera** 相机的句柄。  
[out] **upDelayTimeUs** 触发延时

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetExtTrigIntervalTime()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetExtTrigIntervalTime ( CameraHandle hCamera,  
                               UINT *          upTimeUs  
                             )
```

获得设置的外触发信号间隔时间, 默认为0, 单位为微秒。

## 参数

[in] **hCamera** 相机的句柄。  
[out] **upTimeUs** 触发间隔

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraGetExtTrigJitterTime()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetExtTrigJitterTime( CameraHandle hCamera,  
                           UINT * upTimeUs  
                         )
```

获得设置的相机外触发消抖时间， 默认为0.单位为微秒。

### 参数

[in] **hCamera** 相机的句柄。  
[out] **upTimeUs** 时间

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraGetExtTrigShutterType()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetExtTrigShutterType( CameraHandle hCamera,  
                            INT * ipType  
                          )
```

获得外触发模式下， 相机快门的方式

### 参数

[in] **hCamera** 相机的句柄。  
[out] **ipType** 指针, 返回当前设定的外触发快门方式。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### 参见

[CameraSetExtTrigShutterType](#)

## ◆ CameraGetExtTrigSignalType()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetExtTrigSignalType( CameraHandle hCamera,
INT * ipType )
```

获得相机当前外触发信号的种类。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **ipType** 指针, 返回外触发信号种类, 参考[emExtTrigSignal](#)

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetStrobeDelayTime()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetStrobeDelayTime( CameraHandle hCamera,
UINT * upDelayTimeUs )
```

当STROBE信号处于STROBE\_SYNC\_WITH\_TRIG时, 通过该函数获得其相对触发信号延时时间。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **upDelayTimeUs** 指针, 返回延时时间, 单位us。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetStrobeMode()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetStrobeMode( CameraHandle hCamera,
INT * piMode )
```

获取当前STROBE信号设置的模式。

#### 参数

[in] **hCamera** 相机的句柄。

[out] **piMode** 返回模式

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetStrobePolarity()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetStrobePolarity

( **CameraHandle** hCamera,  
INT \* upPolarity  
)

获得相机当前STROBE信号的有效极性。默认为高电平有效。

#### 参数

[in] **hCamera** 相机的句柄。

[in] **upPolarity** 指针, 返回STROBE信号当前的有效极性。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetStrobePulseWidth()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetStrobePulseWidth

( **CameraHandle** hCamera,  
UINT \* upTimeUs  
)

当STROBE信号处于STROBE\_SYNC\_WITH\_TRIG时, 通过该函数获得其脉冲宽度。

#### 参数

[in] **hCamera** 相机的句柄。

[out] **upTimeUs** 指针, 返回脉冲宽度。单位为us。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetTriggerCount()

MVSDK\_API **CameraSdkStatus** \_\_stdcall

CameraGetTriggerCount

( **CameraHandle** hCamera,  
    INT \*                       piCount  
)

获得一次触发的帧数。

#### 参数

[in] **hCamera** 相机的句柄。

[out] **piCount** 一次触发信号采集的帧数。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetTriggerDelayTime()

MVSDK\_API **CameraSdkStatus** \_\_stdcall

CameraGetTriggerDelayTime

( **CameraHandle** hCamera,  
    UINT \*                       puDelayTimeUs  
)

获得当前设定的硬触发延时时间。

#### 参数

[in] **hCamera** 相机的句柄。

[out] **puDelayTimeUs** 指针, 返回延时时间, 单位微秒。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

- ◆ CameraGetTriggerMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetTriggerMode( CameraHandle hCamera,  
                      INT * piModeSel  
                     )
```

获得相机的触发模式。

## 参数

[in] **hCamera** 相机的句柄。

[out] **piModeSel** 指针，返回当前选择的相机触发模式的索引号。

返回

成功返回 `CAMERA_STATUS_SUCCESS(0)`。否则返回非0值的错误码, 请参考 `CameraStatus.h` 中错误码的定义。

- ◆ CameraPauseLevelTrigger()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraPauseLevelTrigger ( CameraHandle hCamera )
```

当外触发信号为电平模式时，暂时停止触发相机，直到电平信号跳变后继续触发。

参数

[in] **hCamera** 相机的句柄。

返回

成功返回 `CAMERA_STATUS_SUCCESS(0)`。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

- ◆ CameraSetExtTrigBufferedDelayTime()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetExtTrigBufferedDelayTime ( CameraHandle hCamera,  
                                    UINT          uDelayTimeUs  
                                )
```

设置外触发信号延时激活时间， 默认为0， 单位为微秒。当设置的值uDelayTimeUs不为0时， 相机接收到外触发信号后， 将延时uDelayTimeUs个微秒后再进行图像捕获。并且会把延时期间收到的触发信号缓存起来， 被缓存的信号也将延时uDelayTimeUs个微秒后生效（最大缓存个数128）。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **uDelayTimeUs** 延时时间， 单位为微秒， 默认为0.

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraSetExtTrigDelayTime()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetExtTrigDelayTime ( CameraHandle hCamera,  
                           UINT          uDelayTimeUs  
                         )
```

设置外触发信号延时时间， 默认为0， 单位为微秒。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **uDelayTimeUs** 延时时间， 单位为微秒， 默认为0.

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraSetExtTrigIntervalTime()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetExtTrigIntervalTime ( CameraHandle hCamera,  
                               UINT          uTimeUs  
                             )
```

设置外触发信号间隔时间， 默认为0， 单位为微秒。

## 参数

[in] **hCamera** 相机的句柄。  
[in] **uTimeUs** 间隔时间，单位为微秒，默认为0。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetExtTrigJitterTime()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetExtTrigJitterTime ( **CameraHandle** hCamera,  
                                  **UINT**              uTimeUs  
                                  )

设置相机外触发信号的消抖时间。默认为0，单位为微秒。

## 参数

[in] **hCamera** 相机的句柄。  
[in] **uTimeUs** 时间

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetExtTrigShutterType()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetExtTrigShutterType ( **CameraHandle** hCamera,  
                                  **INT**               iType  
                                  )

设置外触发模式下，相机快门的方式，默认为标准快门方式。部分滚动快门的CMOS相机支持GRR方式。

## 参数

[in] **hCamera** 相机的句柄。  
[in] **iType** 外触发快门方式。参考 **emExtTrigShutterMode**

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

- ◆ CameraSetExtTrigSignalType()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetExtTrigSignalType ( CameraHandle hCamera,  
                           INT iType  
                         )
```

设置相机外触发信号的种类。上边沿、下边沿、或者高、低电平方式。

**参数**

[in] **hCamera** 相机的句柄。  
[in] **iType** 外触发信号种类, 参考**emExtTrigSignal**

**返回**

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

- ◆ CameraSetStrobeDelayTime()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetStrobeDelayTime ( CameraHandle hCamera,  
                           UINT uDelayTimeUs  
                         )
```

当STROBE信号处于STROBE\_SYNC\_WITH\_TRIG时, 通过该函数设置其相对触发信号延时时间。

**参数**

[in] **hCamera** 相机的句柄。  
[in] **uDelayTimeUs** 相对触发信号的延时时间, 单位为us。可以为0, 但不能为负数。

**返回**

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

- ◆ CameraSetStrobeMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetStrobeMode( CameraHandle hCamera,  
                      INT iMode  
                    )
```

设置IO引脚端子上的STROBE信号。该信号可以做闪光灯控制，也可以做外部机械快门控制。

**参数**

[in] **hCamera** 相机的句柄。  
[in] **iMode** 闪光灯模式，参考[emStrobeControl](#)

**返回**

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

- ◆ CameraSetStrobePolarity()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetStrobePolarity( CameraHandle hCamera,  
                         INT uPolarity  
                       )
```

当STROBE信号处于STROBE\_SYNC\_WITH\_TRIG时，通过该函数设置其有效电平的极性。默认为高有效，当触发信号到来时，STROBE信号被拉高。

**参数**

[in] **hCamera** 相机的句柄。  
[in] **uPolarity** STROBE信号的极性，0为低电平有效，1为高电平有效。默认为高电平有效。

**返回**

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

- ◆ CameraSetStrobePulseWidth()

```
MVSDK_API CameraSdkStatus __stdcall CameraSetStrobePulseWidth( CameraHandle hCamera,
                                                                    UINT          uTimeUs
                                                                  )
```

当STROBE信号处于STROBE\_SYNC\_WITH\_TRIG时，通过该函数设置其脉冲宽度。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **uTimeUs** 脉冲的宽度，单位为时间us。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraSetTriggerCount()

```
MVSDK_API CameraSdkStatus __stdcall CameraSetTriggerCount( CameraHandle hCamera,
                                                               INT          iCount
                                                             )
```

设置触发模式下的触发帧数。对软件触发和硬件触发模式都有效。默认为1帧，即一次触发信号采集一帧图像。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iCount** 一次触发采集的帧数。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

#### 示例：

[soft\\_trigger.cpp](#).

### ◆ CameraSetTriggerDelayTime()

```
MVSDK_API CameraSdkStatus __stdcall CameraSetTriggerDelayTime( CameraHandle hCamera,
```

```
    UINT          uDelayTimeUs  
)
```

设置硬件触发模式下的触发延时时间，单位微秒。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **uDelayTimeUs** 硬触发延时。单位微秒。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

#### 注解

当硬触发信号来临后，经过指定的延时，再开始采集图像。

### ◆ CameraSetTriggerMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetTriggerMode( CameraHandle hCamera,  
                      int           iModeSel  
)
```

设置相机的触发模式。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iModeSel** 模式选择索引号。0表示连续采集模式；1表示软件触发模式；2表示硬件触发模式。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

#### 示例:

**grab.cpp**, **grab\_ex.cpp**, **roi.cpp**, **soft\_trigger.cpp**, **win\_basic.cpp**,  
**win\_callback.cpp**，以及 **win\_grabber.cpp**.

### ◆ CameraSoftTrigger()

```
MVSDK_API CameraSdkStatus __stdcall          ( CameraHandle hCamera )
```

## CameraSoftTrigger

执行一次软触发。执行后，会触发由**CameraSetTriggerCount**指定的帧数。

### 参数

[in] **hCamera** 相机的句柄。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### 参见

[CameraSetTriggerMode](#)

### 示例:

[soft\\_trigger.cpp](#).

## ◆ CameraSoftTriggerEx()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSoftTriggerEx( CameraHandle hCamera,  
                      UINT uFlags  
                    )
```

执行软触发。

### 参数

[in] **hCamera** 相机的句柄。

[in] **uFlags** 功能标志,详见[emCameraSoftTriggerExFlags](#)中的定义

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### 参见

[CameraSoftTrigger](#)

# MVSDK API

## GPIO API

函数

### 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetIOState** (**CameraHandle** hCamera, INT iOutputIOIndex, UINT uState)  
设置指定IO的电平状态，IO为输出型IO，相机预留可编程输出IO的个数由  
**tSdkCameraCapbility.iOutputIoCounts**决定。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetIOStateEx** (**CameraHandle** hCamera, INT iOutputIOIndex, UINT uState)  
设置指定IO的电平状态，IO为输出型IO，相机预留可编程输出IO的个数由  
**tSdkCameraCapbility.iOutputIoCounts**决定。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetOutPutIOState** (**CameraHandle** hCamera, INT iOutputIOIndex, UINT \*puState)  
读取指定IO的电平状态，IO为输出型IO，相机预留可编程输出IO的个数由  
**tSdkCameraCapbility.iOutputIoCounts**决定。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetOutPutIOStateEx** (**CameraHandle** hCamera, INT iOutputIOIndex, UINT \*puState)  
读取指定IO的电平状态，IO为输出型IO，相机预留可编程输出IO的个数由  
**tSdkCameraCapbility.iOutputIoCounts**决定。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetIOState** (**CameraHandle** hCamera, INT iInputIOIndex, UINT \*puState)  
读取指定IO的电平状态，IO为输入型IO，相机预留可编程输出IO的个数由  
**tSdkCameraCapbility.iInputIoCounts**决定。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetIOStateEx** (**CameraHandle** hCamera, INT iInputIOIndex, UINT \*puState)

读取指定IO的电平状态，IO为输入型IO，相机预留可编程输出IO的个数由  
**tSdkCameraCapbility.iInputIoCounts**决定。更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetInPutIOMode** (**CameraHandle** hCamera, INT iInputOIndex, INT iMode)  
设置输入IO的模式 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetInPutIOMode** (**CameraHandle** hCamera, INT iInputOIndex, INT \*piMode)  
获取输入IO的模式 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetOutPutIOMode** (**CameraHandle** hCamera, INT iOutputOIndex, INT iMode)  
设置输出IO的模式 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetOutPutIOMode** (**CameraHandle** hCamera, INT iOutputOIndex, INT \*piMode)  
获取输出IO的模式 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetInPutIOModeCapbility** (**CameraHandle** hCamera, INT iInputOIndex, UINT \*piCapbility)  
获取输入IO的模式支持能力 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetOutPutIOModeCapbility** (**CameraHandle** hCamera, INT iOutputOIndex, UINT \*piCapbility)  
获取输出IO的模式支持能力 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetOutPutPWM** (**CameraHandle** hCamera, INT iOutputOIndex, UINT iCycle, UINT uDuty)  
设置PWM型输出的参数 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetRotaryEncDir** (**CameraHandle** hCamera, INT dir)  
设置编码器有效方向 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetRotaryEncDir** (**CameraHandle** hCamera, INT \*dir)  
获取编码器有效方向 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetRotaryEncFreq** (**CameraHandle** hCamera, INT mul, INT div)  
设置编码器频率 更多...

MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetRotaryEncFreq</b> ( <b>CameraHandle</b> hCamera, INT *mul, INT *div) 获取编码器频率 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetInPutIOFormat</b> ( <b>CameraHandle</b> hCamera, INT iInputOIndex, INT iFormat) 设置输入IO的格式 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetInPutIOFormat</b> ( <b>CameraHandle</b> hCamera, INT iInputOIndex, INT *piFormat) 获取输入IO的格式 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetOutPutIOFormat</b> ( <b>CameraHandle</b> hCamera, INT iOutputOIndex, INT iFormat) 设置输出IO的格式 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetOutPutIOFormat</b> ( <b>CameraHandle</b> hCamera, INT iOutputOIndex, INT *piFormat) 获取输出IO的格式 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetInPutIOFormatCapbility</b> ( <b>CameraHandle</b> hCamera, INT iInputOIndex, UINT *piCapbility) 获取输入IO的格式支持能力 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetOutPutIOFormatCapbility</b> ( <b>CameraHandle</b> hCamera, INT iOutputOIndex, UINT *piCapbility) 获取输出IO的格式支持能力 <a href="#">更多...</a>

## 详细描述

### 函数说明

#### ◆ CameraGetInPutIOFormat()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetInPutIOFormat( CameraHandle hCamera,  
                        INT iInputIOIndex,  
                        INT * piFormat  
)
```

获取输入IO的格式

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iInputIOIndex** IO的索引号, 从0开始。  
[out] **piFormat** IO格式, 参考[emCameraGPIOFormat](#)

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### ◆ CameraGetInPutIOFormatCapbility()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetInPutIOFormatCapbility( CameraHandle hCamera,  
                                 INT iInputIOIndex,  
                                 UINT * piCapbility  
)
```

获取输入IO的格式支持能力

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iInputIOIndex** IO的索引号, 从0开始。  
[out] **piCapbility** IO格式支持位掩码

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetInPutIOMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetInPutIOMode  
    ( CameraHandle hCamera,  
      INT          iInputIOIndex,  
      INT *        piMode  
    )
```

获取输入IO的模式

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iInputIOIndex** IO的索引号, 从0开始。  
[out] **piMode** IO模式, 参考[emCameraGPIOMode](#)

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetInPutIOModeCapbility()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetInPutIOModeCapbility  
    ( CameraHandle hCamera,  
      INT          iInputIOIndex,  
      UINT *       piCapbility  
    )
```

获取输入IO的模式支持能力

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iInputIOIndex** IO的索引号, 从0开始。  
[out] **piCapbility** IO模式支持位掩码

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetIOState()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetIOState( ( CameraHandle hCamera,  
                  INT iInputOIndex,  
                  UINT * puState  
                )
```

读取指定IO的电平状态，IO为输入型IO，相机预留可编程输出IO的个数由 **tSdkCameraCapbility.iInputIoCounts** 决定。

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iInputOIndex** IO的索引号，从0开始。  
[out] **puState** 指针，返回IO状态(GE、SUA: 0(高) 1(低))

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### 注解

已废弃，使用CameraGetIOStateEx，它对所有型号相机的输入状态值统一为1高 0 低

## ◆ CameraGetIOStateEx()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetIOStateEx( ( CameraHandle hCamera,  
                     INT iInputOIndex,  
                     UINT * puState  
                   )
```

读取指定IO的电平状态，IO为输入型IO，相机预留可编程输出IO的个数由 **tSdkCameraCapbility.iInputIoCounts** 决定。

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iInputOIndex** IO的索引号，从0开始。  
[out] **puState** 指针，返回IO状态,1为高，0为低

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraGetOutPutIOFormat()

MVSDK\_API CameraSdkStatus \_\_stdcall  
CameraGetOutPutIOFormat

( CameraHandle hCamera,  
INT iOutputOIndex,  
INT \* piFormat  
)

获取输出IO的格式

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iOutputOIndex** IO的索引号, 从0开始。  
[out] **piFormat** IO格式, 参考[emCameraGPIOFormat](#)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考  
[CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetOutPutIOFormatCapbility()

MVSDK\_API CameraSdkStatus \_\_stdcall  
CameraGetOutPutIOFormatCapbility

( CameraHandle hCamera,  
INT iOutputOIndex,  
UINT \* piCapbility  
)

获取输出IO的格式支持能力

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iOutputOIndex** IO的索引号, 从0开始。  
[out] **piCapbility** IO格式支持位掩码

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考  
[CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetOutPutIOMode()

MVSDK\_API CameraSdkStatus \_\_stdcall

( CameraHandle hCamera,

## CameraGetOutPutIOMode

```
    INT          iOutputIOIndex,  
    INT *        piMode  
)
```

获取输出IO的模式

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iOutputIOIndex** IO的索引号, 从0开始。  
[out] **piMode** IO模式, 参考[emCameraGPIOMode](#)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考[CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetOutPutIOModeCapbility()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetOutPutIOModeCapbility

```
( CameraHandle hCamera,  
  INT          iOutputIOIndex,  
  UINT *        piCapbility  
)
```

获取输出IO的模式支持能力

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iOutputIOIndex** IO的索引号, 从0开始。  
[out] **piCapbility** IO模式支持位掩码

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考[CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetOutPutIOState()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetOutPutIOState

```
( CameraHandle hCamera,  
  INT          iOutputIOIndex,  
  UINT *        puState  
)
```

读取指定IO的电平状态，IO为输出型IO，相机预留可编程输出IO的个数由**tSdkCameraCapbility.iOutputIoCounts**决定。

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iOutputIoIndex** IO的索引号，从0开始。
- [out] **puState** 返回IO状态(GE、SUA: 0(高) 1(低))

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### 注解

已废弃, 使用**CameraGetOutPutIOStateEx**, 它对所有型号相机的输出状态值统一为1高 0低

### ◆ CameraGetOutPutIOStateEx()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetOutPutIOStateEx

( **CameraHandle** hCamera,  
INT iOutputIoIndex,  
UINT \* puState  
)

读取指定IO的电平状态，IO为输出型IO，相机预留可编程输出IO的个数由**tSdkCameraCapbility.iOutputIoCounts**决定。

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iOutputIoIndex** IO的索引号，从0开始。
- [out] **puState** 返回IO状态, 1为高, 0为低

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetRotaryEncDir()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetRotaryEncDir

( **CameraHandle** hCamera,  
INT \* dir  
)

获取编码器有效方向

#### 参数

- [in] **hCamera** 相机的句柄。  
[out] **dir** 有效方向 (0:正反转都有效 1: 顺时针 (A相超前于B) 2:逆时针)

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetRotaryEncFreq()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetRotaryEncFreq

( **CameraHandle** hCamera,  
    INT \*                        mul,  
    INT \*                        div  
)

获取编码器频率

#### 参数

- [in] **hCamera** 相机的句柄。  
[out] **mul**      倍频  
[out] **div**      分频

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraSetInPutIOFormat()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetInPutIOFormat

( **CameraHandle** hCamera,  
    INT                          iInputIOIndex,  
    INT                          iFormat  
)

设置输入IO的格式

#### 参数

- [in] **hCamera**      相机的句柄。

[in] **iInputIOIndex** IO的索引号, 从0开始。  
[in] **iFormat** IO格式, 参考[emCameraGPIOFormat](#)

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraSetInPutIOMode()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetInPutIOMode ( **CameraHandle** hCamera,  
                          **INT**                  iInputIOIndex,  
                          **INT**                  iMode  
                          )

设置输入IO的模式

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iInputIOIndex** IO的索引号, 从0开始。  
[in] **iMode** IO模式, 参考[emCameraGPIOMode](#)

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraSetIOState()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetIOState ( **CameraHandle** hCamera,  
                          **INT**                  iOutputIOIndex,  
                          **UINT**                uState  
                          )

设置指定IO的电平状态, IO为输出型IO, 相机预留可编程输出IO的个数由 [tSdkCameraCapbility.iOutputIoCounts](#) 决定。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iOutputIOIndex** IO的索引号, 从0开始。  
[in] **uState** 要设定的状态(GE、SUA: 0(高) 1(低))

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### 注解

已废弃, 使用**CameraSetIOStateEx**, 它对所有型号相机的输出状态值统一为1高 0低

#### ◆ CameraSetIOStateEx()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetIOStateEx  
    ( CameraHandle hCamera,  
      INT          iOutputIOIndex,  
      UINT         uState  
    )
```

设置指定IO的电平状态, IO为输出型IO, 相机预留可编程输出IO的个数由 **tSdkCameraCapability.iOutputIOCounts** 决定。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iOutputIOIndex** IO的索引号, 从0开始。  
[in] **uState** 要设定的状态 (1为高, 0为低)

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### ◆ CameraSetOutPutIOFormat()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetOutPutIOFormat  
    ( CameraHandle hCamera,  
      INT          iOutputIOIndex,  
      INT          iFormat  
    )
```

设置输出IO的格式

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iOutputIOIndex** IO的索引号, 从0开始。  
[in] **iFormat** IO格式, 参考 **emCameraGPIOFormat**

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### ◆ CameraSetOutPutIOMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetOutPutIOMode( CameraHandle hCamera,  
                      INT       iOutputIOIndex,  
                      INT       iMode  
                      )
```

设置输出IO的模式

##### 参数

[in] **hCamera** 相机的句柄。  
[in] **iOutputIOIndex** IO的索引号, 从0开始。  
[in] **iMode** IO模式, 参考[emCameraGPIOMode](#)

##### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### ◆ CameraSetOutPutPWM()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetOutPutPWM( CameraHandle hCamera,  
                     INT       iOutputIOIndex,  
                     UINT      iCycle,  
                     UINT      uDuty  
                     )
```

设置PWM型输出的参数

##### 参数

[in] **hCamera** 相机的句柄。  
[in] **iOutputIOIndex** IO的索引号, 从0开始。  
[in] **iCycle** PWM的周期, 单位(us)  
[in] **uDuty** 占用比, 取值1%~99%

##### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetRotaryEncDir()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetRotaryEncDir

( **CameraHandle** hCamera,  
INT dir  
)

设置编码器有效方向

### 参数

[in] **hCamera** 相机的句柄。  
[in] **dir** 有效方向 (0:正反转都有效 1: 顺时针 (A相超前于B) 2:逆时针)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考  
**CameraStatus.h** 中错误码的定义。

## ◆ CameraSetRotaryEncFreq()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetRotaryEncFreq

( **CameraHandle** hCamera,  
INT mul,  
INT div  
)

设置编码器频率

### 参数

[in] **hCamera** 相机的句柄。  
[in] **mul** 倍频  
[in] **div** 分频

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考  
**CameraStatus.h** 中错误码的定义。

# MVSDK API

## 高级设置

API

类型定义 | 函数

### 类型定义

```
typedef void(WINAPI * CAMERA_FRAME_EVENT_CALLBACK) (CameraHandle  
hCamera, tSdkFrameEvent *pEvent, PVOID pContext)
```

### 函数

<b>MVSDK_API CameraSdkStatus __stdcall</b>	<b>CameraGetMediaType</b> ( <b>CameraHandle</b> hCamera, INT *piMediaType) 获得相机当前输出原始数据的格式索引 号。 <a href="#">更多...</a>
--	--

<b>MVSDK_API CameraSdkStatus __stdcall</b>	<b>CameraSetMediaType</b> ( <b>CameraHandle</b> hCamera, INT iMediaType) 设置相机的输出原始数据格式。 <a href="#">更多...</a>
--	---

<b>MVSDK_API CameraSdkStatus __stdcall</b>	<b>CameraGetRawMaxAvailBits</b> ( <b>CameraHandle</b> hCamera, int *pMaxAvailBits) 获取RAW数据的最大有效位数 <a href="#">更多...</a>
--	--

<b>MVSDK_API CameraSdkStatus __stdcall</b>	<b>CameraSetRawStartBit</b> ( <b>CameraHandle</b> hCamera, int startBit) 设置RAW数据的输出起始位 <a href="#">更多...</a>
--	--

<b>MVSDK_API CameraSdkStatus __stdcall</b>	<b>CameraGetRawStartBit</b> ( <b>CameraHandle</b> hCamera, int *startBit) 获取RAW数据的输出起始位 <a href="#">更多...</a>
--	---

<b>MVSDK_API CameraSdkStatus __stdcall</b>	<b>CameraSetFrameSpeed</b> ( <b>CameraHandle</b> hCamera, int iFrameSpeed) 设定相机输出图像的帧率。 <a href="#">更多...</a>
--	---

<b>MVSDK_API CameraSdkStatus __stdcall</b>	<b>CameraGetFrameSpeed</b> ( <b>CameraHandle</b> hCamera, int *piFrameSpeed) 获得相机输出图像的帧率选择索引号。 <a href="#">更多...</a>
--	--

MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetFrameRate (CameraHandle hCamera, int RateHZ)</b> 设定相机的帧频(面阵)或行频(线阵)。 (仅部分网口相机支持) <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetFrameRate (CameraHandle hCamera, int *RateHZ)</b> 获取设定的相机帧频(面阵)或行频(线阵) <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetTransPackLen (CameraHandle hCamera, INT iPackSel)</b> 设置相机传输图像数据的分包大小。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetTransPackLen (CameraHandle hCamera, INT *piPackSel)</b> 获得相机当前传输分包大小的选择索引号。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraRstTimeStamp (CameraHandle hCamera)</b> 复位图像采集的时间戳，从0开始。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetLedEnable (CameraHandle hCamera, int index, BOOL enable)</b> 设置相机的LED使能状态，不带LED的型号，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetLedEnable (CameraHandle hCamera, int index, BOOL *enable)</b> 获得相机的LED使能状态，不带LED的型号，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetLedOnOff (CameraHandle hCamera, int index, BOOL onoff)</b> 设置相机的LED开关状态，不带LED的型号，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetLedOnOff (CameraHandle hCamera, int index, BOOL *onoff)</b> 获得相机的LED开关状态，不带LED的型号，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>

MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetLedDuration (CameraHandle hCamera, int index, UINT duration)</b> 设置相机的LED持续时间，不带LED的型号，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>
-------------------------------------	---

MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetLedDuration (CameraHandle hCamera, int index, UINT *duration)</b> 获得相机的LED持续时间，不带LED的型号，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>
-------------------------------------	--

MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetLedBrightness (CameraHandle hCamera, int index, UINT uBrightness)</b> 设置相机的LED亮度，不带LED的型号，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>
-------------------------------------	---

MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetLedBrightness (CameraHandle hCamera, int index, UINT *uBrightness)</b> 获得相机的LED亮度，不带LED的型号，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>
-------------------------------------	--

MVSDK_API CameraSdkStatus __stdcall	<b>CameraEnableTransferRoi (CameraHandle hCamera, UINT uEnableMask)</b> 使能或者禁止相机的多区域传输功能，不带该功能的型号，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>
-------------------------------------	---

MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetTransferRoi (CameraHandle hCamera, int index, UINT X1, UINT Y1, UINT X2, UINT Y2)</b> 设置相机传输的裁剪区域。在相机端，图像从传感器上被采集后，将会被裁剪成指定的区域来传送，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>
-------------------------------------	---

MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetTransferRoi (CameraHandle hCamera, int index, UINT *pX1, UINT *pY1, UINT *pX2, UINT *pY2)</b> 获取相机传输的裁剪区域。在相机端，图像从传感器上被采集后，将会被裁剪成指定的区域来传送，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>
-------------------------------------	---

MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetSingleGrabMode</b> (CameraHandle hCamera, BOOL bEnable) 启用或禁用单帧抓取模式，默認為禁用。 (本功能仅USB2.0相机支持) <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetSingleGrabMode</b> (CameraHandle hCamera, BOOL *pbEnable) 获得相机的单帧抓取使能状态 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraRestartGrab</b> (CameraHandle hCamera) 当相机处于单帧抓取模式时，每当成功抓取到一帧后SDK会进入暂停状态，调用此函数可使SDK退出暂停状态并开始抓取下一帧 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraEnableFastResponse</b> (CameraHandle hCamera) 使能快速响应 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetHDR</b> (CameraHandle hCamera, float value) 设置相机的HDR，需要相机支持，不带HDR功能的型号，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetHDR</b> (CameraHandle hCamera, float *value) 获取相机的HDR，需要相机支持，不带HDR功能的型号，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetFrameID</b> (CameraHandle hCamera, UINT *id) 获取当前帧的ID，需相机支持(网口全系列支持)，此函数返回错误代码，表示不支持。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetFrameTimeStamp</b> (CameraHandle hCamera, UINT *TimeStampL, UINT *TimeStampH) 获取当前帧的时间戳(单位微秒) <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetHDRGainMode</b> (CameraHandle hCamera, int value)

设置相机的增益模式，需要相机支持，不带增益模式切换功能的型号，此函数返回错误代码，表示不支持。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraGetHDRGainMode](#)  
(**CameraHandle** hCamera, int \*value)  
获取相机的增益模式，需要相机支持，不带增益模式切换功能的型号，此函数返回错误代码，表示不支持。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraSetLightingControllerMode](#)  
(**CameraHandle** hCamera, int index, int mode)  
设置光源控制器的输出模式（智能相机系列且需要硬件支持） [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraSetLightingControllerState](#)  
(**CameraHandle** hCamera, int index, int state)  
设置光源控制器的输出状态（智能相机系列且需要硬件支持） [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraSetFrameResendCount](#)  
(**CameraHandle** hCamera, int count)  
当相机处于触发模式（软触发或硬触发）时，相机发送一帧到PC，如相机未收到PC端的接收确认，相机可以把帧重发几次。  
用本函数设置相机重发次数。（仅网口相机支持） [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraGetMediaCapability](#)  
(**CameraHandle** hCamera, int iMediaType, UINT \*uCap)  
获取输出格式的特性支持。（比如：H264、H265支持设置码率） [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraSetMediaBitRate](#) (**CameraHandle** hCamera, int iMediaType, UINT uRate)  
设置码率。（仅部分输出格式支持，比如：H264、H265） [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraGetMediaBitRate](#) (**CameraHandle** hCamera, int iMediaType, UINT \*uRate)  
获取码率设置。（仅部分输出格式支持，比如：H264、H265） [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraSetFrameEventCallback](#)

(**CameraHandle** hCamera,  
**CAMERA\_FRAME\_EVENT\_CALLBACK**  
pCallBack, PVOID pContext)

设置相机帧事件回调函数。当帧开始以及  
帧完成时，pCallBack所指向的回调函数就  
会被调用。[更多...](#)

---

## 详细描述

---

### 类型定义说明

---

#### ◆ CAMERA\_FRAME\_EVENT\_CALLBACK

```
typedef void(WINAPI* CAMERA_FRAME_EVENT_CALLBACK) (CameraHandle  
hCamera, tSdkFrameEvent *pEvent, PVOID pContext)
```

帧事件回调函数定义

## 函数说明

### ◆ CameraEnableFastResponse()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraEnableFastResponse ( **CameraHandle** hCamera )

使能快速响应

#### 参数

[in] **hCamera** 相机的句柄。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraEnableTransferRoi()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraEnableTransferRoi ( **CameraHandle** hCamera,  
                          **UINT**              uEnableMask  
                          )

使能或者禁止相机的多区域传输功能, 不带该功能的型号, 此函数返回错误代码, 表示不支持。

#### 参数

[in] **hCamera** 相机的句柄。

[in] **uEnableMask** 区域使能状态掩码, 对应的比特位为1表示使能。0为禁止。  
目前SDK支持4个可编辑区域, index范围为0到3, 即bit0, bit1, bit2, bit3控制4个区域的使能状态。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### 注解

该功能主要用于在相机端将采集的整幅画面切分, 只传输指定的多个区域, 以提高传输帧率。多个区域传输到PC上后, 会自动拼接成整幅画面, 没有被传输的部分, 会用黑色填充。

## ◆ CameraGetFrameID()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetFrameID( CameraHandle hCamera,  
                  UINT * id  
                )
```

获取当前帧的ID，需相机支持(网口全系列支持)，此函数返回错误代码，表示不支持。

### 参数

[in] **hCamera** 相机的句柄。  
[out] **id** 帧ID

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetFrameRate()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetFrameRate( CameraHandle hCamera,  
                     int * RateHZ  
                   )
```

获取设定的相机帧频(面阵)或行频(线阵)

### 参数

[in] **hCamera** 相机的句柄。  
[out] **RateHZ** 帧频或行频 (<=0表示最大频率)。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetFrameSpeed()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetFrameSpeed( CameraHandle hCamera,  
                      int * piFrameSpeed  
                    )
```

获得相机输出图像的帧率选择索引号。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **piFrameSpeed** 返回选择的帧率模式索引号。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### 参见

[CameraSetFrameSpeed](#)

### ◆ CameraGetFrameTimeStamp()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetFrameTimeStamp ( **CameraHandle** hCamera,  
                          UINT \*              TimeStampL,  
                          UINT \*              TimeStampH  
                          )

获取当前帧的时间戳(单位微秒)

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **TimeStampL** 时间戳低32位  
[out] **TimeStampH** 时间戳高32位

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetHDR()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetHDR ( **CameraHandle** hCamera,  
                         float \*              value  
                          )

获取相机的HDR, 需要相机支持, 不带HDR功能的型号, 此函数返回错误代码, 表示不支持。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **value** HDR系数，范围0.0到1.0

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetHDRGainMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetHDRGainMode  
    ( CameraHandle hCamera,  
      int *           value  
    )
```

获取相机的增益模式，需要相机支持，不带增益模式切换功能的型号，此函数返回错误代码，表示不支持。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **value** 0: 低增益 1: 高增益

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetLedBrightness()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetLedBrightness  
    ( CameraHandle hCamera,  
      int          index,  
      UINT *       uBrightness  
    )
```

获得相机的LED亮度，不带LED的型号，此函数返回错误代码，表示不支持。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **index** LED灯的索引号，从0开始。如果只有一个可控制亮度的 LED，则该参数为0。  
[out] **uBrightness** 指针，返回LED亮度值，范围0到255. 0表示关闭，255最亮。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetLedDuration()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetLedDuration

( **CameraHandle** hCamera,  
int index,  
UINT \* duration  
)

获得相机的LED持续时间, 不带LED的型号, 此函数返回错误代码, 表示不支持。

## 参数

- [in] **hCamera** 相机的句柄。
- [in] **index** LED灯的索引号, 从0开始。如果只有一个可控制亮度的LED, 则该参数为0。
- [out] **duration** 返回LED持续时间, 单位毫秒

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetLedEnable()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetLedEnable

( **CameraHandle** hCamera,  
int index,  
BOOL \* enable  
)

获得相机的LED使能状态, 不带LED的型号, 此函数返回错误代码, 表示不支持。

## 参数

- [in] **hCamera** 相机的句柄。
- [in] **index** LED灯的索引号, 从0开始。如果只有一个可控制亮度的LED, 则该参数为0。
- [out] **enable** 指针, 返回LED使能状态

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetLedOnOff()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetLedOnOff  
    ( CameraHandle hCamera,  
      int index,  
      BOOL * onoff  
    )
```

获得相机的LED开关状态, 不带LED的型号, 此函数返回错误代码, 表示不支持。

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **index** LED灯的索引号, 从0开始。如果只有一个可控制亮度的LED, 则该参数为0。
- [out] **onoff** 返回LED开关状态

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetMediaBitRate()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetMediaBitRate  
    ( CameraHandle hCamera,  
      int iMediaType,  
      UINT * uRate  
    )
```

获取码率设置。 (仅部分输出格式支持, 比如: H264、H265)

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iMediaType** 输出格式索引
- [out] **uRate** 码率

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetMediaCapability()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetMediaCapability( CameraHandle hCamera,  
                           int iMediaType,  
                           UINT * uCap  
                         )
```

获取输出格式的特性支持。 (比如：H264、H265支持设置码率)

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iMediaType** 输出格式索引  
[out] **uCap** 特性支持

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraGetMediaType()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetMediaType( CameraHandle hCamera,  
                     INT * piMediaType  
                   )
```

获得相机当前输出原始数据的格式索引号。

### 参数

[in] **hCamera** 相机的句柄。  
[out] **piMediaType** 返回当前格式类型的索引号。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### 注解

在**tSdkCameraCapbility.pMediaTypeDesc**成员中, 以数组的形式保存了相机支持的格式, **piMediaType**所指向的索引号, 就是该数组的索引号。

## ◆ CameraGetRawMaxAvailBits()

MVSDK\_API CameraSdkStatus \_\_stdcall  
CameraGetRawMaxAvailBits

( CameraHandle hCamera,  
int \* pMaxAvailBits  
)

获取RAW数据的最大有效位数

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **pMaxAvailBits** 返回RAW的最大有效位数

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考  
**CameraStatus.h** 中错误码的定义。

### ◆ CameraGetRawStartBit()

MVSDK\_API CameraSdkStatus \_\_stdcall  
CameraGetRawStartBit

( CameraHandle hCamera,  
int \* startBit  
)

获取RAW数据的输出起始位

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **startBit** 起始BIT

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考  
**CameraStatus.h** 中错误码的定义。

### ◆ CameraGetSingleGrabMode()

MVSDK\_API CameraSdkStatus \_\_stdcall  
CameraGetSingleGrabMode

( CameraHandle hCamera,  
BOOL \* pbEnable  
)

获得相机的单帧抓取使能状态

## 参数

- [in] **hCamera** 相机的句柄。
- [out] **pbEnable** 返回相机的单帧抓取模式使能状态

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetTransferRoi()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetTransferRoi

```
( CameraHandle hCamera,  
    int index,  
    UINT * pX1,  
    UINT * pY1,  
    UINT * pX2,  
    UINT * pY2  
)
```

获取相机传输的裁剪区域。在相机端, 图像从传感器上被采集后, 将会被裁剪成指定的区域来传送, 此函数返回错误代码, 表示不支持。

## 参数

- [in] **hCamera** 相机的句柄。
- [in] **index** ROI区域的索引号, 从0开始。
- [out] **pX1** ROI区域的左上角X坐标
- [out] **pY1** ROI区域的左上角Y坐标
- [out] **pX2** ROI区域的右下角X坐标
- [out] **pY2** ROI区域的右下角Y坐标

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetTransPackLen()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetTransPackLen

```
( CameraHandle hCamera,  
    INT * piPackSel  
)
```

获得相机当前传输分包大小的选择索引号。

#### 参数

- [in] **hCamera** 相机的句柄。
- [out] **piPackSel** 指针, 返回当前选择的分包大小索引号。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### 参见

[CameraSetTransPackLen](#)

### ◆ CameraRestartGrab()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraRestartGrab ( **CameraHandle** hCamera )

当相机处于单帧抓取模式时, 每当成功抓取到一帧后SDK会进入暂停状态, 调用此函数可使SDK退出暂停状态并开始抓取下一帧

#### 参数

- [in] **hCamera** 相机的句柄。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraRstTimeStamp()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraRstTimeStamp ( **CameraHandle** hCamera )

复位图像采集的时间戳, 从0开始。

#### 参数

- [in] **hCamera** 相机的句柄。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetFrameEventCallback()

MVSDK\_API

```
CameraSdkStatus __stdcall  
CameraSetFrameEventCallback ( CameraHandle hCamera,  
CAMERA_FRAME_EVENT_CALLBACK pCallBack,  
PVOID pContext  
)
```

设置相机帧事件回调函数。当帧开始以及帧完成时，pCallBack所指向的回调函数就会被调用。

### 参数

[in] **hCamera** 相机的句柄。

[in] **pCallBack** 回调函数指针。

[in] **pContext** 回调函数的附加参数，在回调函数被调用时该附加参数会被传入，可以为NULL。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### 注解

对于全局快门相机帧开始表示一帧曝光结束

## ◆ CameraSetFrameRate()

MVSDK\_API CameraSdkStatus \_\_stdcall

```
CameraSetFrameRate ( CameraHandle hCamera,  
int RateHZ  
)
```

设定相机的帧频(面阵)或行频(线阵)。（仅部分网口相机支持）

### 参数

[in] **hCamera** 相机的句柄。

[in] **RateHZ** 帧频或行频（<=0表示最大频率）。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetFrameResendCount()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetFrameResendCount( CameraHandle hCamera,  
                           int count  
                         )
```

当相机处于触发模式（软触发或硬触发）时，相机发送一帧到PC，如相机未收到PC端的接收确认，相机可以把帧重发几次。用本函数设置相机重发次数。（仅网口相机支持）

### 参数

[in] **hCamera** 相机的句柄。  
[in] **count** 重发次数 (<=0表示禁用重发功能)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetFrameSpeed()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetFrameSpeed( CameraHandle hCamera,  
                      int iFrameSpeed  
                    )
```

设定相机输出图像的帧率。

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iFrameSpeed** 选择的帧率模式索引号，范围从0到  
tSdkCameraCapbility.iFrameSpeedDesc - 1

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetHDR()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetHDR( CameraHandle hCamera,
```

```
    float      value  
    )
```

设置相机的HDR，需要相机支持，不带HDR功能的型号，此函数返回错误代码，表示不支持。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **value** HDR系数，范围0.0到1.0

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraSetHDRGainMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetHDRGainMode  
    ( CameraHandle hCamera,  
      int           value  
    )
```

设置相机的增益模式，需要相机支持，不带增益模式切换功能的型号，此函数返回错误代码，表示不支持。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **value** 0：低增益 1：高增益

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraSetLedBrightness()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetLedBrightness  
    ( CameraHandle hCamera,  
      int           index,  
      UINT          uBrightness  
    )
```

设置相机的LED亮度，不带LED的型号，此函数返回错误代码，表示不支持。

## 参数

- [in] **hCamera** 相机的句柄。
- [in] **index** LED灯的索引号，从0开始。如果只有一个可控制亮度的LED，则该参数为0。
- [in] **uBrightness** LED亮度值，范围0到255. 0表示关闭，255最亮。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetLedDuration()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetLedDuration ( **CameraHandle** hCamera,  
int index,  
UINT duration  
)

设置相机的LED持续时间，不带LED的型号，此函数返回错误代码，表示不支持。

## 参数

- [in] **hCamera** 相机的句柄。
- [in] **index** LED灯的索引号，从0开始。如果只有一个可控制亮度的LED，则该参数为0。
- [in] **duration** LED持续时间，单位毫秒

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetLedEnable()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetLedEnable ( **CameraHandle** hCamera,  
int index,  
BOOL enable  
)

设置相机的LED使能状态，不带LED的型号，此函数返回错误代码，表示不支持。

## 参数

- [in] **hCamera** 相机的句柄。

[in] **index** LED灯的索引号，从0开始。如果只有一个可控制亮度的LED，则该参数为0。  
[in] **enable** 使能状态

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraSetLedOnOff()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetLedOnOff ( **CameraHandle** hCamera,  
int index,  
BOOL onoff  
)

设置相机的LED开关状态，不带LED的型号，此函数返回错误代码，表示不支持。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **index** LED灯的索引号，从0开始。如果只有一个可控制亮度的LED，则该参数为0。  
[in] **onoff** LED开关状态

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraSetLightingControllerMode()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetLightingControllerMode ( **CameraHandle** hCamera,  
int index,  
int mode  
)

设置光源控制器的输出模式（智能相机系列且需要硬件支持）

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **index** 控制器索引  
[in] **mode** 输出模式 (0:跟随闪光灯 1:手动)

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraSetLightingControllerState()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetLightingControllerState

( **CameraHandle** hCamera,  
    int                        index,  
    int                        state  
)

设置光源控制器的输出状态 (智能相机系列且需要硬件支持)

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **index**    控制器索引  
[in] **state**    输出状态 (0:关闭 1: 打开)

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraSetMediaBitRate()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetMediaBitRate

( **CameraHandle** hCamera,  
    int                        iMediaType,  
    UINT                       uRate  
)

设置码率。 (仅部分输出格式支持, 比如: H264、H265)

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iMediaType** 输出格式索引  
[in] **uRate**        码率

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetMediaType()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetMediaType( CameraHandle hCamera,  
                    INT iMediaType  
)
```

设置相机的输出原始数据格式。

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iMediaType** 新格式类型的索引号。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### 注解

与**CameraGetMediaType**相同。

## ◆ CameraSetRawStartBit()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetRawStartBit( CameraHandle hCamera,  
                      int startBit  
)
```

设置RAW数据的输出起始位

### 参数

[in] **hCamera** 相机的句柄。  
[in] **startBit** 起始BIT (默认输出高8位)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetSingleGrabMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetSingleGrabMode( CameraHandle hCamera,
```

```
    BOOL  
)  
bEnable
```

启用或禁用单帧抓取模式， 默认为禁用。 (本功能仅USB2.0相机支持)

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **bEnable** 使能单帧抓取模式

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

每当成功抓取到一帧后SDK会进入暂停状态，从而不再占用USB带宽，主要用于多相机轮流拍照的场景。

### ◆ CameraSetTransferRoi()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetTransferRoi

```
( CameraHandle hCamera,  
int index,  
UINT X1,  
UINT Y1,  
UINT X2,  
UINT Y2  
)
```

设置相机传输的裁剪区域。在相机端，图像从传感器上被采集后，将会被裁剪成指定的区域来传送，此函数返回错误代码，表示不支持。

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **index** ROI区域的索引号，从0开始。
- [in] **X1** ROI区域的左上角X坐标
- [in] **Y1** ROI区域的左上角Y坐标
- [in] **X2** ROI区域的右下角X坐标
- [in] **Y2** ROI区域的右下角Y坐标

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetTransPackLen()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetTransPackLen  
    ( CameraHandle hCamera,  
      INT           iPackSel  
    )
```

设置相机传输图像数据的分包大小。

### 参数

[in] **hCamera** 相机的句柄。

[in] **iPackSel** 分包长度选择的索引号。分包长度可由获得相机属性结构体中 **tSdkCameraCapbility.pPackLenDesc** 成员表述, **tSdkCameraCapbility.iPackLenDesc** 成员则表示最大可选的分包模式个数。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### 注解

目前的SDK版本中, 该接口仅对GIGE接口相机有效, 用来控制网络传输的分包大小。

对于支持巨帧的网卡, 我们建议选择8K的分包大小, 可以有效的降低传输所占用的CPU处理时间。

### 警告

新版本的SDK无需调用此函数, SDK会自动根据网络情况协商最优的分包大小

# MVSDK API

## 色彩调节

API

函数

### 函数

MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetWbMode</b> ( <b>CameraHandle</b> hCamera, BOOL bAuto)	设置相机白平衡模式。分为手动 和自动两种方式。 <a href="#">更多...</a>
-------------------------------------	---	--

MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetWbMode</b> ( <b>CameraHandle</b> hCamera, BOOL *pbAuto)	获得当前的白平衡模式。 <a href="#">更 多...</a>
-------------------------------------	---	--

MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetPresetClrTemp</b> ( <b>CameraHandle</b> hCamera, int iSel)	选择指定预设色温模式 <a href="#">更多...</a>
-------------------------------------	--	----------------------------------

MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetPresetClrTemp</b> ( <b>CameraHandle</b> hCamera, int *piSel)	获得当前选择的预设色温模式。 <a href="#">更多...</a>
-------------------------------------	--	---

MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetUserClrTempGain</b> ( <b>CameraHandle</b> hCamera, int iRgain, int iGgain, int iBgain)	设置自定义色温模式下的数字增 益 <a href="#">更多...</a>
-------------------------------------	--	---

MVSDK_API CameraSdkStatus __stdcall	<b>Camera GetUserClrTempGain</b> ( <b>CameraHandle</b> hCamera, int	
-------------------------------------	--	--

\*piRgain, int \*piGgain, int  
\*piBgain)  
获得自定义色温模式下的数字增  
益 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraSetUserClrTempMatrix](#)  
(**CameraHandle** hCamera, float  
\*pMatrix)  
设置自定义色温模式下的颜色矩  
阵 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [Camera GetUserClrTempMatrix](#)  
(**CameraHandle** hCamera, float  
\*pMatrix)  
获得自定义色温模式下的颜色矩  
阵 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraSetClrTempMode](#)  
(**CameraHandle** hCamera, int  
iMode)  
设置白平衡时使用的色温模式  
[更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraGetClrTempMode](#)  
(**CameraHandle** hCamera, int  
\*pimode)  
获得白平衡时使用的色温模式。  
参考[CameraSetClrTempMode](#)  
中功能描述部分。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraSetOnceWB](#)  
(**CameraHandle** hCamera)  
在手动白平衡模式下，调用该函  
数会进行一次白平衡。生效的时  
间为接收到下一帧图像数据时。  
[更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameraSetOnceBB](#)  
(**CameraHandle** hCamera)  
执行一次黑平衡操作。（需要相  
机支持本功能） [更多...](#)

MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetGain</b> ( <b>CameraHandle</b> hCamera, int iRGain, int iGGain, int iBGain) 设置图像的数字增益。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetGain</b> ( <b>CameraHandle</b> hCamera, int *piRGain, int *piGGain, int *piBGain) 获得图像处理的数字增益。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetWbWindow</b> ( <b>CameraHandle</b> hCamera, INT *PiHOff, INT *PiVOff, INT *PiWidth, INT *PiHeight) 获得白平衡参考窗口的位置。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetWbWindow</b> ( <b>CameraHandle</b> hCamera, INT iHOff, INT iVOff, INT iWidth, INT iHeight) 设置白平衡参考窗口的位置。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CamerasWbWinVisible</b> ( <b>CameraHandle</b> hCamera, BOOL *pbShow) 获得白平衡窗口的显示状态。 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetWbWinVisible</b> ( <b>CameraHandle</b> hCamera, BOOL bShow) 设置白平衡窗口的显示状态。 <a href="#">更多...</a>

## 详细描述

## 函数说明

### ◆ CameraGetClrTempMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetClrTempMode( CameraHandle hCamera,  
                      int * pimode  
)
```

获得白平衡时使用的色温模式。参考[CameraSetClrTempMode](#)中功能描述部分。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **pimode** 指针，返回模式选择，参考[emSdkClrTmpMode](#)类型定义

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetGain()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetGain( CameraHandle hCamera,  
                int * piRGain,  
                int * piGGain,  
                int * piBGain  
)
```

获得图像处理的数字增益。

## 参数

- [in] **hCamera** 相机的句柄。
- [out] **piRGain** 指针，返回红色通道的数字增益值。
- [out] **piGGain** 指针，返回绿色通道的数字增益值。
- [out] **piBGain** 指针，返回蓝色通道的数字增益值。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## 参见

[CameraSetGain](#)

## ◆ CameraGetPresetClrTemp()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetPresetClrTemp ( CameraHandle hCamera,  
                         int * piSel  
                       )
```

获得当前选择的预设色温模式。

## 参数

- [in] **hCamera** 相机的句柄。
- [out] **piSel** 返回选择的预设色温索引号

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ Camera GetUserClrTempGain()

```
MVSDK_API CameraSdkStatus __stdcall  
Camera GetUserClrTempGain ( CameraHandle hCamera,  
                           int * piRgain,
```

```
    int *          piGgain,  
    int *          piBgain  
)
```

获得自定义色温模式下的数字增益

#### 参数

- [in] **hCamera** 相机的句柄。
- [out] **piRgain** 指针, 返回红色增益, 范围0到400, 表示0到4倍
- [out] **piGgain** 指针, 返回绿色增益, 范围0到400, 表示0到4倍
- [out] **piBgain** 指针, 返回蓝色增益, 范围0到400, 表示0到4倍

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ Camera GetUserClrTempMatrix()

```
MVSDK_API CameraSdkStatus __stdcall  
Camera GetUserClrTempMatrix ( CameraHandle hCamera,  
                           float *          pMatrix  
)
```

获得自定义色温模式下的颜色矩阵

#### 参数

- [in] **hCamera** 相机的句柄。
- [out] **pMatrix** 指向一个float[3][3]数组的首地址

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetWbMode()

```
MVSDK_API CameraSdkStatus __stdcall ( CameraHandle hCamera,
```

## CameraGetWbMode

```
    BOOL * pbAuto  
)
```

获得当前的白平衡模式。

### 参数

- [in] **hCamera** 相机的句柄。
- [out] **pbAuto** 指针，返回TRUE表示自动模式，FALSE为手动模式。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetWbWindow()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetWbWindow
```

```
( CameraHandle hCamera,  
    INT *          PiHOff,  
    INT *          PiVOff,  
    INT *          PiWidth,  
    INT *          PiHeight  
)
```

获得白平衡参考窗口的位置。

### 参数

- [in] **hCamera** 相机的句柄。
- [out] **PiHOff** 指针，返回参考窗口的左上角横坐标。
- [out] **PiVOff** 指针，返回参考窗口的左上角纵坐标。
- [out] **PiWidth** 指针，返回参考窗口的宽度。
- [out] **PiHeight** 指针，返回参考窗口的高度。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraIsWbWinVisible()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraIsWbWinVisible( CameraHandle hCamera,  
                      BOOL * pbShow  
)
```

获得白平衡窗口的显示状态。

### 参数

- [in] **hCamera** 相机的句柄。
- [out] **pbShow** 指针，返回TRUE，则表示窗口是可见的。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetClrTempMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetClrTempMode( CameraHandle hCamera,  
                      int iMode  
)
```

设置白平衡时使用的色温模式

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iMode** 模式，只能是**emSdkClrTmpMode**中定义的一种

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### 注解

支持的模式有三种，分别是自动，预设和自定义。

自动模式下，会自动选择合适的色温模式  
预设模式下，会使用用户指定的色温模式  
自定义模式下，使用用户自定义的色温数字增益和矩阵

### ◆ CameraSetGain()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetGain

( **CameraHandle** hCamera,  
int iRGain,  
int iGGain,  
int iBGain  
)

设置图像的数字增益。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iRGain** 红色通道的增益值。  
[in] **iGGain** 绿色通道的增益值。  
[in] **iBGain** 蓝色通道的增益值。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

#### 注解

设定范围由**tRgbGainRange**成员表述。实际的放大倍数是设定值/100。

### ◆ CameraSetOnceBB()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetOnceBB

( **CameraHandle** hCamera )

执行一次黑平衡操作。（需要相机支持本功能）

## 参数

[in] **hCamera** 相机的句柄。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetOnceWB()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetOnceWB ( **CameraHandle** hCamera )

在手动白平衡模式下, 调用该函数会进行一次白平衡。生效的时间为接收  
到下一帧图像数据时。

## 参数

[in] **hCamera** 相机的句柄。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetPresetClrTemp()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetPresetClrTemp ( **CameraHandle** hCamera,  
int iSel  
)

选择指定预设色温模式

## 参数

[in] **hCamera** 相机的句柄。

[in] **iSel** 预设色温的模式索引号, 从0开始

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

调用[CameraSetClrTempMode](#)设置为预设模式。

### ◆ CameraSetUserClrTempGain()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetUserClrTempGain ( CameraHandle hCamera,  
                           int          iRgain,  
                           int          iGgain,  
                           int          iBgain  
                         )
```

设置自定义色温模式下的数字增益

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iRgain** 红色增益, 范围0到400, 表示0到4倍
- [in] **iGgain** 绿色增益, 范围0到400, 表示0到4倍
- [in] **iBgain** 蓝色增益, 范围0到400, 表示0到4倍

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

调用[CameraSetClrTempMode](#)设置为自定义模式。

### ◆ CameraSetUserClrTempMatrix()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetUserClrTempMatrix ( CameraHandle hCamera,  
                            float *      pMatrix  
                          )
```

设置自定义色温模式下的颜色矩阵

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **pMatrix** 指向一个float[3][3]数组的首地址

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### 注解

调用**CameraSetClrTempMode**设置为自定义模式。

## ◆ CameraSetWbMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetWbMode  
    ( CameraHandle hCamera,  
      BOOL          bAuto  
    )
```

设置相机白平衡模式。分为手动和自动两种方式。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **bAuto** TRUE, 则表示使能自动模式。FALSE, 则表示使用手动模式, 通过调用**CameraSetOnceWB**来进行一次白平衡。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetWbWindow()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetWbWindow  
    ( CameraHandle hCamera,  
      INT          iHOff,
```

```
    INT          iVOff,  
    INT          iWidth,  
    INT          iHeight  
)
```

设置白平衡参考窗口的位置。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iHOff** 参考窗口的左上角横坐标。
- [in] **iVOff** 参考窗口的左上角纵坐标。
- [in] **iWidth** 参考窗口的宽度。
- [in] **iHeight** 参考窗口的高度。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetWbWinVisible()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetWbWinVisible ( CameraHandle hCamera,  
                         BOOL          bShow  
)
```

设置白平衡窗口的显示状态。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **bShow** TRUE, 则表示设置为可见。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### 注解

在调用**CameralImageOverlay**后, 图像内容上将以矩形的方式叠加白平衡参考窗口的位置。



# MVSDK API

## 图像增强

API

函数

### 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetSharpness** (CameraHandle hCamera, int iSharpness)  
设置图像的处理的锐化参数。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetSharpness** (CameraHandle hCamera, int \*piSharpness)  
获取当前锐化设定值。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetLutMode** (CameraHandle hCamera, int emLutMode)  
设置相机的查表变换模式LUT模式。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetLutMode** (CameraHandle hCamera, int \*pemLutMode)  
获得相机的查表变换模式LUT模式。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSelectLutPreset** (CameraHandle hCamera, int iSel)  
选择预设LUT模式下的LUT表。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetLutPresetSel** (CameraHandle hCamera, int \*piSel)  
获得预设LUT模式下的LUT表索引号。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetCustomLut** (CameraHandle hCamera, int iChannel, USHORT \*pLut)  
设置自定义的LUT表。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetCustomLut** (CameraHandle hCamera, int iChannel, USHORT \*pLut)  
获得当前使用的自定义LUT表。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetCurrentLut** (CameraHandle hCamera, int iChannel, USHORT \*pLut)  
获得相机当前的LUT表，在任何LUT模式下都可以调用,用来直观的观察LUT曲线的变化。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetGamma** (CameraHandle hCamera, int iGamma)  
设定LUT动态生成模式下的Gamma值。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetGamma** (CameraHandle hCamera, int

\*piGamma)  
获得LUT动态生成模式下的Gamma值 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetContrast** (CameraHandle hCamera, int iContrast)  
设定LUT动态生成模式下的对比度值。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetContrast** (CameraHandle hCamera, int \*piContrast)  
获得LUT动态生成模式下的对比度值。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetSaturation** (CameraHandle hCamera, int iSaturation)  
设定图像处理的饱和度。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetSaturation** (CameraHandle hCamera, int \*piSaturation)  
获得图像处理的饱和度。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetMonochrome** (CameraHandle hCamera, BOOL bEnable)  
设置彩色转为黑白功能的使能。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetMonochrome** (CameraHandle hCamera, BOOL \*pbEnable)  
获得彩色转换黑白功能的使能状况。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetInverse** (CameraHandle hCamera, BOOL bEnable)  
设置彩图像颜色翻转功能的使能。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetInverse** (CameraHandle hCamera, BOOL \*pbEnable)  
获得图像颜色反转功能的使能状态。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetNoiseFilter** (CameraHandle hCamera, BOOL bEnable)  
设置图像降噪模块的使能状态。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetNoiseFilterState** (CameraHandle hCamera, BOOL \*pEnable)  
获得图像降噪模块的使能状态。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraFlatFieldingCorrectSetEnable** (CameraHandle hCamera, BOOL bEnable)  
使能平场校正 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraFlatFieldingCorrectGetEnable** (CameraHandle hCamera, BOOL \*pbEnable)  
获取平场校正使能状态 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraFlatFieldingCorrectSetParameter**  
(**CameraHandle** hCamera, BYTE const  
\*pDarkFieldingImage, **tSdkFrameHead** const  
\*pDarkFieldingFrInfo, BYTE const  
\*pLightFieldingImage, **tSdkFrameHead** const  
\*pLightFieldingFrInfo)  
设置平场校正参数 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraFlatFieldingCorrectGetParameterState**  
(**CameraHandle** hCamera, BOOL \*pbValid, char  
\*pFilePath)  
获取平场校正参数的状态 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraFlatFieldingCorrectSaveParameterToFile**  
(**CameraHandle** hCamera, char const \*pszFileName)  
保存平场校正参数到文件 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraFlatFieldingCorrectLoadParameterFromFile**  
(**CameraHandle** hCamera, char const \*pszFileName)  
从文件中加载平场校正参数 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetDenoise3DParams** (**CameraHandle**  
hCamera, BOOL bEnable, int nCount, float \*Weights)  
设置3D降噪参数 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetDenoise3DParams** (**CameraHandle**  
hCamera, BOOL \*bEnable, int \*nCount, BOOL  
\*bUseWeight, float \*Weights)  
获取当前的3D降噪参数 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraManualDenoise3D** (**tSdkFrameHead**  
\*InFramesHead, BYTE \*\*InFramesData, int nCount,  
float \*Weights, **tSdkFrameHead** \*OutFrameHead,  
BYTE \*OutFrameData)  
对一组帧进行一次降噪处理 [更多...](#)

## 详细描述

### 函数说明

#### ◆ CameraFlatFieldingCorrectGetEnable()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraFlatFieldingCorrectGetEnable

( **CameraHandle** hCamera,  
BOOL \* pbEnable  
)

获取平场校正使能状态

##### 参数

[in] **hCamera** 相机的句柄。  
[out] **pbEnable** 返回使能状态

##### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考  
**CameraStatus.h** 中错误码的定义。

#### ◆ CameraFlatFieldingCorrectGetParameterState()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraFlatFieldingCorrectGetParameterState

( **CameraHandle** hCamera,  
BOOL \* pbValid,  
char \* pFilePath  
)

获取平场校正参数的状态

##### 参数

[in] **hCamera** 相机的句柄。  
[out] **pbValid** 返回参数是否有效  
[out] **pFilePath** 返回参数文件的路径

##### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考  
**CameraStatus.h** 中错误码的定义。

#### ◆ CameraFlatFieldingCorrectLoadParameterFromFile()

MVSDK\_API CameraSdkStatus \_\_stdcall  
CameraFlatFieldingCorrectLoadParameterFromFile

( CameraHandle hCamera,  
char const \* pszFileName  
)

从文件中加载平场校正参数

**参数**

[in] **hCamera** 相机的句柄。  
[in] **pszFileName** 文件路径

**返回**

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考  
**CameraStatus.h** 中错误码的定义。

◆ CameraFlatFieldingCorrectSaveParameterToFile()

MVSDK\_API CameraSdkStatus \_\_stdcall  
CameraFlatFieldingCorrectSaveParameterToFile

( CameraHandle hCamera,  
char const \* pszFileName  
)

保存平场校正参数到文件

**参数**

[in] **hCamera** 相机的句柄。  
[in] **pszFileName** 文件路径

**返回**

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考  
**CameraStatus.h** 中错误码的定义。

◆ CameraFlatFieldingCorrectSetEnable()

MVSDK\_API CameraSdkStatus \_\_stdcall  
CameraFlatFieldingCorrectSetEnable

( CameraHandle hCamera,  
BOOL bEnable  
)

使能平场校正

**参数**

[in] **hCamera** 相机的句柄。  
[in] **bEnable** TRUE: 使能平场校正 FALSE: 关闭平场校正

**返回**

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### ◆ CameraFlatFieldingCorrectSetParameter()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraFlatFieldingCorrectSetParameter ( CameraHandle hCamera,  
BYTE const * pDarkFieldingImage,  
tSdkFrameHead const * pDarkFieldingFrInfo,  
BYTE const * pLightFieldingImage,  
tSdkFrameHead const * pLightFieldingFrInfo  
)
```

设置平场校正参数

##### 参数

[in] **hCamera** 相机的句柄。  
[in] **pDarkFieldingImage** 暗场图片  
[in] **pDarkFieldingFrInfo** 暗场图片信息  
[in] **pLightFieldingImage** 明场图片  
[in] **pLightFieldingFrInfo** 明场图片信息

##### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### ◆ CameraGetContrast()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetContrast ( CameraHandle hCamera,  
int * piContrast  
)
```

获得LUT动态生成模式下的对比度值。

##### 参数

[in] **hCamera** 相机的句柄。  
[out] **piContrast** 指针, 返回当前的对比度值。

##### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

##### 参见

[CameraSetContrast](#)

## ◆ CameraGetCurrentLut()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetCurrentLut ( CameraHandle hCamera,
                                                       int iChannel,
                                                       USHORT * pLut
)
```

获得相机当前的LUT表，在任何LUT模式下都可以调用，用来直观的观察LUT曲线的变化。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iChannel** 指定要获取的LUT颜色通道，当为**LUT\_CHANNEL\_ALL**时，返回红色通道的LUT表。

### 参见

[emSdkLutChannel](#)

### 参数

- [out] **pLut** 指向LUT表的地址。LUT表为无符号短整形数组，数组大小为4096，分别代码颜色通道从0到4096(12bit颜色精度)对应的映射值。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetCustomLut()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetCustomLut ( CameraHandle hCamera,
                                                       int iChannel,
                                                       USHORT * pLut
)
```

获得当前使用的自定义LUT表。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iChannel** 指定要获取的LUT颜色通道，当为**LUT\_CHANNEL\_ALL**时，返回红色通道的LUT表。

### 参见

[emSdkLutChannel](#)

### 参数

- [out] **pLut** 指向LUT表的地址。LUT表为无符号短整形数组，数组大小为4096，分别代码颜色通道从0到4096(12bit颜色精度)对应的映射值。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetDenoise3DParams()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetDenoise3DParams( CameraHandle hCamera,  
                           BOOL * bEnable,  
                           int * nCount,  
                           BOOL * bUseWeight,  
                           float * Weights  
                         )
```

获取当前的3D降噪参数

### 参数

[in] **hCamera** 相机的句柄。  
[out] **bEnable** 启用或禁用  
[out] **nCount** 使用了几张图片进行降噪  
[out] **bUseWeight** 是否使用了降噪权重  
[out] **Weights** 降噪权重

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraGetGamma()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetGamma( CameraHandle hCamera,  
                                                       int * piGamma  
                                                     )
```

获得LUT动态生成模式下的Gamma值

### 参数

[in] **hCamera** 相机的句柄。  
[out] **piGamma** 指针, 返回当前的Gamma值。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### 参见

[CameraSetGamma](#)

## ◆ CameraGetInverse()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetInverse( CameraHandle hCamera,
```

```
    BOOL * pbEnable  
)
```

获得图像颜色反转功能的使能状态。

#### 参数

- [in] **hCamera** 相机的句柄。
- [out] **pbEnable** 指针, 返回该功能使能状态。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetLutMode()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetLutMode ( CameraHandle hCamera,  
                                         int * pemLutMode  
)
```

获得相机的查表变换模式LUT模式。

#### 参数

- [in] **hCamera** 相机的句柄。
- [out] **pemLutMode** 返回当前LUT模式。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetLutPresetSel()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetLutPresetSel ( CameraHandle hCamera,  
                                         int * piSel  
)
```

获得预设LUT模式下的LUT表索引号。

#### 参数

- [in] **hCamera** 相机的句柄。
- [out] **piSel** 返回表的索引号。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraGetMonochrome()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetMonochrome ( CameraHandle hCamera,
                                                               BOOL *          pbEnable
                                                               )
```

获得彩色转换黑白功能的使能状况。

### 参数

[in] **hCamera** 相机的句柄。

[out] **pbEnable** 指针。返回TRUE表示开启了彩色图像转换为黑白图像的功能。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### 参见

[CameraSetMonochrome](#)

## ◆ CameraGetNoiseFilterState()

```
MVSDK_API CameraSdkStatus __stdcall
CameraGetNoiseFilterState
( CameraHandle hCamera,
  BOOL *          pEnable
  )
```

获得图像降噪模块的使能状态。

### 参数

[in] **hCamera** 相机的句柄。

[out] **pEnable** 指针, 返回状态。TRUE, 为使能。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraGetSaturation()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetSaturation ( CameraHandle hCamera,
                                                               int *           piSaturation
                                                               )
```

获得图像处理的饱和度。

### 参数

[in] **hCamera** 相机的句柄。

[out] **piSaturation** 指针，返回当前图像处理的饱和度值。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

#### 参见

[CameraSetSaturation](#)

### ◆ CameraGetSharpness()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetSharpness ( CameraHandle hCamera,
                                                               int * piSharpness
                                                               )
```

获取当前锐化设定值。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **piSharpness** 返回当前设定的锐化的设定值。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraManualDenoise3D()

```
MVSDK_API CameraSdkStatus __stdcall
CameraManualDenoise3D
(
    tSdkFrameHead * InFramesHead,
    BYTE ** InFramesData,
    int nCount,
    float * Weights,
    tSdkFrameHead * OutFrameHead,
    BYTE * OutFrameData
)
```

对一组帧进行一次降噪处理

#### 参数

[in] **InFramesHead** 输入帧头  
[in] **InFramesData** 输入帧数据  
[in] **nCount** 输入帧的数量  
[in] **Weights** 降噪权重  
[out] **OutFrameHead** 输出帧头  
[out] **OutFrameData** 输出帧数据

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### ◆ CameraSelectLutPreset()

```
MVSDK_API CameraSdkStatus __stdcall CameraSelectLutPreset ( CameraHandle hCamera,  
                                         int iSel  
                                         )
```

选择预设LUT模式下的LUT表。

##### 参数

[in] **hCamera** 相机的句柄。

[in] **iSel** 表的索引号。表的个数由**tSdkCameraCapability.iPresetLut**获得。

##### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

##### 注解

必须先使用**CameraSetLutMode**将LUT模式设置为预设模式。

#### ◆ CameraSetContrast()

```
MVSDK_API CameraSdkStatus __stdcall CameraSetContrast ( CameraHandle hCamera,  
                                         int iContrast  
                                         )
```

设定LUT动态生成模式下的对比度值。

##### 参数

[in] **hCamera** 相机的句柄。

[in] **iContrast** 设定的对比度值。

##### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

##### 注解

设定的值会马上保存在SDK内部, 但是只有当相机处于动态参数生成的LUT模式时, 才会生效。请参考**CameraSetLutMode**的函数说明部分。

#### ◆ CameraSetCustomLut()

```
MVSDK_API CameraSdkStatus __stdcall CameraSetCustomLut ( CameraHandle hCamera,  
                                         int iChannel,
```

```
USHORT *      pLut  
 )
```

设置自定义的LUT表。

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iChannel** 指定要设定的LUT颜色通道，当为**LUT\_CHANNEL\_ALL**时，三个通道的LUT将被同时替换。

#### 参见

[emSdkLutChannel](#)

#### 参数

- [in] **pLut** 指针，指向LUT表的地址。LUT表为无符号短整形数组，数组大小为4096，分别代表颜色通道从0到4096(12bit颜色精度)对应的映射值。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

必须先使用[CameraSetLutMode](#)将LUT模式设置为自定义模式。

## ◆ CameraSetDenoise3DParams()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetDenoise3DParams

```
( CameraHandle hCamera,  
  BOOL          bEnable,  
  int           nCount,  
  float *       Weights  
)
```

设置3D降噪参数

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **bEnable** 启用或禁用
- [in] **nCount** 使用几张图片进行降噪(2-8张)
- [in] **Weights** 降噪权重，如当使用3张图片进行降噪则这个参数可以传入3个浮点(0.3,0.3,0.4)，最后一张图片的权重大于前2张。如果不使用权重，则把这个参数传入0，表示所有图片的权重相同(0.33,0.33,0.33)

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetGamma()

```
MVSDK_API CameraSdkStatus __stdcall CameraSetGamma ( CameraHandle hCamera,
                                                    int             iGamma
                                                    )
```

设定LUT动态生成模式下的Gamma值。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **iGamma** 要设定的Gamma值。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### 注解

设定的值会马上保存在SDK内部, 但是只有当相机处于动态参数生成的LUT模式时, 才会生效。请参考**CameraSetLutMode**的函数说明部分。

### ◆ CameraSetInverse()

```
MVSDK_API CameraSdkStatus __stdcall CameraSetInverse ( CameraHandle hCamera,
                                                       BOOL            bEnable
                                                       )
```

设置彩图像颜色翻转功能的使能。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **bEnable** TRUE, 表示开启图像颜色翻转功能, 可以获得类似胶卷底片的效果。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraSetLutMode()

```
MVSDK_API CameraSdkStatus __stdcall CameraSetLutMode ( CameraHandle hCamera,
                                                       int             emLutMode
                                                       )
```

设置相机的查表变换模式LUT模式。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **emLutMode** 定义参考**emSdkLutMode**类型。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### ◆ CameraSetMonochrome()

```
MVSDK_API CameraSdkStatus __stdcall CameraSetMonochrome ( CameraHandle hCamera,  
                                         BOOL             bEnable  
                                         )
```

设置彩色转为黑白功能的使能。

##### 参数

[in] **hCamera** 相机的句柄。  
[in] **bEnable** TRUE, 表示将彩色图像转为黑白。

##### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### ◆ CameraSetNoiseFilter()

```
MVSDK_API CameraSdkStatus __stdcall CameraSetNoiseFilter ( CameraHandle hCamera,  
                                         BOOL             bEnable  
                                         )
```

设置图像降噪模块的使能状态。

##### 参数

[in] **hCamera** 相机的句柄。  
[in] **bEnable** TRUE, 使能; FALSE, 禁止。

##### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### ◆ CameraSetSaturation()

```
MVSDK_API CameraSdkStatus __stdcall CameraSetSaturation ( CameraHandle hCamera,  
                                         int              iSaturation  
                                         )
```

设定图像处理的饱和度。

##### 参数

[in] **hCamera** 相机的句柄。

[in] **iSaturation** 设定的饱和度值。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

对黑白相机无效。设定范围由**tSaturationRange**获得。100表示原始色度, 不增强。

### ◆ CameraSetSharpness()

```
MVSDK_API CameraSdkStatus __stdcall CameraSetSharpness ( CameraHandle hCamera,
                                                               int           iSharpness
                                                               )
```

设置图像的处理的锐化参数。

#### 参数

[in] **hCamera** 相机的句柄。

[in] **iSharpness** 锐化参数, 一般是[0,100], 0表示关闭锐化处理。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

# MVSDK API

## 实用功能

API

函数

### 函数

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraSetCrossLine**  
(**CameraHandle** hCamera, int iLine,  
INT x, INT y, UINT uColor, BOOL  
bVisible)  
设置指定十字线的参数。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetCrossLine**  
(**CameraHandle** hCamera, INT iLine,  
INT \*px, INT \*py, UINT \*pcolor, BOOL  
\*pbVisible)  
获得指定十字线的状态。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraCustomizeReferWin**  
(**CameraHandle** hCamera, INT  
iWinType, HWND hParent, INT  
\*piHOFF, INT \*piVOFF, INT \*piWidth,  
INT \*piHeight)  
打开参考窗口自定义面板。并通过可  
视化的方式来获得一个自定义窗口的  
位置。一般是用自定义白平衡和自动  
曝光的参考窗口。 [更多...](#)

MVSDK\_API char \* \_\_stdcall **CameraGetErrorString**  
(**CameraSdkStatus** iStatusCode)  
获得错误码对应的描述字符串 [更多...](#)

MVSDK\_API BYTE \* \_\_stdcall **CameraAlignMalloc** (int size, int  
align)  
申请一段对齐的内存空间。功能和  
malloc类似，但是返回的内存是以  
align指定的字节数对齐的。 [更多...](#)

MVSDK\_API void \_\_stdcall **CameraAlignFree** (BYTE  
\*membuffer)

释放由**CameraAlignMalloc**函数分配的内存空间。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraEvaluateImageDefinition** (**CameraHandle** hCamera, INT iAlgorithmSel, BYTE \*pbvln, **tSdkFrameHead** \*pFrInfo, double \*DefinitionValue)  
图片清晰度评估 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraDrawText** (BYTE \*pRgbBuffer, **tSdkFrameHead** \*pFrInfo, char const \*pFontFileName, UINT FontWidth, UINT FontHeight, char const \*pText, INT Left, INT Top, UINT Width, UINT Height, UINT TextColor, UINT uFlags)  
在输入的图像数据中绘制文字 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGigeGetIp** (**tSdkCameraDevInfo** \*pCameraInfo, char \*CamIp, char \*CamMask, char \*CamGateWay, char \*EtIp, char \*EtMask, char \*EtGateWay)  
获取GIGE相机的IP地址 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGigeSetIp** (**tSdkCameraDevInfo** \*pCameraInfo, char const \*Ip, char const \*SubMask, char const \*GateWay, BOOL bPersistent)  
设置GIGE相机的IP地址 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGigeGetMac** (**tSdkCameraDevInfo** \*pCameraInfo, char \*CamMac, char \*EtMac)  
获取GIGE相机的MAC地址 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraCreateDIBitmap** (HDC hDC, BYTE \*pFrameBuffer, **tSdkFrameHead** \*pFrameHead, HBITMAP \*outBitmap)  
从帧数据创建HBITMAP [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraDrawFrameBuffer** (BYTE \*pFrameBuffer, **tSdkFrameHead** \*pFrameHead, HWND hWnd, int Algorithm, int Mode)  
绘制帧到指定窗口 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraFlipFrameBuffer** (BYTE \*pFrameBuffer, **tSdkFrameHead** \*pFrameHead, int Flags)  
翻转帧数据 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraConvertFrameBufferFormat** (**CameraHandle** hCamera, BYTE \*pInFrameBuffer, BYTE \*pOutFrameBuffer, int outWidth, int outHeight, UINT outMediaType, **tSdkFrameHead** \*pFrameHead)  
转换帧数据格式 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetRegionAverageGray** (BYTE \*pFrameBuffer, **tSdkFrameHead** \*pFrameHead, int Left, int Top, int Width, int Height, int \*AvgGray)  
计算区域的平均灰度值 [更多...](#)

## 详细描述

### 函数说明

#### ◆ CameraAlignFree()

```
MVSDK_API void __stdcall CameraAlignFree ( BYTE * membuffer )
```

释放由**CameraAlignMalloc**函数分配的内存空间。

#### 参数

[in] **membuffer** 内存地址

#### 示例:

[grab.cpp](#), [roi.cpp](#), [soft\\_trigger.cpp](#), [win\\_basic.cpp](#), 以及  
[win\\_callback.cpp](#).

#### ◆ CameraAlignMalloc()

```
MVSDK_API BYTE* __stdcall CameraAlignMalloc ( int size,
                                              int align
                                            )
```

申请一段对齐的内存空间。功能和malloc类似，但是返回的内存是以align指定的字节数对齐的。

#### 参数

[in] **size** 空间的大小。  
[in] **align** 地址对齐的字节数。

#### 返回

成功时，返回非0值，表示内存首地址。失败返回NULL。

#### 注解

分配的内存必须使用**CameraAlignFree**释放

示例:

[grab.cpp](#), [roi.cpp](#), [soft\\_trigger.cpp](#), [win\\_basic.cpp](#), 以及  
[win\\_callback.cpp](#).

## ◆ CameraConvertFrameBufferFormat()

MVSDK\_API **CameraSdkStatus**

\_\_stdcall

CameraConvertFrameBufferFormat

```
( CameraHandle hCamera,  
    BYTE * pInFrameBuffer,  
    BYTE * pOutFrameBuffer,  
    int outWidth,  
    int outHeight,  
    UINT outMediaType,  
    tSdkFrameHead * pFrameHead  
)
```

转换帧数据格式

### 参数

[in]	<b>hCamera</b>	相机的句柄。
[in]	<b>pInFrameBuffer</b>	输入帧数据
[out]	<b>pOutFrameBuffer</b>	输出帧数据
[in]	<b>outWidth</b>	输出宽度
[in]	<b>outHeight</b>	输出高度
[in]	<b>outMediaType</b>	输出格式

### 参见

[CameraSetIspOutFormat](#)

### 参数

[in, out] **pFrameHead** 帧头信息 (转换成功后, 里面的信息会被修改为输出帧的信息)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraCreateDIBitmap()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraCreateDIBitmap( HDC hDC,  
                      BYTE * pFrameBuffer,  
                      tSdkFrameHead * pFrameHead,  
                      HBITMAP * outBitmap  
)
```

从帧数据创建HBITMAP

### 参数

[in] <b>hDC</b>	Handle to a device context (WIN32 API CreateDIBitmap的参数hdc)
[in] <b>pFrameBuffer</b>	帧数据
[in] <b>pFrameHead</b>	帧头
[out] <b>outBitmap</b>	新创建的HBITMAP (使用完后需要调用WIN32 API DeleteObject释放)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraCustomizeReferWin()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraCustomizeReferWin( CameraHandle hCamera,  
                         INT iWinType,  
                         HWND hParent,  
                         INT * piHOFF,  
                         INT * piVOFF,  
                         INT * piWidth,  
                         INT * piHeight  
)
```

打开参考窗口自定义面板。并通过可视化的方式来获得一个自定义窗口的位置。  
一般是用自定义白平衡和自动曝光的参考窗口。

## 参数

- [in] **hCamera** 相机的句柄。
- [in] **iWinType** 要生成的参考窗口的用途。0:自动曝光参考窗口；1:白平衡参考窗口。
- [in] **hParent** 调用该函数的窗口的句柄。可以为NULL。
- [out] **piHOff** 指针，返回自定义窗口的左上角横坐标。
- [out] **piVOff** 指针，返回自定义窗口的左上角纵坐标。
- [out] **piWidth** 指针，返回自定义窗口的宽度。
- [out] **piHeight** 指针，返回自定义窗口的高度。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraDrawFrameBuffer()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraDrawFrameBuffer( BYTE * pFrameBuffer,  
                      tSdkFrameHead * pFrameHead,  
                      HWND hWnd,  
                      int Algorithm,  
                      int Mode  
)
```

绘制帧到指定窗口

## 参数

- [in] **pFrameBuffer** 帧数据
- [in] **pFrameHead** 帧头
- [in] **hWnd** 目的窗口
- [in] **Algorithm** 缩放算法 0: 快速但质量稍差 1: 速度慢但质量好
- [in] **Mode** 缩放模式 0: 等比缩放 1: 拉伸缩放

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraDrawText()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraDrawText( BYTE * pRgbBuffer,  
                tSdkFrameHead * pFrInfo,  
                char const * pFontFileName,  
                UINT FontWidth,  
                UINT FontHeight,  
                char const * pText,  
                INT Left,  
                INT Top,  
                UINT Width,  
                UINT Height,  
                UINT TextColor,  
                UINT uFlags  
)
```

在输入的图像数据中绘制文字

### 参数

[in,out]	<b>pRgbBuffer</b>	图像数据缓冲区
[in]	<b>pFrInfo</b>	图像的帧头信息
[in]	<b>pFontFileName</b>	字体文件名
[in]	<b>FontWidth</b>	字体宽度
[in]	<b>FontHeight</b>	字体高度
[in]	<b>pText</b>	要输出的文字
[in]	<b>Left</b>	文字的输出矩形
[in]	<b>Top</b>	文字的输出矩形
[in]	<b>Width</b>	文字的输出矩形
[in]	<b>Height</b>	文字的输出矩形
[in]	<b>TextColor</b>	文字颜色RGB
[in]	<b>uFlags</b>	输出标志,详见 <b>emCameraDrawTextFlags</b> 中的定义

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码,请参考 **CameraStatus.h** 中错误码的定义。

- ◆ CameraEvaluateImageDefinition()

```
MVSDK_API CameraSdkStatus __stdcall ( CameraHandle hCamera,
```

## CameraEvaluateImageDefinition

```
INT iAlgorithSel,
BYTE * pbyIn,
tSdkFrameHead * pFrInfo,
double * DefinitionValue
)
```

### 图片清晰度评估

#### 参数

[in] hCamera	相机的句柄。
[in] iAlgorithSel	使用的评估算法,参考 <a href="#">emEvaluateDefinitionAlgorithm</a> 的定义
[in] pbyIn	输入图像数据的缓冲区地址,不能为NULL。
[in] pFrInfo	输入图像的帧头信息
[out] DefinitionValue	返回的清晰度估值 (越大越清晰)

#### 返回

成功返回 [CAMERA\\_STATUS\\_SUCCESS\(0\)](#)。否则返回非0值的错误码,请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraFlipFrameBuffer()

```
MVSDK_API CameraSdkStatus __stdcall
CameraFlipFrameBuffer( BYTE * pFrameBuffer,
                      tSdkFrameHead * pFrameHead,
                      int Flags
)
```

### 翻转帧数据

#### 参数

[in,out] pFrameBuffer	帧数据
[in] pFrameHead	帧头
[in] Flags	1:上下 2: 左右 3: 上下、左右皆做一次翻转(相当于旋转180度)

#### 返回

成功返回 [CAMERA\\_STATUS\\_SUCCESS\(0\)](#)。否则返回非0值的错误码,请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetCrossLine()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetCrossLine( CameraHandle hCamera,  
                    INT iLine,  
                    INT * px,  
                    INT * py,  
                    UINT * pcolor,  
                    BOOL * pbVisible  
)
```

获得指定十字线的状态。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iLine** 表示要获取的第几条十字线的状态。范围为[0,8], 共9条。
- [out] **px** 指针, 返回该十字线中心位置的横坐标。
- [out] **py** 指针, 返回该十字线中心位置的横坐标。
- [out] **pcolor** 指针, 返回该十字线的颜色, 格式为(RI(G<<8)I(B<<16))。
- [out] **pbVisible** 指针, 返回TRUE, 则表示该十字线可见。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGetErrorString()

```
MVSDK_API char* __stdcall  
CameraGetErrorString( CameraSdkStatus iStatusCode )
```

获得错误码对应的描述字符串

### 参数

- [in] **iStatusCode** 错误码。(定义于[CameraStatus.h](#)中)

### 返回

成功时，返回错误码对应的字符串首地址;否则返回NULL。

### ◆ CameraGetRegionAverageGray()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetRegionAverageGray( BYTE * pFrameBuffer,  
                           tSdkFrameHead * pFrameHead,  
                           int Left,  
                           int Top,  
                           int Width,  
                           int Height,  
                           int * AvgGray  
                           )
```

计算区域的平均灰度值

#### 参数

[in]	<b>pFrameBuffer</b>	帧数据
[in]	<b>pFrameHead</b>	帧头
[in]	<b>Left</b>	矩形区域的起始x坐标
[in]	<b>Top</b>	矩形区域的起始y坐标
[in]	<b>Width</b>	矩形区域的宽度
[in]	<b>Height</b>	矩形区域的高度
[out]	<b>AvgGray</b>	返回计算结果

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

Width或者Height为0则忽略Left、Top并返回整帧的平均灰度值

### ◆ CameraGigeGetIp()

```
MVSDK_API CameraSdkStatus  
__stdcall CameraGigeGetIp( tSdkCameraDevInfo * pCameraInfo,  
                           char * CamIp,  
                           char * CamMask,
```

```
    char * CamGateWay,  
    char * EtIp,  
    char * EtMask,  
    char * EtGateWay  
)
```

获取GIGE相机的IP地址

### 参数

[in] <b>pCameraInfo</b>	相机的设备描述信息，可由 <a href="#">CameraEnumerateDevice</a> 函数获得。
[out] <b>CamIp</b>	相机IP(注意：必须保证传入的缓冲区大于等于16字节)
[out] <b>CamMask</b>	相机子网掩码(注意：必须保证传入的缓冲区大于等于16字节)
[out] <b>CamGateWay</b>	相机网关(注意：必须保证传入的缓冲区大于等于16字节)
[out] <b>EtIp</b>	网卡IP(注意：必须保证传入的缓冲区大于等于16字节)
[out] <b>EtMask</b>	网卡子网掩码(注意：必须保证传入的缓冲区大于等于16字节)
[out] <b>EtGateWay</b>	网卡网关(注意：必须保证传入的缓冲区大于等于16字节)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraGigeGetMac()

MVSDK\_API **CameraSdkStatus**  
\_\_stdcall CameraGigeGetMac

```
( tSdkCameraDevInfo * pCameraInfo,  
  char * CamMac,  
  char * EtMac  
)
```

获取GIGE相机的MAC地址

### 参数

[in] <b>pCameraInfo</b>	相机的设备描述信息，可由
-------------------------	--------------

[out] <b>CamMac</b>	<b>CameraEnumerateDevice</b> 函数获得。 相机MAC(注意：必须保证传入的缓冲区大于等于18字节)
[out] <b>EtMac</b>	网卡MAC(注意：必须保证传入的缓冲区大于等于18字节)

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraGigeSetIp()

MVSDK\_API **CameraSdkStatus**  
\_\_stdcall CameraGigeSetIp

```
( tSdkCameraDevInfo * pCameraInfo,
  char const *          Ip,
  char const *          SubMask,
  char const *          GateWay,
  BOOL                  bPersistent
)
```

设置GIGE相机的IP地址

## 参数

[in] <b>pCameraInfo</b>	相机的设备描述信息, 可由 <b>CameraEnumerateDevice</b> 函数获得。
[in] <b>Ip</b>	相机IP(如: 192.168.1.100)
[in] <b>SubMask</b>	相机子网掩码(如: 255.255.255.0)
[in] <b>GateWay</b>	相机网关(如: 192.168.1.1)
[in] <b>bPersistent</b>	TRUE: 设置相机为固定IP, FALSE: 设置相机自动分配IP (忽略参数Ip, SubMask, GateWay)

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetCrossLine()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetCrossLine

( **CameraHandle** hCamera,

```
int          iLine,
INT         x,
INT         y,
UINT        uColor,
BOOL        bVisible
)
```

设置指定十字线的参数。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iLine** 表示要设置第几条十字线的状态。范围为[0,8], 共9条。
- [in] **x** 十字线中心位置的横坐标值。
- [in] **y** 十字线中心位置的纵坐标值。
- [in] **uColor** 十字线的颜色, 格式为(R|(G<<8)|(B<<16))
- [in] **bVisible** 十字线的显示状态。TRUE, 表示显示。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### 注解

只有设置为显示状态的十字线, 在调用[CameralImageOverlay](#)后才会被叠加到图像上。

# MVSDK API

## 镜像旋转

API

函数

### 函数

MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetMirror</b> <b>(CameraHandle hCamera,</b> int iDir, BOOL bEnable) 设置图像镜像操作。镜像操作分为水平和垂直两个方向。 <a href="#">更多...</a>
-------------------------------------	---

MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetMirror</b> <b>(CameraHandle hCamera,</b> int iDir, BOOL *pbEnable) 获得图像的镜像状态。 <a href="#">更多...</a>
-------------------------------------	--

MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetHardwareMirror</b> <b>(CameraHandle hCamera,</b> int iDir, BOOL bEnable) 设置硬件镜像。分为水平和垂直两个方向。(仅部分网口、U3相机支持此功能) <a href="#">更多...</a>
-------------------------------------	---

MVSDK_API CameraSdkStatus __stdcall	<b>CameraGetHardwareMirror</b> <b>(CameraHandle hCamera,</b> int iDir, BOOL *pbEnable) 获取设置的硬件镜像状态。 <a href="#">更多...</a>
-------------------------------------	---

MVSDK_API CameraSdkStatus __stdcall	<b>CameraSetRotate</b> <b>(CameraHandle hCamera,</b> int iRot)
-------------------------------------	--

设置图像旋转操作 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetRotate**  
(**CameraHandle** hCamera,  
int \*iRot)  
获得图像的旋转状态。 [更多...](#)

## 详细描述

## 函数说明

### ◆ CameraGetHardwareMirror()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetHardwareMirror( CameraHandle hCamera,  
                        int iDir,  
                        BOOL * pbEnable  
)
```

获取设置的硬件镜像状态。

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iDir** 表示要获得的镜像方向。0，表示水平方向；  
1，表示垂直方向。
- [out] **pbEnable** 指针，返回TRUE，则表示iDir所指的方向镜像被使能。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetMirror()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetMirror( CameraHandle hCamera,  
                  int iDir,
```

```
    BOOL * pbEnable  
)
```

获得图像的镜像状态。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iDir** 表示要获得的镜像方向。0, 表示水平方向;  
1, 表示垂直方向。
- [out] **pbEnable** 指针, 返回TRUE, 则表示iDir所指的方向镜像被使能。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraGetRotate()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetRotate( CameraHandle hCamera,  
                  int * iRot  
)
```

获得图像的旋转状态。

### 参数

- [in] **hCamera** 相机的句柄。
- [out] **iRot** 表示要获得的旋转方向。 (逆时针方向) (0:  
不旋转 1:90度 2:180度 3:270度)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetHardwareMirror()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetHardwareMirror( CameraHandle hCamera,  
                        int iDir,  
                        BOOL bEnable  
)
```

设置硬件镜像。分为水平和垂直两个方向。（仅部分网口、U3相机支持此功能）

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iDir** 表示镜像的方向。0，表示水平方向；1，表示垂直方向。
- [in] **bEnable** TRUE，使能镜像;FALSE，禁止镜像

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetMirror()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetMirror( CameraHandle hCamera,  
                  int iDir,  
                  BOOL bEnable  
)
```

设置图像镜像操作。镜像操作分为水平和垂直两个方向。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iDir** 表示镜像的方向。0，表示水平方向；1，表示垂

直方向。

[in] **bEnable** TRUE, 使能镜像;FALSE, 禁止镜像

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetRotate()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetRotate( CameraHandle hCamera,  
                  int          iRot  
                )
```

设置图像旋转操作

## 参数

[in] **hCamera** 相机的句柄。

[in] **iRot** 表示旋转的角度 (逆时针方向) (0: 不旋转 1:90 度 2:180度 3:270度)

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

# MVSDK API

## 图片保存

API

函数

### 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSaveImage**  
**(CameraHandle**  
hCamera, char  
\*lpszFileName, BYTE  
\*pbImageBuffer,  
**tSdkFrameHead**  
\*pFrInfo, UINT  
byFileType, BYTE  
byQuality)  
将图像缓冲区的数据保  
存成图片文件。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSaveImageEx**  
**(CameraHandle**  
hCamera, char  
\*lpszFileName, BYTE  
\*pbImageBuffer, UINT  
ulImageFormat, int  
iWidth, int iHeight,  
UINT byFileType,  
BYTE byQuality)  
将图像缓冲区的数据保  
存成图片文件。 [更多...](#)

## 详细描述

## 函数说明

### ◆ CameraSaveImage()

```
MVSDK_API  
CameraSdkStatus __stdcall  
CameraSaveImage ( CameraHandle hCamera,  
                  char * lpszFileName,  
                  BYTE * pbyImageBuffer,  
                  tSdkFrameHead * pFrInfo,  
                  UINT byFileType,  
                  BYTE byQuality  
)
```

将图像缓冲区的数据保存成图片文件。

#### 参数

[in] <b>hCamera</b>	相机的句柄。
[in] <b>lpszFileName</b>	图片保存文件完整路径。
[in] <b>pbyImageBuffer</b>	图像的数据缓冲区。
[in] <b>pFrInfo</b>	图像的帧头信息。
[in] <b>byFileType</b>	图像保存的格式。取值范围参见 <b>emSdkFileType</b> 的定义。
[in] <b>byQuality</b>	图像保存的质量因子，仅当保存为 JPG格式时该参数有效，范围1到 100。其余格式可以写成0。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## 注解

目前支持 BMP、JPG、PNG、RAW四种格式。其中RAW表示相机输出的原始数据, 保存RAW格式文件要求pbImageBuffer和pFrInfo是由**CameraGetImageBuffer**获得的数据, 而且未经**CameralImageProcess**转换成BMP格式; 反之, 如果要保存成BMP、JPG或者PNG格式, 则pbImageBuffer和pFrInfo是由**CameralImageProcess**处理后的RGB格式数据。具体用法可以参考Advanced的例程。

## 示例:

[grab.cpp](#), [grab\\_ex.cpp](#), [roi.cpp](#) , 以及 [soft\\_trigger.cpp](#).

## ◆ CameraSaveImageEx()

MVSDK\_API

**CameraSdkStatus \_\_stdcall**  
CameraSaveImageEx

```
( CameraHandle hCamera,  
    char *           lpszFileName,  
    BYTE *          pbImageBuffer,  
    UINT            ulImageFormat,  
    int             iWidth,  
    int             iHeight,  
    UINT            byFileType,  
    BYTE            byQuality  
)
```

将图像缓冲区的数据保存成图片文件。

## 参数

[in] **hCamera** 相机的句柄。

[in] <b>IpszFileName</b>	图片保存文件完整路径。
[in] <b>pbyImageBuffer</b>	图像的数据缓冲区。
[in] <b>ulImageFormat</b>	0:8 BIT gray 1:rgb24 2:rgba32 3:bgr24 4:bgra32
[in] <b>iWidth</b>	图片宽度
[in] <b>iHeight</b>	图片高度
[in] <b>byFileType</b>	图像保存的格式。取值范围参见 <b>emSdkFileType</b> 的定义。
[in] <b>byQuality</b>	图像保存的质量因子，仅当保存为 JPG格式时该参数有效，范围1到 100。其余格式可以写成0。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## 注解

与 [CameraSaveImage](#) 相同

# MVSDK API

## 录像功能

API

函数

### 函数

MVSDK_API CameraSdkStatus __stdcall	<b>CameralinitRecord</b> ( <b>CameraHandle</b> hCamera, int iFormat, char *pcSavePath, BOOL b2GLimit, DWORD dwQuality, int iFrameRate) 初始化一次录像。 <a href="#">更多...</a>
-------------------------------------	---

MVSDK_API CameraSdkStatus __stdcall	<b>CameraStopRecord</b> ( <b>CameraHandle</b> hCamera) 结束本次录像。当 <b>CameralinitRecord</b> 后，可 以通过该函数来结束一次 录像，并完成文件保存操 作。 <a href="#">更多...</a>
-------------------------------------	--

MVSDK_API CameraSdkStatus __stdcall	<b>CameraPushFrame</b> ( <b>CameraHandle</b> hCamera, BYTE *pbImageBuffer, <b>tSdkFrameHead</b> *pFrInfo) 将一帧数据存入录像流 中。由于我们的帧头信息 中携带了图像采集的时 间戳信息，因此录像可以精 准的时间同步，而不受帧 率不稳定的影响。 <a href="#">更多...</a>
-------------------------------------	--

## 详细描述

## 函数说明

### ◆ CameraInitRecord()

MVSDK\_API CameraSdkStatus  
\_\_stdcall CameraInitRecord

( CameraHandle hCamera,  
int iFormat,  
char \* pcSavePath,  
BOOL b2GLimit,  
DWORD dwQuality,  
int iFrameRate  
)

初始化一次录像。

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iFormat** 录像的格式 (0:不压缩 1:MSCV方式压缩 4:H264)
- [in] **pcSavePath** 录像文件保存的路径。
- [in] **b2GLimit** 如果为TRUE,则文件大于2G时自动分割。  
(功能未实现)
- [in] **dwQuality** 录像的质量因子, 越大, 则质量越好。范围1到100.
- [in] **iFrameRate** 录像的帧率。建议设定的比实际采集帧率大, 这样就不会漏帧。

[返回](#)

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraPushFrame()

```
MVSDK_API  
CameraSdkStatus __stdcall  
CameraPushFrame ( CameraHandle hCamera,  
                  BYTE * pbyImageBuffer,  
                  tSdkFrameHead * pFrInfo  
                )
```

将一帧数据存入录像流中。由于我们的帧头信息中携带了图像采集的时间戳信息, 因此录像可以精准的时间同步, 而不受帧率不稳定的影响。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **pbyImageBuffer** 图像的数据缓冲区。
- [in] **pFrInfo** 图像的帧头信息。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraStopRecord()

```
MVSDK_API CameraSdkStatus  
__stdcall CameraStopRecord ( CameraHandle hCamera )
```

结束本次录像。当**CameralInitRecord**后, 可以通过该函数来结束一次录像, 并完成文件保存操作。

### 参数

[in] **hCamera** 相机的句柄。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

# MVSDK API

## 用户数据

API

函数

### 函数

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraWriteSN</a> <a href="#">(CameraHandle</a> hCamera, BYTE *pbySN, INT iLevel)	设置相机的序列号。 更 多...
-------------------------------------	--	---------------------

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraReadSN</a> <a href="#">(CameraHandle</a> hCamera, BYTE *pbySN, INT iLevel)	读取相机指定级别的序列 号。 更多...
-------------------------------------	---	-------------------------

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraSaveUserData</a> <a href="#">(CameraHandle</a> hCamera, UINT uStartAddr, BYTE *pbData, int ilen)	将用户自定义的数据保存 到相机的非易性存储器 中。 更多...
-------------------------------------	--	---------------------------------------

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraLoadUserData</a> <a href="#">(CameraHandle</a> hCamera, UINT uStartAddr, BYTE *pbData, int ilen)	从相机的非易性存储器中
-------------------------------------	--	-------------

读取用户自定义的数据。

[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall [\*\*CameraGetFriendlyName\*\*](#)  
**(CameraHandle**  
hCamera, char \*pName)  
读取用户自定义的设备昵称。[更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall [\*\*CameraSetFriendlyName\*\*](#)  
**(CameraHandle**  
hCamera, char \*pName)  
设置用户自定义的设备昵称。[更多...](#)

## 详细描述

## 函数说明

### ◆ CameraGetFriendlyName()

MVSDK\_API CameraSdkStatus

\_stdcall CameraGetFriendlyName

( CameraHandle hCamera,  
char \* pName  
)

读取用户自定义的设备昵称。

#### 参数

[in] **hCamera** 相机的句柄。

[out] **pName** 指针，返回指向0结尾的字符串，设备昵称不超过32个字节，因此该指针指向的缓冲区必须大于等于32个字节空间。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraLoadUserData()

MVSDK\_API CameraSdkStatus

\_stdcall CameraLoadUserData

( CameraHandle hCamera,  
UINT uStartAddr,  
BYTE \* pbData,

```
    int         ilen  
    )
```

从相机的非易性存储器中读取用户自定义的数据。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **uStartAddr** 起始地址，从0开始。
- [out] **pbData** 数据缓冲区指针
- [in] **ilen** 数据的长度，**ilen + uStartAddr**必须小于用户区最大长度

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraReadSN()

### MVSDK\_API CameraSdkStatus

```
_stdcall CameraReadSN
```

```
( CameraHandle hCamera,  
    BYTE *         pbySN,  
    INT            iLevel  
    )
```

读取相机指定级别的序列号。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **pbySN** 序列号的缓冲区。
- [in] **iLevel** 要读取的序列号级别。可以为0、1和2。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

参见

[CameraWriteSN](#)

## ◆ CameraSaveUserData()

MVSDK\_API **CameraSdkStatus**  
\_\_stdcall CameraSaveUserData

( **CameraHandle** hCamera,  
    **UINT**                  uStartAddr,  
    **BYTE \***              pbData,  
    **int**                  ilen  
)

将用户自定义的数据保存到相机的非易性存储器中。

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **uStartAddr** 起始地址，从0开始。
- [in] **pbData** 数据缓冲区指针
- [in] **ilen** 写入数据的长度，ilen + uStartAddr必须小于用户区最大长度

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

### 注解

每个型号的相机可能支持的用户数据区最大长度不一样。可以从设备的特性描述中获取该长度信息。

## ◆ CameraSetFriendlyName()

MVSDK\_API **CameraSdkStatus**  
\_\_stdcall CameraSetFriendlyName

( **CameraHandle** hCamera,

```
    char *          pName  
)
```

设置用户自定义的设备昵称。

### 参数

[in] **hCamera** 相机的句柄。

[in] **pName** 指针，指向0结尾的字符串，设备昵称不超过32个字节，因此该指针指向字符串必须小于等于32个字节空间。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraWriteSN()

MVSDK\_API **CameraSdkStatus**

```
_stdcall CameraWriteSN
```

```
( CameraHandle hCamera,  
    BYTE *          pbySN,  
    INT             iLevel  
)
```

设置相机的序列号。

### 参数

[in] **hCamera** 相机的句柄。

[in] **pbySN** 序列号的缓冲区。

[in] **iLevel** 要设定的序列号级别，只能是1或者2。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### 注解

我公司相机序列号分为3级。0级的是我公司自定义的相机序列号，出厂时已经设定好且无法修改，1级和2级留给二次开发使用。每级序列号长度都是32个字节。

# MVSDK API

## 参数加载

函数

API

### 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetParameterMode**  
(**CameraHandle** hCamera, int iMode)  
设定参数存取的目标对象。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetParameterMode**  
(**CameraHandle** hCamera, int  
\*piTarget)  
获取参数存取的目标对象。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetParameterMask**  
(**CameraHandle** hCamera, UINT  
uMask)  
设置参数存取的掩码。参数加载和保  
存时会根据该掩码来决定各个模块参  
数的是否加载或者保存。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSaveParameter**  
(**CameraHandle** hCamera, int iTeam)  
保存当前相机参数到指定的参数组  
中。相机提供了A,B,C,D四组空间来  
进行参数的保存。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSaveParameterToFile**  
(**CameraHandle** hCamera, char  
\*sFileName)  
保存当前相机参数到指定的文件中。  
该文件可以复制到别的电脑上供其他  
相机加载，也可以做参数备份用。 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraReadParameterFromFile**  
(**CameraHandle** hCamera, char  
\*sFileName)  
从PC上指定的参数文件中加载参数。

我公司相机参数保存在PC上为.config  
后缀的文件，位于安装下的  
Camera\Configs文件夹中。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraLoadParameter**  
(**CameraHandle** hCamera, int iTeam)  
加载指定组的参数到相机中。 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetCurrentParameterGroup**  
(**CameraHandle** hCamera, int  
\*piTeam)  
获得当前选择的参数组。 [更多...](#)

## 详细描述

### 函数说明

#### ◆ CameraGetCurrentParameterGroup()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetCurrentParameterGroup( CameraHandle hCamera,  
                                int * piTeam  
                               )
```

获得当前选择的参数组。

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **piTeam** 指针，返回当前选择的参数组。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

#### ◆ CameraGetParameterMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetParameterMode( CameraHandle hCamera,  
                        int * piTarget  
                       )
```

获取参数存取的目标对象。

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **piTarget** 返回参数存取的对象。参考**emSdkParameterMode**

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraLoadParameter()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraLoadParameter( CameraHandle hCamera,  
                      int iTeam  
                    )
```

加载指定组的参数到相机中。

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iTeam** 参数组, 参考[emSdkParameterTeam](#)

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraReadParameterFromFile()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraReadParameterFromFile( CameraHandle hCamera,  
                            char * sFileName  
                          )
```

从PC上指定的参数文件中加载参数。我公司相机参数保存在PC上为.config后缀的文件, 位于安装下的Camera\Configs文件夹中。

### 参数

[in] **hCamera** 相机的句柄。  
[in] **sFileName** 参数文件的完整路径。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSaveParameter()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSaveParameter

( **CameraHandle** hCamera,  
int iTeam  
)

保存当前相机参数到指定的参数组中。相机提供了A,B,C,D四组空间来进行参数的保存。

### 参数

[in] **hCamera** 相机的句柄。  
[in] **iTeam** 参数组，参考[emSdkParameterTeam](#)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSaveParameterToFile()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSaveParameterToFile

( **CameraHandle** hCamera,  
char \* sFileName  
)

保存当前相机参数到指定的文件中。该文件可以复制到别的电脑上供其他相机加载，也可以做参数备份用。

### 参数

[in] **hCamera** 相机的句柄。  
[in] **sFileName** 参数文件的完整路径。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetParameterMask()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetParameterMask  
    ( CameraHandle hCamera,  
        UINT          uMask  
    )
```

设置参数存取的掩码。参数加载和保存时会根据该掩码来决定各个模块参数的是否加载或者保存。

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **uMask** 掩码。参考[emSdkPropSheetMask](#)

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraSetParameterMode()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraSetParameterMode  
    ( CameraHandle hCamera,  
        int           iMode  
    )
```

设定参数存取的目标对象。

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iMode** 参数存取的对象。参考[emSdkParameterMode](#)

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

# MVSDK API

类型定义 | 函数

## 掉线重连

API

### 类型定义

```
typedef void(WINAPI * CAMERA_CONNECTION_STATUS_CALLBACK) (CameraHandle  
hCamera, UINT MSG, UINT uParam, PVOID pContext)
```

### 函数

**MVSDK\_API CameraSdkStatus \_\_stdcall CameraReConnect (CameraHandle hCamera)**  
重新连接设备，用于连接恢复后手动重连 [更多...](#)

**MVSDK\_API CameraSdkStatus \_\_stdcall CameraConnectTest (CameraHandle hCamera)**  
测试相机的连接状态，用于检测相机是否掉线 [更多...](#)

**MVSDK\_API CameraSdkStatus \_\_stdcall CameraSetAutoConnect (CameraHandle hCamera,  
BOOL bEnable)**  
启用或禁用自动重连，默認為启用。 [更多...](#)

**MVSDK\_API CameraSdkStatus \_\_stdcall CameraGetAutoConnect (CameraHandle hCamera,  
BOOL \*pbEnable)**  
获取自动重连使能状态 [更多...](#)

**MVSDK\_API CameraSdkStatus \_\_stdcall CameraGetReConnectCounts (CameraHandle  
hCamera, **UINT** \*puCounts)**  
获得相机自动重连的次数，前提是  
**CameraSetAutoConnect**使能相机自动重连功能。默认是使能的。 [更多...](#)

**MVSDK\_API CameraSdkStatus \_\_stdcall CameraSetConnectionStatusCallback  
(CameraHandle hCamera,  
**CAMERA\_CONNECTION\_STATUS\_CALLBACK**  
pCallBack, **PVOID** pContext)**  
设置相机连接状态改变的回调通知函数。当相机掉线、重连时，pCallBack所指向的回调函数就会被调用。 [更多...](#)

## 详细描述

## 类型定义说明

### ◆ CAMERA\_CONNECTION\_STATUS\_CALLBACK

```
typedef void(WINAPI* CAMERA_CONNECTION_STATUS_CALLBACK) (CameraHandle hCamera,  
UINT MSG, UINT uParam, PVOID pContext)
```

相机连接状态回调

#### 参数

[in] **hCamera** 相机句柄  
[in] **MSG** 消息, 0: 相机连接断开 1: 相机连接恢复  
[in] **uParam** 附加信息  
[in] **pContext** 用户数据

#### 返回

无

#### 注解

USB相机uParam取值:

未定义

网口相机uParam取值:

当MSG=0时: 未定义

当MSG=1时:

0: 上次掉线原因, 网络通讯失败

1: 上次掉线原因, 相机掉电

## 函数说明

- ◆ CameraConnectTest()

**MVSDK\_API CameraSdkStatus \_\_stdcall CameraConnectTest ( CameraHandle hCamera )**

测试相机的连接状态，用于检测相机是否掉线

参数

[in] **hCamera** 相机的句柄。

返回

连接正常返回CAMERA STATUS SUCCESS(0)。否则表示已掉线

- ◆ CameraGetAutoConnect()

```
MVSDK_API CameraSdkStatus __stdcall CameraGetAutoConnect ( CameraHandle hCamera,  
                                         BOOL *          pbEnable  
                                         )
```

### 获取自动重连使能状态

参数

[in] **hCamera** 相机的句柄。

[out] **pbEnable** 返回相机自动重连使能状态

返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

- ◆ CameraGetReConnectCounts()

**MVSDK\_API CameraSdkStatus \_\_stdcall CameraGetReConnectCounts**

```
( CameraHandle hCamera,  
    UINT * puCounts  
)
```

获得相机自动重连的次数，前提是**CameraSetAutoConnect**使能相机自动重连功能。默认是使能的。

参数

[ɪn] **hCamera** 相机的句柄。

[out] **puCounts** 返回自动重连的次数

返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

- ◆ CameraReConnect()

**MVSDK\_API CameraSdkStatus \_\_stdcall CameraReConnect ( CameraHandle hCamera )**

重新连接设备，用于连接恢复后手动重连

参数

[in] **hCamera** 相机的句柄。

返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

警告

相机默认使能了自动重连，在自动重连模式中请勿调用本函数。

参见

## CameraSetAutoConnect

- ◆ CameraSetAutoConnect()

启用或禁用自动重连，默认为启用。

参数

[in] **hCamera** 相机的句柄。

[in] **bEnable** 使能相机重连，当位TRUE时，SDK内部自动检测相机是否掉线，掉线后自己重连。

返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

- ◆ CameraSetConnectionStatusCallback()

## MVSDK\_API CameraSdkStatus stdcall

设置相机连接状态改变的回调通知函数。当相机掉线、重连时，pCallBack所指向的回调函数就会被调用。

#### 参数

- [in] **hCamera** 相机的句柄。
- [in] **pCallBack** 回调函数指针。
- [in] **pContext** 回调函数的附加参数，在回调函数被调用时该附加参数会被传入，可以为NULL。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

# MVSDK API

## 彩点修正

API

函数

### 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraSetCorrectDeadPixel**  
(CameraHandle hCamera, BOOL bEnable)  
使能坏点修正 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGetCorrectDeadPixel**  
(CameraHandle hCamera, BOOL \*pbEnable)  
获取坏点修正使能状态 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraCustomizeDeadPixels**  
(CameraHandle hCamera, HWND hParent)  
打开坏点编辑面板 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraReadDeadPixels**  
(CameraHandle hCamera,  
USHORT \*pRows, USHORT  
\*pCols, UINT \*pNumPixel)  
读取相机坏点 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraAddDeadPixels**  
(CameraHandle hCamera,  
USHORT \*pRows, USHORT  
\*pCols, UINT NumPixel)  
添加相机坏点 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraRemoveDeadPixels**  
(CameraHandle hCamera,  
USHORT \*pRows, USHORT  
\*pCols, UINT NumPixel)  
删除相机指定坏点 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall CameraRemoveAllDeadPixels  
(CameraHandle hCamera)  
删除相机的所有坏点 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall CameraSaveDeadPixels  
(CameraHandle hCamera)  
保存相机坏点到相机存储中 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall CameraSaveDeadPixelsToFile  
(CameraHandle hCamera, char const \*sFileName)  
保存相机坏点到文件中 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall CameraLoadDeadPixelsFromFile  
(CameraHandle hCamera, char const \*sFileName)  
从文件加载相机坏点 [更多...](#)

---

## 详细描述

## 函数说明

### ◆ CameraAddDeadPixels()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraAddDeadPixels

( **CameraHandle** hCamera,  
USHORT \* pRows,  
USHORT \* pCols,  
UINT NumPixel  
)

添加相机坏点

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **pRows** 坏点y坐标  
[in] **pCols** 坏点x坐标  
[in] **NumPixel** 行列缓冲区中的坏点个数

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，  
请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraCustomizeDeadPixels()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraCustomizeDeadPixels

( **CameraHandle** hCamera,  
HWND hParent  
)

打开坏点编辑面板

#### 参数

[in] **hCamera** 相机的句柄。

[in] **hParent** 调用该函数的窗口的句柄。可以为NULL。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码,  
请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGetCorrectDeadPixel()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGetCorrectDeadPixel

( **CameraHandle** hCamera,  
  BOOL \*               pbEnable  
)

获取坏点修正使能状态

#### 参数

[in] **hCamera** 相机的句柄。

[out] **pbEnable** 返回使能状态

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码,  
请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraLoadDeadPixelsFromFile()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraLoadDeadPixelsFromFile

( **CameraHandle** hCamera,  
  char const \*        sFileName  
)

从文件加载相机坏点

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **sFileName** 文件的完整路径。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码,  
请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraReadDeadPixels()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraReadDeadPixels ( **CameraHandle** hCamera,  
                          USHORT \*          pRows,  
                          USHORT \*          pCols,  
                          UINT \*           pNumPixel  
                          )

读取相机坏点

#### 参数

[in] **hCamera** 相机的句柄。  
[out] **pRows** 坏点y坐标  
[out] **pCols** 坏点x坐标  
[out] **pNumPixel** 输入时表示行列缓冲区的大小，返回时表示行列缓冲区中返回的坏点数量。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码,  
请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

当pRows或者pCols为NULL时函数会把相机当前的坏点个数通过pNumPixel返回

### ◆ CameraRemoveAllDeadPixels()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraRemoveAllDeadPixels ( **CameraHandle** hCamera )

删除相机的所有坏点

### 参数

[in] **hCamera** 相机的句柄。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码,  
请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraRemoveDeadPixels()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraRemoveDeadPixels

( **CameraHandle** hCamera,  
USHORT \* pRows,  
USHORT \* pCols,  
UINT NumPixel  
)

删除相机指定坏点

### 参数

[in] **hCamera** 相机的句柄。  
[in] **pRows** 坏点y坐标  
[in] **pCols** 坏点x坐标  
[in] **NumPixel** 行列缓冲区中的坏点个数

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码,  
请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSaveDeadPixels()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSaveDeadPixels

( **CameraHandle** hCamera )

保存相机坏点到相机存储中

## 参数

[in] **hCamera** 相机的句柄。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码,  
请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSaveDeadPixelsToFile()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSaveDeadPixelsToFile

( **CameraHandle** hCamera,  
char const \* sFileName  
)

保存相机坏点到文件中

## 参数

[in] **hCamera** 相机的句柄。

[in] **sFileName** 文件的完整路径。

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码,  
请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraSetCorrectDeadPixel()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraSetCorrectDeadPixel

( **CameraHandle** hCamera,  
BOOL bEnable  
)

使能坏点修正

## 参数

[in] **hCamera** 相机的句柄。

[in] **bEnable** TRUE: 使能坏点修正 FALSE: 关闭坏点修正

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，  
请参考 [CameraStatus.h](#) 中错误码的定义。

# MVSDK API

## 镜头校正 API

函数

### 函数

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraSetUndistortParams</a> ( <b>CameraHandle</b> hCamera, int width, int height, double cameraMatrix[4], double distCoeffs[5])	设置校正参数 <a href="#">更多...</a>
-------------------------------------	---	------------------------------

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraGetUndistortParams</a> ( <b>CameraHandle</b> hCamera, int *width, int *height, double cameraMatrix[4], double distCoeffs[5])	获取校正参数 <a href="#">更多...</a>
-------------------------------------	---	------------------------------

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraSetUndistortEnable</a> ( <b>CameraHandle</b> hCamera, BOOL bEnable)	使能镜头校正 <a href="#">更多...</a>
-------------------------------------	--	------------------------------

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraGetUndistortEnable</a> ( <b>CameraHandle</b> hCamera, BOOL *bEnable)	获取镜头校正使能状态 <a href="#">更多...</a>
-------------------------------------	---	----------------------------------

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraCustomizeUndistort</a> ( <b>CameraHandle</b> hCamera, HWND hParent)	打开校正编辑面板 <a href="#">更多...</a>
-------------------------------------	--	--------------------------------

## 详细描述

## 函数说明

### ◆ CameraCustomizeUndistort()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraCustomizeUndistort( CameraHandle hCamera,  
                           HWND           hParent  
                         )
```

打开校正编辑面板

#### 参数

[in] **hCamera** 相机的句柄。  
[in] **hParent** 调用该函数的窗口的句柄。可以为NULL。

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGetUndistortEnable()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetUndistortEnable( CameraHandle hCamera,  
                           BOOL *        bEnable  
                         )
```

获取镜头校正使能状态

## 参数

- [in] **hCamera** 相机的句柄。
- [out] **bEnable** 使能校正

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraGetUndistortParams()

MVSDK\_API **CameraSdkStatus**

\_\_stdcall

CameraGetUndistortParams

( **CameraHandle** hCamera,  
    int \*                       width,  
    int \*                       height,  
    double                      cameraMatrix[4],  
    double                      distCoeffs[5]  
)

## 获取校正参数

### 参数

- [in] **hCamera**      相机的句柄。
- [out] **width**        图片宽度
- [out] **height**       图片高度
- [out] **cameraMatrix** 内参(fx, fy, cx, cy)
- [out] **distCoeffs**   畸变系数(k1,k2,p1,p2,k3)

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetUndistortEnable()

MVSDK\_API **CameraSdkStatus** \_\_stdcall ( **CameraHandle** hCamera,

## CameraSetUndistortEnable

```
    BOOL bEnable  
)
```

使能镜头校正

### 参数

[in] **hCamera** 相机的句柄。  
[in] **bEnable** 使能校正

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraSetUndistortParams()

### MVSDK\_API **CameraSdkStatus**

```
_stdcall  
CameraSetUndistortParams ( CameraHandle hCamera,  
                           int width,  
                           int height,  
                           double cameraMatrix[4],  
                           double distCoeffs[5]  
)
```

设置校正参数

### 参数

[in] **hCamera** 相机的句柄。  
[in] **width** 图片宽度  
[in] **height** 图片高度  
[in] **cameraMatrix** 内参(fx, fy, cx, cy)  
[in] **distCoeffs** 畸变系数(k1,k2,p1,p2,k3)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

# MVSDK API

## 多目相关

API

函数

### 函数

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGetEyeCount**  
(**CameraHandle** hCamera, int  
\*EyeCount)  
获取多目相机的目数 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraMultiEyeImageProcess**  
(**CameraHandle** hCamera, int  
iEyeIndex, BYTE \*pbyIn,  
**tSdkFrameHead** \*pInFrInfo,  
BYTE \*pbyOut,  
**tSdkFrameHead** \*pOutFrInfo,  
UINT uOutFormat, UINT  
uReserved)  
对多目相机帧内的某个单目图做  
ISP [更多...](#)

## 详细描述

## 函数说明

- ◆ CameraGetEyeCount()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGetEyeCount( CameraHandle hCamera,  
                    int *           EyeCount  
                  )
```

### 获取多目相机的目数

参数

[in] **hCamera** 相机的句柄。

[out] **EyeCount** 目数

返  
回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

- ◆ CameraMultiEyeImageProcess()

	UINT	uReserved
)		

对多目相机帧内的某个单目图做ISP

### 参数

- [in] **hCamera** 相机的句柄。
- [in] **iEyeIndex** 单目索引。
- [in] **pbyIn** 输入图像数据的缓冲区地址，不能为NULL。
- [in] **pInFrInfo** 输入图像数据的帧头，不能为NULL。
- [out] **pbyOut** 处理后图像输出的缓冲区地址，不能为NULL。
- [out] **pOutFrInfo** 处理后图像的帧头信息，不能为NULL。
- [in] **uOutFormat** 处理完后图像的输出格式。
- [in] **uReserved** 预留参数，必须设置为0。

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

# MVSDK API

## 缩放显示

API

函数

### 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraZoomTool\_Create** (void \*\*ZoomTool)  
创建缩放工具 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraZoomTool\_Destroy** (void \*ZoomTool)  
销毁缩放工具 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraZoomTool\_SetViewOrg** (void \*ZoomTool, float x, float y)  
设置视口原点 (左上角坐标) 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraZoomTool\_GetViewOrg** (void \*ZoomTool, float \*x, float \*y)  
获取视口原点 (左上角坐标) 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraZoomTool\_SetViewCenter** (void \*ZoomTool, float x, float y)  
设置视口中心点坐标 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraZoomTool\_GetViewCenter** (void \*ZoomTool, float \*x, float \*y)  
获取视口中心点坐标 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraZoomTool\_SetViewSize** (void \*ZoomTool, int w, int h)  
设置视口大小 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraZoomTool\_GetViewSize** (void \*ZoomTool, int \*w, int \*h)  
获取视口大小 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraZoomTool\_SetViewScrollPos** (void \*ZoomTool, float xPos, float yPos)  
设置视口的滚动位置 更多...

MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraZoomTool_GetViewScrollPos</a> (void *ZoomTool, float *xPos, float *yPos) 获取视口的滚动位置 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraZoomTool_GetViewScrollRange</a> (void *ZoomTool, float *xRange, float *yRange) 获取视口的滚动范围 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraZoomTool_SetFrameSize</a> (void *ZoomTool, int w, int h) 设置图片帧的大小 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraZoomTool_GetFrameSize</a> (void *ZoomTool, int *w, int *h) 获取图片帧的大小 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraZoomTool_SetScale</a> (void *ZoomTool, float ratio) 设置缩放比例 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraZoomTool_GetScale</a> (void *ZoomTool, float *ratio) 获取缩放比例 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraZoomTool_SetScaleAnchor</a> (void *ZoomTool, float xAnchor, float yAnchor) 设置缩放参考点 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraZoomTool_GetScaleAnchor</a> (void *ZoomTool, float *xAnchor, float *yAnchor) 获取缩放参考点 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraZoomTool_GetDrawRect</a> (void *ZoomTool, int *FrameX, int *FrameY, int *FrameW, int *FrameH, int *ViewX, int *ViewY, int *ViewW, int *ViewH) 获取绘制矩形 <a href="#">更多...</a>
MVSDK_API CameraSdkStatus __stdcall	<a href="#">CameraZoomTool_Transform</a> (void *ZoomTool, int Type, float *PointX, float *PointY)

坐标变换 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraZoomTool\_Draw** (void  
\*ZoomTool, int Algorithm, void  
\*pFrameBuffer, **tSdkFrameHead**  
\*pFrameHead, HWND hWnd, int xDst, int  
yDst, HBRUSH hBackBrush)  
把帧数据按设定的视口和比例绘制到窗口  
[更多...](#)

---

## 详细描述

### 函数说明

- ◆ CameraZoomTool\_Create()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraZoomTool\_Create ( void \*\* ZoomTool )

创建缩放工具

**参数**

[out] **ZoomTool** 返回新创建的缩放工具

**返回**

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

- ◆ CameraZoomTool\_Destory()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraZoomTool\_Destory ( void \* ZoomTool )

销毁缩放工具

**参数**

[in] **ZoomTool** 缩放工具

**返回**

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

- ◆ CameraZoomTool\_Draw()

MVSDK\_API **CameraSdkStatus** \_\_stdcall ( void \* ZoomTool,

## CameraZoomTool\_Draw

```
int          Algorithm,
void *      pFrameBuffer,
tSdkFrameHead * pFrameHead,
HWND        hWnd,
int          xDst,
int          yDst,
HBRUSH      hBackBrush
)
```

把帧数据按设定的视口和比例绘制到窗口

### 参数

[in] <b>ZoomTool</b>	缩放工具
[in] <b>Algorithm</b>	缩放算法 0: 快速但质量稍差 1: 速度慢但质量好
[in] <b>pFrameBuffer</b>	帧数据
[in] <b>pFrameHead</b>	帧头
[in] <b>hWnd</b>	目的窗口句柄
[in] <b>xDst</b>	目标矩形左上角X坐标
[in] <b>yDst</b>	目标矩形左上角Y坐标
[in] <b>hBackBrush</b>	背景画刷(NULL:不填充背景)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraZoomTool\_GetDrawRect()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraZoomTool\_GetDrawRect

```
( void * ZoomTool,
  int * FrameX,
  int * FrameY,
  int * FrameW,
  int * FrameH,
  int * ViewX,
  int * ViewY,
  int * ViewW,
  int * ViewH
)
```

获取绘制矩形

## 参数

[in]	<b>ZoomTool</b>	缩放工具
[out]	<b>FrameX</b>	帧内矩形X坐标
[out]	<b>FrameY</b>	帧内矩形Y坐标
[out]	<b>FrameW</b>	帧内矩形宽度
[out]	<b>FrameH</b>	帧内矩形高度
[out]	<b>ViewX</b>	视口内矩形X坐标
[out]	<b>ViewY</b>	视口内矩形Y坐标
[out]	<b>ViewW</b>	视口内矩形宽度
[out]	<b>ViewH</b>	视口内矩形高度

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraZoomTool\_GetFrameSize()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraZoomTool_GetFrameSize( void * ZoomTool,  
                           int * w,  
                           int * h  
                         )
```

获取图片帧的大小

## 参数

[in]	<b>ZoomTool</b>	缩放工具
[out]	<b>w</b>	帧宽度
[out]	<b>h</b>	帧高度

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameraZoomTool\_GetScale()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraZoomTool_GetScale( void * ZoomTool,  
                        float * ratio  
                      )
```

## 获取缩放比例

### 参数

[in] **ZoomTool** 缩放工具  
[out] **ratio** 缩放比例

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraZoomTool\_GetScaleAnchor()

```
MVSDK_API CameraSdkStatus __stdcall
CameraZoomTool_GetScaleAnchor(
    void * ZoomTool,
    float * xAnchor,
    float * yAnchor
)
```

## 获取缩放参考点

### 参数

[in] **ZoomTool** 缩放工具  
[out] **xAnchor** 水平参考点(0-1)  
[out] **yAnchor** 垂直参考点(0-1)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraZoomTool\_GetViewCenter()

```
MVSDK_API CameraSdkStatus __stdcall
CameraZoomTool_GetViewCenter(
    void * ZoomTool,
    float * x,
    float * y
)
```

## 获取视口中心点坐标

### 参数

[in] **ZoomTool** 缩放工具

[out] **x** 横坐标值  
[out] **y** 纵坐标值

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraZoomTool\_GetViewOrg()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraZoomTool\_GetViewOrg ( void \* **ZoomTool**,  
float \* **x**,  
float \* **y**  
)

获取视口原点 (左上角坐标)

#### 参数

[in] **ZoomTool** 缩放工具  
[out] **x** 横坐标值  
[out] **y** 纵坐标值

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraZoomTool\_GetViewScrollPos()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraZoomTool\_GetViewScrollPos ( void \* **ZoomTool**,  
float \* **xPos**,  
float \* **yPos**  
)

获取视口的滚动位置

#### 参数

[in] **ZoomTool** 缩放工具  
[out] **xPos** 横向值 (0-1) (<0表示横向可完全显示无需滚动)  
[out] **yPos** 纵向值 (0-1) (<0表示纵向可完全显示无需滚动)

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraZoomTool\_GetViewScrollRange()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraZoomTool_GetViewScrollRange  
    ( void * ZoomTool,  
      float * xRange,  
      float * yRange  
    )
```

获取视口的滚动范围

### 参数

[in] **ZoomTool** 缩放工具  
[out] **xRange** 横向滚动范围  
[out] **yRange** 纵向滚动范围

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraZoomTool\_GetViewSize()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraZoomTool_GetViewSize  
    ( void * ZoomTool,  
      int * w,  
      int * h  
    )
```

获取视口大小

### 参数

[in] **ZoomTool** 缩放工具  
[out] **w** 视口宽度  
[out] **h** 视口高度

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraZoomTool\_SetFrameSize()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraZoomTool_SetFrameSize( void * ZoomTool,  
                           int     w,  
                           int     h  
                         )
```

设置图片帧的大小

### 参数

[in] **ZoomTool** 缩放工具  
[in] **w** 帧宽度  
[in] **h** 帧高度

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraZoomTool\_SetScale()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraZoomTool_SetScale( void * ZoomTool,  
                        float   ratio  
                      )
```

设置缩放比例

### 参数

[in] **ZoomTool** 缩放工具  
[in] **ratio** 缩放比例

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraZoomTool\_SetScaleAnchor()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraZoomTool_SetScaleAnchor  
(  
    void * ZoomTool,  
    float xAnchor,  
    float yAnchor  
)
```

设置缩放参考点

#### 参数

- [in] **ZoomTool** 缩放工具
- [in] **xAnchor** 水平参考点(0-1)
- [in] **yAnchor** 垂直参考点(0-1)

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraZoomTool\_SetViewCenter()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraZoomTool_SetViewCenter  
(  
    void * ZoomTool,  
    float x,  
    float y  
)
```

设置视口中心点坐标

#### 参数

- [in] **ZoomTool** 缩放工具
- [in] **x** 横坐标值
- [in] **y** 纵坐标值

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraZoomTool\_SetViewOrg()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraZoomTool_SetViewOrg  
(  
    void * ZoomTool,  
    float x,
```

```
    float y  
)
```

设置视口原点（左上角坐标）

#### 参数

[in] **ZoomTool** 缩放工具  
[in] **x** 横坐标值  
[in] **y** 纵坐标值

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraZoomTool\_SetViewScrollPos()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraZoomTool\_SetViewScrollPos

```
( void * ZoomTool,  
float xPos,  
float yPos  
)
```

设置视口的滚动位置

#### 参数

[in] **ZoomTool** 缩放工具  
[in] **xPos** 横向值 (0-1)  
[in] **yPos** 纵向值 (0-1)

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraZoomTool\_SetViewSize()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraZoomTool\_SetViewSize

```
( void * ZoomTool,  
int w,  
int h  
)
```

## 设置视口大小

### 参数

[in] **ZoomTool** 缩放工具  
[in] **w** 视口宽度  
[in] **h** 视口高度

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraZoomTool\_Transform()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraZoomTool\_Transform ( void \* ZoomTool,  
int Type,  
float \* PointX,  
float \* PointY  
)

## 坐标变换

### 参数

[in] **ZoomTool** 缩放工具  
[in] **Type** 变换类型 0: 视口转帧坐标 1: 视口转世界坐标(帧坐标\*  
缩放因子)  
[in,out] **PointX** X坐标 (返回变换后的X坐标)  
[in,out] **PointY** Y坐标 (返回变换后的Y坐标)

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

# MVSDK API

## 采集器

模块

模块

创建

控制

取图回调

抓图

销毁

## 详细描述

---

# MVSDK API

## 创建

采集器

函数

### 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_CreateFromDevicePage** (void \*\*Grabber)  
弹出相机列表让用户选择要打开的相机 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_CreateByIndex** (void \*\*Grabber, int Index)  
使用相机列表索引创建Grabber [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_CreateByName** (void \*\*Grabber, char \*Name)  
使用相机名称创建Grabber [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_Create** (void \*\*Grabber, tSdkCameraDevInfo \*pDevInfo)  
从设备信息创建Grabber [更多...](#)

## 详细描述

### 函数说明

#### ◆ CameraGrabber\_Create()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGrabber_Create( void ** Grabber,  
                      tSdkCameraDevInfo * pDevInfo  
                      )
```

从设备信息创建Grabber

#### 参数

[out] **Grabber** 返回新创建的采集器  
[in] **pDevInfo** 设备信息。[CameraEnumerateDevice](#)

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

本函数内部使用了 [CameralInit](#) 打开相机, 因此可以使用 [CameraGrabber\\_GetCameraHandle](#) 获取相机句柄, 进而使用其他SDK API来操作相机。

#### 示例:

[win\\_grabber.cpp](#).

#### ◆ CameraGrabber\_CreateByIndex()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGrabber_CreateByIndex( void ** Grabber,  
                           int Index  
                           )
```

使用相机列表索引创建Grabber

## 参数

[out] **Grabber** 返回新创建的采集器  
[in] **Index** 相机索引

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## 注解

本函数内部使用了 [CameraInit](#) 打开相机, 因此可以使用 [CameraGrabber\\_GetCameraHandle](#) 获取相机句柄, 进而使用其他SDK API来操作相机。

## ◆ CameraGrabber\_CreateByName()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGrabber_CreateByName( void ** Grabber,  
                           char * Name  
                         )
```

使用相机名称创建Grabber

## 参数

[out] **Grabber** 返回新创建的采集器  
[in] **Name** 相机名称。[tSdkCameraDevInfo.acFriendlyName](#)

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## 注解

本函数内部使用了 [CameraInit](#) 打开相机, 因此可以使用 [CameraGrabber\\_GetCameraHandle](#) 获取相机句柄, 进而使用其他SDK API来操作相机。

## ◆ CameraGrabber\_CreateFromDevicePage()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGrabber_CreateFromDevicePage( void ** Grabber )
```

弹出相机列表让用户选择要打开的相机

## 参数

[out] **Grabber** 返回新创建的采集器

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### 注解

本函数内部使用了 [Cameralinit](#) 打开相机, 因此可以使用 [CameraGrabber\\_GetCameraHandle](#) 获取相机句柄, 进而使用其他SDK API来操作相机。

# MVSDK API

## 控制

采集器

函数

## 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_SetHWnd** (void \*Grabber, HWND hWnd)  
设置预览视频的显示窗口 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_SetPriority** (void \*Grabber, UINT Priority)  
设置Grabber取图时使用的优先级 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_StartLive** (void \*Grabber)  
开始采集 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_StopLive** (void \*Grabber)  
停止采集 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_GetCameraHandle** (void \*Grabber, CameraHandle \*hCamera)  
获取相机句柄 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_GetStat** (void \*Grabber, tSdkGrabberStat \*stat)  
获取帧统计信息 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_GetCameraDevInfo** (void \*Grabber, tSdkCameraDevInfo \*DevInfo)  
获取相机DevInfo 更多...

## 详细描述

### 函数说明

#### ◆ CameraGrabber\_GetCameraDevInfo()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGrabber_GetCameraDevInfo( void * Grabber,  
                                tSdkCameraDevInfo * DevInfo  
                                )
```

获取相机DevInfo

#### 参数

[in] **Grabber** 采集器  
[out] **DevInfo** 返回的相机DevInfo

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

#### ◆ CameraGrabber\_GetCameraHandle()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGrabber_GetCameraHandle( void * Grabber,  
                               CameraHandle * hCamera  
                               )
```

获取相机句柄

#### 参数

[in] **Grabber** 采集器  
[out] **hCamera** 返回的相机句柄

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## 示例:

[win\\_grabber.cpp](#).

### ◆ CameraGrabber\_GetStat()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGrabber_GetStat( void * Grabber,  
                        tSdkGrabberStat * stat  
                      )
```

获取帧统计信息

## 参数

[in] **Grabber** 采集器  
[out] **stat** 返回的统计信息

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraGrabber\_SetHWnd()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGrabber_SetHWnd( void * Grabber,  
                       HWND hWnd  
                     )
```

设置预览视频的显示窗口

## 参数

[in] **Grabber** 采集器  
[in] **hWnd** 显示窗口的窗口句柄

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

示例：  
[win\\_grabber.cpp](#).

◆ CameraGrabber\_SetPriority()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGrabber_SetPriority( void * Grabber,  
                           UINT Priority  
                         )
```

设置Grabber取图时使用的优先级

参数

[in] **Grabber** 采集器  
[in] **Priority** 取图优先级 详见：[emCameraGetImagePriority](#)

返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

◆ CameraGrabber\_StartLive()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGrabber_StartLive( void * Grabber )
```

开始采集

参数

[in] **Grabber** 采集器

返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 [CameraStatus.h](#) 中错误码的定义。

注解

Grabber必须进入采集状态，采集回调、抓图等功能才能正常运作

示例：  
[win\\_grabber.cpp](#).

- ◆ CameraGrabber\_StopLive()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
CameraGrabber\_StopLive ( void \* Grabber )

停止采集

**参数**

[in] **Grabber** 采集器

**返回**

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

**警告**

本函数内部会等待所有回调函数结束后才返回调用者, 并且在等待时会派发 windows消息。

# MVSDK API

## 取图回调

采集器

类型定义 | 函数

### 类型定义

```
typedef void(__stdcall * pfnCameraGrabberFrameCallback) (void *Grabber,  
BYTE *pFrameBuffer, tSdkFrameHead *pFrameHead,  
void *Context)
```

```
typedef int(__stdcall * pfnCameraGrabberFrameListener) (void *Grabber, int  
Phase, BYTE *pFrameBuffer, tSdkFrameHead  
*pFrameHead, void *Context)
```

### 函数

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGrabber\_SetFrameListener**  
(void \*Grabber,  
**pfnCameraGrabberFrameListener**  
Listener, void \*Context)  
设置帧监听函数 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGrabber\_SetRawCallback**  
(void \*Grabber,  
**pfnCameraGrabberFrameCallback**  
Callback, void \*Context)  
设置RAW数据回调函数 [更多...](#)

MVSDK\_API **CameraSdkStatus** \_\_stdcall **CameraGrabber\_SetRGBCallback**  
(void \*Grabber,  
**pfnCameraGrabberFrameCallback**  
Callback, void \*Context)  
设置RGB回调函数 [更多...](#)

## 详细描述

### 类型定义说明

#### ◆ pfnCameraGrabberFrameCallback

```
typedef void(__stdcall * pfnCameraGrabberFrameCallback) (void *Grabber, BYTE  
*pFrameBuffer, tSdkFrameHead *pFrameHead, void *Context)
```

图像捕获的回调函数定义

#### ◆ pfnCameraGrabberFrameListener

```
typedef int(__stdcall * pfnCameraGrabberFrameListener) (void *Grabber, int  
Phase, BYTE *pFrameBuffer, tSdkFrameHead *pFrameHead, void *Context)
```

帧监听函数定义

#### 参数

[in] Grabber
[in] Phase 图像处理阶段
[in] pFrameBuffer 帧数据
[in] pFrameHead 帧头
[in] Context 用户数据

#### 返回

0:Grabber将会丢弃此帧并结束针对此帧的所有后续处理阶段 1:继续下一阶段  
处理

#### 注解

每当Grabber捕获到一帧图像时，会分3个阶段来依次调用FrameListener

阶段0: RAW数据处理， pFrameBuffer=Raw数据

阶段1: 截图前处理， pFrameBuffer=RGB数据

阶段2: 显示前处理， pFrameBuffer=RGB数据

特别的，当相机掉线后此回调也会被调用，此时Phase=-1，  
pFrameBuffer=NULL,pFrameHead=NULL。

## 函数说明

### ◆ CameraGrabber\_SetFrameListener()

MVSDK\_API **CameraSdkStatus**

stdcall

```
CameraGrabber_SetFrameListener ( void * Grabber,  
                                pfnCameraGrabberFrameListener Listener,  
                                void * Context  
                               )
```

设置帧监听函数

#### 参数

[in] **Grabber** 采集器

[in] **Listener** 监听函数

[in] **Context** 当Listener被调用时，作为参数传入Listener

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameraGrabber\_SetRawCallback()

MVSDK\_API **CameraSdkStatus**

stdcall

```
CameraGrabber_SetRawCallback ( void * Grabber,  
                               pfnCameraGrabberFrameCallback Callback,  
                               void * Context  
                              )
```

设置RAW数据回调函数

#### 参数

[in] **Grabber** 采集器

[in] **Callback** Raw回调函数

[in] **Context** 当Callback被调用时，作为参数传入Callback

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameraGrabber\_SetRGBCallback()

MVSDK\_API **CameraSdkStatus**

```
__stdcall  
CameraGrabber_SetRGBCallback ( void * Grabber,  
                                pfnCameraGrabberFrameCallback Callback,  
                                void * Context  
                                )
```

设置RGB回调函数

### 参数

[in] **Grabber** 采集器

[in] **Callback** RGB回调函数

[in] **Context** 当Callback被调用时, 作为参数传入Callback

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### 示例:

[win\\_grabber.cpp](#).

# MVSDK API

## 抓图

采集器

类型定义 | 函数

## 类型定义

```
typedef void(__stdcall * pfnCameraGrabberSavelImageComplete)(void *Grabber, void *Image,  
CameraSdkStatus Status, void *Context)
```

## 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_SavelImage** (void \*Grabber, void  
\*\*Image, DWORD TimeOut)  
同步抓图 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_SavelImageAsync** (void \*Grabber)  
提交一个异步的抓图请求，提交成功后待抓图完成会回调  
用户设置的完成函数 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_SavelImageAsyncEx** (void \*Grabber,  
void \*UserData)  
提交一个异步的抓图请求，提交成功后待抓图完成会回调  
用户设置的完成函数 [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_SetSavelImageCompleteCallback** (void  
\*Grabber, pfnCameraGrabberSavelImageComplete  
Callback, void \*Context)  
设置异步方式抓图的完成函数 [更多...](#)

## 详细描述

---

## 类型定义说明

---

- ◆ pfnCameraGrabberSaveImageComplete

```
typedef void(__stdcall * pfnCameraGrabberSaveImageComplete) (void *Grabber, void *Image,  
CameraSdkStatus Status, void *Context)
```

异步抓图的回调函数定义

**警告**

Image需要调用 **CameralImage\_Destroy** 释放

## 函数说明

- ◆ CameraGrabber\_SaveImage()

```
MVSDK_API CameraSdkStatus __stdcall CameraGrabber_SaveImage( void * Grabber,  
                                         void ** Image,  
                                         DWORD TimeOut  
                                     )
```

## 同步抓图

参数

[in] **Grabber** 采集器  
[out] **Image** 返回抓取到的图像

注解

需要调用**CameralImage\_Destroy**释放

参数

[in] **TimeOut** 超时时间 (毫秒)

[返回](#)

成功返回 `CAMERA_STATUS_SUCCESS(0)`。否则返回非0值的错误码, 请参考 `CameraStatus.h` 中错误码的定义。

- ◆ CameraGrabber\_SaveImageAsync()

MVSDK\_API CameraSdkStatus \_\_stdcall CameraGrabber\_SaveImageAsync( void \* Grabber )

提交一个异步的抓图请求，提交成功后待抓图完成会回调用户设置的完成函数

参数

[in] **Grabber** 采集器

返回

成功返回 `CAMERA_STATUS_SUCCESS(0)`。否则返回非0值的错误码, 请参考 `CameraStatus.h` 中错误码的定义。

参见

## CameraGrabber\_SetSaveImageCompleteCallback

- ◆ CameraGrabber\_SaveImageAsyncEx()

```
MVSDK_API CameraSdkStatus __stdcall CameraGrabber_SaveImageAsyncEx ( void * Grabber,  
                                         void * UserData  
                                         )
```

提交一个异步的抓图请求，提交成功后待抓图完成会回调用户设置的完成函数

#### 参数

[in] **Grabber** 采集器

[in] **UserData** 用户数据，可使用 **CameralImage\_GetUserData** 从Image获取此值

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

#### 参见

[CameraGrabber\\_SetSaveImageCompleteCallback](#)

### ◆ CameraGrabber\_SetSaveImageCompleteCallback()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGrabber_SetSaveImageCompleteCallback ( void * Grabber  
                                             pfnCameraGrabberSaveImageComplete Callbac  
                                             void * Context  
                                           )
```

设置异步方式抓图的完成函数

#### 参数

[in] **Grabber** 采集器

[in] **Callback** 当异步抓图任务完成时被调用

[in] **Context** 当Callback被调用时，作为参数传入Callback

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码，请参考 **CameraStatus.h** 中错误码的定义。

#### 参见

[CameraGrabber\\_SaveImageAsync](#) [CameraGrabber\\_SaveImageAsyncEx](#)

# MVSDK API

销毁  
采集器

函数

## 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameraGrabber\_Destroy**  
(void \*Grabber)  
销毁Grabber [更多...](#)

## 详细描述

---

## 函数说明

---

### ◆ CameraGrabber\_Destroy()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraGrabber_Destroy( void * Grabber )
```

销毁Grabber

#### 参数

[in] **Grabber** 采集器

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

#### 示例:

[win\\_grabber.cpp](#).

# MVSDK API

## Image

函数

### 函数

MVSDK\_API CameraSdkStatus \_\_stdcall **CameralImage\_Create** (void \*\*Image, BYTE \*pFrameBuffer, **tSdkFrameHead** \*pFrameHead, BOOL bCopy)  
创建一个新的Image [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameralImage\_CreateEmpty** (void \*\*Image)  
创建一个空的Image [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameralImage\_Destroy** (void \*Image)  
销毁Image [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameralImage\_GetData** (void \*Image, BYTE \*\*DataBuffer, **tSdkFrameHead** \*\*Head)  
从Image获取帧数据和帧头  
[更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameralImage\_GetUserData** (void \*Image, void \*\*UserData)  
获取Image的用户自定义数据  
[更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameralImage\_SetUserData** (void \*Image, void \*UserData)  
设置Image的用户自定义数据  
[更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall **CameralImage\_IsEmpty**  
(void \*Image, BOOL  
\*IsEmpty)  
判断一个Image是否为空 更  
多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameralImage\_Draw** (void  
\*Image, HWND hWnd, int  
Algorithm)  
绘制Image到指定窗口 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameralImage\_DrawFit** (void  
\*Image, HWND hWnd, int  
Algorithm)  
拉升绘制Image到指定窗口  
更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameralImage\_DrawToDC**  
(void \*Image, HDC hDC, int  
Algorithm, int xDst, int yDst,  
int cxDst, int cyDst)  
绘制Image到指定DC 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameralImage\_DrawToDCFit**  
(void \*Image, HDC hDC, int  
Algorithm, int xDst, int yDst,  
int cxDst, int cyDst)  
拉升绘制Image到指定DC 更  
多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameralImage\_BitBlt** (void  
\*Image, HWND hWnd, int  
xDst, int yDst, int cxDst, int  
cyDst, int xSrc, int ySrc)  
绘制Image到指定窗口（不缩  
放） 更多...

MVSDK\_API CameraSdkStatus \_\_stdcall **CameralImage\_BitBltToDC**

(void \*Image, HDC hDC, int xDst, int yDst, int cxDst, int cyDst, int xSrc, int ySrc)  
绘制Image到指定DC (不缩放) [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameralImage\\_SaveAsBmp](#)  
(void \*Image, char const \*FileName)  
以bmp格式保存Image [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameralImage\\_SaveAsJpeg](#)  
(void \*Image, char const \*FileName, BYTE Quality)  
以jpg格式保存Image [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameralImage\\_SaveAsPng](#)  
(void \*Image, char const \*FileName)  
以png格式保存Image [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameralImage\\_SaveAsRaw](#)  
(void \*Image, char const \*FileName, int Format)  
以raw格式保存Image [更多...](#)

MVSDK\_API CameraSdkStatus \_\_stdcall [CameralImage\\_IPicture](#) (void \*Image, IPicture \*\*NewPic)  
从Image创建一个IPicture [更多...](#)

## 详细描述

## 函数说明

### ◆ CameralImage\_BitBlt()

MVSDK\_API CameraSdkStatus \_\_stdcall  
CameralImage\_BitBlt

```
( void * Image,  
  HWND hWnd,  
  int xDst,  
  int yDst,  
  int cxDst,  
  int cyDst,  
  int xSrc,  
  int ySrc  
)
```

绘制Image到指定窗口 (不缩放)

#### 参数

- [in] **Image**
- [in] **hWnd** 目的窗口
- [in] **xDst** 目标矩形的左上角X坐标
- [in] **yDst** 目标矩形的左上角Y坐标
- [in] **cxDst** 目标矩形的宽度
- [in] **cyDst** 目标矩形的高度
- [in] **xSrc** 图像矩形的左上角X坐标
- [in] **ySrc** 图像矩形的左上角Y坐标

返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameralImage\_BitBltToDC()

MVSDK\_API **CameraSdkStatus** \_\_stdcall  
Cameralimage\_BitBltToDC

```
( void * Image,  
  HDC   hDC,  
  int    xDst,  
  int    yDst,  
  int    cxDst,  
  int    cyDst,  
  int    xSrc,  
  int    ySrc  
)
```

绘制Image到指定DC (不缩放)

### 参数

- [in] **Image**
- [in] **hDC** 目的DC
- [in] **xDst** 目标矩形的左上角X坐标
- [in] **yDst** 目标矩形的左上角Y坐标
- [in] **cxDst** 目标矩形的宽度
- [in] **cyDst** 目标矩形的高度
- [in] **xSrc** 图像矩形的左上角X坐标
- [in] **ySrc** 图像矩形的左上角Y坐标

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameralImage\_Create()

```
MVSDK_API CameraSdkStatus __stdcall CameralImage_Create( void ** Image,
                                                               BYTE * pFrameBuffer,
                                                               tSdkFrameHead * pFrameHead,
                                                               BOOL bCopy
)
```

创建一个新的Image

#### 参数

[out] <b>Image</b>	新创建的图片
[in] <b>pFrameBuffer</b>	帧数据
[in] <b>pFrameHead</b>	帧头
[in] <b>bCopy</b>	TRUE: 复制出一份新的帧数据 FALSE: 不复制, 直接使用pFrameBuffer指向的缓冲区

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameralImage\_CreateEmpty()

```
MVSDK_API CameraSdkStatus __stdcall
CameralImage_CreateEmpty( void ** Image )
```

创建一个空的Image

#### 参数

[out] <b>Image</b>	新创建的图片
--------------------	--------

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

## ◆ CameralImage\_Destroy()

```
MVSDK_API CameraSdkStatus __stdcall  
CameralImage_Destroy( void * Image )
```

销毁Image

### 参数

[in] **Image**

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameralImage\_Draw()

```
MVSDK_API CameraSdkStatus __stdcall  
CameralImage_Draw( void * Image,  
                    HWND hWnd,  
                    int Algorithm  
                )
```

绘制Image到指定窗口

### 参数

[in] **Image**

[in] **hWnd** 目的窗口

[in] **Algorithm** 缩放算法 0: 快速但质量稍差 1: 速度慢但质量好

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameralImage\_DrawFit()

```
MVSDK_API CameraSdkStatus __stdcall  
CameralImage_DrawFit( void * Image,  
                      HWND hWnd,  
                      int Algorithm  
)
```

拉升绘制Image到指定窗口

### 参数

[in] **Image**  
[in] **hWnd** 目的窗口  
[in] **Algorithm** 缩放算法 0: 快速但质量稍差 1: 速度慢但质量好

### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameralImage\_DrawToDC()

```
MVSDK_API CameraSdkStatus __stdcall  
CameralImage_DrawToDC( void * Image,  
                       HDC hDC,  
                       int Algorithm,  
                       int xDst,  
                       int yDst,  
                       int cxDst,  
                       int cyDst  
)
```

绘制Image到指定DC

### 参数

[in] **Image**  
[in] **hDC** 目的DC  
[in] **Algorithm** 缩放算法 0: 快速但质量稍差 1: 速度慢但质量好  
[in] **xDst** 目标矩形的左上角X坐标  
[in] **yDst** 目标矩形的左上角Y坐标  
[in] **cxDst** 目标矩形的宽度  
[in] **cyDst** 目标矩形的高度

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameralImage\_DrawToDCFit()

```
MVSDK_API CameraSdkStatus __stdcall
CameralImage_DrawToDCFit(
    void * Image,
    HDC hDC,
    int Algorithm,
    int xDst,
    int yDst,
    int cxDst,
    int cyDst
)
```

拉升绘制Image到指定DC

### 参数

[in] **Image**  
[in] **hDC** 目的DC  
[in] **Algorithm** 缩放算法 0: 快速但质量稍差 1: 速度慢但质量好  
[in] **xDst** 目标矩形的左上角X坐标  
[in] **yDst** 目标矩形的左上角Y坐标  
[in] **cxDst** 目标矩形的宽度  
[in] **cyDst** 目标矩形的高度

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameralImage\_GetData()

```
MVSDK_API CameraSdkStatus  
__stdcall CameralImage_GetData( void *  
                                BYTE **  
                                tSdkFrameHead ** Head  
)
```

从Image获取帧数据和帧头

#### 参数

[in] **Image**  
[out] **DataBuffer** 帧数据  
[out] **Head** 帧头

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameralImage\_GetUserData()

```
MVSDK_API CameraSdkStatus __stdcall  
CameralImage_GetUserData( void * Image,  
                           void ** UserData  
)
```

获取Image的用户自定义数据

#### 参数

[in] **Image**  
[out] **UserData** 返回用户自定义数据

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameralImage\_IPicture()

```
MVSDK_API CameraSdkStatus __stdcall  
CameralImage_IPicture( void * Image,  
                        IPicture ** NewPic  
                      )
```

从Image创建一个IPicture

### 参数

[in] **Image**  
[out] **NewPic** 新创建的IPicture

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameralImage\_IsEmpty()

```
MVSDK_API CameraSdkStatus __stdcall  
CameralImage_IsEmpty( void * Image,  
                      BOOL * IsEmpty  
                    )
```

判断一个Image是否为空

### 参数

[in] **Image**  
[out] **IsEmpty** 为空返回:TRUE(1) 否则返回:FALSE(0)

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameralImage\_SaveAsBmp()

```
MVSDK_API CameraSdkStatus __stdcall  
CameralImage_SaveAsBmp ( void * Image,  
                          char const * FileName  
                        )
```

以bmp格式保存Image

### 参数

[in] **Image**  
[in] **FileName** 文件名

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

## ◆ CameralImage\_SaveAsJpeg()

```
MVSDK_API CameraSdkStatus __stdcall  
CameralImage_SaveAsJpeg ( void * Image,  
                           char const * FileName,  
                           BYTE Quality  
                         )
```

以jpg格式保存Image

### 参数

[in] **Image**  
[in] **FileName** 文件名  
[in] **Quality** 保存质量(1-100), 100为质量最佳但文件也最大

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameralImage\_SaveAsPng()

```
MVSDK_API CameraSdkStatus __stdcall  
CameralImage_SaveAsPng ( void * Image,  
                          char const * FileName  
                        )
```

以png格式保存Image

#### 参数

[in] **Image**  
[in] **FileName** 文件名

## 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 **CameraStatus.h** 中错误码的定义。

### ◆ CameralImage\_SaveAsRaw()

```
MVSDK_API CameraSdkStatus __stdcall  
CameralImage_SaveAsRaw ( void * Image,  
                         char const * FileName,  
                         int Format  
                       )
```

以raw格式保存Image

#### 参数

[in] **Image**  
[in] **FileName** 文件名

[in] **Format** 0: 8Bit Raw 1: 16Bit Raw

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

### ◆ CameraImage\_SetUserData()

```
MVSDK_API CameraSdkStatus __stdcall  
CameraImage_SetUserData( void * Image,  
                           void * UserData  
                         )
```

设置Image的用户自定义数据

#### 参数

[in] **Image**  
[in] **UserData** 用户自定义数据

#### 返回

成功返回 **CAMERA\_STATUS\_SUCCESS(0)**。否则返回非0值的错误码, 请参考 [CameraStatus.h](#) 中错误码的定义。

# MVSDK API

## 类型定义

类型定义

### 类型定义

`typedef int CameraSdkStatus`

`typedef int CameraHandle`

## 详细描述

---

### 类型定义说明

---

- ◆ CameraHandle

```
typedef int CameraHandle
```

相机的句柄类型定义

- ◆ CameraSdkStatus

```
typedef int CameraSdkStatus
```

SDK错误码

# MVSDK API

## 枚举类型

枚举

### 枚举

```
enum emSdkLutMode { LUTMODE_PARAM_GEN =0, LUTMODE_PRESET =1,
LUTMODE_USER_DEF =2 }

enum emSdkRunMode { RUNMODE_PLAY =0, RUNMODE_PAUSE =1, RUNMODE_STOP =2 }

enum emSdkDisplayMode {
    DISPLAYMODE_SCALE =0, DISPLAYMODE_REAL =1, DISPLAYMODE_2X =2,
    DISPLAYMODE_4X =3,
    DISPLAYMODE_8X =4, DISPLAYMODE_16X =5, DISPLAYMODE_SCALE_FIT =6
}

enum emSdkRecordMode { RECORD_STOP =0, RECORD_START =1, RECORD_PAUSE =2 }

enum emSdkMirrorDirection { MIRROR_DIRECTION_HORIZONTAL =0,
MIRROR_DIRECTION_VERTICAL =1 }

enum emSdkRotateDirection { ROTATE_DIRECTION_0 =0, ROTATE_DIRECTION_90 =1,
ROTATE_DIRECTION_180 =2, ROTATE_DIRECTION_270 =3 }

enum emSdkFrameSpeed { FRAME_SPEED_LOW =0, FRAME_SPEED_NORMAL =1,
FRAME_SPEED_HIGH =2, FRAME_SPEED_SUPER =3 }

enum emSdkFileType {
    FILE_JPG =1, FILE_BMP =2, FILE_RAW =4, FILE_PNG =8,
    FILE_BMP_8BIT =16, FILE_PNG_8BIT =32, FILE_RAW_16BIT =64
}

enum emSdkSnapMode {
    CONTINUATION =0, SOFT_TRIGGER =1, EXTERNAL_TRIGGER =2,
    ROTARYENC_TRIGGER =3,
    ROTARYENC_COND_TRIGGER =4
}

enum emSdkLightFrequency { LIGHT_FREQUENCY_50HZ =0, LIGHT_FREQUENCY_60HZ =1 }

enum emSdkParameterTeam {
    PARAMETER_TEAM_DEFAULT =0xff, PARAMETER_TEAM_A =0,
    PARAMETER_TEAM_B =1, PARAMETER_TEAM_C =2,
    PARAMETER_TEAM_D =3
}

enum emSdkParameterMode { PARAM_MODE_BY_MODEL =0, PARAM_MODE_BY_NAME =1,
PARAM_MODE_BY_SN =2, PARAM_MODE_IN_DEVICE =3 }
相机参数加载模式 更多...

enum emSdkPropSheetMask {
    PROP_SHEET_INDEX_EXPOSURE =0, PROP_SHEET_INDEX_ISP_COLOR =1,
```

```
    PROP_SHEET_INDEX_ISP_LUT =2, PROP_SHEET_INDEX_ISP_SHAPE =3,
    PROP_SHEET_INDEX_VIDEO_FORMAT =4, PROP_SHEET_INDEX_RESOLUTION =5,
    PROP_SHEET_INDEX_IO_CTRL =6, PROP_SHEET_INDEX_TRIGGER_SET =7,
    PROP_SHEET_INDEX_OVERLAY =8, PROP_SHEET_INDEX_DEVICE_INFO =9,
    PROP_SHEET_INDEX_WDR =10, PROP_SHEET_INDEX_MULTI_EXPOSURE =11
}
```

```
enum emSdkPropSheetMsg { SHEET_MSG_LOAD_PARAM_DEFAULT =0,
    SHEET_MSG_LOAD_PARAM_GROUP =1, SHEET_MSG_LOAD_PARAM_FROMFILE =2,
    SHEET_MSG_SAVE_PARAM_GROUP =3 }
```

```
enum emSdkRefWinType { REF_WIN_AUTO_EXPOSURE =0, REF_WIN_WHITE_BALANCE =1 }
```

```
enum emSdkResolutionMode { RES_MODE_PREVIEW =0, RES_MODE_SNAPSHOT =1 }
```

```
enum emSdkClrTmpMode { CT_MODE_AUTO =0, CT_MODE_PRESET =1,
    CT_MODE_USER_DEF =2 }
```

```
enum emSdkLutChannel { LUT_CHANNEL_ALL =0, LUT_CHANNEL_RED =1,
    LUT_CHANNEL_GREEN =2, LUT_CHANNEL_BLUE =3 }
```

```
enum emSdkIspProcessor { ISP_PROCESSOR_PC =0, ISP_PROCESSOR_DEVICE =1 }
```

```
enum emStrobeControl { STROBE_SYNC_WITH_TRIG_AUTO =0,
    STROBE_SYNC_WITH_TRIG_MANUAL =1, STROBE_ALWAYS_HIGH =2,
    STROBE_ALWAYS_LOW =3 }
```

```
enum emExtTrigSignal {
    EXT_TRIG.LEADING_EDGE =0, EXT_TRIG.TRAILING_EDGE =1,
    EXT_TRIG.HIGH_LEVEL =2, EXT_TRIG.LOW_LEVEL =3,
    EXT_TRIG.DOUBLE_EDGE =4
}
```

```
enum emExtTrigShutterMode { EXT_TRIG_EXP_STANDARD =0, EXT_TRIG_EXP_GRR =1 }
```

```
enum emEvaluateDefinitionAlgorithm {
    EVALUATE_DEFINITION_DEVIATION =0, EVALUATE_DEFINITION_SMD =1,
    EVALUATE_DEFINITION_GRADIENT =2, EVALUATE_DEFINITION_SOBEL =3,
    EVALUATE_DEFINITION_ROBERT =4, EVALUATE_DEFINITION_LAPLACE =5,
    EVALUATE_DEFINITION_ALG_MAX =6
}
```

```
enum emCameraDrawTextFlags {
    CAMERA_DT_VCENTER = 0x1, CAMERA_DT_BOTTOM = 0x2, CAMERA_DT_HCENTER
    = 0x4, CAMERA_DT_RIGHT = 0x8,
    CAMERA_DT_SINGLELINE = 0x10, CAMERA_DT_ALPHA_BLEND = 0x20,
    CAMERA_DT_ANTI_ALIASING = 0x40
}
```

```
enum emCameraGPIOMode {
    IOMODE_TRIG_INPUT =0, IOMODE_STROBE_OUTPUT =1, IOMODE_GP_INPUT =2,
    IOMODE_GP_OUTPUT =3,
    IOMODE_PWM_OUTPUT =4, IOMODE_ROTARYENC_INPUT =5
}
```

```
enum emCameraGPIOFormat { IOFORMAT_SINGLE =0, IOFORMAT_RS422 =1,
```

```
IOFORMAT_RS422_TERM =2 }
```

```
enum emCameraGetImagePriority { CAMERA_GET_IMAGE_PRIORITY_OLDEST =0,  
CAMERA_GET_IMAGE_PRIORITY_NEWEST =1,  
CAMERA_GET_IMAGE_PRIORITY_NEXT =2 }
```

```
enum emCameraSoftTriggerExFlags { CAMERA_ST_CLEAR_BUFFER_BEFORE =0x1 }
```

## 详细描述

### 枚举类型说明

- ◆ emCameraDrawTextFlags

enum **emCameraDrawTextFlags**

文字输出标志

枚举值

CAMERA_DT_VCENTER	垂直居中
CAMERA_DT_BOTTOM	底部对齐
CAMERA_DT_HCENTER	水平居中
CAMERA_DT_RIGHT	右对齐
CAMERA_DT_SINGLELINE	单行显示
CAMERA_DT_ALPHA_BLEND	Alpha混合
CAMERA_DT_ANTI_ALIASING	抗锯齿

- ◆ emCameraGetImagePriority

enum **emCameraGetImagePriority**

取图优先级

枚举值

CAMERA_GET_IMAGE_PRIORITY_OLEDEST	获取缓存中最旧的一帧
CAMERA_GET_IMAGE_PRIORITY_NEWEST	获取缓存中最新的一帧（比此帧旧的将全部丢弃）
CAMERA_GET_IMAGE_PRIORITY_NEXT	丢弃缓存中的所有帧，并且如果此刻相机正在曝光或传输将会被立即打断，等待接收下一帧

#### 注解

某些型号的相机不支持此功能，对于不支持此功能的相机这个标志相当于  
**CAMERA\_GET\_IMAGE\_PRIORITY\_OLEDEST**

- ◆ emCameraGPIOFormat

enum **emCameraGPIOFormat**

## GPIO 格式

枚举值	
IOFORMAT_SINGLE	单端
IOFORMAT_RS422	差分RS422
IOFORMAT_RS422_TERM	差分RS422带终端电阻

## ◆ emCameraGPIOMode

### enum emCameraGPIOMode

#### GPIO模式

枚举值	
IOMODE_TRIG_INPUT	触发输入
IOMODE_STROBE_OUTPUT	闪光灯输出
IOMODE_GP_INPUT	通用型输入
IOMODE_GP_OUTPUT	通用型输出
IOMODE_PWM_OUTPUT	PWM型输出
IOMODE_ROTARYENC_INPUT	编码器输入

## ◆ emCameraSoftTriggerExFlags

### enum emCameraSoftTriggerExFlags

#### 软触发功能标志

枚举值	
CAMERA_ST_CLEAR_BUFFER_BEFORE	在软触发之前先清空相机已缓存的帧

## ◆ emEvaluateDefinitionAlgorithm

### enum emEvaluateDefinitionAlgorithm

#### 清晰度评估算法

枚举值	
EVALUATE_DEFINITION_DEVIATION	方差法
EVALUATE_DEFINITION_SMD	相邻像素灰度方差法
EVALUATE_DEFINITION_GRADIENT	梯度统计
EVALUATE_DEFINITION_SOBEL	Sobel
EVALUATE_DEFINITION_ROBERT	Robert
EVALUATE_DEFINITION_LAPLACE	Laplace

EVALUATE\_DEFINITION\_ALG\_MAX 算法个数

◆ emExtTrigShutterMode

enum **emExtTrigShutterMode**

硬件外触发时的快门方式

枚举值

EXT_TRIG_EXP_STANDARD	标准方式， 默认为该方式。
EXT_TRIG_EXP_GRR	全局复位方式， 部分滚动快门的CMOS型号的相机支持该方式， 配合外部机械快门， 可以达到全局快门的效果， 适合拍高速运动的物体

◆ emExtTrigSignal

enum **emExtTrigSignal**

硬件外触发的信号种类

枚举值

EXT_TRIG.LEADING_EDGE	上升沿触发， 默认为该方式
EXT_TRIG.TRAILING_EDGE	下降沿触发
EXT_TRIG.HIGH_LEVEL	高电平触发,电平宽度决定曝光时间， 仅部分型号的相机支持电平触发方式。
EXT_TRIG.LOW_LEVEL	低电平触发
EXT_TRIG.DOUBLE_EDGE	双边沿触发

◆ emSdkCirTmpMode

enum **emSdkCirTmpMode**

白平衡时色温模式

枚举值

CT_MODE_AUTO	自动识别色温
CT_MODE_PRESET	使用指定的预设色温
CT_MODE_USER_DEF	自定义色温(增益和矩阵)

◆ emSdkDisplayMode

enum **emSdkDisplayMode**

## SDK内部显示接口的显示方式

枚举值

DISPLAYMODE_SCALE	缩放显示模式，缩放到显示控件的尺寸
DISPLAYMODE_REAL	1:1显示模式，当图像尺寸大于显示控件的尺寸时，只显示局部
DISPLAYMODE_2X	放大2X
DISPLAYMODE_4X	放大4X
DISPLAYMODE_8X	放大8X
DISPLAYMODE_16X	放大16X
DISPLAYMODE_SCALE_FIT	拉伸缩放，填满显示区域

## ◆ emSdkFileType

enum **emSdkFileType**

保存文件的格式类型

枚举值

FILE_JPG	JPG
FILE_BMP	BMP 24bit
FILE_RAW	RAW
FILE_PNG	PNG 24bit
FILE_BMP_8BIT	BMP 8bit
FILE_PNG_8BIT	PNG 8bit
FILE_RAW_16BIT	RAW 16bit

## ◆ emSdkFrameSpeed

enum **emSdkFrameSpeed**

相机视频的帧率

枚举值

FRAME_SPEED_LOW	低速模式
FRAME_SPEED_NORMAL	普通模式
FRAME_SPEED_HIGH	高速模式(需要较高的传输带宽,多设备共享传输带宽时会对帧率的稳定性有影响)
FRAME_SPEED_SUPER	超高速模式(需要较高的传输带宽,多设备共享传输带宽时会对帧率的稳定性有影响)

## ◆ emSdkIspProcessor

enum **emSdkIspProcessor**

## ISP处理单元

枚举值

ISP_PROCESSSOR_PC	使用PC的软件ISP模块
ISP_PROCESSSOR_DEVICE	使用相机自带的硬件ISP模块

## ◆ emSdkLightFrequency

### enum emSdkLightFrequency

自动曝光时抗频闪的频率

枚举值

LIGHT_FREQUENCY_50HZ	50HZ,一般的灯光都是50HZ
LIGHT_FREQUENCY_60HZ	60HZ,主要是指显示器的

## ◆ emSdkLutChannel

### enum emSdkLutChannel

LUT的颜色通道

枚举值

LUT_CHANNEL_ALL	R,B,G三通道同时调节
LUT_CHANNEL_RED	红色通道
LUT_CHANNEL_GREEN	绿色通道
LUT_CHANNEL_BLUE	蓝色通道

## ◆ emSdkLutMode

### enum emSdkLutMode

图像查表变换的方式

枚举值

LUTMODE_PARAM_GEN	通过调节参数动态生成LUT表。
LUTMODE_PRESET	使用预设的LUT表
LUTMODE_USER_DEF	使用用户自定义的LUT表

## ◆ emSdkMirrorDirection

### enum emSdkMirrorDirection

## 图像的镜像操作

### 枚举值

MIRROR_DIRECTION_HORIZONTAL	水平镜像
MIRROR_DIRECTION_VERTICAL	垂直镜像

## ◆ emSdkParameterMode

### enum emSdkParameterMode

相机参数加载模式

#### 注解

您可以根据自己的使用环境，灵活使用以上几种方式加载参数。例如，以MV-U300为例，您希望多台该型号的相机在您的电脑上都共用4组参数，那么就使用PARAM\_MODE\_BY\_MODEL方式；如果您希望其中某一台或者某几台MV-U300能使用自己参数文件而其余的MV-U300又要使用相同的参数文件，那么使用PARAM\_MODE\_BY\_NAME方式；如果您希望每台MV-U300都使用不同的参数文件，那么使用PARAM\_MODE\_BY\_SN方式。

参数文件存在安装目录的\Camera\Configs目录下，以config为后缀名的文件。

### 枚举值

PARAM_MODE_BY_MODEL	根据相机型号名从文件中加载参数，例如MV-U300  <b>注解</b> 所有同型号的相机共用ABCD四组参数文件。修改一台相机的参数文件，会影响到整个同型号的相机参数加载。
PARAM_MODE_BY_NAME	根据设备昵称( <a href="#">tSdkCameraDevInfo.acFriendlyName</a> )从文件中加载参数，例如MV-U300,该昵称可自定义  <b>注解</b> 所有设备名相同的相机，共用ABCD四组参数文件。 默认情况下，当电脑上只接了某型号一台相机时， 设备名都是一样的，而您希望某一台相机能够加载 不同的参数文件，则可以通过修改其设备名的方式 来让其加载指定的参数文件。
PARAM_MODE_BY_SN	根据设备的唯一序列号从文件中加载参数，序列号在出厂时已经写入 设备，每台相机拥有不同的序列号。  <b>注解</b> 相机按照自己的唯一序列号来加载ABCD四组参数文件， 序列号在出厂时已经固化在相机内，每台相机的序列号 都不相同，通过这种方式，每台相机的参数文件都是独立的。
PARAM_MODE_IN_DEVICE	从设备的固态存储器中加载参数。不是所有的型号都支持从相机中读 写参数组，由 <a href="#">tSdkCameraCapbility.bParamInDevice</a> 决定

◆ emSdkParameterTeam

enum **emSdkParameterTeam**

相机的配置参数，分为A,B,C,D 4组进行保存。

枚举值

PARAMETER_TEAM_DEFAULT	默认参数
PARAMETER_TEAM_A	参数A
PARAMETER_TEAM_B	参数B
PARAMETER_TEAM_C	参数C
PARAMETER_TEAM_D	参数D

◆ emSdkPropSheetMask

enum **emSdkPropSheetMask**

SDK生成的相机配置页面掩码值

枚举值

PROP_SHEET_INDEX_EXPOSURE	曝光设置
PROP_SHEET_INDEX_ISP_COLOR	颜色矩阵设置
PROP_SHEET_INDEX_ISP_LUT	LUT设置
PROP_SHEET_INDEX_ISP_SHAPE	变换设置
PROP_SHEET_INDEX_VIDEO_FORMAT	格式设置
PROP_SHEET_INDEX_RESOLUTION	分辨率设置
PROP_SHEET_INDEX_IO_CTRL	IO控制
PROP_SHEET_INDEX_TRIGGER_SET	触发模式
PROP_SHEET_INDEX_OVERLAY	十字线
PROP_SHEET_INDEX_DEVICE_INFO	设备信息
PROP_SHEET_INDEX_WDR	宽动态
PROP_SHEET_INDEX_MULTI_EXPOSURE	多重曝光

◆ emSdkPropSheetMsg

enum **emSdkPropSheetMsg**

SDK生成的相机配置页面的回调消息类型

枚举值

SHEET_MSG_LOAD_PARAM_DEFAULT	参数被恢复成默认后，触发该消息
SHEET_MSG_LOAD_PARAM_GROUP	加载指定参数组，触发该消息
SHEET_MSG_LOAD_PARAM_FROMFILE	从指定文件加载参数后，触发该消息

## SHEET\_MSG\_SAVE\_PARAM\_GROUP

当前参数组被保存时，触发该消息

### ◆ emSdkRecordMode

#### enum emSdkRecordMode

录像状态

枚举值

RECORD_STOP	停止
RECORD_START	录像中
RECORD_PAUSE	暂停

### ◆ emSdkRefWinType

#### enum emSdkRefWinType

可视化选择参考窗口的类型

枚举值

REF_WIN_AUTO_EXPOSURE	自动曝光窗口
REF_WIN_WHITE_BALANCE	白平衡窗口

### ◆ emSdkResolutionMode

#### enum emSdkResolutionMode

可视化选择参考窗口的类型

枚举值

RES_MODE_PREVIEW	预览
RES_MODE_SNAPSHOT	抓拍

### ◆ emSdkRotateDirection

#### enum emSdkRotateDirection

图像的旋转操作

枚举值

ROTATE_DIRECTION_0	不旋转
ROTATE_DIRECTION_90	逆时针90度
ROTATE_DIRECTION_180	逆时针180度

## ROTATE\_DIRECTION\_270 | 逆时针270度

### ◆ emSdkRunMode

#### enum **emSdkRunMode**

相机的视频流控制

##### 枚举值

RUNMODE_PLAY	正常预览，捕获到图像就显示。（如果相机处于触发模式，则会等待触发帧的到来）
RUNMODE_PAUSE	暂停，会暂停相机的图像输出，同时也不会去捕获图像
RUNMODE_STOP	停止相机工作。反初始化后，相机就处于停止模式

### ◆ emSdkSnapMode

#### enum **emSdkSnapMode**

相机中的图像传感器的工作模式

##### 枚举值

CONTINUATION	连续采集模式
SOFT_TRIGGER	软件触发模式，由软件发送指令后，传感器开始采集指定帧数的图像，采集完成后，停止输出
EXTERNAL_TRIGGER	硬件触发模式，当接收到外部信号，传感器开始采集指定帧数的图像，采集完成后，停止输出
ROTARYENC_TRIGGER	编码器触发模式（仅用于线阵相机）
ROTARYENC_COND_TRIGGER	编码器条件触发模式（仅用于线阵相机）

### ◆ emStrobeControl

#### enum **emStrobeControl**

闪光灯信号控制方式

##### 枚举值

STROBE_SYNC_WITH_TRIG_AUTO	和触发信号同步，触发后，相机进行曝光时，自动生成STROBE信号。此时，有效极性可设置( <a href="#">CameraSetStrobePolarity</a> )。
STROBE_SYNC_WITH_TRIG_MANUAL	和触发信号同步，触发后，STROBE延时指定的时间后( <a href="#">CameraSetStrobeDelayTime</a> )，再持续指定时间的脉冲( <a href="#">CameraSetStrobePulseWidth</a> )，有效极性可设置( <a href="#">CameraSetStrobePolarity</a> )。
STROBE_ALWAYS_HIGH	始终为高，忽略STROBE信号的其他设置
STROBE_ALWAYS_LOW	始终为低，忽略STROBE信号的其他设置

# MVSDK API

## 宏定义

```
#define CAMERA_STATUS_SUCCESS 0
```

操作成功

```
#define CAMERA_STATUS_FAILED -1
```

操作失败

```
#define CAMERA_STATUS_INTERNAL_ERROR -2
```

内部错误

```
#define CAMERA_STATUS_UNKNOW -3
```

未知错误

```
#define CAMERA_STATUS_NOT_SUPPORTED -4
```

不支持该功能

```
#define CAMERA_STATUS_NOT_INITIALIZED -5
```

初始化未完成

```
#define CAMERA_STATUS_PARAMETER_INVALID -6
```

参数无效

```
#define CAMERA_STATUS_PARAMETER_OUT_OF_BOUND -7
```

参数越界

```
#define CAMERA_STATUS_UNENABLED -8
```

未使能

```
#define CAMERA_STATUS_USER_CANCEL -9
```

用户手动取消了，比如roi面板点击取消，返回

```
#define CAMERA_STATUS_PATH_NOT_FOUND -10
```

注册表中没有找到对应的路径

```
#define CAMERA_STATUS_SIZE_DISMATCH -11
```

获得图像数据长度和定义的尺寸不匹配

```
#define CAMERA_STATUS_TIME_OUT -12
```

超时错误

```
#define CAMERA_STATUS_IO_ERROR -13
```

硬件IO错误

```
#define CAMERA_STATUS_COMM_ERROR -14
```

通讯错误

```
#define CAMERA_STATUS_BUS_ERROR -15
```

总线错误

```
#define CAMERA_STATUS_NO_DEVICE_FOUND -16
    没有发现设备

#define CAMERA_STATUS_NO_LOGIC_DEVICE_FOUND -17
    未找到逻辑设备

#define CAMERA_STATUS_DEVICE_IS_OPENED -18
    设备已经打开

#define CAMERA_STATUS_DEVICE_IS_CLOSED -19
    设备已经关闭

#define CAMERA_STATUS_DEVICE_VEDIO_CLOSED -20
    没有打开设备视频，调用录像相关的函数时，如果相机视频没有打开，则会返回该错误。

#define CAMERA_STATUS_NO_MEMORY -21
    没有足够的系统内存

#define CAMERA_STATUS_FILE_CREATE_FAILED -22
    创建文件失败

#define CAMERA_STATUS_FILE_INVALID -23
    文件格式无效

#define CAMERA_STATUS_WRITE_PROTECTED -24
    写保护，不可写

#define CAMERA_STATUS_GRAB_FAILED -25
    数据采集失败

#define CAMERA_STATUS_LOST_DATA -26
    数据丢失，不完整

#define CAMERA_STATUS_EOF_ERROR -27
    未接收到帧结束符

#define CAMERA_STATUS_BUSY -28
    正忙(上一次操作还在进行中)，此次操作不能进行

#define CAMERA_STATUS_WAIT -29
    需要等待(进行操作的条件不成立)，可以再次尝试

#define CAMERA_STATUS_IN_PROCESS -30
    正在进行，已经被操作过

#define CAMERA_STATUS_IIC_ERROR -31
    IIC传输错误

#define CAMERA_STATUS_SPI_ERROR -32
    SPI传输错误
```

```
#define CAMERA_STATUS_USB_CONTROL_ERROR -33
USB控制传输错误

#define CAMERA_STATUS_USB_BULK_ERROR -34
USB BULK传输错误

#define CAMERA_STATUS_SOCKET_INIT_ERROR -35
网络传输套件初始化失败

#define CAMERA_STATUS_GIGE_FILTER_INIT_ERROR -36
网络相机内核过滤驱动初始化失败, 请检查是否正确安装了驱动, 或者重新安装。

#define CAMERA_STATUS_NET_SEND_ERROR -37
网络数据发送错误

#define CAMERA_STATUS_DEVICE_LOST -38
与网络相机失去连接, 心跳检测超时

#define CAMERA_STATUS_DATA_RECV_LESS -39
接收到的字节数比请求的少

#define CAMERA_STATUS_FUNCTION_LOAD_FAILED -40
从文件中加载程序失败

#define CAMERA_STATUS_CRITICAL_FILE_LOST -41
程序运行所必须的文件丢失。

#define CAMERA_STATUS_SENSOR_ID_DISMATCH -42
固件和程序不匹配, 原因是下载了错误的固件。

#define CAMERA_STATUS_OUT_OF_RANGE -43
参数超出有效范围。

#define CAMERA_STATUS_REGISTRY_ERROR -44
安装程序注册错误。请重新安装程序, 或者运行安装目录Setup\Installer.exe

#define CAMERA_STATUS_ACCESS_DENY -45
禁止访问。指定相机已经被其他程序占用时, 再申请访问该相机, 会返回该状态。(一个相机不能被多个程序同时访问)

#define CAMERA_STATUS_CAMERA_NEED_RESET -46
表示相机需要复位后才能正常使用, 此时请让相机断电重启, 或者重启操作系统后, 便可正常使用。

#define CAMERA_STATUS_ISP_MOODULE_NOT_INITIALIZED -47
ISP模块未初始化

#define CAMERA_STATUS_ISP_DATA_CRC_ERROR -48
数据校验错误

#define CAMERA_STATUS_MV_TEST_FAILED -49
数据测试失败
```

```
#define CAMERA_STATUS_INTERNAL_ERR1 -50
    内部错误1

#define CAMERA_STATUS_U3V_NO_CONTROL_EP -51
    U3V控制端点未找到

#define CAMERA_STATUS_U3V_CONTROL_ERROR -52
    U3V控制通讯错误

#define CAMERA_STATUS_INVALID_FRIENDLY_NAME -53
    无效的设备名，名字里不能包含以下字符(V:/*?"<>|")"

#define CAMERA_STATUS_FORMAT_ERROR -54
    格式错误

#define CAMERA_AIA_PACKET_RESEND 0x0100
    该帧需要重传

#define CAMERA_AIA_NOT_IMPLEMENTED 0x8001
    设备不支持的命令

#define CAMERA_AIA_INVALID_PARAMETER 0x8002
    命令参数非法

#define CAMERA_AIA_INVALID_ADDRESS 0x8003
    不可访问的地址

#define CAMERA_AIA_WRITE_PROTECT 0x8004
    访问的对象不可写

#define CAMERA_AIA_BAD_ALIGNMENT 0x8005
    访问的地址没有按照要求对齐

#define CAMERA_AIA_ACCESS_DENIED 0x8006
    没有访问权限

#define CAMERA_AIA_BUSY 0x8007
    命令正在处理中

#define CAMERA_AIA_DEPRECATED 0x8008
    0x8008-0x800B 0x800F 该指令已经废弃

#define CAMERA_AIA_PACKET_UNAVAILABLE 0x800C
    包无效

#define CAMERA_AIA_DATA_OVERRUN 0x800D
    数据溢出，通常是收到的数据比需要的多

#define CAMERA_AIA_INVALID_HEADER 0x800E
    数据包头部中某些区域与协议不匹配
```

```
#define CAMERA_AIA_PACKET_NOT_YET_AVAILABLE 0x8010
    图像分包数据还未准备好，多用于触发模式，应用程序访问超时

#define CAMERA_AIA_PACKET_AND_PREV_REMOVED_FROM_MEMORY 0x8011
    需要访问的分包已经不存在。多用于重传时数据已经不在缓冲区中

#define CAMERA_AIA_PACKET_REMOVED_FROM_MEMORY 0x8012
    CAMERA_AIA_PACKET_AND_PREV_REMOVED_FROM_MEMORY

#define CAMERA_AIA_NO_REF_TIME 0x0813
    没有参考时钟源。多用于时间同步的命令执行时

#define CAMERA_AIA_PACKET_TEMPORARILY_UNAVAILABLE 0x0814
    由于信道带宽问题，当前分包暂时不可用，需稍后进行访问

#define CAMERA_AIA_OVERFLOW 0x0815
    设备端数据溢出，通常是队列已满

#define CAMERA_AIA_ACTION_LATE 0x0816
    命令执行已经超过有效的指定时间

#define CAMERA_AIA_ERROR 0xFFFF
    错误

#define EXT_TRIG_MASK_GRR_SHUTTER 1
    快门支持GRR模式

#define EXT_TRIG_MASK_LEVEL_MODE 2
    支持电平触发

#define EXT_TRIG_MASK_DOUBLE_EDGE 4
    支持双边沿触发

#define EXT_TRIG_MASK_BUFFERED_DELAY 8
    支持信号后延

#define MASK_2X2_HD (1<<0)

#define MASK_3X3_HD (1<<1)

#define MASK_4X4_HD (1<<2)

#define MASK_5X5_HD (1<<3)

#define MASK_6X6_HD (1<<4)

#define MASK_7X7_HD (1<<5)

#define MASK_8X8_HD (1<<6)

#define MASK_9X9_HD (1<<7)

#define MASK_10X10_HD (1<<8)
```

```
#define MASK_11X11_HD (1<<9)

#define MASK_12X12_HD (1<<10)

#define MASK_13X13_HD (1<<11)

#define MASK_14X14_HD (1<<12)

#define MASK_15X15_HD (1<<13)

#define MASK_16X16_HD (1<<14)

#define MASK_17X17_HD (1<<15)

#define MASK_2X2_SW (1<<16)

#define MASK_3X3_SW (1<<17)

#define MASK_4X4_SW (1<<18)

#define MASK_5X5_SW (1<<19)

#define MASK_6X6_SW (1<<20)

#define MASK_7X7_SW (1<<21)

#define MASK_8X8_SW (1<<22)

#define MASK_9X9_SW (1<<23)

#define MASK_10X10_SW (1<<24)

#define MASK_11X11_SW (1<<25)

#define MASK_12X12_SW (1<<26)

#define MASK_13X13_SW (1<<27)

#define MASK_14X14_SW (1<<28)

#define MASK_15X15_SW (1<<29)

#define MASK_16X16_SW (1<<30)

#define MASK_17X17_SW (1<<31)

#define CAMERA_MEDIA_TYPE_MONO 0x01000000

#define CAMERA_MEDIA_TYPE_RGB 0x02000000

#define CAMERA_MEDIA_TYPE_COLOR 0x02000000

#define CAMERA_MEDIA_TYPE_CUSTOM 0x80000000
```

```
#define CAMERA_MEDIA_TYPE_COLOR_MASK 0xFF000000

#define CAMERA_MEDIA_TYPE_OCCUPY1BIT 0x00010000

#define CAMERA_MEDIA_TYPE_OCCUPY2BIT 0x00020000

#define CAMERA_MEDIA_TYPE_OCCUPY4BIT 0x00040000

#define CAMERA_MEDIA_TYPE_OCCUPY8BIT 0x00080000

#define CAMERA_MEDIA_TYPE_OCCUPY10BIT 0x000A0000

#define CAMERA_MEDIA_TYPE_OCCUPY12BIT 0x000C0000

#define CAMERA_MEDIA_TYPE_OCCUPY16BIT 0x00100000

#define CAMERA_MEDIA_TYPE_OCCUPY24BIT 0x00180000

#define CAMERA_MEDIA_TYPE_OCCUPY32BIT 0x00200000

#define CAMERA_MEDIA_TYPE_OCCUPY36BIT 0x00240000

#define CAMERA_MEDIA_TYPE_OCCUPY48BIT 0x00300000

#define CAMERA_MEDIA_TYPE_OCCUPY64BIT 0x00400000

#define CAMERA_MEDIA_TYPE_EFFECTIVE_PIXEL_SIZE_MASK 0x00FF0000

#define CAMERA_MEDIA_TYPE_EFFECTIVE_PIXEL_SIZE_SHIFT 16

#define CAMERA_MEDIA_TYPE_PIXEL_SIZE(type) (((type) &
CAMERA_MEDIA_TYPE_EFFECTIVE_PIXEL_SIZE_MASK) >>
CAMERA_MEDIA_TYPE_EFFECTIVE_PIXEL_SIZE_SHIFT)

#define CAMERA_MEDIA_TYPE_ID_MASK 0x0000FFFF

#define CAMERA_MEDIA_TYPE_COUNT 0x46

#define CAMERA_MEDIA_TYPE_MONO1P (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY1BIT | 0x0037)

#define CAMERA_MEDIA_TYPE_MONO2P (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY2BIT | 0x0038)

#define CAMERA_MEDIA_TYPE_MONO4P (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY4BIT | 0x0039)

#define CAMERA_MEDIA_TYPE_MONO8 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0001)

#define CAMERA_MEDIA_TYPE_MONO8S (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0002)
```

```
#define CAMERA_MEDIA_TYPE_MONO10 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0003)  
  
#define CAMERA_MEDIA_TYPE_MONO10_PACKED (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0004)  
  
#define CAMERA_MEDIA_TYPE_MONO12 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0005)  
  
#define CAMERA_MEDIA_TYPE_MONO12_PACKED (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0006)  
  
#define CAMERA_MEDIA_TYPE_MONO14 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0025)  
  
#define CAMERA_MEDIA_TYPE_MONO16 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0007)  
  
#define CAMERA_MEDIA_TYPE_BAYGR8 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0008)  
  
#define CAMERA_MEDIA_TYPE_BAYRG8 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0009)  
  
#define CAMERA_MEDIA_TYPE_BAYGB8 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x000A)  
  
#define CAMERA_MEDIA_TYPE_BAYBG8 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x000B)  
  
#define CAMERA_MEDIA_TYPE_BAYGR10_MIPI (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY10BIT | 0x0026)  
  
#define CAMERA_MEDIA_TYPE_BAYRG10_MIPI (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY10BIT | 0x0027)  
  
#define CAMERA_MEDIA_TYPE_BAYGB10_MIPI (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY10BIT | 0x0028)  
  
#define CAMERA_MEDIA_TYPE_BAYBG10_MIPI (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY10BIT | 0x0029)  
  
#define CAMERA_MEDIA_TYPE_BAYGR10 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x000C)  
  
#define CAMERA_MEDIA_TYPE_BAYRG10 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x000D)  
  
#define CAMERA_MEDIA_TYPE_BAYGB10 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x000E)  
  
#define CAMERA_MEDIA_TYPE_BAYBG10 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x000F)  
  
#define CAMERA_MEDIA_TYPE_BAYGR12 (CAMERA_MEDIA_TYPE_MONO |
```

```
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0010)
```

```
#define CAMERA_MEDIA_TYPE_BAYRG12 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0011)
```

```
#define CAMERA_MEDIA_TYPE_BAYGB12 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0012)
```

```
#define CAMERA_MEDIA_TYPE_BAYBG12 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0013)
```

```
#define CAMERA_MEDIA_TYPE_BAYGR10_PACKED (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0026)
```

```
#define CAMERA_MEDIA_TYPE_BAYRG10_PACKED (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0027)
```

```
#define CAMERA_MEDIA_TYPE_BAYGB10_PACKED (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0028)
```

```
#define CAMERA_MEDIA_TYPE_BAYBG10_PACKED (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0029)
```

```
#define CAMERA_MEDIA_TYPE_BAYGR12_PACKED (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x002A)
```

```
#define CAMERA_MEDIA_TYPE_BAYRG12_PACKED (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x002B)
```

```
#define CAMERA_MEDIA_TYPE_BAYGB12_PACKED (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x002C)
```

```
#define CAMERA_MEDIA_TYPE_BAYBG12_PACKED (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x002D)
```

```
#define CAMERA_MEDIA_TYPE_BAYGR16 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x002E)
```

```
#define CAMERA_MEDIA_TYPE_BAYRG16 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x002F)
```

```
#define CAMERA_MEDIA_TYPE_BAYGB16 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0030)
```

```
#define CAMERA_MEDIA_TYPE_BAYBG16 (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0031)
```

```
#define CAMERA_MEDIA_TYPE_RGB8 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0014)
```

```
#define CAMERA_MEDIA_TYPE_BGR8 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0015)
```

```
#define CAMERA_MEDIA_TYPE_RGBA8 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY32BIT | 0x0016)
```

```
#define CAMERA_MEDIA_TYPE_BGRA8 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY32BIT | 0x0017)  
  
#define CAMERA_MEDIA_TYPE_RGB10 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0018)  
  
#define CAMERA_MEDIA_TYPE_BGR10 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0019)  
  
#define CAMERA_MEDIA_TYPE_RGB12 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x001A)  
  
#define CAMERA_MEDIA_TYPE_BGR12 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x001B)  
  
#define CAMERA_MEDIA_TYPE_RGB16 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0033)  
  
#define CAMERA_MEDIA_TYPE_BGR16 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x004B)  
  
#define CAMERA_MEDIA_TYPE_RGBA16 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY64BIT | 0x0064)  
  
#define CAMERA_MEDIA_TYPE_BGRA16 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY64BIT | 0x0051)  
  
#define CAMERA_MEDIA_TYPE_RGB10V1_PACKED (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY32BIT | 0x001C)  
  
#define CAMERA_MEDIA_TYPE_RGB10P32 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY32BIT | 0x001D)  
  
#define CAMERA_MEDIA_TYPE_RGB12V1_PACKED (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY36BIT | 0X0034)  
  
#define CAMERA_MEDIA_TYPE_RGB565P (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0035)  
  
#define CAMERA_MEDIA_TYPE_BGR565P (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0X0036)  
  
#define CAMERA_MEDIA_TYPE_YUV411_8_UYYYVYY (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x001E)  
  
#define CAMERA_MEDIA_TYPE_YUV422_8_UYVY (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x001F)  
  
#define CAMERA_MEDIA_TYPE_YUV422_8 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0032)  
  
#define CAMERA_MEDIA_TYPE_YUV8_UYV (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0020)
```

```
#define CAMERA_MEDIA_TYPE_YCBCR8_CBYCR (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x003A)  
  
#define CAMERA_MEDIA_TYPE_YCBCR422_8 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x003B)  
  
#define CAMERA_MEDIA_TYPE_YCBCR422_8_CBYCRY (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0043)  
  
#define CAMERA_MEDIA_TYPE_YCBCR411_8_CBYYCRY (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x003C)  
  
#define CAMERA_MEDIA_TYPE_YCBCR601_8_CBYCR (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x003D)  
  
#define CAMERA_MEDIA_TYPE_YCBCR601_422_8 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x003E)  
  
#define CAMERA_MEDIA_TYPE_YCBCR601_422_8_CBYCRY (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0044)  
  
#define CAMERA_MEDIA_TYPE_YCBCR601_411_8_CBYYCRY (CAMERA_MEDIA_TYPE_COLO  
| CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x003F)  
  
#define CAMERA_MEDIA_TYPE_YCBCR709_8_CBYCR (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0040)  
  
#define CAMERA_MEDIA_TYPE_YCBCR709_422_8 (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0041)  
  
#define CAMERA_MEDIA_TYPE_YCBCR709_422_8_CBYCRY (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0045)  
  
#define CAMERA_MEDIA_TYPE_YCBCR709_411_8_CBYYCRY (CAMERA_MEDIA_TYPE_COLO  
| CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0042)  
  
#define CAMERA_MEDIA_TYPE_RGB8_PLANAR (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0021)  
  
#define CAMERA_MEDIA_TYPE_RGB10_PLANAR (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0022)  
  
#define CAMERA_MEDIA_TYPE_RGB12_PLANAR (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0023)  
  
#define CAMERA_MEDIA_TYPE_RGB16_PLANAR (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0024)  
  
#define CAMERA_MEDIA_TYPE_BAYGR12_PACKED_MV (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0060)  
  
#define CAMERA_MEDIA_TYPE_BAYRG12_PACKED_MV (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0061)  
  
#define CAMERA_MEDIA_TYPE_BAYGB12_PACKED_MV (CAMERA_MEDIA_TYPE_MONO |
```

```
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0062)
```

```
#define CAMERA_MEDIA_TYPE_BAYBG12_PACKED_MV (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0063)
```

```
#define CAMERA_MEDIA_TYPE_MONO12_PACKED_MV (CAMERA_MEDIA_TYPE_MONO |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0064)
```

```
#define CAMERA_MEDIA_TYPE_YUV420P_MV (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0065)
```

```
#define CAMERA_MEDIA_TYPE_YUV_NV21_MV (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0066)
```

```
#define CAMERA_MEDIA_TYPE_H264_MV (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0067)
```

```
#define CAMERA_MEDIA_TYPE_H265_MV (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0068)
```

```
#define CAMERA_MEDIA_TYPE_JPEG_MV (CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0069)
```

## 详细描述

---

# MVSDK API

## 结构体

这里列出了所有结构体，并附带简要说明：

<a href="#">C sCameraFrame</a>	
<a href="#">C tContrastRange</a>	
<a href="#">C tGammaRange</a>	
<a href="#">C tRgbGainRange</a>	
<a href="#">C tSaturationRange</a>	
<a href="#">C tSdkAeAlgorithm</a>	
<a href="#">C tSdkBayerDecodeAlgorithm</a>	
<a href="#">C tSdkCameraCapbility</a>	
<a href="#">C tSdkCameraDevInfo</a>	
<a href="#">C tSdkColorTemperatureDes</a>	
<a href="#">C tSdkExpose</a>	
<a href="#">C tSdkFrameEvent_</a>	
<a href="#">C tSdkFrameHead</a>	
<a href="#">C tSdkFrameSpeed</a>	
<a href="#">C tSdkFrameStatistic</a>	
<a href="#">C tSdkGrabberStat</a>	
<a href="#">C tSdkImageResolution</a>	
<a href="#">C tSdkIspCapacity</a>	
<a href="#">C tSdkMediaType</a>	
<a href="#">C tSdkPackLength</a>	
<a href="#">C tSdkPresetLut</a>	
<a href="#">C tSdkResolutionRange</a>	
<a href="#">C tSdkTrigger</a>	
<a href="#">C tSharpnessRange</a>	



# MVSDK API

## sCameraFrame结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

tSdkFrameHead head

帧头

BYTE \* pBuffer

数据区

## 详细描述

---

### 图像帧描述

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tContrastRange结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iMin**  
最小值

INT **iMax**  
最大值

## 详细描述

---

对比度的设定范围

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tGammaRange结构体参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iMin**  
最小值

INT **iMax**  
最大值

## 详细描述

---

伽马的设定范围

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tRgbGainRange结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iRGainMin**

红色增益的最小值

INT **iRGainMax**

红色增益的最大值

INT **iGGainMin**

绿色增益的最小值

INT **iGGainMax**

绿色增益的最大值

INT **iBGainMin**

蓝色增益的最小值

INT **iBGainMax**

蓝色增益的最大值

## 详细描述

---

RGB三通道数字增益的设定范围

---

该结构体的文档由以下文件生成:

- [CameraDefine.h](#)

# MVSDK API

## tSaturationRange结 构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iMin**  
最小值

INT **iMax**  
最大值

## 详细描述

---

饱和度设定的范围

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tSdkAeAlgorithm结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iIndex**

索引

char **acDescription** [32]

描述信息

## 详细描述

---

### AE算法描述

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tSdkBayerDecodeAlg orithm结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iIndex**

索引

char **acDescription** [32]

描述信息

## 详细描述

---

RAW转RGB算法描述

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tSdkCameraCapbility 结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

**tSdkTrigger \* pTriggerDesc**  
触发模式

**INT iTriggerDesc**  
触发模式的个数，即pTriggerDesc数组的大小

**tSdkImageResolution \* plImageSizeDesc**  
预设分辨率

**INT ilImageSizeDesc**  
预设分辨率的个数，即plImageSizeDesc数组的大小

**tSdkColorTemperatureDes \* pClrTempDesc**  
预设色温，用于白平衡

**INT iClrTempDesc**  
预设色温个数

**tSdkMediaType \* pMediaTypeDesc**  
相机输出图像格式

**INT iMediaTypdeDesc**

相机输出图像格式的种类个数，即  
pMediaTypeDesc数组的大小。

**tSdkFrameSpeed \* pFrameSpeedDesc**

可调节帧速类型，对应界面上普通  
高速 和超级三种速度设置

**INT iFrameSpeedDesc**

可调节帧速类型的个数，即  
pFrameSpeedDesc数组的大小。

**tSdkPackLength \* pPackLenDesc**

传输包长度，一般用于网络设备

**INT iPakLenDesc**

可供选择的传输分包长度的个数，  
即pPackLenDesc数组的大小。

**INT iOutputIoCounts**

可编程输出IO的个数

**INT iInputIoCounts**

可编程输入IO的个数

**tSdkPresetLut \* pPresetLutDesc**

相机预设的LUT表

**INT iPresetLut**

相机预设的LUT表的个数，即  
pPresetLutDesc数组的大小

**INT iUserDataMaxLen**

指示该相机中用于保存用户数据区  
的最大长度。为0表示无。

**BOOL bParamInDevice**  
指示该设备是否支持从设备中读写参数组。1为支持，0不支持。

**tSdkAeAlgorithm \* pAeAlmSwDesc**  
软件自动曝光算法描述

**int iAeAlmSwDesc**  
软件自动曝光算法个数

**tSdkAeAlgorithm \* pAeAlmHdDesc**  
硬件自动曝光算法描述，为NULL表示不支持硬件自动曝光

**int iAeAlmHdDesc**  
硬件自动曝光算法个数，为0表示不支持硬件自动曝光

**tSdkBayerDecodeAlgorithm \* pBayerDecAlmSwDesc**  
软件Bayer转换为RGB数据的算法描述

**int iBayerDecAlmSwDesc**  
软件Bayer转换为RGB数据的算法个数

**tSdkBayerDecodeAlgorithm \* pBayerDecAlmHdDesc**  
硬件Bayer转换为RGB数据的算法描述，为NULL表示不支持

**int iBayerDecAlmHdDesc**  
硬件Bayer转换为RGB数据的算法个数，为0表示不支持

**tSdkExpose** **sExposeDesc**  
曝光的范围值

**tSdkResolutionRange** **sResolutionRange**  
分辨率范围描述

**tRgbGainRange** **sRgbGainRange**  
图像数字增益范围描述

**tSaturationRange** **sSaturationRange**  
饱和度范围描述

**tGammaRange** **sGammaRange**  
伽马范围描述

**tContrastRange** **sContrastRange**  
对比度范围描述

**tSharpnessRange** **sSharpnessRange**  
锐化范围描述

**tSdkIspCapacity** **sIspCapacity**  
ISP能力描述

## 详细描述

---

定义整合的设备描述信息，这些信息可以用于动态构建UI

### 注解

调用**CameraGetCapability**获取本结构

### 示例：

[grab.cpp](#), [grab\\_ex.cpp](#), [roi.cpp](#), [soft\\_trigger.cpp](#),  
[win\\_basic.cpp](#), [win\\_callback.cpp](#) , 以及 [win\\_grabber.cpp](#).

---

该结构体的文档由以下文件生成:

- [CameraDefine.h](#)

# MVSDK API

## tSdkCameraDevInfo 结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

char **acProductSeries** [32]

产品系列

char **acProductName** [32]

产品名称

char **acFriendlyName** [32]

char **acLinkName** [32]

内核符号连接名，内部使用

char **acDriverVersion** [32]

驱动版本

char **acSensorType** [32]

sensor类型

char **acPortType** [32]

接口类型

char **acSn** [32]

产品唯一序列号

**UINT ulnstance**

该型号相机在该电脑上的实例索引号，用于区分同型号多相机

---

## 详细描述

---

相机的设备信息

**示例:**

[grab.cpp](#), [grab\\_ex.cpp](#), [roi.cpp](#), [soft\\_trigger.cpp](#),  
[win\\_basic.cpp](#), [win\\_callback.cpp](#) , 以及 [win\\_grabber.cpp](#).

## 结构体成员变量说明

---

### ◆ acFriendlyName

char acFriendlyName[32]

产品昵称，用户可自定义改昵称，保存在相机内，用于区分多个相机同时使用，可以用**CameraSetFriendlyName**接口改变该昵称，设备重启后生效。

**示例：**

[grab\\_ex.cpp.](#)

---

该结构体的文档由以下文件生成：

- [CameraDefine.h](#)

# MVSDK API

## tSdkColorTemperatu reDes结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iIndex**

模式索引号

char **acDescription** [32]

描述信息

## 详细描述

---

相机白平衡色温模式描述信息

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tSdkExpose结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

UINT **uiTargetMin**

自动曝光亮度目标最小值

UINT **uiTargetMax**

自动曝光亮度目标最大值

UINT **uiAnalogGainMin**

模拟增益的最小值，单位为fAnalogGainStep中定义

UINT **uiAnalogGainMax**

模拟增益的最大值，单位为fAnalogGainStep中定义

float **fAnalogGainStep**

模拟增益每增加1，对应的增加的放大倍数。例如，  
uiAnalogGainMin一般为16，fAnalogGainStep一般为  
0.125，那么最小放大倍数就是 $16 * 0.125 = 2$ 倍

UINT **uiExposeTimeMin**

手动模式下，曝光时间的最小值，单位:行。根据  
CameraGetExposureLineTime可以获得一行对应的时间(微  
秒)，从而得到整帧的曝光时间

UINT **uiExposeTimeMax**

手动模式下，曝光时间的最大值，单位:行



## 详细描述

---

相机曝光功能范围定义

参见

[\*\*tSdkCameraCapbility.sExposeDesc\*\*](#)

---

该结构体的文档由以下文件生成:

- [\*\*CameraDefine.h\*\*](#)

# MVSDK API

## tSdkFrameEvent\_结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

UINT **uType**

事件类型(1:帧开始 2:帧结束)

UINT **uStatus**

状态(0:成功 非0:错误)

UINT **uFrameID**

帧ID

UINT **uWidth**

宽度

UINT **uHeight**

高度

UINT **uPixelFormat**

图像格式

UINT **TimeStampL**

时间戳低32位

UINT **TimeStampH**

时间戳高32位



## 详细描述

---

### 帧事件

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tSdkFrameHead结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

UINT **uiMediaType**

图像格式

UINT **uBytes**

图像数据字节数

INT **iWidth**

图像的宽度, 调用图像处理函数后, 该变量可能被动态修改, 来指示处理后的图像尺寸

INT **iHeight**

图像的高度, 调用图像处理函数后, 该变量可能被动态修改, 来指示处理后的图像尺寸

INT **iWidthZoomSw**

软件缩放的宽度, 不需要进行软件裁剪的图像, 此变量设置为0.

INT **iHeightZoomSw**

软件缩放的高度, 不需要进行软件裁剪的图像, 此变量设置为0.

BOOL **bIsTrigger**

指示是否为触发帧

**UINT uiTimeStamp**

该帧的采集时间，单位0.1毫秒

**UINT uiExpTime**

当前图像的曝光值，单位为微秒us

**float fAnalogGain**

当前图像的模拟增益倍数

**INT iGamma**

该帧图像的伽马设定值，仅当LUT模式为动态参数生成时有效，其余模式下为-1

**INT iContrast**

该帧图像的对比度设定值，仅当LUT模式为动态参数生成时有效，其余模式下为-1

**INT iSaturation**

该帧图像的饱和度设定值，对于黑白相机无意义，为0

**float fRgain**

该帧图像处理的红色数字增益倍数，对于黑白相机无意义，为1

**float fGgain**

该帧图像处理的绿色数字增益倍数，对于黑白相机无意义，为1

**float fBgain**

该帧图像处理的蓝色数字增益倍数，对于黑白相机无意义，为1

## 详细描述

---

图像帧头信息

示例:

`grab.cpp`, `grab_ex.cpp`, `roi.cpp`, `soft_trigger.cpp`,  
`win_basic.cpp`, `win_callback.cpp`, 以及 `win_grabber.cpp`.

---

该结构体的文档由以下文件生成:

- `CameraDefine.h`

# MVSDK API

## tSdkFrameSpeed结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iIndex**

帧率索引号，一般0对应于低速模式，1对应于普通模式，2对应于高速模式

char **acDescription** [32]

描述信息

## 详细描述

---

相机帧率描述信息

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tSdkFrameStatistic结 构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iTotal**

当前采集的总帧数（包括错误帧）

INT **iCapture**

当前采集的有效帧的数量

INT **iLost**

当前丢帧的数量

## 详细描述

---

帧统计信息

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tSdkGrabberStat结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

int **Width**

帧宽度

int **Height**

帧高度

int **Disp**

显示帧数量

int **Capture**

采集的有效帧的数量

int **Lost**

丢帧的数量

int **Error**

错帧的数量

float **DispFps**

显示帧率

float **CapFps**

捕获帧率



## 详细描述

---

Grabber统计信息

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tSdkImageResolution 结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

#### INT iIndex

索引号, [0,N]表示预设的分辨率(N 为预设分辨率的最大个数, 一般不超过20),0XFF 表示自定义分辨率(ROI)

#### char acDescription [32]

该分辨率的描述信息。仅预设分辨率时该信息有效。自定义分辨率可忽略该信息

#### UINT uBinSumMode

BIN(求和)的模式, 范围不能超过  
tSdkResolutionRange.uBinSumModeMask

#### UINT uBinAverageMode

BIN(求均值)的模式, 范围不能超过  
tSdkResolutionRange.uBinAverageModeMask

#### UINT uSkipMode

是否SKIP的尺寸, 为0表示禁止SKIP模式, 范围不能超过  
tSdkResolutionRange.uSkipModeMask

#### UINT uResampleMask

硬件重采样的掩码

#### INT iHOffsetFOV

采集视场相对于Sensor最大视场左上角的水平偏移

INT **iVOffsetFOV**

采集视场相对于Sensor最大视场左上角的垂直偏移

INT **iWidthFOV**

采集视场的宽度

INT **iHeightFOV**

采集视场的高度

INT **iWidth**

相机最终输出的图像的宽度

INT **iHeight**

相机最终输出的图像的高度

INT **iWidthZoomHd**

硬件缩放的宽度,不需要进行此操作的分辨率, 此变量设置为0.

INT **iHeightZoomHd**

硬件缩放的高度,不需要进行此操作的分辨率, 此变量设置为0.

INT **iWidthZoomSw**

软件缩放的宽度,不需要进行此操作的分辨率, 此变量设置为0.

INT **iHeightZoomSw**

软件缩放的高度,不需要进行此操作的分辨率, 此变量设置为0.

## 详细描述

---

相机的分辨率描述

示例:  
[roi.cpp](#).

---

该结构体的文档由以下文件生成:

- [CameraDefine.h](#)

# MVSDK API

## tSdkIspCapacity结构体参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

BOOL **bMonoSensor**

表示该型号相机是否为黑白相机,如果是黑白相机, 则颜色相关的功能都无法调节

BOOL **bWbOnce**

表示该型号相机是否支持手动白平衡功能

BOOL **bAutoWb**

表示该型号相机是否支持自动白平衡功能

BOOL **bAutoExposure**

表示该型号相机是否支持自动曝光功能

BOOL **bManualExposure**

表示该型号相机是否支持手动曝光功能

BOOL **bAntiFlick**

表示该型号相机是否支持抗频闪功能

BOOL **bDeviceisp**

表示该型号相机是否支持硬件ISP功能

BOOL **bForceUseDeviceisp**

bDeviceISP和bForceUseDeviceISP同时为TRUE时，表示强制只用硬件ISP，不可取消。

BOOL **bZoomHD**

相机硬件是否支持图像缩放输出(只能是缩小)。

---

## 详细描述

---

ISP模块的使能信息

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tSdkMediaType结构体参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iIndex**

格式种类编号

char **acDescription** [32]

描述信息

UINT **iMediaType**

对应的图像格式编码，如

CAMERA\_MEDIA\_TYPE\_BAYGR8。

## 详细描述

---

相机输出的图像数据格式

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tSdkPackLength结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iIndex**

分包大小索引号

char **acDescription** [32]

对应的描述信息

UINT **iPackSize**

包大小

## 详细描述

---

传输分包大小描述(针对某些网络相机有效)

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tSdkPresetLut结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iIndex**

索引

char **acDescription** [32]

描述信息

## 详细描述

---

预设的LUT表描述

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tSdkResolutionRange e结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iHeightMax**  
图像最大高度

INT **iHeightMin**  
图像最小高度

INT **iWidthMax**  
图像最大宽度

INT **iWidthMin**  
图像最小宽度

UINT **uSkipModeMask**  
SKIP模式掩码，为0，表示不支持SKIP。bit0为1,表示支持  
SKIP 2x2 ;bit1为1， 表示支持SKIP 3x3....

UINT **uBinSumModeMask**  
BIN(求和)模式掩码，为0，表示不支持BIN 。bit0为1,表示支  
持BIN 2x2 ;bit1为1， 表示支持BIN 3x3....

UINT **uBinAverageModeMask**  
BIN(求均值)模式掩码，为0，表示不支持BIN 。bit0为1,表示  
支持BIN 2x2 ;bit1为1， 表示支持BIN 3x3....

**UINT uResampleMask**  
硬件重采样的掩码

## 详细描述

---

相机的分辨率设定范围，可用于构件UI

---

该结构体的文档由以下文件生成：

- **CameraDefine.h**

# MVSDK API

## tSdkTrigger结构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iIndex**

模式索引号

char **acDescription** [32]

该模式的描述信息

## 详细描述

---

触发模式描述

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## tSharpnessRange结 构体 参考

成员变量

```
#include <CameraDefine.h>
```

### 成员变量

INT **iMin**  
最小值

INT **iMax**  
最大值

## 详细描述

---

锐化的设定范围

---

该结构体的文档由以下文件生成:

- **CameraDefine.h**

# MVSDK API

## 结构体索引

s | t

<b>s</b>	tGammaRange tRgbGainRange tSaturationRange tSdkAeAlgorithm tSdkBayerDecodeAlgorithm tContrastRange	tSdkCameraDevInfo tSdkColorTemperatureDes tSdkExpose tSdkFrameEvent_	tSdkFrameStatistic tSdkGrabberStat tSdkImageResolut tSdkIspCapacity tSdkMediaType tSdkPackLength
----------	---	---	---

s | t

# MVSDK API

---

这里列出了所有文档化的结构体和联合体的成员变量，并附带结构或联合所属的文件：

## - a -

- acDescription : **tSdkAeAlgorithm** ,  
**tSdkBayerDecodeAlgorithm** , **tSdkColorTemperatureDes** ,  
**tSdkFrameSpeed** , **tSdkImageResolution** , **tSdkMediaType** ,  
**tSdkPackLength** , **tSdkPresetLut** , **tSdkTrigger**
- acDriverVersion : **tSdkCameraDevInfo**
- acFriendlyName : **tSdkCameraDevInfo**
- acLinkName : **tSdkCameraDevInfo**
- acPortType : **tSdkCameraDevInfo**
- acProductName : **tSdkCameraDevInfo**
- acProductSeries : **tSdkCameraDevInfo**
- acSensorType : **tSdkCameraDevInfo**
- acSn : **tSdkCameraDevInfo**

## - b -

- bAntiFlick : **tSdkIspCapacity**
- bAutoExposure : **tSdkIspCapacity**
- bAutoWb : **tSdkIspCapacity**
- bDeviceIsp : **tSdkIspCapacity**
- bForceUseDeviceIsp : **tSdkIspCapacity**
- bIsTrigger : **tSdkFrameHead**
- bManualExposure : **tSdkIspCapacity**
- bMonoSensor : **tSdkIspCapacity**
- bParamInDevice : **tSdkCameraCapbility**
- bWbOnce : **tSdkIspCapacity**
- bZoomHD : **tSdkIspCapacity**

## - c -

- CapFps : **tSdkGrabberStat**

- Capture : **tSdkGrabberStat**

- d -

- Disp : **tSdkGrabberStat**
- DispFps : **tSdkGrabberStat**

- e -

- Error : **tSdkGrabberStat**

- f -

- fAnalogGain : **tSdkFrameHead**
- fAnalogGainStep : **tSdkExpose**
- fBgain : **tSdkFrameHead**
- fGgain : **tSdkFrameHead**
- fRgain : **tSdkFrameHead**

- h -

- head : **sCameraFrame**
- Height : **tSdkGrabberStat**

- i -

- iAeAlmHdDesc : **tSdkCameraCapbility**
- iAeAlmSwDesc : **tSdkCameraCapbility**
- iBayerDecAlmHdDesc : **tSdkCameraCapbility**
- iBayerDecAlmSwDesc : **tSdkCameraCapbility**
- iBGainMax : **tRgbGainRange**
- iBGainMin : **tRgbGainRange**
- iCapture : **tSdkFrameStatistic**
- iClrTempDesc : **tSdkCameraCapbility**
- iContrast : **tSdkFrameHead**
- iFrameSpeedDesc : **tSdkCameraCapbility**
- iGamma : **tSdkFrameHead**

- iGGainMax : **tRgbGainRange**
- iGGainMin : **tRgbGainRange**
- iHeight : **tSdkFrameHead , tSdkImageResolution**
- iHeightFOV : **tSdkImageResolution**
- iHeightMax : **tSdkResolutionRange**
- iHeightMin : **tSdkResolutionRange**
- iHeightZoomHd : **tSdkImageResolution**
- iHeightZoomSw : **tSdkFrameHead , tSdkImageResolution**
- iOffsetFOV : **tSdkImageResolution**
- iImageSizeDesc : **tSdkCameraCapbility**
- iIndex : **tSdkAeAlgorithm , tSdkBayerDecodeAlgorithm , tSdkColorTemperatureDes , tSdkFrameSpeed , tSdkImageResolution , tSdkMediaType , tSdkPackLength , tSdkPresetLut , tSdkTrigger**
- iInputIoCounts : **tSdkCameraCapbility**
- iLost : **tSdkFrameStatistic**
- iMax : **tContrastRange , tGammaRange , tSaturationRange , tSharpnessRange**
- iMediaTypdeDesc : **tSdkCameraCapbility**
- iMediaType : **tSdkMediaType**
- iMin : **tContrastRange , tGammaRange , tSaturationRange , tSharpnessRange**
- iOutputIoCounts : **tSdkCameraCapbility**
- iPackLenDesc : **tSdkCameraCapbility**
- iPackSize : **tSdkPackLength**
- iPresetLut : **tSdkCameraCapbility**
- iRGainMax : **tRgbGainRange**
- iRGainMin : **tRgbGainRange**
- iSaturation : **tSdkFrameHead**
- iTotla : **tSdkFrameStatistic**
- iTriggerDesc : **tSdkCameraCapbility**
- iUserDataMaxLen : **tSdkCameraCapbility**
- iOffsetFOV : **tSdkImageResolution**
- iWidth : **tSdkFrameHead , tSdkImageResolution**
- iWidthFOV : **tSdkImageResolution**
- iWidthMax : **tSdkResolutionRange**
- iWidthMin : **tSdkResolutionRange**
- iWidthZoomHd : **tSdkImageResolution**
- iWidthZoomSw : **tSdkFrameHead , tSdkImageResolution**

- | -

- Lost : **tSdkGrabberStat**

- p -

- pAeAlmHdDesc : **tSdkCameraCapbility**
- pAeAlmSwDesc : **tSdkCameraCapbility**
- pBayerDecAlmHdDesc : **tSdkCameraCapbility**
- pBayerDecAlmSwDesc : **tSdkCameraCapbility**
- pBuffer : **sCameraFrame**
- pClrTempDesc : **tSdkCameraCapbility**
- pFrameSpeedDesc : **tSdkCameraCapbility**
- pImageSizeDesc : **tSdkCameraCapbility**
- pMediaTypeDesc : **tSdkCameraCapbility**
- pPackLenDesc : **tSdkCameraCapbility**
- pPresetLutDesc : **tSdkCameraCapbility**
- pTriggerDesc : **tSdkCameraCapbility**

- s -

- sContrastRange : **tSdkCameraCapbility**
- sExposeDesc : **tSdkCameraCapbility**
- sGammaRange : **tSdkCameraCapbility**
- slspCapacity : **tSdkCameraCapbility**
- sResolutionRange : **tSdkCameraCapbility**
- sRgbGainRange : **tSdkCameraCapbility**
- sSaturationRange : **tSdkCameraCapbility**
- sSharpnessRange : **tSdkCameraCapbility**

- t -

- TimeStampH : **tSdkFrameEvent\_**
- TimeStampL : **tSdkFrameEvent\_**

- u -

- uBinAverageMode : **tSdkImageResolution**
- uBinAverageModeMask : **tSdkResolutionRange**
- uBinSumMode : **tSdkImageResolution**
- uBinSumModeMask : **tSdkResolutionRange**
- uBytes : **tSdkFrameHead**
- uFrameID : **tSdkFrameEvent\_**
- uHeight : **tSdkFrameEvent\_**
- uiAnalogGainMax : **tSdkExpose**
- uiAnalogGainMin : **tSdkExpose**
- uiExposeTimeMax : **tSdkExpose**
- uiExposeTimeMin : **tSdkExpose**
- uiExpTime : **tSdkFrameHead**
- uiMediaType : **tSdkFrameHead**
- ulnstance : **tSdkCameraDevInfo**
- uiTargetMax : **tSdkExpose**
- uiTargetMin : **tSdkExpose**
- uiTimeStamp : **tSdkFrameHead**
- uPixelFormat : **tSdkFrameEvent\_**
- uResampleMask : **tSdkImageResolution , tSdkResolutionRange**
- uSkipMode : **tSdkImageResolution**
- uSkipModeMask : **tSdkResolutionRange**
- uStatus : **tSdkFrameEvent\_**
- uType : **tSdkFrameEvent\_**
- uWidth : **tSdkFrameEvent\_**

**- W -**

- Width : **tSdkGrabberStat**

# MVSDK API

---

## - a -

- acDescription : **tSdkAeAlgorithm** ,  
**tSdkBayerDecodeAlgorithm** , **tSdkColorTemperatureDes** ,  
**tSdkFrameSpeed** , **tSdkImageResolution** , **tSdkMediaType** ,  
**tSdkPackLength** , **tSdkPresetLut** , **tSdkTrigger**
- acDriverVersion : **tSdkCameraDevInfo**
- acFriendlyName : **tSdkCameraDevInfo**
- acLinkName : **tSdkCameraDevInfo**
- acPortType : **tSdkCameraDevInfo**
- acProductName : **tSdkCameraDevInfo**
- acProductSeries : **tSdkCameraDevInfo**
- acSensorType : **tSdkCameraDevInfo**
- acSn : **tSdkCameraDevInfo**

## - b -

- bAntiFlick : **tSdkIspCapacity**
- bAutoExposure : **tSdkIspCapacity**
- bAutoWb : **tSdkIspCapacity**
- bDeviceIsp : **tSdkIspCapacity**
- bForceUseDeviceIsp : **tSdkIspCapacity**
- bIsTrigger : **tSdkFrameHead**
- bManualExposure : **tSdkIspCapacity**
- bMonoSensor : **tSdkIspCapacity**
- bParamInDevice : **tSdkCameraCapbility**
- bWbOnce : **tSdkIspCapacity**
- bZoomHD : **tSdkIspCapacity**

## - c -

- CapFps : **tSdkGrabberStat**
- Capture : **tSdkGrabberStat**

- d -

- Disp : **tSdkGrabberStat**
- DispFps : **tSdkGrabberStat**

- e -

- Error : **tSdkGrabberStat**

- f -

- fAnalogGain : **tSdkFrameHead**
- fAnalogGainStep : **tSdkExpose**
- fBgain : **tSdkFrameHead**
- fGgain : **tSdkFrameHead**
- fRgain : **tSdkFrameHead**

- h -

- head : **sCameraFrame**
- Height : **tSdkGrabberStat**

- i -

- iAeAlmHdDesc : **tSdkCameraCapbility**
- iAeAlmSwDesc : **tSdkCameraCapbility**
- iBayerDecAlmHdDesc : **tSdkCameraCapbility**
- iBayerDecAlmSwDesc : **tSdkCameraCapbility**
- iBGainMax : **tRgbGainRange**
- iBGainMin : **tRgbGainRange**
- iCapture : **tSdkFrameStatistic**
- iClrTempDesc : **tSdkCameraCapbility**
- iContrast : **tSdkFrameHead**
- iFrameSpeedDesc : **tSdkCameraCapbility**
- iGamma : **tSdkFrameHead**
- iGGainMax : **tRgbGainRange**
- iGGainMin : **tRgbGainRange**

- iHeight : **tSdkFrameHead** , **tSdkImageResolution**
- iHeightFOV : **tSdkImageResolution**
- iHeightMax : **tSdkResolutionRange**
- iHeightMin : **tSdkResolutionRange**
- iHeightZoomHd : **tSdkImageResolution**
- iHeightZoomSw : **tSdkFrameHead** , **tSdkImageResolution**
- iOffsetFOV : **tSdkImageResolution**
- iImageSizeDesc : **tSdkCameraCapbility**
- iIndex : **tSdkAeAlgorithm** , **tSdkBayerDecodeAlgorithm** ,  
**tSdkColorTemperatureDes** , **tSdkFrameSpeed** ,  
**tSdkImageResolution** , **tSdkMediaType** , **tSdkPackLength** ,  
**tSdkPresetLut** , **tSdkTrigger**
- iInputIoCounts : **tSdkCameraCapbility**
- iLost : **tSdkFrameStatistic**
- iMax : **tContrastRange** , **tGammaRange** , **tSaturationRange** ,  
**tSharpnessRange**
- iMediaTypdeDesc : **tSdkCameraCapbility**
- iMediaType : **tSdkMediaType**
- iMin : **tContrastRange** , **tGammaRange** , **tSaturationRange** ,  
**tSharpnessRange**
- iOutputIoCounts : **tSdkCameraCapbility**
- iPackLenDesc : **tSdkCameraCapbility**
- iPackSize : **tSdkPackLength**
- iPresetLut : **tSdkCameraCapbility**
- iRGainMax : **tRgbGainRange**
- iRGainMin : **tRgbGainRange**
- iSaturation : **tSdkFrameHead**
- iTTotal : **tSdkFrameStatistic**
- iTriggerDesc : **tSdkCameraCapbility**
- iUserDataMaxLen : **tSdkCameraCapbility**
- iVOffsetFOV : **tSdkImageResolution**
- iWidth : **tSdkFrameHead** , **tSdkImageResolution**
- iWidthFOV : **tSdkImageResolution**
- iWidthMax : **tSdkResolutionRange**
- iWidthMin : **tSdkResolutionRange**
- iWidthZoomHd : **tSdkImageResolution**
- iWidthZoomSw : **tSdkFrameHead** , **tSdkImageResolution**

- Lost : **tSdkGrabberStat**

- p -

- pAeAlmHdDesc : **tSdkCameraCapbility**
- pAeAlmSwDesc : **tSdkCameraCapbility**
- pBayerDecAlmHdDesc : **tSdkCameraCapbility**
- pBayerDecAlmSwDesc : **tSdkCameraCapbility**
- pBuffer : **sCameraFrame**
- pClrTempDesc : **tSdkCameraCapbility**
- pFrameSpeedDesc : **tSdkCameraCapbility**
- pImageSizeDesc : **tSdkCameraCapbility**
- pMediaTypeDesc : **tSdkCameraCapbility**
- pPackLenDesc : **tSdkCameraCapbility**
- pPresetLutDesc : **tSdkCameraCapbility**
- pTriggerDesc : **tSdkCameraCapbility**

- s -

- sContrastRange : **tSdkCameraCapbility**
- sExposeDesc : **tSdkCameraCapbility**
- sGammaRange : **tSdkCameraCapbility**
- sIspCapacity : **tSdkCameraCapbility**
- sResolutionRange : **tSdkCameraCapbility**
- sRgbGainRange : **tSdkCameraCapbility**
- sSaturationRange : **tSdkCameraCapbility**
- sSharpnessRange : **tSdkCameraCapbility**

- t -

- TimeStampH : **tSdkFrameEvent\_**
- TimeStampL : **tSdkFrameEvent\_**

- u -

- uBinAverageMode : **tSdkImageResolution**
- uBinAverageModeMask : **tSdkResolutionRange**
- uBinSumMode : **tSdkImageResolution**

- uBinSumModeMask : **tSdkResolutionRange**
- uBytes : **tSdkFrameHead**
- uFrameID : **tSdkFrameEvent\_**
- uHeight : **tSdkFrameEvent\_**
- uiAnalogGainMax : **tSdkExpose**
- uiAnalogGainMin : **tSdkExpose**
- uiExposeTimeMax : **tSdkExpose**
- uiExposeTimeMin : **tSdkExpose**
- uiExpTime : **tSdkFrameHead**
- uiMediaType : **tSdkFrameHead**
- ulnstance : **tSdkCameraDevInfo**
- uiTargetMax : **tSdkExpose**
- uiTargetMin : **tSdkExpose**
- uiTimeStamp : **tSdkFrameHead**
- uPixelFormat : **tSdkFrameEvent\_**
- uResampleMask : **tSdkImageResolution , tSdkResolutionRange**
- uSkipMode : **tSdkImageResolution**
- uSkipModeMask : **tSdkResolutionRange**
- uStatus : **tSdkFrameEvent\_**
- uType : **tSdkFrameEvent\_**
- uWidth : **tSdkFrameEvent\_**

**- W -**

- Width : **tSdkGrabberStat**

# MVSDK API

## 文件列表

这里列出了所有文档化的文件，并附带简要说明：

<b>CameraApi.h</b>	
<b>CameraDefine.h</b>	
<b>CameraGrabber.h</b>	
<b>CameralImage.h</b>	
<b>CameraStatus.h</b>	
<b>CameraZoomTool.h</b>	

# MVSDK API

Include >

## CameraApi.h

```
1 #ifndef _MVCAMAPI_H_
2 #define _MVCAMAPI_H_
3
4
5 #ifdef DLL_EXPORT
6 #define MVSDK_API extern "C" __declspec(dllexport)
7 #else
8 #define MVSDK_API extern "C" __declspec(dllimport)
9 #endif
10
11 #include "CameraDefine.h"
12 #include "CameraStatus.h"
13
14 MVSDK_API CameraSdkStatus __stdcall CameraSdkInit(
15     int      iLanguageSel
16 );
17
18 MVSDK_API CameraSdkStatus __stdcall CameraSetSysOption(
19     char const* optionName,
20     char const* value
21 );
22
23 MVSDK_API CameraSdkStatus __stdcall CameraEnumerateDevice(
24     tSdkCameraDevInfo* pCameraList,
25     INT*               piNums
26 );
27
28 MVSDK_API INT __stdcall CameraEnumerateDeviceEx(
29 );
30
31 MVSDK_API CameraSdkStatus __stdcall CameraIsOpened(
32     tSdkCameraDevInfo* pCameraInfo,
33     BOOL*              pOpened
34 );
35
36 MVSDK_API CameraSdkStatus __stdcall CameraInit(
37     tSdkCameraDevInfo* pCameraInfo,
38     int                emParamLoadMode,
39     int                emTeam,
40     CameraHandle*      pCameraHandle
41 );
42
43 MVSDK_API CameraSdkStatus __stdcall CameraInitEx(
44     int                iDeviceIndex,
45     int                emParamLoadMode,
46     int                emTeam,
47     CameraHandle*      pCameraHandle
48 );
```

```
131 );
132
144 MVSDK_API CameraSdkStatus __stdcall CameraInitEx2(
145     char* CameraName,
146     CameraHandle *pCameraHandle
147 );
148
164 MVSDK_API CameraSdkStatus __stdcall CameraSetCallbackFunction(
165     CameraHandle hCamera,
166     CAMERA_SNAP_PROC pCallBack,
167     PVOID pContext,
168     CAMERA_SNAP_PROC* pCallbackOld
169 );
170
180 MVSDK_API CameraSdkStatus __stdcall CameraUnInit(
181     CameraHandle hCamera
182 );
183
195 MVSDK_API CameraSdkStatus __stdcall CameraGetInformation(
196     CameraHandle hCamera,
197     char** pBuffer
198 );
199
215 MVSDK_API CameraSdkStatus __stdcall CameraImageProcess(
216     CameraHandle hCamera,
217     BYTE* pbyIn,
218     BYTE* pbyOut,
219     tSdkFrameHead* pFrInfo
220 );
221
241 MVSDK_API CameraSdkStatus __stdcall CameraImageProcessEx(
242     CameraHandle hCamera,
243     BYTE* pbyIn,
244     BYTE* pbyOut,
245     tSdkFrameHead* pFrInfo,
246     UINT uOutFormat,
247     UINT uReserved
248 );
249
261 MVSDK_API CameraSdkStatus __stdcall CameraDisplayInit(
262     CameraHandle hCamera,
263     HWND hWndDisplay
264 );
265
279 MVSDK_API CameraSdkStatus __stdcall CameraDisplayRGB24(
280     CameraHandle hCamera,
281     BYTE* pFrameBuffer,
282     tSdkFrameHead* pFrInfo
283 );
284
296 MVSDK_API CameraSdkStatus __stdcall CameraSetDisplayMode(
297     CameraHandle hCamera,
298     INT iMode
299 );
```

```
300
314 MVSDK_API CameraSdkStatus __stdcall CameraSetDisplayOffset(
315     CameraHandle    hCamera,
316     int             iOffsetX,
317     int             iOffsetY
318 );
319
333 MVSDK_API CameraSdkStatus __stdcall CameraSetDisplaySize(
334     CameraHandle    hCamera,
335     INT            iWidth,
336     INT            iHeight
337 );
338
356 MVSDK_API CameraSdkStatus __stdcall CameraGetImageBuffer(
357     CameraHandle    hCamera,
358     tSdkFrameHead* pFrameInfo,
359     BYTE**          pbyBuffer,
360     UINT            wTimes
361 );
362
380 MVSDK_API unsigned char* __stdcall CameraGetImageBufferEx(
381     CameraHandle    hCamera,
382     INT*           piWidth,
383     INT*           piHeight,
384     UINT            wTimes
385 );
386
406 MVSDK_API CameraSdkStatus __stdcall CameraSnapToBuffer(
407     CameraHandle    hCamera,
408     tSdkFrameHead* pFrameInfo,
409     BYTE**          pbyBuffer,
410     UINT            wTimes
411 );
412
428 MVSDK_API CameraSdkStatus __stdcall CameraSnapJpegToFile(
429     CameraHandle    hCamera,
430     char const*    lpszFileName,
431     BYTE           byQuality,
432     UINT            wTimes
433 );
434
446 MVSDK_API CameraSdkStatus __stdcall CameraReleaseImageBuffer(
447     CameraHandle    hCamera,
448     BYTE*           pbyBuffer
449 );
450
460 MVSDK_API CameraSdkStatus __stdcall CameraPlay(
461     CameraHandle   hCamera
462 );
463
473 MVSDK_API CameraSdkStatus __stdcall CameraPause(
474     CameraHandle   hCamera
475 );
476
```

```
486 MVSDK_API CameraSdkStatus __stdcall CameraStop(
487     CameraHandle hCamera
488 );
489
509 MVSDK_API CameraSdkStatus __stdcall CameraInitRecord(
510     CameraHandle    hCamera,
511     int             iFormat,
512     char*          pcSavePath,
513     BOOL            b2GLimit,
514     DWORD           dwQuality,
515     int             iFrameRate
516 );
517
527 MVSDK_API CameraSdkStatus __stdcall CameraStopRecord(
528     CameraHandle    hCamera
529 );
530
544 MVSDK_API CameraSdkStatus __stdcall CameraPushFrame(
545     CameraHandle    hCamera,
546     BYTE*          pbyImageBuffer,
547     tSdkFrameHead* pFrInfo
548 );
549
571 MVSDK_API CameraSdkStatus __stdcall CameraSaveImage(
572     CameraHandle    hCamera,
573     char*          lpszFileName,
574     BYTE*          pbyImageBuffer,
575     tSdkFrameHead* pFrInfo,
576     UINT            byFileType,
577     BYTE            byQuality
578 );
579
605 MVSDK_API CameraSdkStatus __stdcall CameraSaveImageEx(
606     CameraHandle    hCamera,
607     char*          lpszFileName,
608     BYTE*          pbyImageBuffer,
609     UINT            uImageFormat,
610     int             iWidth,
611     int             iHeight,
612     UINT            byFileType,
613     BYTE            byQuality
614 );
615
627 MVSDK_API CameraSdkStatus __stdcall CameraGetImageResolution(
628     CameraHandle    hCamera,
629     tSdkImageResolution* psCurVideoSize
630 );
631
661 MVSDK_API CameraSdkStatus __stdcall CameraGetImageResolutionEx(
662     CameraHandle    hCamera,
663     int*            iIndex,
664     char*          acDescription[32],
665     int*            Mode,
666     UINT*           ModeSize,
```

```
667     int*           x,
668     int*           y,
669     int*           width,
670     int*           height,
671     int*           ZoomWidth,
672     int*           ZoomHeight
673 );
674
686 MVSDK_API CameraSdkStatus __stdcall CameraSetImageResolution(
687     CameraHandle      hCamera,
688     tSdkImageResolution* pImageResolution
689 );
690
718 MVSDK_API CameraSdkStatus __stdcall CameraSetImageResolutionEx(
719     CameraHandle      hCamera,
720     int               iIndex,
721     int               Mode,
722     UINT              ModeSize,
723     int               x,
724     int               y,
725     int               width,
726     int               height,
727     int               ZoomWidth,
728     int               ZoomHeight
729 );
730
744 MVSDK_API CameraSdkStatus __stdcall CameraGetMediaType(
745     CameraHandle      hCamera,
746     INT*              piMediaType
747 );
748
762 MVSDK_API CameraSdkStatus __stdcall CameraSetMediaType(
763     CameraHandle      hCamera,
764     INT               iMediaType
765 );
766
778 MVSDK_API CameraSdkStatus __stdcall CameraGetRawMaxAvailBits(
779     CameraHandle      hCamera,
780     int*              pMaxAvailBits
781 );
782
794 MVSDK_API CameraSdkStatus __stdcall CameraSetRawStartBit(
795     CameraHandle      hCamera,
796     int               startBit
797 );
798
810 MVSDK_API CameraSdkStatus __stdcall CameraGetRawStartBit(
811     CameraHandle      hCamera,
812     int*              startBit
813 );
814
826 MVSDK_API CameraSdkStatus __stdcall CameraSetAeState(
827     CameraHandle      hCamera,
828     BOOL              bAeState
```

```
829 );
830
842 MVSDK_API CameraSdkStatus __stdcall CameraGetAeState(
843     CameraHandle    hCamera,
844     BOOL*          pAestate
845 );
846
858 MVSDK_API CameraSdkStatus __stdcall CameraSetSharpness(
859     CameraHandle    hCamera,
860     int             iSharpness
861 );
862
874 MVSDK_API CameraSdkStatus __stdcall CameraGetSharpness(
875     CameraHandle    hCamera,
876     int*           piSharpness
877 );
878
890 MVSDK_API CameraSdkStatus __stdcall CameraSetLutMode(
891     CameraHandle    hCamera,
892     int             emLutMode
893 );
894
906 MVSDK_API CameraSdkStatus __stdcall CameraGetLutMode(
907     CameraHandle    hCamera,
908     int*           pemLutMode
909 );
910
924 MVSDK_API CameraSdkStatus __stdcall CameraSelectLutPreset(
925     CameraHandle    hCamera,
926     int             iSel
927 );
928
940 MVSDK_API CameraSdkStatus __stdcall CameraGetLutPresetSel(
941     CameraHandle    hCamera,
942     int*           piSel
943 );
944
960 MVSDK_API CameraSdkStatus __stdcall CameraSetCustomLut(
961     CameraHandle    hCamera,
962     int             iChannel,
963     USHORT*         pLut
964 );
965
979 MVSDK_API CameraSdkStatus __stdcall CameraGetCustomLut(
980     CameraHandle    hCamera,
981     int             iChannel,
982     USHORT*         pLut
983 );
984
998 MVSDK_API CameraSdkStatus __stdcall CameraGetCurrentLut(
999     CameraHandle    hCamera,
1000    int             iChannel,
1001    USHORT*         pLut
1002 );
```

```
1003
1015 MVSDK_API CameraSdkStatus __stdcall CameraSetWbMode(
1016     CameraHandle    hCamera,
1017     BOOL           bAuto
1018 );
1019
1031 MVSDK_API CameraSdkStatus __stdcall CameraGetWbMode(
1032     CameraHandle    hCamera,
1033     BOOL*          pbAuto
1034 );
1035
1049 MVSDK_API CameraSdkStatus __stdcall CameraSetPresetClrTemp(
1050     CameraHandle    hCamera,
1051     int            iSel
1052 );
1053
1065 MVSDK_API CameraSdkStatus __stdcall CameraGetPresetClrTemp(
1066     CameraHandle    hCamera,
1067     int*           piSel
1068 );
1069
1087 MVSDK_API CameraSdkStatus __stdcall CameraSetUserClrTempGain(
1088     CameraHandle   hCamera,
1089     int           iRgain,
1090     int           iGgain,
1091     int           iBgain
1092 );
1093
1109 MVSDK_API CameraSdkStatus __stdcall Camera GetUserClrTempGain(
1110     CameraHandle   hCamera,
1111     int*           piRgain,
1112     int*           piGgain,
1113     int*           piBgain
1114 );
1115
1129 MVSDK_API CameraSdkStatus __stdcall CameraSetUserClrTempMatrix(
1130     CameraHandle   hCamera,
1131     float*         pMatrix
1132 );
1133
1145 MVSDK_API CameraSdkStatus __stdcall Camera GetUserClrTempMatrix(
1146     CameraHandle   hCamera,
1147     float*         pMatrix
1148 );
1149
1169 MVSDK_API CameraSdkStatus __stdcall CameraSetClrTempMode(
1170     CameraHandle   hCamera,
1171     int            iMode
1172 );
1173
1185 MVSDK_API CameraSdkStatus __stdcall CameraGetClrTempMode(
1186     CameraHandle   hCamera,
1187     int*           pimode
1188 );
```

```
1189 MVSDK_API CameraSdkStatus __stdcall CameraSetOnceWB(
1190     CameraHandle     hCamera
1201 );
1202
1212 MVSDK_API CameraSdkStatus __stdcall CameraSetOnceBB(
1213     CameraHandle     hCamera
1214 );
1215
1227 MVSDK_API CameraSdkStatus __stdcall CameraSetAeTarget(
1228     CameraHandle     hCamera,
1229     int              iAeTarget
1230 );
1231
1243 MVSDK_API CameraSdkStatus __stdcall CameraGetAeTarget(
1244     CameraHandle     hCamera,
1245     int*             piAeTarget
1246 );
1247
1261 MVSDK_API CameraSdkStatus __stdcall CameraSetAeExposureRange(
1262     CameraHandle     hCamera,
1263     double           fMinExposureTime,
1264     double           fMaxExposureTime
1265 );
1266
1280 MVSDK_API CameraSdkStatus __stdcall CameraGetAeExposureRange(
1281     CameraHandle     hCamera,
1282     double*          fMinExposureTime,
1283     double*          fMaxExposureTime
1284 );
1285
1299 MVSDK_API CameraSdkStatus __stdcall CameraSetAeAnalogGainRange(
1300     CameraHandle     hCamera,
1301     int              iMinAnalogGain,
1302     int              iMaxAnalogGain
1303 );
1304
1318 MVSDK_API CameraSdkStatus __stdcall CameraGetAeAnalogGainRange(
1319     CameraHandle     hCamera,
1320     int*             iMinAnalogGain,
1321     int*             iMaxAnalogGain
1322 );
1323
1335 MVSDK_API CameraSdkStatus __stdcall CameraSetAeThreshold(
1336     CameraHandle     hCamera,
1337     int              iThreshold
1338 );
1339
1351 MVSDK_API CameraSdkStatus __stdcall CameraGetAeThreshold(
1352     CameraHandle     hCamera,
1353     int*             iThreshold
1354 );
1355
1369 MVSDK_API CameraSdkStatus __stdcall CameraSetExposureTime(
```

```
1370     CameraHandle    hCamera,
1371     double          fExposureTime
1372 );
1373
1387 MVSDK_API CameraSdkStatus __stdcall CameraGetExposureLineTime(
1388     CameraHandle    hCamera,
1389     double*         pfLineTime
1390 );
1391
1405 MVSDK_API CameraSdkStatus __stdcall CameraGetExposureTime(
1406     CameraHandle    hCamera,
1407     double*         pfExposureTime
1408 );
1409
1425 MVSDK_API CameraSdkStatus __stdcall CameraGetExposureTimeRange(
1426     CameraHandle    hCamera,
1427     double*         pfMin,
1428     double*         pfMax,
1429     double*         pfStep
1430 );
1431
1447 MVSDK_API CameraSdkStatus __stdcall CameraSetMultiExposureTime(
1448     CameraHandle    hCamera,
1449     int              index,
1450     double           fExposureTime
1451 );
1452
1466 MVSDK_API CameraSdkStatus __stdcall CameraGetMultiExposureTime(
1467     CameraHandle    hCamera,
1468     int              index,
1469     double*         fExposureTime
1470 );
1471
1483 MVSDK_API CameraSdkStatus __stdcall CameraSetMultiExposureCount(
1484     CameraHandle    hCamera,
1485     int              count
1486 );
1487
1499 MVSDK_API CameraSdkStatus __stdcall CameraGetMultiExposureCount(
1500     CameraHandle    hCamera,
1501     int*             count
1502 );
1503
1515 MVSDK_API CameraSdkStatus __stdcall CameraGetMultiExposureMaxCount(
1516     CameraHandle    hCamera,
1517     int*             max_count
1518 );
1519
1533 MVSDK_API CameraSdkStatus __stdcall CameraSetAnalogGain(
1534     CameraHandle    hCamera,
1535     INT              iAnalogGain
1536 );
1537
1551 MVSDK_API CameraSdkStatus __stdcall CameraGetAnalogGain(
```

```
1552     CameraHandle    hCamera,
1553     INT*           piAnalogGain
1554 );
1555
1573 MVSDK_API CameraSdkStatus __stdcall CameraSetGain(
1574     CameraHandle    hCamera,
1575     int             iRGain,
1576     int             iGGain,
1577     int             iBGain
1578 );
1579
1597 MVSDK_API CameraSdkStatus __stdcall CameraGetGain(
1598     CameraHandle    hCamera,
1599     int*            piRGain,
1600     int*            piGGain,
1601     int*            piBGain
1602 );
1603
1617 MVSDK_API CameraSdkStatus __stdcall CameraSetGamma(
1618     CameraHandle    hCamera,
1619     int             iGamma
1620 );
1621
1635 MVSDK_API CameraSdkStatus __stdcall CameraGetGamma(
1636     CameraHandle    hCamera,
1637     int*            piGamma
1638 );
1639
1653 MVSDK_API CameraSdkStatus __stdcall CameraSetContrast(
1654     CameraHandle    hCamera,
1655     int             iContrast
1656 );
1657
1671 MVSDK_API CameraSdkStatus __stdcall CameraGetContrast(
1672     CameraHandle    hCamera,
1673     int*            piContrast
1674 );
1675
1689 MVSDK_API CameraSdkStatus __stdcall CameraSetSaturation(
1690     CameraHandle    hCamera,
1691     int             iSaturation
1692 );
1693
1707 MVSDK_API CameraSdkStatus __stdcall CameraGetSaturation(
1708     CameraHandle    hCamera,
1709     int*            piSaturation
1710 );
1711
1723 MVSDK_API CameraSdkStatus __stdcall CameraSetMonochrome(
1724     CameraHandle    hCamera,
1725     BOOL            bEnable
1726 );
1727
1741 MVSDK_API CameraSdkStatus __stdcall CameraGetMonochrome(
```

```
1742     CameraHandle    hCamera,
1743     BOOL*          pbEnable
1744 );
1745
1757 MVSDK_API CameraSdkStatus __stdcall CameraSetInverse(
1758     CameraHandle    hCamera,
1759     BOOL           bEnable
1760 );
1761
1773 MVSDK_API CameraSdkStatus __stdcall CameraGetInverse(
1774     CameraHandle    hCamera,
1775     BOOL*          pbEnable
1776 );
1777
1791 MVSDK_API CameraSdkStatus __stdcall CameraSetAntiFlick(
1792     CameraHandle    hCamera,
1793     BOOL           bEnable
1794 );
1795
1807 MVSDK_API CameraSdkStatus __stdcall CameraGetAntiFlick(
1808     CameraHandle    hCamera,
1809     BOOL*          pbEnable
1810 );
1811
1823 MVSDK_API CameraSdkStatus __stdcall CameraGetLightFrequency(
1824     CameraHandle    hCamera,
1825     int*           piFrequencySel
1826 );
1827
1839 MVSDK_API CameraSdkStatus __stdcall CameraSetLightFrequency(
1840     CameraHandle    hCamera,
1841     int            iFrequencySel
1842 );
1843
1855 MVSDK_API CameraSdkStatus __stdcall CameraSetFrameSpeed(
1856     CameraHandle    hCamera,
1857     int             iFrameSpeed
1858 );
1859
1873 MVSDK_API CameraSdkStatus __stdcall CameraGetFrameSpeed(
1874     CameraHandle    hCamera,
1875     int*           piFrameSpeed
1876 );
1877
1889 MVSDK_API CameraSdkStatus __stdcall CameraSetFrameRate(
1890     CameraHandle    hCamera,
1891     int            RateHZ
1892 );
1893
1905 MVSDK_API CameraSdkStatus __stdcall CameraGetFrameRate(
1906     CameraHandle    hCamera,
1907     int*           RateHZ
1908 );
1909
```

```
1921 MVSDK_API CameraSdkStatus __stdcall CameraSetParameterMode(
1922     CameraHandle    hCamera,
1923     int             iMode
1924 );
1925
1937 MVSDK_API CameraSdkStatus __stdcall CameraGetParameterMode(
1938     CameraHandle    hCamera,
1939     int*            piTarget
1940 );
1941
1953 MVSDK_API CameraSdkStatus __stdcall CameraSetParameterMask(
1954     CameraHandle    hCamera,
1955     UINT            uMask
1956 );
1957
1969 MVSDK_API CameraSdkStatus __stdcall CameraSaveParameter(
1970     CameraHandle    hCamera,
1971     int             iTeam
1972 );
1973
1985 MVSDK_API CameraSdkStatus __stdcall CameraSaveParameterToFile(
1986     CameraHandle    hCamera,
1987     char*           sFileName
1988 );
1989
2001 MVSDK_API CameraSdkStatus __stdcall CameraReadParameterFromFile(
2002     CameraHandle    hCamera,
2003     char*           sFileName
2004 );
2005
2017 MVSDK_API CameraSdkStatus __stdcall CameraLoadParameter(
2018     CameraHandle    hCamera,
2019     int             iTeam
2020 );
2021
2033 MVSDK_API CameraSdkStatus __stdcall CameraGetCurrentParameterGroup(
2034     CameraHandle    hCamera,
2035     int*            piTeam
2036 );
2037
2055 MVSDK_API CameraSdkStatus __stdcall CameraSetTransPackLen(
2056     CameraHandle    hCamera,
2057     INT             iPackSel
2058 );
2059
2073 MVSDK_API CameraSdkStatus __stdcall CameraGetTransPackLen(
2074     CameraHandle    hCamera,
2075     INT*            piPackSel
2076 );
2077
2089 MVSDK_API CameraSdkStatus __stdcall CameraIsAeWinVisible(
2090     CameraHandle    hCamera,
2091     BOOL*           pbIsVisible
2092 );
```

```
2093  
2107 MVSDK_API CameraSdkStatus __stdcall CameraSetAeWinVisible(  
2108     CameraHandle    hCamera,  
2109     BOOL           bIsVisible  
2110 );  
2111  
2129 MVSDK_API CameraSdkStatus __stdcall CameraGetAeWindow(  
2130     CameraHandle    hCamera,  
2131     INT*          piHOFF,  
2132     INT*          piVOFF,  
2133     INT*          piWidth,  
2134     INT*          piHeight  
2135 );  
2136  
2158 MVSDK_API CameraSdkStatus __stdcall CameraSetAeWindow(  
2159     CameraHandle    hCamera,  
2160     int            iHOFF,  
2161     int            iVOFF,  
2162     int            iWidth,  
2163     int            iHeight  
2164 );  
2165  
2179 MVSDK_API CameraSdkStatus __stdcall CameraSetMirror(  
2180     CameraHandle    hCamera,  
2181     int            iDir,  
2182     BOOL           bEnable  
2183 );  
2184  
2198 MVSDK_API CameraSdkStatus __stdcall CameraGetMirror(  
2199     CameraHandle    hCamera,  
2200     int            iDir,  
2201     BOOL*          pbEnable  
2202 );  
2203  
2217 MVSDK_API CameraSdkStatus __stdcall CameraSetHardwareMirror(  
2218     CameraHandle    hCamera,  
2219     int            iDir,  
2220     BOOL           bEnable  
2221 );  
2222  
2236 MVSDK_API CameraSdkStatus __stdcall CameraGetHardwareMirror(  
2237     CameraHandle    hCamera,  
2238     int            iDir,  
2239     BOOL*          pbEnable  
2240 );  
2241  
2253 MVSDK_API CameraSdkStatus __stdcall CameraSetRotate(  
2254     CameraHandle    hCamera,  
2255     int            iRot  
2256 );  
2257  
2269 MVSDK_API CameraSdkStatus __stdcall CameraGetRotate(  
2270     CameraHandle    hCamera,  
2271     int*          iRot
```

```
2272     );
2273
2291 MVSDK_API CameraSdkStatus __stdcall CameraGetWbWindow(
2292     CameraHandle    hCamera,
2293     INT*           PiHOff,
2294     INT*           PiVOff,
2295     INT*           PiWidth,
2296     INT*           PiHeight
2297 );
2298
2316 MVSDK_API CameraSdkStatus __stdcall CameraSetWbWindow(
2317     CameraHandle    hCamera,
2318     INT            iHOff,
2319     INT            iVOff,
2320     INT            iWidth,
2321     INT            iHeight
2322 );
2323
2335 MVSDK_API CameraSdkStatus __stdcall CameraIsWbWinVisible(
2336     CameraHandle    hCamera,
2337     BOOL*          pbShow
2338 );
2339
2353 MVSDK_API CameraSdkStatus __stdcall CameraSetWbWinVisible(
2354     CameraHandle    hCamera,
2355     BOOL            bShow
2356 );
2357
2371 MVSDK_API CameraSdkStatus __stdcall CameraImageOverlay(
2372     CameraHandle    hCamera,
2373     BYTE*          pRgbBuffer,
2374     tSdkFrameHead* pFrInfo
2375 );
2376
2398 MVSDK_API CameraSdkStatus __stdcall CameraSetCrossLine(
2399     CameraHandle    hCamera,
2400     int             iLine,
2401     INT            x,
2402     INT            y,
2403     UINT           uColor,
2404     BOOL           bVisible
2405 );
2406
2426 MVSDK_API CameraSdkStatus __stdcall CameraGetCrossLine(
2427     CameraHandle    hCamera,
2428     INT            iLine,
2429     INT*           px,
2430     INT*           py,
2431     UINT*          pcolor,
2432     BOOL*          pbVisible
2433 );
2434
2446 MVSDK_API CameraSdkStatus __stdcall CameraGetCapability(
2447     CameraHandle    hCamera,
```

```
2448     tSdkCameraCapbility*      pCameraInfo
2449 );
2450
2451 /****** */
2452 // 函数名   : CameraGetCapabilityEx
2453 // 功能描述 : 获得相机的特性描述结构体。该结构体中包含了相机
2454 //           可设置的各种参数的范围信息。决定了相关函数的参数
2455 //           返回，也可用于动态创建相机的配置界面。
2456 // 参数     : sDeviceModel    相机的型号，由扫描列表中获取
2457 //           pCameraInfo 指针，返回该相机特性描述的结构体。
2458 //           tSdkCameraCapbility在CameraDefine.h中定义。
2459 // 返回值   : 成功时，返回CAMERA_STATUS_SUCCESS (0);
2460 //           否则返回非0值的错误码，请参考CameraStatus.h
2461 //           中错误码的定义。
2462 /****** */
2463 MVSDK_API CameraSdkStatus __stdcall CameraGetCapabilityEx(
2464     char*          sDeviceModel,
2465     tSdkCameraCapbility* pCameraInfo,
2466     PVOID          hCameraHandle
2467 );
2468
2469 MVSDK_API CameraSdkStatus __stdcall CameraWriteSN(
2470     CameraHandle    hCamera,
2471     BYTE*          pbySN,
2472     INT            iLevel
2473 );
2474
2475 MVSDK_API CameraSdkStatus __stdcall CameraReadSN(
2476     CameraHandle    hCamera,
2477     BYTE*          pbySN,
2478     INT            iLevel
2479 );
2480
2481 MVSDK_API CameraSdkStatus __stdcall CameraSetTriggerDelayTime(
2482     CameraHandle    hCamera,
2483     UINT           uDelayTimeUs
2484 );
2485
2486 MVSDK_API CameraSdkStatus __stdcall CameraGetTriggerDelayTime(
2487     CameraHandle    hCamera,
2488     UINT*          puDelayTimeUs
2489 );
2490
2491 MVSDK_API CameraSdkStatus __stdcall CameraSetTriggerCount(
2492     CameraHandle    hCamera,
2493     INT            iCount
2494 );
2495
2496 MVSDK_API CameraSdkStatus __stdcall CameraGetTriggerCount(
2497     CameraHandle    hCamera,
2498     INT*           piCount
2499 );
2500
2501 MVSDK_API CameraSdkStatus __stdcall CameraSoftTrigger(
```

```
2589     CameraHandle    hCamera
2590 );
2591
2603 MVSDK_API CameraSdkStatus __stdcall CameraSetTriggerMode(
2604     CameraHandle    hCamera,
2605     int             iModeSel
2606 );
2607
2619 MVSDK_API CameraSdkStatus __stdcall CameraGetTriggerMode(
2620     CameraHandle    hCamera,
2621     INT*            piModeSel
2622 );
2623
2635 MVSDK_API CameraSdkStatus __stdcall CameraSetStrobeMode(
2636     CameraHandle    hCamera,
2637     INT             iMode
2638 );
2639
2651 MVSDK_API CameraSdkStatus __stdcall CameraGetStrobeMode(
2652     CameraHandle    hCamera,
2653     INT*            piMode
2654 );
2655
2667 MVSDK_API CameraSdkStatus __stdcall CameraSetStrobeDelayTime(
2668     CameraHandle    hCamera,
2669     UINT            uDelayTimeUs
2670 );
2671
2683 MVSDK_API CameraSdkStatus __stdcall CameraGetStrobeDelayTime(
2684     CameraHandle    hCamera,
2685     UINT*           upDelayTimeUs
2686 );
2687
2699 MVSDK_API CameraSdkStatus __stdcall CameraSetStrobePulseWidth(
2700     CameraHandle    hCamera,
2701     UINT            uTimeUs
2702 );
2703
2715 MVSDK_API CameraSdkStatus __stdcall CameraGetStrobePulseWidth(
2716     CameraHandle    hCamera,
2717     UINT*           upTimeUs
2718 );
2719
2731 MVSDK_API CameraSdkStatus __stdcall CameraSetStrobePolarity(
2732     CameraHandle    hCamera,
2733     INT             uPolarity
2734 );
2735
2747 MVSDK_API CameraSdkStatus __stdcall CameraGetStrobePolarity(
2748     CameraHandle    hCamera,
2749     INT*            upPolarity
2750 );
2751
2763 MVSDK_API CameraSdkStatus __stdcall CameraSetExtTrigSignalType(
```

```
2764     CameraHandle    hCamera,
2765     INT           iType
2766 );
2767
2779 MVSDK_API CameraSdkStatus __stdcall CameraGetExtTrigSignalType(
2780     CameraHandle    hCamera,
2781     INT*          ipType
2782 );
2783
2795 MVSDK_API CameraSdkStatus __stdcall CameraSetExtTrigShutterType(
2796     CameraHandle    hCamera,
2797     INT           iType
2798 );
2799
2813 MVSDK_API CameraSdkStatus __stdcall CameraGetExtTrigShutterType(
2814     CameraHandle    hCamera,
2815     INT*          ipType
2816 );
2817
2829 MVSDK_API CameraSdkStatus __stdcall CameraSetExtTrigDelayTime(
2830     CameraHandle    hCamera,
2831     UINT          uDelayTimeUs
2832 );
2833
2845 MVSDK_API CameraSdkStatus __stdcall CameraGetExtTrigDelayTime(
2846     CameraHandle    hCamera,
2847     UINT*         upDelayTimeUs
2848 );
2849
2861 MVSDK_API CameraSdkStatus __stdcall CameraSetExtTrigBufferedDelayTime(
2862     CameraHandle    hCamera,
2863     UINT          uDelayTimeUs
2864 );
2865
2877 MVSDK_API CameraSdkStatus __stdcall CameraGetExtTrigBufferedDelayTime(
2878     CameraHandle    hCamera,
2879     UINT*         puDelayTimeUs
2880 );
2881
2893 MVSDK_API CameraSdkStatus __stdcall CameraSetExtTrigIntervalTime(
2894     CameraHandle    hCamera,
2895     UINT          uTimeUs
2896 );
2897
2909 MVSDK_API CameraSdkStatus __stdcall CameraGetExtTrigIntervalTime(
2910     CameraHandle    hCamera,
2911     UINT*         upTimeUs
2912 );
2913
2925 MVSDK_API CameraSdkStatus __stdcall CameraSetExtTrigJitterTime(
2926     CameraHandle    hCamera,
2927     UINT          uTimeUs
2928 );
2929
```

```
2941 MVSDK_API CameraSdkStatus __stdcall CameraGetExtTrigJitterTime(
2942     CameraHandle      hCamera,
2943     UINT*             upTimeUs
2944 );
2945
2957 MVSDK_API CameraSdkStatus __stdcall CameraGetExtTrigCapability(
2958     CameraHandle      hCamera,
2959     UINT*             puCapabilityMask
2960 );
2961
2971 MVSDK_API CameraSdkStatus __stdcall CameraPauseLevelTrigger(
2972     CameraHandle      hCamera
2973 );
2974
2986 MVSDK_API CameraSdkStatus __stdcall CameraGetResolutionForSnap(
2987     CameraHandle      hCamera,
2988     tSdkImageResolution* pImageResolution
2989 );
2990
3004 MVSDK_API CameraSdkStatus __stdcall CameraSetResolutionForSnap(
3005     CameraHandle      hCamera,
3006     tSdkImageResolution* pImageResolution
3007 );
3008
3020 MVSDK_API CameraSdkStatus __stdcall CameraCustomizeResolution(
3021     CameraHandle      hCamera,
3022     tSdkImageResolution* pImageCustom
3023 );
3024
3046 MVSDK_API CameraSdkStatus __stdcall CameraCustomizeReferWin(
3047     CameraHandle      hCamera,
3048     INT                iWinType,
3049     HWND               hParent,
3050     INT*              piHOff,
3051     INT*              piVOff,
3052     INT*              piWidth,
3053     INT*              piHeight
3054 );
3055
3069 MVSDK_API CameraSdkStatus __stdcall CameraShowSettingPage(
3070     CameraHandle      hCamera,
3071     BOOL               bShow
3072 );
3073
3093 MVSDK_API CameraSdkStatus __stdcall CameraCreateSettingPage(
3094     CameraHandle      hCamera,
3095     HWND               hParent,
3096     char*              pWinText,
3097     CAMERA_PAGE_MSG_PROC pCallbackFunc,
3098     PVOID              pCallbackCtx,
3099     UINT               uReserved
3100 );
3101
3111 MVSDK_API CameraSdkStatus __stdcall CameraCreateSettingPageEx(
```

```
3112     CameraHandle           hCamera
3113 );
3114
3126 MVSDK_API CameraSdkStatus __stdcall CameraSetActiveSettingSubPage(
3127     CameraHandle    hCamera,
3128     INT           index
3129 );
3130
3144 MVSDK_API CameraSdkStatus __stdcall CameraSetSettingPageParent(
3145     CameraHandle    hCamera,
3146     HWND          hParentWnd,
3147     DWORD          Flags
3148 );
3149
3161 MVSDK_API CameraSdkStatus __stdcall CameraGetSettingPageHWnd(
3162     CameraHandle    hCamera,
3163     HWND*          hWnd
3164 );
3165
3175 MVSDK_API CameraSdkStatus __stdcall CameraUpdateSettingPage(
3176     CameraHandle    hCamera
3177 );
3178
3179 MVSDK_API CameraSdkStatus __stdcall CameraSpecialControl(
3180     CameraHandle    hCamera,
3181     DWORD          dwCtrlCode,
3182     DWORD          dwParam,
3183     LPVOID         lpData
3184 );
3185
3197 MVSDK_API CameraSdkStatus __stdcall CameraGetFrameStatistic(
3198     CameraHandle    hCamera,
3199     tSdkFrameStatistic* psFrameStatistic
3200 );
3201
3213 MVSDK_API CameraSdkStatus __stdcall CameraSetNoiseFilter(
3214     CameraHandle    hCamera,
3215     BOOL           bEnable
3216 );
3217
3229 MVSDK_API CameraSdkStatus __stdcall CameraGetNoiseFilterState(
3230     CameraHandle    hCamera,
3231     BOOL*          pEnable
3232 );
3233
3243 MVSDK_API CameraSdkStatus __stdcall CameraRstTimeStamp(
3244     CameraHandle    hCamera
3245 );
3246
3264 MVSDK_API CameraSdkStatus __stdcall CameraSaveUserData(
3265     CameraHandle    hCamera,
3266     UINT           uStartAddr,
3267     BYTE*          pbData,
3268     int            ilen
```

```
3269 );
3270
3286 MVSDK_API CameraSdkStatus __stdcall CameraLoadUserData(
3287     CameraHandle    hCamera,
3288     UINT           uStartAddr,
3289     BYTE            *pbData,
3290     int             ilen
3291 );
3292
3304 MVSDK_API CameraSdkStatus __stdcall CameraGetFriendlyName(
3305     CameraHandle   hCamera,
3306     char*          pName
3307 );
3308
3320 MVSDK_API CameraSdkStatus __stdcall CameraSetFriendlyName(
3321     CameraHandle   hCamera,
3322     char*          pName
3323 );
3324
3334 MVSDK_API CameraSdkStatus __stdcall CameraSdkGetVersionString(
3335     char*          pVersionString
3336 );
3337
3338 // ****
3339 // 函数名    : CameraCheckFwUpdate
3340 // 功能描述  : 检测固件版本, 是否需要升级。
3341 // 参数      : hCamera 相机的句柄, 由CameraInit函数获得。
3342 //           : pNeedUpdate 指针, 返回固件检测状态, TRUE表示需要更新
3343 // 返回值    : 成功时, 返回CAMERA_STATUS_SUCCESS (0);
3344 //           : 否则返回非0值的错误码, 请参考CameraStatus.h
3345 //           : 中错误码的定义。
3346 // ****
3347 MVSDK_API CameraSdkStatus __stdcall CameraCheckFwUpdate(
3348     CameraHandle   hCamera,
3349     BOOL*          pNeedUpdate
3350 );
3351
3363 MVSDK_API CameraSdkStatus __stdcall CameraGetFirmwareVersion(
3364     CameraHandle   hCamera,
3365     char*          pVersion
3366 );
3367
3368 // 功能与CameraGetFirmwareVersion相同。Version拼写错误, 为了兼容性保留
3369 // Same function as CameraGetFirmwareVersion. Version misspelled,
3370 // reserved for compatibility
3370 MVSDK_API CameraSdkStatus __stdcall CameraGetFirmwareVision(
3371     CameraHandle   hCamera,
3372     char*          pVersion
3373 );
3374
3386 MVSDK_API CameraSdkStatus __stdcall CameraGetEnumInfo(
3387     CameraHandle   hCamera,
3388     tSdkCameraDevInfo* pCameraInfo
3389 );
```

```
3390
3402 MVSDK_API CameraSdkStatus __stdcall CameraGetInterfaceVersion(
3403     CameraHandle     hCamera,
3404     char*           pVersion
3405 );
3406
3422 MVSDK_API CameraSdkStatus __stdcall CameraSetIOState(
3423     CameraHandle     hCamera,
3424     INT              iOutputIOIndex,
3425     UINT             uState
3426 );
3427
3441 MVSDK_API CameraSdkStatus __stdcall CameraSetIOStateEx(
3442     CameraHandle     hCamera,
3443     INT              iOutputIOIndex,
3444     UINT             uState
3445 );
3446
3462 MVSDK_API CameraSdkStatus __stdcall CameraGetOutPutIOState(
3463     CameraHandle     hCamera,
3464     INT              iOutputIOIndex,
3465     UINT*            puState
3466 );
3467
3481 MVSDK_API CameraSdkStatus __stdcall CameraGetOutPutIOStateEx(
3482     CameraHandle     hCamera,
3483     INT              iOutputIOIndex,
3484     UINT*            puState
3485 );
3486
3502 MVSDK_API CameraSdkStatus __stdcall CameraGetIOState(
3503     CameraHandle     hCamera,
3504     INT              iInputIOIndex,
3505     UINT*            puState
3506 );
3507
3521 MVSDK_API CameraSdkStatus __stdcall CameraGetIOStateEx(
3522     CameraHandle     hCamera,
3523     INT              iInputIOIndex,
3524     UINT*            puState
3525 );
3526
3540 MVSDK_API CameraSdkStatus __stdcall CameraSetInPutIOMode(
3541     CameraHandle     hCamera,
3542     INT              iInputIOIndex,
3543     INT              iMode
3544 );
3545
3559 MVSDK_API CameraSdkStatus __stdcall CameraGetInPutIOMode(
3560     CameraHandle     hCamera,
3561     INT              iInputIOIndex,
3562     INT*             piMode
3563 );
3564
```

```
3578 MVSDK_API CameraSdkStatus __stdcall CameraSetOutPutIOMode(
3579     CameraHandle     hCamera,
3580     INT             iOutputIOIndex,
3581     INT             iMode
3582 );
3583
3597 MVSDK_API CameraSdkStatus __stdcall CameraGetOutPutIOMode(
3598     CameraHandle     hCamera,
3599     INT             iOutputIOIndex,
3600     INT*            piMode
3601 );
3602
3616 MVSDK_API CameraSdkStatus __stdcall CameraGetInPutIOModeCapbility(
3617     CameraHandle     hCamera,
3618     INT             iInputIOIndex,
3619     UINT*           piCapbility
3620 );
3621
3635 MVSDK_API CameraSdkStatus __stdcall CameraGetOutPutIOModeCapbility(
3636     CameraHandle     hCamera,
3637     INT             iOutputIOIndex,
3638     UINT*           piCapbility
3639 );
3640
3656 MVSDK_API CameraSdkStatus __stdcall CameraSetOutPutPWM(
3657     CameraHandle     hCamera,
3658     INT             iOutputIOIndex,
3659     UINT            iCycle,
3660     UINT            uDuty
3661 );
3662
3674 MVSDK_API CameraSdkStatus __stdcall CameraSetRotaryEncDir(
3675     CameraHandle     hCamera,
3676     INT             dir
3677 );
3678
3690 MVSDK_API CameraSdkStatus __stdcall CameraGetRotaryEncDir(
3691     CameraHandle     hCamera,
3692     INT*            dir
3693 );
3694
3708 MVSDK_API CameraSdkStatus __stdcall CameraSetRotaryEncFreq(
3709     CameraHandle     hCamera,
3710     INT             mul,
3711     INT             div
3712 );
3713
3727 MVSDK_API CameraSdkStatus __stdcall CameraGetRotaryEncFreq(
3728     CameraHandle     hCamera,
3729     INT*            mul,
3730     INT*            div
3731 );
3732
3746 MVSDK_API CameraSdkStatus __stdcall CameraSetInPutIOFormat(
```

```
3747     CameraHandle     hCamera,
3748     INT             iInputIOIndex,
3749     INT             iFormat
3750 );
3751
3765 MVSDK_API CameraSdkStatus __stdcall CameraGetInPutIOFormat(
3766     CameraHandle     hCamera,
3767     INT             iInputIOIndex,
3768     INT*            piFormat
3769 );
3770
3784 MVSDK_API CameraSdkStatus __stdcall CameraSetOutPutIOFormat(
3785     CameraHandle     hCamera,
3786     INT             iOutputIOIndex,
3787     INT             iFormat
3788 );
3789
3803 MVSDK_API CameraSdkStatus __stdcall CameraGetOutPutIOFormat(
3804     CameraHandle     hCamera,
3805     INT             iOutputIOIndex,
3806     INT*            piFormat
3807 );
3808
3822 MVSDK_API CameraSdkStatus __stdcall CameraGetInPutIOFormatCapbility(
3823     CameraHandle     hCamera,
3824     INT             iInputIOIndex,
3825     UINT*           piCapbility
3826 );
3827
3841 MVSDK_API CameraSdkStatus __stdcall CameraGetOutPutIOFormatCapbility(
3842     CameraHandle     hCamera,
3843     INT             iOutputIOIndex,
3844     UINT*           piCapbility
3845 );
3846
3847 // @ingroup API_EXPOSURE
3860 MVSDK_API CameraSdkStatus __stdcall CameraSetAeAlgorithm(
3861     CameraHandle     hCamera,
3862     INT             iIspProcessor,
3863     INT             iAeAlgorithmSel
3864 );
3865
3866 // @ingroup API_EXPOSURE
3879 MVSDK_API CameraSdkStatus __stdcall CameraGetAeAlgorithm(
3880     CameraHandle     hCamera,
3881     INT             iIspProcessor,
3882     INT*            piAlgorithmSel
3883 );
3884
3898 MVSDK_API CameraSdkStatus __stdcall CameraSetBayerDecAlgorithm(
3899     CameraHandle     hCamera,
3900     INT             iIspProcessor,
3901     INT             iAlgorithmSel
3902 );
```

```
3903  
3917 MVSDK_API CameraSdkStatus __stdcall CameraGetBayerDecAlgorithm(  
3918     CameraHandle    hCamera,  
3919     INT            iIspProcessor,  
3920     INT*           piAlgorithmSel  
3921 );  
3922  
3923 // @ingroup API_ISP  
3924 MVSDK_API CameraSdkStatus __stdcall CameraSetIspProcessor(  
3925     CameraHandle    hCamera,  
3926     INT            iIspProcessor  
3927 );  
3928  
3929 // @ingroup API_ISP  
3930 MVSDK_API CameraSdkStatus __stdcall CameraGetIspProcessor(  
3931     CameraHandle    hCamera,  
3932     INT*           piIspProcessor  
3933 );  
3934  
3935 MVSDK_API CameraSdkStatus __stdcall CameraSetBlackLevel(  
3936     CameraHandle    hCamera,  
3937     INT            iBlackLevel  
3938 );  
3939  
3940 MVSDK_API CameraSdkStatus __stdcall CameraGetBlackLevel(  
3941     CameraHandle    hCamera,  
3942     INT*           piBlackLevel  
3943 );  
3944  
3945 MVSDK_API CameraSdkStatus __stdcall CameraSetWhiteLevel(  
3946     CameraHandle    hCamera,  
3947     INT            iWhiteLevel  
3948 );  
3949  
3950 MVSDK_API CameraSdkStatus __stdcall CameraGetWhiteLevel(  
3951     CameraHandle    hCamera,  
3952     INT*           piWhiteLevel  
3953 );  
3954  
3955 MVSDK_API CameraSdkStatus __stdcall CameraSetIspOutFormat(  
3956     CameraHandle    hCamera,  
3957     UINT           uFormat  
3958 );  
3959  
3960 MVSDK_API CameraSdkStatus __stdcall CameraGetIspOutFormat(  
3961     CameraHandle    hCamera,  
3962     UINT*          puFormat  
3963 );  
3964  
3965 MVSDK_API char* __stdcall CameraGetErrorString(  
3966     CameraSdkStatus    iStatusCode  
3967 );  
3968  
3969 MVSDK_API CameraSdkStatus __stdcall CameraGetImageBufferEx2(
```

```
4090     CameraHandle    hCamera,
4091     BYTE*          pImageData,
4092     UINT           uOutFormat,
4093     int*           piWidth,
4094     int*           piHeight,
4095     UINT           wTimes
4096 );
4097
4121 MVSDK_API CameraSdkStatus __stdcall CameraGetImageBufferEx3(
4122     CameraHandle   hCamera,
4123     BYTE*pImageData,
4124     UINT uOutFormat,
4125     int *piWidth,
4126     int *piHeight,
4127     UINT* puTimeStamp,
4128     UINT wTimes
4129 );
4130
4146 MVSDK_API CameraSdkStatus __stdcall CameraGetCapabilityEx2(
4147     CameraHandle   hCamera,
4148     int*          pMaxWidth,
4149     int*          pMaxHeight,
4150     int*          pbColorCamera
4151 );
4152
4164 MVSDK_API CameraSdkStatus __stdcall CameraReConnect(
4165     CameraHandle   hCamera
4166 );
4167
4177 MVSDK_API CameraSdkStatus __stdcall CameraConnectTest(
4178     CameraHandle   hCamera
4179 );
4180
4194 MVSDK_API CameraSdkStatus __stdcall CameraSetLedEnable(
4195     CameraHandle   hCamera,
4196     int           index,
4197     BOOL          enable
4198 );
4199
4213 MVSDK_API CameraSdkStatus __stdcall CameraGetLedEnable(
4214     CameraHandle   hCamera,
4215     int           index,
4216     BOOL*         enable
4217 );
4218
4232 MVSDK_API CameraSdkStatus __stdcall CameraSetLedOnOff(
4233     CameraHandle   hCamera,
4234     int           index,
4235     BOOL          onoff
4236 );
4237
4251 MVSDK_API CameraSdkStatus __stdcall CameraGetLedOnOff(
4252     CameraHandle   hCamera,
4253     int           index,
```

```
4254     BOOL*          onoff
4255 );
4256
4270 MVSDK_API CameraSdkStatus __stdcall CameraSetLedDuration(
4271     CameraHandle    hCamera,
4272     int              index,
4273     UINT             duration
4274 );
4275
4289 MVSDK_API CameraSdkStatus __stdcall CameraGetLedDuration(
4290     CameraHandle    hCamera,
4291     int              index,
4292     UINT*           duration
4293 );
4294
4308 MVSDK_API CameraSdkStatus __stdcall CameraSetLedBrightness(
4309     CameraHandle    hCamera,
4310     int              index,
4311     UINT             uBrightness
4312 );
4313
4327 MVSDK_API CameraSdkStatus __stdcall CameraGetLedBrightness(
4328     CameraHandle    hCamera,
4329     int              index,
4330     UINT*           uBrightness
4331 );
4332
4346 MVSDK_API CameraSdkStatus __stdcall CameraEnableTransferRoi(
4347     CameraHandle    hCamera,
4348     UINT            uEnableMask
4349 );
4350
4370 MVSDK_API CameraSdkStatus __stdcall CameraSetTransferRoi(
4371     CameraHandle    hCamera,
4372     int              index,
4373     UINT             X1,
4374     UINT             Y1,
4375     UINT             X2,
4376     UINT             Y2
4377 );
4378
4398 MVSDK_API CameraSdkStatus __stdcall CameraGetTransferRoi(
4399     CameraHandle    hCamera,
4400     int              index,
4401     UINT*           pX1,
4402     UINT*           pY1,
4403     UINT*           pX2,
4404     UINT*           pY2
4405 );
4406
4420 MVSDK_API BYTE* __stdcall CameraAlignMalloc(
4421     int              size,
4422     int              align
4423 );
```

```
4424
4432 MVSDK_API void __stdcall CameraAlignFree(
4433     BYTE*           membuffer
4434 );
4435
4447 MVSDK_API CameraSdkStatus __stdcall CameraSetAutoConnect(CameraHandle
4448     hCamera,BOOL bEnable);
4460 MVSDK_API CameraSdkStatus __stdcall CameraGetAutoConnect(CameraHandle
4461     hCamera,BOOL *pbEnable);
4461
4473 MVSDK_API CameraSdkStatus __stdcall
4474     CameraGetReConnectCounts(CameraHandle hCamera,UINT* puCounts);
4474
4488 MVSDK_API CameraSdkStatus __stdcall
4489     CameraSetSingleGrabMode(CameraHandle hCamera, BOOL bEnable);
4489
4501 MVSDK_API CameraSdkStatus __stdcall
4502     CameraGetSingleGrabMode(CameraHandle hCamera, BOOL* pbEnable);
4502
4512 MVSDK_API CameraSdkStatus __stdcall CameraRestartGrab(CameraHandle hCar
4513
4531 MVSDK_API CameraSdkStatus __stdcall CameraEvaluateImageDefinition(
4532     CameraHandle      hCamera,
4533     INT                iAlgorithSel,
4534     BYTE*              pbyIn,
4535     tSdkFrameHead*    pFrInfo,
4536     double*            DefinitionValue
4537 );
4538
4570 MVSDK_API CameraSdkStatus __stdcall CameraDrawText(
4571     BYTE*              pRgbBuffer,
4572     tSdkFrameHead*    pFrInfo,
4573     char const*        pFontFileName,
4574     UINT               FontWidth,
4575     UINT               FontHeight,
4576     char const*        pText,
4577     INT                Left,
4578     INT                Top,
4579     UINT               Width,
4580     UINT               Height,
4581     UINT               TextColor,
4582     UINT               uFlags
4583 );
4584
4604 MVSDK_API CameraSdkStatus __stdcall CameraGigeEnumerateDevice(
4605     char const**       ppIpList,
4606     int                numIp,
4607     tSdkCameraDevInfo* pCameraList,
4608     int*               piNums
4609 );
4610
4632 MVSDK_API CameraSdkStatus __stdcall CameraGigeGetIp(
4633     tSdkCameraDevInfo* pCameraInfo,
```

```
4634     char* CamIp,
4635     char* CamMask,
4636     char* CamGateWay,
4637     char* EtIp,
4638     char* EtMask,
4639     char* EtGateWay
4640 );
4641
4659 MVSDK_API CameraSdkStatus __stdcall CameraGigeSetIp(
4660     tSdkCameraDevInfo* pCameraInfo,
4661     char const* Ip,
4662     char const* SubMask,
4663     char const* GateWay,
4664     BOOL bPersistent
4665 );
4666
4680 MVSDK_API CameraSdkStatus __stdcall CameraGigeGetMac(
4681     tSdkCameraDevInfo* pCameraInfo,
4682     char* CamMac,
4683     char* EtMac
4684 );
4685
4695 MVSDK_API CameraSdkStatus __stdcall CameraEnableFastResponse(
4696     CameraHandle hCamera
4697 );
4698
4710 MVSDK_API CameraSdkStatus __stdcall CameraSetCorrectDeadPixel(
4711     CameraHandle hCamera,
4712     BOOL bEnable
4713 );
4714
4726 MVSDK_API CameraSdkStatus __stdcall CameraGetCorrectDeadPixel(
4727     CameraHandle hCamera,
4728     BOOL* pbEnable
4729 );
4730
4742 MVSDK_API CameraSdkStatus __stdcall CameraFlatFieldingCorrectSetEnable(
4743     CameraHandle hCamera,
4744     BOOL bEnable
4745 );
4746
4758 MVSDK_API CameraSdkStatus __stdcall CameraFlatFieldingCorrectGetEnable(
4759     CameraHandle hCamera,
4760     BOOL* pbEnable
4761 );
4762
4780 MVSDK_API CameraSdkStatus __stdcall CameraFlatFieldingCorrectSetParamet
4781     CameraHandle hCamera,
4782     BYTE const* pDarkFieldingImage,
4783     tSdkFrameHead const* pDarkFieldingFrInfo,
4784     BYTE const* pLightFieldingImage,
4785     tSdkFrameHead const* pLightFieldingFrInfo
4786 );
4787
```

```
4801 MVSDK_API CameraSdkStatus __stdcall CameraFlatFieldingCorrectGetParamet
4802     CameraHandle hCamera,
4803     BOOL *pbValid,
4804     char *pFilePath
4805 );
4806
4818 MVSDK_API CameraSdkStatus __stdcall CameraFlatFieldingCorrectSaveParamet
4819     CameraHandle hCamera,
4820     char const* pszFileName
4821 );
4822
4834 MVSDK_API CameraSdkStatus __stdcall CameraFlatFieldingCorrectLoadParamet
4835     CameraHandle hCamera,
4836     char const* pszFileName
4837 );
4838
4839 /***** */
4840 // 函数名 : CameraCommonCall
4841 // 功能描述 : 相机的一些特殊功能调用, 二次开发时一般不需要调用。
4842 // 参数 : hCamera 相机的句柄, 由CameraInit函数获得。
4843 //         pszCall 功能及参数
4844 //         pszResult 调用结果, 不同的pszCall时, 意义不同。
4845 //         uResultBufSize pszResult指向的缓冲区的字节大小
4846 // 返回值 : 成功时, 返回CAMERA_STATUS_SUCCESS (0);
4847 //           否则返回非0值的错误码, 请参考CameraStatus.h
4848 //           中错误码的定义。
4849 /***** */
4850 MVSDK_API CameraSdkStatus __stdcall CameraCommonCall(
4851     CameraHandle hCamera,
4852     char const* pszCall,
4853     char* pszResult,
4854     UINT uResultBufSize
4855 );
4856
4872 MVSDK_API CameraSdkStatus __stdcall CameraSetDenoise3DParams(
4873     CameraHandle hCamera,
4874     BOOL bEnable,
4875     int nCount,
4876     float *Weights
4877 );
4878
4896 MVSDK_API CameraSdkStatus __stdcall CameraGetDenoise3DParams(
4897     CameraHandle hCamera,
4898     BOOL *bEnable,
4899     int *nCount,
4900     BOOL *bUseWeight,
4901     float *Weights
4902 );
4903
4923 MVSDK_API CameraSdkStatus __stdcall CameraManualDenoise3D(
4924     tSdkFrameHead *InFramesHead,
4925     BYTE **InFramesData,
4926     int nCount,
4927     float *Weights,
```

```
4928     tSdkFrameHead    *OutFrameHead,
4929     BYTE             *OutFrameData
4930 );
4931
4943 MVSDK_API CameraSdkStatus __stdcall CameraCustomizeDeadPixels(
4944     CameraHandle      hCamera,
4945     HWND              hParent
4946 );
4947
4965 MVSDK_API CameraSdkStatus __stdcall CameraReadDeadPixels(
4966     CameraHandle      hCamera,
4967     USHORT*           pRows,
4968     USHORT*           pCols,
4969     UINT*             pNumPixel
4970 );
4971
4987 MVSDK_API CameraSdkStatus __stdcall CameraAddDeadPixels(
4988     CameraHandle      hCamera,
4989     USHORT*           pRows,
4990     USHORT*           pCols,
4991     UINT              NumPixel
4992 );
4993
5009 MVSDK_API CameraSdkStatus __stdcall CameraRemoveDeadPixels(
5010     CameraHandle      hCamera,
5011     USHORT*           pRows,
5012     USHORT*           pCols,
5013     UINT              NumPixel
5014 );
5015
5025 MVSDK_API CameraSdkStatus __stdcall CameraRemoveAllDeadPixels(
5026     CameraHandle      hCamera
5027 );
5028
5038 MVSDK_API CameraSdkStatus __stdcall CameraSaveDeadPixels(
5039     CameraHandle      hCamera
5040 );
5041
5053 MVSDK_API CameraSdkStatus __stdcall CameraSaveDeadPixelsToFile(
5054     CameraHandle      hCamera,
5055     char const*        sFileName
5056 );
5057
5069 MVSDK_API CameraSdkStatus __stdcall CameraLoadDeadPixelsFromFile(
5070     CameraHandle      hCamera,
5071     char const*        sFileName
5072 );
5073
5093 MVSDK_API CameraSdkStatus __stdcall CameraGetImageBufferPriority(
5094     CameraHandle      hCamera,
5095     tSdkFrameHead*    pFrameInfo,
5096     BYTE**            pbyBuffer,
5097     UINT              wTimes,
5098     UINT              Priority
```

```
5099     );
5100
5120 MVSDK_API unsigned char* __stdcall CameraGetImageBufferPriorityEx(
5121     CameraHandle      hCamera,
5122     INT*              piWidth,
5123     INT*              piHeight,
5124     UINT               wTimes,
5125     UINT               Priority
5126 );
5127
5151 MVSDK_API CameraSdkStatus __stdcall CameraGetImageBufferPriorityEx2(
5152     CameraHandle      hCamera,
5153     BYTE*             pImageData,
5154     UINT              uOutFormat,
5155     int*              piWidth,
5156     int*              piHeight,
5157     UINT               wTimes,
5158     UINT               Priority
5159 );
5160
5186 MVSDK_API CameraSdkStatus __stdcall CameraGetImageBufferPriorityEx3(
5187     CameraHandle      hCamera,
5188     BYTE*pImageData,
5189     UINT uOutFormat,
5190     int *piWidth,
5191     int *piHeight,
5192     UINT* puTimeStamp,
5193     UINT wTimes,
5194     UINT Priority
5195 );
5196
5206 MVSDK_API CameraSdkStatus __stdcall CameraClearBuffer(
5207     CameraHandle      hCamera
5208 );
5209
5223 MVSDK_API CameraSdkStatus __stdcall CameraSoftTriggerEx(
5224     CameraHandle      hCamera,
5225     UINT uFlags
5226 );
5227
5239 MVSDK_API CameraSdkStatus __stdcall CameraSetHDR(
5240     CameraHandle      hCamera,
5241     float             value
5242 );
5243
5255 MVSDK_API CameraSdkStatus __stdcall CameraGetHDR(
5256     CameraHandle      hCamera,
5257     float*            value
5258 );
5259
5271 MVSDK_API CameraSdkStatus __stdcall CameraGetFrameID(
5272     CameraHandle      hCamera,
5273     UINT*             id
5274 );
```

```
5275 MVSDK_API CameraSdkStatus __stdcall CameraGetFrameTimeStamp(
5276     CameraHandle    hCamera,
5277     UINT*          TimeStampL,
5278     UINT*          TimeStampH
5279 );
5280
5281
5282 MVSDK_API CameraSdkStatus __stdcall CameraSetHDRGainMode(
5283     CameraHandle    hCamera,
5284     int             value
5285 );
5286
5287 MVSDK_API CameraSdkStatus __stdcall CameraGetHDRGainMode(
5288     CameraHandle    hCamera,
5289     int*            value
5290 );
5291
5292
5293 MVSDK_API CameraSdkStatus __stdcall CameraCreateDIBitmap(
5294     HDC   hDC,
5295     BYTE *pFrameBuffer,
5296     tSdkFrameHead* pFrameHead,
5297     HBITMAP* outBitmap
5298 );
5299
5300
5301 MVSDK_API CameraSdkStatus __stdcall CameraDrawFrameBuffer(
5302     BYTE *pFrameBuffer,
5303     tSdkFrameHead* pFrameHead,
5304     HWND  hWnd,
5305     int   Algorithm,
5306     int   Mode
5307 );
5308
5309
5310 MVSDK_API CameraSdkStatus __stdcall CameraFlipFrameBuffer(
5311     BYTE *pFrameBuffer,
5312     tSdkFrameHead* pFrameHead,
5313     int   Flags
5314 );
5315
5316
5317 MVSDK_API CameraSdkStatus __stdcall CameraConvertFrameBufferFormat(
5318     CameraHandle hCamera,
5319     BYTE *pInFrameBuffer,
5320     BYTE *pOutFrameBuffer,
5321     int   outWidth,
5322     int   outHeight,
5323     UINT  outMediaType,
5324     tSdkFrameHead* pFrameHead
5325 );
5326
5327
5328 MVSDK_API CameraSdkStatus __stdcall CameraSetConnectionStatusCallback(
5329     CameraHandle      hCamera,
5330     CAMERA_CONNECTION_STATUS_CALLBACK pCallBack,
5331     PVOID             pContext
5332 );
5333
5334
5335 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionStatus(
5336     CameraHandle      hCamera,
5337     PUINT             pStatus
5338 );
5339
5340
5341 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionError(
5342     CameraHandle      hCamera,
5343     PUINT             pError
5344 );
5345
5346
5347 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastError(
5348     CameraHandle      hCamera,
5349     PUINT             pError
5350 );
5351
5352
5353 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5354     CameraHandle      hCamera,
5355     PCHAR             pErrorString
5356 );
5357
5358
5359 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5360     CameraHandle      hCamera,
5361     PCHAR             pErrorString
5362 );
5363
5364
5365 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5366     CameraHandle      hCamera,
5367     PCHAR             pErrorString
5368 );
5369
5370
5371 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5372     CameraHandle      hCamera,
5373     PCHAR             pErrorString
5374 );
5375
5376
5377 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5378     CameraHandle      hCamera,
5379     PCHAR             pErrorString
5380 );
5381
5382
5383 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5384     CameraHandle      hCamera,
5385     PCHAR             pErrorString
5386 );
5387
5388
5389 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5390     CameraHandle      hCamera,
5391     PCHAR             pErrorString
5392 );
5393
5394
5395 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5396     CameraHandle      hCamera,
5397     PCHAR             pErrorString
5398 );
5399
5400
5401 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5402     CameraHandle      hCamera,
5403     PCHAR             pErrorString
5404 );
5405
5406
5407 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5408     CameraHandle      hCamera,
5409     PCHAR             pErrorString
5410 );
5411
5412
5413 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5414     CameraHandle      hCamera,
5415     PCHAR             pErrorString
5416 );
5417
5418
5419 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5420     CameraHandle      hCamera,
5421     PCHAR             pErrorString
5422 );
5423
5424
5425 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5426     CameraHandle      hCamera,
5427     PCHAR             pErrorString
5428 );
5429
5430
5431 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5432     CameraHandle      hCamera,
5433     PCHAR             pErrorString
5434 );
5435
5436
5437 MVSDK_API CameraSdkStatus __stdcall CameraGetConnectionLastErrorString(
5438     CameraHandle      hCamera,
5439     PCHAR             pErrorString
5440 );
5441
5442
```

```
5456 MVSDK_API CameraSdkStatus __stdcall CameraSetLightingControllerMode(
5457     CameraHandle          hCamera,
5458     int                   index,
5459     int                   mode
5460 );
5461
5475 MVSDK_API CameraSdkStatus __stdcall CameraSetLightingControllerState(
5476     CameraHandle          hCamera,
5477     int                   index,
5478     int                   state
5479 );
5480
5492 MVSDK_API CameraSdkStatus __stdcall CameraSetFrameResendCount(
5493     CameraHandle          hCamera,
5494     int                   count
5495 );
5496
5514 MVSDK_API CameraSdkStatus __stdcall CameraSetUndistortParams(
5515     CameraHandle          hCamera,
5516     int                   width,
5517     int                   height,
5518     double                cameraMatrix[4],
5519     double                distCoeffs[5]
5520 );
5521
5539 MVSDK_API CameraSdkStatus __stdcall CameraGetUndistortParams(
5540     CameraHandle          hCamera,
5541     int                   *width,
5542     int                   *height,
5543     double                cameraMatrix[4],
5544     double                distCoeffs[5]
5545 );
5546
5558 MVSDK_API CameraSdkStatus __stdcall CameraSetUndistortEnable(
5559     CameraHandle          hCamera,
5560     BOOL                 bEnable
5561 );
5562
5574 MVSDK_API CameraSdkStatus __stdcall CameraGetUndistortEnable(
5575     CameraHandle          hCamera,
5576     BOOL*                bEnable
5577 );
5578
5590 MVSDK_API CameraSdkStatus __stdcall CameraCustomizeUndistort(
5591     CameraHandle          hCamera,
5592     HWND                  hParent
5593 );
5594
5606 MVSDK_API CameraSdkStatus __stdcall CameraGetEyeCount(
5607     CameraHandle          hCamera,
5608     int*                 EyeCount
5609 );
5610
5634 MVSDK_API CameraSdkStatus __stdcall CameraMultiEyeImageProcess(
```

```
5635     CameraHandle      hCamera,
5636     int             iEyeIndex,
5637     BYTE*           pbyIn,
5638     tSdkFrameHead* pInFrInfo,
5639     BYTE*           pbyOut,
5640     tSdkFrameHead* pOutFrInfo,
5641     UINT            uOutFormat,
5642     UINT            uReserved
5643 );
5644
5668 MVSDK_API CameraSdkStatus __stdcall CameraGetRegionAverageGray(
5669     BYTE *pFrameBuffer,
5670     tSdkFrameHead* pFrameHead,
5671     int Left,
5672     int Top,
5673     int Width,
5674     int Height,
5675     int *AvgGray
5676 );
5677
5691 MVSDK_API CameraSdkStatus __stdcall CameraGetMediaCapability(
5692     CameraHandle    hCamera,
5693     int             iMediaType,
5694     UINT            *uCap
5695 );
5696
5710 MVSDK_API CameraSdkStatus __stdcall CameraSetMediaBitRate(
5711     CameraHandle    hCamera,
5712     int             iMediaType,
5713     UINT            uRate
5714 );
5715
5729 MVSDK_API CameraSdkStatus __stdcall CameraGetMediaBitRate(
5730     CameraHandle    hCamera,
5731     int             iMediaType,
5732     UINT            *uRate
5733 );
5734
5750 MVSDK_API CameraSdkStatus __stdcall CameraSetFrameEventCallback(
5751     CameraHandle    hCamera,
5752     CAMERA_FRAME_EVENT_CALLBACK pCallBack,
5753     PVOID           pContext
5754 );
5755
5756 #endif
```

# MVSDK API

Include >

## CameraDefine.h

```
1 #pragma once
2 #ifndef _CAMERA_DEFINE_H_
3 #define _CAMERA_DEFINE_H_
4
5 #include "CameraStatus.h"
6
7 #define MAX_CROSS_LINE 9
8
12 typedef int CameraHandle;
13
14
15
19 typedef enum
20 {
21     LUTMODE_PARAM_GEN=0,
22     LUTMODE_PRESET=1,
23     LUTMODE_USER_DEF=2
24 }emSdkLutMode;
25
29 typedef enum
30 {
33     RUNMODE_PLAY=0,
34     RUNMODE_PAUSE=1,
35     RUNMODE_STOP=2
36 }emSdkRunMode;
37
41 typedef enum
42 {
43     DISPLAYMODE_SCALE=0,
44     DISPLAYMODE_REAL=1,
```

```
45     DISPLAYMODE_2X=2,
46     DISPLAYMODE_4X=3,
47     DISPLAYMODE_8X=4,
48     DISPLAYMODE_16X=5,
49     DISPLAYMODE_SCALE_FIT=6
50 }emSdkDisplayMode;
51
55 typedef enum
56 {
57     RECORD_STOP=0,
58     RECORD_START=1,
59     RECORD_PAUSE=2
60 }emSdkRecordMode;
61
65 typedef enum
66 {
67     MIRROR_DIRECTION_HORIZONTAL=0,
68     MIRROR_DIRECTION_VERTICAL=1
69 }emSdkMirrorDirection;
70
74 typedef enum
75 {
76     ROTATE_DIRECTION_0=0,
77     ROTATE_DIRECTION_90=1,
78     ROTATE_DIRECTION_180=2,
79     ROTATE_DIRECTION_270=3,
80 }emSdkRotateDirection;
81
85 typedef enum
86 {
87     FRAME_SPEED_LOW=0,
88     FRAME_SPEED_NORMAL=1,
89     FRAME_SPEED_HIGH=2,
90     FRAME_SPEED_SUPER=3
91 }emSdkFrameSpeed;
92
96 typedef enum
```

```
97 {  
98     FILE_JPG = 1,  
99     FILE_BMP = 2,  
100    FILE_RAW = 4,  
101    FILE_PNG = 8,  
102    FILE_BMP_8BIT = 16,  
103    FILE_PNG_8BIT = 32,  
104    FILE_RAW_16BIT = 64  
105 }emSdkFileType;  
106  
110 typedef enum  
111 {  
114     CONTINUATION=0,  
115  
118     SOFT_TRIGGER=1,  
119  
122     EXTERNAL_TRIGGER=2,  
123  
126     ROTARYENC_TRIGGER=3,  
127  
130     ROTARYENC_COND_TRIGGER=4,  
131 } emSdkSnapMode;  
132  
136 typedef enum  
137 {  
140     LIGHT_FREQUENCY_50HZ=0,  
141  
144     LIGHT_FREQUENCY_60HZ=1  
145 }emSdkLightFrequency;  
146  
150 typedef enum  
151 {  
152     PARAMETER_TEAM_DEFAULT = 0xff,  
153     PARAMETER_TEAM_A = 0,  
154     PARAMETER_TEAM_B = 1,  
155     PARAMETER_TEAM_C = 2,  
156     PARAMETER_TEAM_D = 3
```

```
157 }emSdkParameterTeam;
158
159
179 typedef enum
180 {
185     PARAM_MODE_BY_MODEL=0,
186
199     PARAM_MODE_BY_NAME=1,
200
209     PARAM_MODE_BY_SN=2,
210
213     PARAM_MODE_IN_DEVICE=3
214 }emSdkParameterMode;
215
219 typedef enum
220 {
221     PROP_SHEET_INDEX_EXPOSURE=0,
222     PROP_SHEET_INDEX_ISP_COLOR=1,
223     PROP_SHEET_INDEX_ISP_LUT=2,
224     PROP_SHEET_INDEX_ISP_SHAPE=3,
225     PROP_SHEET_INDEX_VIDEO_FORMAT=4,
226     PROP_SHEET_INDEX_RESOLUTION=5,
227     PROP_SHEET_INDEX_IO_CTRL=6,
228     PROP_SHEET_INDEX_TRIGGER_SET=7,
229     PROP_SHEET_INDEX_OVERLAY=8,
230     PROP_SHEET_INDEX_DEVICE_INFO=9,
231     PROP_SHEET_INDEX_WDR=10,
232     PROP_SHEET_INDEX_MULTI_EXPOSURE=11,
233 }emSdkPropSheetMask;
234
238 typedef enum
239 {
240     SHEET_MSG_LOAD_PARAM_DEFAULT=0,
241     SHEET_MSG_LOAD_PARAM_GROUP=1,
242     SHEET_MSG_LOAD_PARAM_FROMFILE=2,
243     SHEET_MSG_SAVE_PARAM_GROUP=3
244 }emSdkPropSheetMsg;
```

```
245
249 typedef enum
250 {
251     REF_WIN_AUTO_EXPOSURE=0,
252     REF_WIN_WHITE_BALANCE=1,
253 }emSdkRefWinType;
254
258 typedef enum
259 {
260     RES_MODE_PREVIEW=0,
261     RES_MODE_SNAPSHOT=1,
262 }emSdkResolutionMode;
263
267 typedef enum
268 {
269     CT_MODE_AUTO=0,
270     CT_MODE_PRESET=1,
271     CT_MODE_USER_DEF=2
272 }emSdkClrTmpMode;
273
277 typedef enum
278 {
279     LUT_CHANNEL_ALL=0,
280     LUT_CHANNEL_RED=1,
281     LUT_CHANNEL_GREEN=2,
282     LUT_CHANNEL_BLUE=3,
283 }emSdkLutChannel;
284
288 typedef enum
289 {
290     ISP_PROCESSSOR_PC=0,
291     ISP_PROCESSSOR_DEVICE=1
292 }emSdkIspProcessor;
293
297 typedef enum
298 {
299     STROBE_SYNC_WITH_TRIG_AUTO=0,
```

```
300     STROBE_SYNC_WITH_TRIG_MANUAL=1,
301     STROBE_ALWAYS_HIGH=2,
302     STROBE_ALWAYS_LOW=3
303 }emStrobeControl;
304
308 typedef enum
309 {
310     EXT_TRIG.LEADING_EDGE=0,
311     EXT_TRIG.TRAILING_EDGE=1,
312     EXT_TRIG.HIGH_LEVEL=2,
313     EXT_TRIG.LOW_LEVEL=3,
314     EXT_TRIG.DOUBLE_EDGE=4,
315 }emExtTrigSignal;
316
320 typedef enum
321 {
322     EXT_TRIG.EXP_STANDARD=0,
323     EXT_TRIG.EXP_GRR=1,
324 }emExtTrigShutterMode;
325
329 typedef enum
330 {
331     EVALUATE_DEFINITION_DEVIATION=0,
332     EVALUATE_DEFINITION_SMD=1,
333     EVALUATE_DEFINITION_GRADIENT=2,
334     EVALUATE_DEFINITION_SOBEL=3,
335     EVALUATE_DEFINITION_ROBERT=4,
336     EVALUATE_DEFINITION_LAPLACE=5,
337
338     EVALUATE_DEFINITION_ALG_MAX=6,
339 }emEvaluateDefinitionAlgorith;
340
344 typedef enum
345 {
346     CAMERA_DT_VCENTER      = 0x1,
347     CAMERA_DT_BOTTOM        = 0x2,
348     CAMERA_DT_HCENTER       = 0x4,
```

```
349     CAMERA_DT_RIGHT          = 0x8,
350     CAMERA_DT_SINGLELINE      = 0x10,
351     CAMERA_DT_ALPHA_BLEND    = 0x20,
352     CAMERA_DT_ANTI_ALIASING = 0x40,
353 }emCameraDrawTextFlags;
354
358 typedef enum
359 {
360     IOMODE_TRIG_INPUT=0,
361     IOMODE_STROBE_OUTPUT=1,
362     IOMODE_GP_INPUT=2,
363     IOMODE_GP_OUTPUT=3,
364     IOMODE_PWM_OUTPUT=4,
365     IOMODE_ROTARYENC_INPUT=5,
366 }emCameraGPIOMode;
367
371 typedef enum
372 {
373     IOFORMAT_SINGLE=0,
374     IOFORMAT_RS422=1,
375     IOFORMAT_RS422_TERM=2,
376 }emCameraGPIOFormat;
377
381 typedef enum
382 {
383     CAMERA_GET_IMAGE_PRIORITY_OLEDEST=0,
384     CAMERA_GET_IMAGE_PRIORITY_NEWEST=1,
385
390     CAMERA_GET_IMAGE_PRIORITY_NEXT=2,
391 }emCameraGetImagePriority;
392
396 typedef enum
397 {
398     CAMERA_ST_CLEAR_BUFFER_BEFORE    = 0x1,
399 }emCameraSoftTriggerExFlags;
400
403 typedef struct
```

```
404 {  
405     char acProductSeries[32];  
406     char acProductName[32];  
407  
410     char acFriendlyName[32];  
411     char acLinkName[32];  
412     char acDriverVersion[32];  
413     char acSensorType[32];  
414     char acPortType[32];  
415     char acSn[32];  
416     UINT uInstance;  
417 } tSdkCameraDevInfo;  
418  
421 #define EXT_TRIGGER_MASK_GRR_SHUTTER 1  
422 #define EXT_TRIGGER_MASK_LEVEL_MODE 2  
423 #define EXT_TRIGGER_MASK_DOUBLE_EDGE 4  
424 #define EXT_TRIGGER_MASK_BUFFERED_DELAY 8  
425  
426 //tSdkResolutionRange结构体中SKIP、BIN、  
    RESAMPLE模式的掩码值  
427 #define MASK_2X2_HD      (1<<0)      //硬件SKIP、  
    BIN、重采样 2X2  
428 #define MASK_3X3_HD      (1<<1)  
429 #define MASK_4X4_HD      (1<<2)  
430 #define MASK_5X5_HD      (1<<3)  
431 #define MASK_6X6_HD      (1<<4)  
432 #define MASK_7X7_HD      (1<<5)  
433 #define MASK_8X8_HD      (1<<6)  
434 #define MASK_9X9_HD      (1<<7)  
435 #define MASK_10X10_HD     (1<<8)  
436 #define MASK_11X11_HD     (1<<9)  
437 #define MASK_12X12_HD     (1<<10)  
438 #define MASK_13X13_HD     (1<<11)  
439 #define MASK_14X14_HD     (1<<12)  
440 #define MASK_15X15_HD     (1<<13)  
441 #define MASK_16X16_HD     (1<<14)  
442 #define MASK_17X17_HD     (1<<15)
```

```
443 #define MASK_2X2_SW      (1<<16) //软件SKIP、  
    BIN、重采样 2X2  
444 #define MASK_3X3_SW      (1<<17)  
445 #define MASK_4X4_SW      (1<<18)  
446 #define MASK_5X5_SW      (1<<19)  
447 #define MASK_6X6_SW      (1<<20)  
448 #define MASK_7X7_SW      (1<<21)  
449 #define MASK_8X8_SW      (1<<22)  
450 #define MASK_9X9_SW      (1<<23)  
451 #define MASK_10X10_SW     (1<<24)  
452 #define MASK_11X11_SW     (1<<25)  
453 #define MASK_12X12_SW     (1<<26)  
454 #define MASK_13X13_SW     (1<<27)  
455 #define MASK_14X14_SW     (1<<28)  
456 #define MASK_15X15_SW     (1<<29)  
457 #define MASK_16X16_SW     (1<<30)  
458 #define MASK_17X17_SW     (1<<31)  
459  
463 typedef struct  
464 {  
465     INT iHeightMax;  
466     INT iHeightMin;  
467     INT iWidthMax;  
468     INT iWidthMin;  
469     UINT uSkipModeMask;  
470     UINT uBinSumModeMask;  
471     UINT uBinAverageModeMask;  
472     UINT uResampleMask;  
473 } tSdkResolutionRange;  
474  
477 typedef struct  
478 {  
479     INT     iIndex;  
480     char    acDescription[ 32 ];  
481     UINT    uBinSumMode;  
482     UINT    uBinAverageMode;  
483     UINT    uSkipMode;
```

```
484     UINT    uResampleMask;
485     INT     iHOffsetFOV;
486     INT    iVOffsetFOV;
487     INT    iWidthFOV;
488     INT    iHeightFOV;
489     INT    iWidth;
490     INT    iHeight;
491     INT    iWidthZoomHd;
492     INT    iHeightZoomHd;
493     INT    iWidthZoomSw;
494     INT    iHeightZoomSw;
495 } tSdkImageResolution;
496
499 typedef struct
500 {
501     INT    iIndex;
502     char   acDescription[32];
503 } tSdkColorTemperatureDes;
504
507 typedef struct
508 {
509     INT    iIndex;
510     char   acDescription[32];
511 } tSdkFrameSpeed;
512
517 typedef struct
518 {
519     UINT    uiTargetMin;
520     UINT    uiTargetMax;
521     UINT    uiAnalogGainMin;
522     UINT    uiAnalogGainMax;
523     float   fAnalogGainStep;
524     UINT    uiExposeTimeMin;
525     UINT    uiExposeTimeMax;
526 } tSdkExpose;
527
530 typedef struct
```

```
531 {  
532     INT    iIndex;  
533     char   acDescription[32];  
534 } tSdkTrigger;  
535  
536  
537     typedef struct  
538 {  
539         INT    iIndex;  
540         char   acDescription[32];  
541         UINT   iPackSize;  
542 } tSdkPackLength;  
543  
544  
545     typedef struct  
546 {  
547         INT    iIndex;  
548         char   acDescription[32];  
549 } tSdkPresetLut;  
550  
551  
552     typedef struct  
553 {  
554         INT    iIndex;  
555         char   acDescription[32];  
556 } tSdkAeAlgorithm;  
557  
558  
559     typedef struct  
560 {  
561         INT    iIndex;  
562         char   acDescription[32];  
563 } tSdkBayerDecodeAlgorithm;  
564  
565  
566     typedef struct  
567 {  
568         INT    iTotals;  
569         INT    iCapture;  
570         INT    iLost;  
571 } tSdkFrameStatistic;  
572  
573  
574  
575  
576  
577 }
```

```
580 | typedef struct
581 | {
582 |     INT      iIndex;
583 |     char     acDescription[ 32 ];
584 |     UINT     iMediaType;
585 | } tSdkMediaType;
586 |
589 | typedef struct
590 | {
591 |     INT  iMin;
592 |     INT  iMax;
593 | } tGammaRange;
594 |
597 | typedef struct
598 | {
599 |     INT  iMin;
600 |     INT  iMax;
601 | } tContrastRange;
602 |
605 | typedef struct
606 | {
607 |     INT  iRGainMin;
608 |     INT  iRGainMax;
609 |     INT  iGGainMin;
610 |     INT  iGGainMax;
611 |     INT  iBGainMin;
612 |     INT  iBGainMax;
613 | } tRgbGainRange;
614 |
617 | typedef struct
618 | {
619 |     INT  iMin;
620 |     INT  iMax;
621 | } tSaturationRange;
622 |
625 | typedef struct
626 | {
```

```
627     INT iMin;
628     INT iMax;
629 } tSharpnessRange;
630
633 typedef struct
634 {
635     BOOL bMonoSensor;
636     BOOL bWbOnce;
637     BOOL bAutoWb;
638     BOOL bAutoExposure;
639     BOOL bManualExposure;
640     BOOL bAntiFlick;
641     BOOL bDeviceIsp;
642     BOOL bForceUseDeviceIsp;
643     BOOL bZoomHD;
644 } tSdkIspCapacity;
645
650 typedef struct
651 {
652
653     tSdkTrigger *pTriggerDesc;
654     INT         iTriggerDesc;
655
656     tSdkImageResolution *pImageSizeDesc;
657     INT                 iImageSizeDesc;
658
659     tSdkColorTemperatureDes *pClrTempDesc;
660     INT                     iClrTempDesc;
661
662     tSdkMediaType      *pMediaTypeDesc;
663     INT                  iMediaTypdeDesc;
664
665     tSdkFrameSpeed    *pFrameSpeedDesc;
666     INT                  iFrameSpeedDesc;
667
668     tSdkPackLength    *pPackLenDesc;
669     INT                  iPackLenDesc;
```

```
670  
671     INT          iOutputIoCounts;  
672     INT          iInputIoCounts;  
673  
674     tSdkPresetLut *pPresetLutDesc;  
675     INT          iPresetLut;  
676  
677     INT          iUserDataMaxLen;  
678     BOOL         bParamInDevice;  
679  
680     tSdkAeAlgorithm *pAeAlmSwDesc;  
681     int           iAeAlmSwDesc;  
682  
683     tSdkAeAlgorithm *pAeAlmHdDesc;  
684     int           iAeAlmHdDesc;  
685  
686     tSdkBayerDecodeAlgorithm  
*pBayerDecAlmSwDesc;  
687     int           iBayerDecAlmSwDesc;  
688  
689     tSdkBayerDecodeAlgorithm  
*pBayerDecAlmHdDesc;  
690     int           iBayerDecAlmHdDesc;  
691  
692     /* 图像参数的调节范围定义,用于动态构建UI */  
693     tSdkExpose      sExposeDesc;  
694     tSdkResolutionRange sResolutionRange;  
695     tRgbGainRange   sRgbGainRange;  
696     tSaturationRange sSaturationRange;  
697     tGammaRange    sGammaRange;  
698     tContrastRange sContrastRange;  
699     tSharpnessRange sSharpnessRange;  
700     tSdkIspCapacity sIspCapacity;  
701  
702
```

```
703 } tSdkCameraCapbility;
704
705
708 typedef struct
709 {
710     UINT      uiMediaType;
711     UINT      uBytes;
712     INT       iWidth;
713     INT       iHeight;
714     INT       iWidthZoomSw;
715     INT       iHeightZoomSw;
716     BOOL      bIsTrigger;
717     UINT      uiTimeStamp;
718     UINT      uiExpTime;
719     float    fAnalogGain;
720     INT       iGamma;
721     INT       iContrast;
722     INT       iSaturation;
723     float    fRgain;
724     float    fGgain;
725     float    fBgain;
726 }tSdkFrameHead;
727
730 typedef struct sCameraFrame
731 {
732     tSdkFrameHead    head;
733     BYTE *          pBuffer;
734 }tSdkFrame;
735
738 typedef struct tSdkFrameEvent_
739 {
740     UINT      uType;
741     UINT      uStatus;
742     UINT      uFrameID;
743     UINT      uWidth;
744     UINT      uHeight;
745     UINT      uPixelFormat;
```

```
746     UINT      TimeStampL;
747     UINT      TimeStampH;
748 }tSdkFrameEvent;
749
753 typedef void (WINAPI* CAMERA_SNAP_PROC)
    (CameraHandle hCamera, BYTE *pFrameBuffer,
     tSdkFrameHead* pFrameHead,PVOID pContext);
754
758 typedef void (WINAPI* CAMERA_PAGE_MSG_PROC)
    (CameraHandle hCamera,UINT MSG,UINT
     uParam,PVOID pContext);
759
787 typedef void (WINAPI*
    CAMERA_CONNECTION_STATUS_CALLBACK)(CameraHandle
    hCamera,UINT MSG,UINT uParam,PVOID pContext);
788
792 typedef void (WINAPI*
    CAMERA_FRAME_EVENT_CALLBACK)(CameraHandle
    hCamera, tSdkFrameEvent* pEvent, PVOID pContext);
793
794
796 // Grabber 相关
797
800 typedef struct
801 {
802     int Width;
803     int Height;
804     int Disp;
805     int Capture;
806     int Lost;
807     int Error;
808     float DispFps;
809     float CapFps;
810 }tSdkGrabberStat;
811
815 typedef void (_stdcall
    *pfnCameraGrabberFrameCallback)(
```

```
816     void* Grabber,
817     BYTE *pFrameBuffer,
818     tSdkFrameHead* pFrameHead,
819     void* Context);
820
846     typedef int (_stdcall
847         *pfnCameraGrabberFrameListener)(
848             void* Grabber,
849             int Phase,
850             BYTE *pFrameBuffer,
851             tSdkFrameHead* pFrameHead,
852             void* Context);
853
858     typedef void (_stdcall
859         *pfnCameraGrabberSaveImageComplete)(
860             void* Grabber,
861             void* Image, // 需要调用
862             CameraImage _Destroy释放
863             CameraSdkStatus Status,
864             void* Context
865             );
866
867
868 //-----IMAGE FORMAT
869 //-----图像格式定义-----
870
871
872
873
874
```

```
875 | #define CAMERA_MEDIA_TYPE_OCCUPY1BIT  
     0x00010000  
876 | #define CAMERA_MEDIA_TYPE_OCCUPY2BIT  
     0x00020000  
877 | #define CAMERA_MEDIA_TYPE_OCCUPY4BIT  
     0x00040000  
878 | #define CAMERA_MEDIA_TYPE_OCCUPY8BIT  
     0x00080000  
879 | #define CAMERA_MEDIA_TYPE_OCCUPY10BIT  
     0x000A0000  
880 | #define CAMERA_MEDIA_TYPE_OCCUPY12BIT  
     0x000C0000  
881 | #define CAMERA_MEDIA_TYPE_OCCUPY16BIT  
     0x00100000  
882 | #define CAMERA_MEDIA_TYPE_OCCUPY24BIT  
     0x00180000  
883 | #define CAMERA_MEDIA_TYPE_OCCUPY32BIT  
     0x00200000  
884 | #define CAMERA_MEDIA_TYPE_OCCUPY36BIT  
     0x00240000  
885 | #define CAMERA_MEDIA_TYPE_OCCUPY48BIT  
     0x00300000  
886 | #define CAMERA_MEDIA_TYPE_OCCUPY64BIT  
     0x00400000  
887 |  
888 | #define  
      CAMERA_MEDIA_TYPE_EFFECTIVE_PIXEL_SIZE_MASK  
      0x00FF0000  
889 | #define  
      CAMERA_MEDIA_TYPE_EFFECTIVE_PIXEL_SIZE_SHIFT  
      16  
890 |  
891 | #define CAMERA_MEDIA_TYPE_PIXEL_SIZE(type)  
     (((type) &  
       CAMERA_MEDIA_TYPE_EFFECTIVE_PIXEL_SIZE_MASK) >>  
       CAMERA_MEDIA_TYPE_EFFECTIVE_PIXEL_SIZE_SHIFT)  
892 |
```

```
893 |
894 | #define CAMERA_MEDIA_TYPE_ID_MASK
895 |     0x0000FFFF
896 |
897 | /*mono*/
898 | #define CAMERA_MEDIA_TYPE_MONO1P
899 |     (CAMERA_MEDIA_TYPE_MONO |
900 |      CAMERA_MEDIA_TYPE_OCCUPY1BIT | 0x0037)
901 | #define CAMERA_MEDIA_TYPE_MONO2P
902 |     (CAMERA_MEDIA_TYPE_MONO |
903 |      CAMERA_MEDIA_TYPE_OCCUPY2BIT | 0x0038)
904 | #define CAMERA_MEDIA_TYPE_MONO4P
905 |     (CAMERA_MEDIA_TYPE_MONO |
906 |      CAMERA_MEDIA_TYPE_OCCUPY4BIT | 0x0039)
907 | #define CAMERA_MEDIA_TYPE_MONO8
908 |     (CAMERA_MEDIA_TYPE_MONO |
909 |      CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0001)
910 | #define CAMERA_MEDIA_TYPE_MONO8S
911 |     (CAMERA_MEDIA_TYPE_MONO |
912 |      CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0002)
913 | #define CAMERA_MEDIA_TYPE_MONO10
914 |     (CAMERA_MEDIA_TYPE_MONO |
915 |      CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0003)
916 | #define CAMERA_MEDIA_TYPE_MONO10_PACKED
917 |     (CAMERA_MEDIA_TYPE_MONO |
918 |      CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0004)
919 | #define CAMERA_MEDIA_TYPE_MONO12
920 |     (CAMERA_MEDIA_TYPE_MONO |
921 |      CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0005)
922 | #define CAMERA_MEDIA_TYPE_MONO12_PACKED
923 |     (CAMERA_MEDIA_TYPE_MONO |
924 |      CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0006)
925 | #define CAMERA_MEDIA_TYPE_MONO14
926 |     (CAMERA_MEDIA_TYPE_MONO |
927 |      CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0025)
```

```
908 | #define CAMERA_MEDIA_TYPE_MONO16  
909 | (CAMERA_MEDIA_TYPE_MONO |  
910 | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0007)  
911 | /*Bayer */  
912 | #define CAMERA_MEDIA_TYPE_BAYGR8  
913 | (CAMERA_MEDIA_TYPE_MONO |  
914 | CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0008)  
915 | #define CAMERA_MEDIA_TYPE_BAYRG8  
916 | (CAMERA_MEDIA_TYPE_MONO |  
917 | CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0009)  
918 | #define CAMERA_MEDIA_TYPE_BAYGB8  
919 | (CAMERA_MEDIA_TYPE_MONO |  
920 | CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x000A)  
921 | #define CAMERA_MEDIA_TYPE_BAYBG8  
922 | (CAMERA_MEDIA_TYPE_MONO |  
923 | CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x000B)  
924 |  
925 | #define CAMERA_MEDIA_TYPE_BAYGR10_MIPI  
926 | (CAMERA_MEDIA_TYPE_MONO |  
927 | CAMERA_MEDIA_TYPE_OCCUPY10BIT | 0x0026)  
928 | #define CAMERA_MEDIA_TYPE_BAYRG10_MIPI  
929 | (CAMERA_MEDIA_TYPE_MONO |  
930 | CAMERA_MEDIA_TYPE_OCCUPY10BIT | 0x0027)  
931 | #define CAMERA_MEDIA_TYPE_BAYGB10_MIPI  
932 | (CAMERA_MEDIA_TYPE_MONO |  
933 | CAMERA_MEDIA_TYPE_OCCUPY10BIT | 0x0028)  
934 | #define CAMERA_MEDIA_TYPE_BAYBG10_MIPI  
935 | (CAMERA_MEDIA_TYPE_MONO |  
936 | CAMERA_MEDIA_TYPE_OCCUPY10BIT | 0x0029)  
937 |  
938 | #define CAMERA_MEDIA_TYPE_BAYGR10  
939 | (CAMERA_MEDIA_TYPE_MONO |  
940 | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x000C)  
941 | #define CAMERA_MEDIA_TYPE_BAYRG10  
942 | (CAMERA_MEDIA_TYPE_MONO |
```

```
    CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x000D)
924 | #define CAMERA_MEDIA_TYPE_BAYGB10
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x000E)
925 | #define CAMERA_MEDIA_TYPE_BAYBG10
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x000F)
926 |
927 | #define CAMERA_MEDIA_TYPE_BAYGR12
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0010)
928 | #define CAMERA_MEDIA_TYPE_BAYRG12
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0011)
929 | #define CAMERA_MEDIA_TYPE_BAYGB12
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0012)
930 | #define CAMERA_MEDIA_TYPE_BAYBG12
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0013)
931 |
932 |
933 | #define CAMERA_MEDIA_TYPE_BAYGR10_PACKED
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0026)
934 | #define CAMERA_MEDIA_TYPE_BAYRG10_PACKED
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0027)
935 | #define CAMERA_MEDIA_TYPE_BAYGB10_PACKED
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0028)
936 | #define CAMERA_MEDIA_TYPE_BAYBG10_PACKED
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0029)
937 |
938 | #define CAMERA_MEDIA_TYPE_BAYGR12_PACKED
    (CAMERA_MEDIA_TYPE_MONO |
```

```
    CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x002A)
939 | #define CAMERA_MEDIA_TYPE_BAYRG12_PACKED
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x002B)
940 | #define CAMERA_MEDIA_TYPE_BAYGB12_PACKED
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x002C)
941 | #define CAMERA_MEDIA_TYPE_BAYBG12_PACKED
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x002D)
942 |
943 | #define CAMERA_MEDIA_TYPE_BAYGR16
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x002E)
944 | #define CAMERA_MEDIA_TYPE_BAYRG16
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x002F)
945 | #define CAMERA_MEDIA_TYPE_BAYGB16
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0030)
946 | #define CAMERA_MEDIA_TYPE_BAYBG16
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0031)
947 |
948 | /*RGB */
949 | #define CAMERA_MEDIA_TYPE_RGB8
    (CAMERA_MEDIA_TYPE_COLOR |
     CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0014)
950 | #define CAMERA_MEDIA_TYPE_BGR8
    (CAMERA_MEDIA_TYPE_COLOR |
     CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0015)
951 | #define CAMERA_MEDIA_TYPE_RGBA8
    (CAMERA_MEDIA_TYPE_COLOR |
     CAMERA_MEDIA_TYPE_OCCUPY32BIT | 0x0016)
952 | #define CAMERA_MEDIA_TYPE_BGRA8
    (CAMERA_MEDIA_TYPE_COLOR |
     CAMERA_MEDIA_TYPE_OCCUPY32BIT | 0x0017)
```

```
953 | #define CAMERA_MEDIA_TYPE_RGB10
      (CAMERA_MEDIA_TYPE_COLOR |
       CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0018)
954 | #define CAMERA_MEDIA_TYPE_BGR10
      (CAMERA_MEDIA_TYPE_COLOR |
       CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0019)
955 | #define CAMERA_MEDIA_TYPE_RGB12
      (CAMERA_MEDIA_TYPE_COLOR |
       CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x001A)
956 | #define CAMERA_MEDIA_TYPE_BGR12
      (CAMERA_MEDIA_TYPE_COLOR |
       CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x001B)
957 | #define CAMERA_MEDIA_TYPE_RGB16
      (CAMERA_MEDIA_TYPE_COLOR |
       CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0033)
958 | #define CAMERA_MEDIA_TYPE_BGR16
      (CAMERA_MEDIA_TYPE_COLOR |
       CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x004B)
959 | #define CAMERA_MEDIA_TYPE_RGBA16
      (CAMERA_MEDIA_TYPE_COLOR |
       CAMERA_MEDIA_TYPE_OCCUPY64BIT | 0x0064)
960 | #define CAMERA_MEDIA_TYPE_BGRA16
      (CAMERA_MEDIA_TYPE_COLOR |
       CAMERA_MEDIA_TYPE_OCCUPY64BIT | 0x0051)
961 | #define CAMERA_MEDIA_TYPE_RGB10V1_PACKED
      (CAMERA_MEDIA_TYPE_COLOR |
       CAMERA_MEDIA_TYPE_OCCUPY32BIT | 0x001C)
962 | #define CAMERA_MEDIA_TYPE_RGB10P32
      (CAMERA_MEDIA_TYPE_COLOR |
       CAMERA_MEDIA_TYPE_OCCUPY32BIT | 0x001D)
963 | #define CAMERA_MEDIA_TYPE_RGB12V1_PACKED
      (CAMERA_MEDIA_TYPE_COLOR |
       CAMERA_MEDIA_TYPE_OCCUPY36BIT | 0X0034)
964 | #define CAMERA_MEDIA_TYPE_RGB565P
      (CAMERA_MEDIA_TYPE_COLOR |
       CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0035)
```

```
965 | #define CAMERA_MEDIA_TYPE_BGR565P  
966 |     (CAMERA_MEDIA_TYPE_COLOR |  
967 |     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0X0036)  
966 |  
967 | /*YUV and YCbCr*/  
968 | #define CAMERA_MEDIA_TYPE_YUV411_8_UYYVYY  
969 |     (CAMERA_MEDIA_TYPE_COLOR |  
970 |     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x001E)  
969 | #define CAMERA_MEDIA_TYPE_YUV422_8_UYVY  
971 |     (CAMERA_MEDIA_TYPE_COLOR |  
972 |     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x001F)  
970 | #define CAMERA_MEDIA_TYPE_YUV422_8  
971 |     (CAMERA_MEDIA_TYPE_COLOR |  
972 |     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0032)  
971 | #define CAMERA_MEDIA_TYPE_YUV8_UYV  
972 |     (CAMERA_MEDIA_TYPE_COLOR |  
973 |     CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0020)  
972 | #define CAMERA_MEDIA_TYPE_YCBCR8_CBYCR  
973 |     (CAMERA_MEDIA_TYPE_COLOR |  
974 |     CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x003A)  
973 | //CAMERA_MEDIA_TYPE_YCBCR422_8 : YYYYCbCrCbCr  
974 | #define CAMERA_MEDIA_TYPE_YCBCR422_8  
975 |     (CAMERA_MEDIA_TYPE_COLOR |  
976 |     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x003B)  
975 | #define CAMERA_MEDIA_TYPE_YCBCR422_8_CBYCRY  
976 |     (CAMERA_MEDIA_TYPE_COLOR |  
977 |     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0043)  
976 | #define CAMERA_MEDIA_TYPE_YCBCR411_8_CBYYCRY  
977 |     (CAMERA_MEDIA_TYPE_COLOR |  
978 |     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x003C)  
977 | #define CAMERA_MEDIA_TYPE_YCBCR601_8_CBYCR  
978 |     (CAMERA_MEDIA_TYPE_COLOR |  
978 |     CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x003D)  
978 | #define CAMERA_MEDIA_TYPE_YCBCR601_422_8  
978 |     (CAMERA_MEDIA_TYPE_COLOR |  
978 |     CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x003E)
```

```
979 | #define  
CAMERA_MEDIA_TYPE_YCBCR601_422_8_CBYCRY  
(CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0044)  
980 | #define  
CAMERA_MEDIA_TYPE_YCBCR601_411_8_CBYYCRY  
(CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x003F)  
981 | #define CAMERA_MEDIA_TYPE_YCBCR709_8_CBYCR  
(CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0040)  
982 | #define CAMERA_MEDIA_TYPE_YCBCR709_422_8  
(CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0041)  
983 | #define  
CAMERA_MEDIA_TYPE_YCBCR709_422_8_CBYCRY  
(CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0045)  
984 | #define  
CAMERA_MEDIA_TYPE_YCBCR709_411_8_CBYYCRY  
(CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0042)  
985 |  
986 | /*RGB Planar */  
987 | #define CAMERA_MEDIA_TYPE_RGB8_PLANAR  
(CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0021)  
988 | #define CAMERA_MEDIA_TYPE_RGB10_PLANAR  
(CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0022)  
989 | #define CAMERA_MEDIA_TYPE_RGB12_PLANAR  
(CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0023)  
990 | #define CAMERA_MEDIA_TYPE_RGB16_PLANAR  
(CAMERA_MEDIA_TYPE_COLOR |  
CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0024)  
991 |
```

```
992 /*MindVision 12bit packed bayer*/
993 #define CAMERA_MEDIA_TYPE_BAYGR12_PACKED_MV
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0060)
994 #define CAMERA_MEDIA_TYPE_BAYRG12_PACKED_MV
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0061)
995 #define CAMERA_MEDIA_TYPE_BAYGB12_PACKED_MV
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0062)
996 #define CAMERA_MEDIA_TYPE_BAYBG12_PACKED_MV
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0063)
997
998 /*MindVision 12bit packed monochome*/
999 #define CAMERA_MEDIA_TYPE_MONO12_PACKED_MV
    (CAMERA_MEDIA_TYPE_MONO |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0064)
1000 #define CAMERA_MEDIA_TYPE_YUV420P_MV
    (CAMERA_MEDIA_TYPE_COLOR |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0065)
1001
1002 /*planar YUV 4:2:0, 12bpp, 1 plane for Y and
   1 plane for the UV components, which are
   interleaved (first byte V and the following
   byte U)*/
1003 #define CAMERA_MEDIA_TYPE_YUV_NV21_MV
    (CAMERA_MEDIA_TYPE_COLOR |
     CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0066)
1004
1005 /* H264 H265 */
1006 #define CAMERA_MEDIA_TYPE_H264_MV
    (CAMERA_MEDIA_TYPE_COLOR |
     CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0067)
1007 #define CAMERA_MEDIA_TYPE_H265_MV
    (CAMERA_MEDIA_TYPE_COLOR |
     CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0068)
```

```
1008 | /* JPEG */
1009 | #define CAMERA_MEDIA_TYPE_JPEG_MV
1010 |     (CAMERA_MEDIA_TYPE_COLOR |
1011 |      CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0069)
1013 |
1014 | #endif
```

# MVSDK API

Include >

## CameraGrabber.h

```
1 #ifndef _MV_CAMERA_GRABBER_H_
2 #define _MV_CAMERA_GRABBER_H_
3
4 #include "CameraDefine.h"
5 #include "CameraStatus.h"
6
7
19 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_CreateFromDevicePage(
20     void** Grabber
21 );
22
36 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_CreateByIndex(
37     void** Grabber,
38     int Index
39 );
40
54 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_CreateByName(
55     void** Grabber,
56     char* Name
57 );
58
72 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_Create(
73     void** Grabber,
74     tSdkCameraDevInfo* pDevInfo
75 );
76
86 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_Destroy(
87     void* Grabber
88 );
89
101 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_SetHWnd(
102     void* Grabber,
103     HWND hWnd
104 );
105
117 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_SetPriority(
118     void* Grabber,
119     UINT Priority
120 );
121
133 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_StartLive(
134     void* Grabber
135 );
136
148 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_StopLive(
149     void* Grabber
150 );
```

```
151
165 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_SaveImage(
166     void* Grabber,
167     void** Image,
168     DWORD TimeOut
169 );
170
182 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_SaveImageAsync(
183     void* Grabber
184 );
185
199 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_SaveImageAsyncEx(
200     void* Grabber,
201     void* UserData
202 );
203
219 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_SetSaveImageComplete(
220     void* Grabber,
221     pfnCameraGrabberSaveImageComplete Callback,
222     void* Context
223 );
224
238 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_SetFrameListener(
239     void* Grabber,
240     pfnCameraGrabberFrameListener Listener,
241     void* Context
242 );
243
257 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_SetRawCallback(
258     void* Grabber,
259     pfnCameraGrabberFrameCallback Callback,
260     void* Context
261 );
262
276 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_SetRGBCallback(
277     void* Grabber,
278     pfnCameraGrabberFrameCallback Callback,
279     void* Context
280 );
281
293 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_GetCameraHandle(
294     void* Grabber,
295     CameraHandle *hCamera
296 );
297
309 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_GetStat(
310     void* Grabber,
311     tSdkGrabberStat *stat
312 );
313
325 MVSDK_API CameraSdkStatus __stdcall CameraGrabber_GetCameraDevInfo(
326     void* Grabber,
327     tSdkCameraDevInfo *DevInfo
328 );
```

```
329  
330  
331  
332  
333 #endif // _MV_CAMERA_GRABBER_H_
```

# MVSDK API

Include >

## CameraImage.h

```
1 #ifndef _MV_CAMERA_IMAGE_H_
2 #define _MV_CAMERA_IMAGE_H_
3
4 #include "CameraDefine.h"
5 #include "CameraStatus.h"
6
7
23 MVSDK_API CameraSdkStatus __stdcall CameraImage_Create(
24     void** Image,
25     BYTE *pFrameBuffer,
26     tSdkFrameHead* pFrameHead,
27     BOOL bCopy
28 );
29
39 MVSDK_API CameraSdkStatus __stdcall CameraImage_CreateEmpty(
40     void** Image
41 );
42
52 MVSDK_API CameraSdkStatus __stdcall CameraImage_Destroy(
53     void* Image
54 );
55
69 MVSDK_API CameraSdkStatus __stdcall CameraImage_GetData(
70     void* Image,
71     BYTE** DataBuffer,
72     tSdkFrameHead** Head
73 );
74
86 MVSDK_API CameraSdkStatus __stdcall CameraImage_GetUserData(
87     void* Image,
88     void** UserData
89 );
90
102 MVSDK_API CameraSdkStatus __stdcall CameraImage_SetUserData(
103     void* Image,
104     void* UserData
105 );
106
118 MVSDK_API CameraSdkStatus __stdcall CameraImage_IsEmpty(
```

```
119     void* Image,
120     BOOL* IsEmpty
121 );
122
136 MVSDK_API CameraSdkStatus __stdcall CameraImage_Draw(
137     void* Image,
138     HWND hWnd,
139     int Algorithm
140 );
141
155 MVSDK_API CameraSdkStatus __stdcall CameraImage_DrawFit(
156     void* Image,
157     HWND hWnd,
158     int Algorithm
159 );
160
182 MVSDK_API CameraSdkStatus __stdcall CameraImage_DrawToDC(
183     void* Image,
184     HDC hDC,
185     int Algorithm,
186     int xDst,
187     int yDst,
188     int cxDst,
189     int cyDst
190 );
191
213 MVSDK_API CameraSdkStatus __stdcall CameraImage_DrawToDCFit(
214     void* Image,
215     HDC hDC,
216     int Algorithm,
217     int xDst,
218     int yDst,
219     int cxDst,
220     int cyDst
221 );
222
246 MVSDK_API CameraSdkStatus __stdcall CameraImage_BitBlt(
247     void* Image,
248     HWND hWnd,
249     int xDst,
250     int yDst,
251     int cxDst,
252     int cyDst,
253     int xSrc,
254     int ySrc
255 );
```

```
256
280 MVSDK_API CameraSdkStatus __stdcall CameraImage_BitBltToDC(
281     void* Image,
282     HDC hDC,
283     int xDst,
284     int yDst,
285     int cxDst,
286     int cyDst,
287     int xSrc,
288     int ySrc
289 );
290
302 MVSDK_API CameraSdkStatus __stdcall CameraImage_SaveAsBmp(
303     void* Image,
304     char const* FileName
305 );
306
320 MVSDK_API CameraSdkStatus __stdcall CameraImage_SaveAsJpeg(
321     void* Image,
322     char const* FileName,
323     BYTE Quality
324 );
325
337 MVSDK_API CameraSdkStatus __stdcall CameraImage_SaveAsPng(
338     void* Image,
339     char const* FileName
340 );
341
355 MVSDK_API CameraSdkStatus __stdcall CameraImage_SaveAsRaw(
356     void* Image,
357     char const* FileName,
358     int Format
359 );
360
372 MVSDK_API CameraSdkStatus __stdcall CameraImage_IPicture(
373     void* Image,
374     IPicture** NewPic
375 );
376
377
378
379
380 #endif // _MV_CAMERA_IMAGE_H_
```

# MVSDK API

Include >

## CameraStatus.h

```
1 #ifndef __CAMERA_STATUS_DEF__
2 #define __CAMERA_STATUS_DEF__
3
4
5 typedef int CameraSdkStatus;
6
7
8
9
10 /*常用的宏*/
11 #define SDK_SUCCESS(_FUC_)
12     (_FUC_ == CAMERA_STATUS_SUCCESS)
13 #define SDK_UNSUCCESS(_FUC_)
14     (_FUC_ != CAMERA_STATUS_SUCCESS)
15 #define SDK_UNSUCCESS_RETURN(_FUC_,RET)
16     if((RET = _FUC_) != CAMERA_STATUS_SUCCESS) \
17         { \
18             return RET; \
19         }
20 #define SDK_UNSUCCESS_BREAK(_FUC_)
21     if(_FUC_ != CAMERA_STATUS_SUCCESS) \
22         { \
23             break; \
24         }
25
26
27
28 /* 常用错误 */
```

```
29 | #define CAMERA_STATUS_SUCCESS
30 | 0
31 | #define CAMERA_STATUS_FAILED
32 | -1
33 | #define CAMERA_STATUS_INTERNAL_ERROR
34 | -2
35 | #define CAMERA_STATUS_UNKNOW
36 | -3
37 | #define
38 |   CAMERA_STATUS_PARAMETER_OUT_OF_BOUND      -7
39 | #define CAMERA_STATUS_UNENABLED
40 | -8
41 | #define CAMERA_STATUS_USER_CANCEL
42 | -9
43 | #define CAMERA_STATUS_PATH_NOT_FOUND
44 | -10
45 | #define CAMERA_STATUS_SIZE_DISMATCH
46 | -11
47 | #define CAMERA_STATUS_TIME_OUT
48 | -12
49 | #define CAMERA_STATUS_IO_ERROR
50 | -13
51 | #define CAMERA_STATUS_COMM_ERROR
52 | -14
53 | #define CAMERA_STATUS_BUS_ERROR
54 | -15
55 | #define CAMERA_STATUS_NO_DEVICE_FOUND
56 | -16
```

```
47 | #define CAMERA_STATUS_NO_LOGIC_DEVICE_FOUND  
-17  
48 | #define CAMERA_STATUS_DEVICE_IS_OPENED  
-18  
49 | #define CAMERA_STATUS_DEVICE_IS_CLOSED  
-19  
50 | #define CAMERA_STATUS_DEVICE_VEDIO_CLOSED  
-20  
51 | #define CAMERA_STATUS_NO_MEMORY  
-21  
52 | #define CAMERA_STATUS_FILE_CREATE_FAILED  
-22  
53 | #define CAMERA_STATUS_FILE_INVALID  
-23  
54 | #define CAMERA_STATUS_WRITE_PROTECTED  
-24  
55 | #define CAMERA_STATUS_GRAB_FAILED  
-25  
56 | #define CAMERA_STATUS_LOST_DATA  
-26  
57 | #define CAMERA_STATUS_EOF_ERROR  
-27  
58 | #define CAMERA_STATUS_BUSY  
-28  
59 | #define CAMERA_STATUS_WAIT  
-29  
60 | #define CAMERA_STATUS_IN_PROCESS  
-30  
61 | #define CAMERA_STATUS_IIC_ERROR  
-31  
62 | #define CAMERA_STATUS_SPI_ERROR  
-32  
63 | #define CAMERA_STATUS_USB_CONTROL_ERROR  
-33  
64 | #define CAMERA_STATUS_USB_BULK_ERROR  
-34
```

```
65 | #define CAMERA_STATUS_SOCKET_INIT_ERROR  
-35  
66 | #define  
CAMERA_STATUS_GIGE_FILTER_INIT_ERROR -36  
67 | #define CAMERA_STATUS_NET_SEND_ERROR  
-37  
68 | #define CAMERA_STATUS_DEVICE_LOST  
-38  
69 | #define CAMERA_STATUS_DATA_RECV_LESS  
-39  
70 | #define CAMERA_STATUS_FUNCTION_LOAD_FAILED  
-40  
71 | #define CAMERA_STATUS_CRITICAL_FILE_LOST  
-41  
72 | #define CAMERA_STATUS_SENSOR_ID_DISMATCH  
-42  
73 | #define CAMERA_STATUS_OUT_OF_RANGE  
-43  
74 | #define CAMERA_STATUS_REGISTRY_ERROR  
-44  
75 | #define CAMERA_STATUS_ACCESS_DENY  
-45  
76 | #define CAMERA_STATUS_CAMERA_NEED_RESET  
-46  
77 | #define  
CAMERA_STATUS_ISP_MOUDLE_NOT_INITIALIZED -47  
78 | #define CAMERA_STATUS_ISP_DATA_CRC_ERROR  
-48  
79 | #define CAMERA_STATUS_MV_TEST_FAILED  
-49  
80 | #define CAMERA_STATUS_INTERNAL_ERR1  
-50  
81 | #define CAMERA_STATUS_U3V_NO_CONTROL_EP  
-51  
82 | #define CAMERA_STATUS_U3V_CONTROL_ERROR  
-52
```

```
83 | #define CAMERA_STATUS_INVALID_FRIENDLY_NAME  
-53  
84 | #define CAMERA_STATUS_FORMAT_ERROR  
-54  
85 |  
86 |  
87 |  
88 | //和AIA制定的标准相同  
89 | /*#define CAMERA_AIA_SUCCESS  
0x0000 */  
90 | #define CAMERA_AIA_PACKET_RESEND  
0x0100  
91 | #define CAMERA_AIA_NOT_IMPLEMENTED  
0x8001  
92 | #define CAMERA_AIA_INVALID_PARAMETER  
0x8002  
93 | #define CAMERA_AIA_INVALID_ADDRESS  
0x8003  
94 | #define CAMERA_AIA_WRITE_PROTECT  
0x8004  
95 | #define CAMERA_AIA_BAD_ALIGNMENT  
0x8005  
96 | #define CAMERA_AIA_ACCESS_DENIED  
0x8006  
97 | #define CAMERA_AIA_BUSY  
0x8007  
98 | #define CAMERA_AIA_DEPRECATED  
0x8008  
99 | #define CAMERA_AIA_PACKET_UNAVAILABLE  
0x800C  
100 | #define CAMERA_AIA_DATA_OVERRUN  
0x800D  
101 | #define CAMERA_AIA_INVALID_HEADER  
0x800E  
102 | #define CAMERA_AIA_PACKET_NOT_YET_AVAILABLE  
0x8010
```

```
103 | #define  
    CAMERA_AIA_PACKET_AND_PREV_REMOVED_FROM_MEMORY  
    Y      0x8011  
104 | #define  
    CAMERA_AIA_PACKET_REMOVED_FROM_MEMORY  
    0x8012  
105 | #define CAMERA_AIA_NO_REF_TIME  
    0x0813  
106 | #define  
    CAMERA_AIA_PACKET_TEMPORARILY_UNAVAILABLE  
    0x0814  
107 | #define CAMERA_AIA_OVERFLOW  
    0x0815  
108 | #define CAMERA_AIA_ACTION_LATE  
    0x0816  
109 | #define CAMERA_AIA_ERROR  
    0x8FFF  
110 |  
111 |  
113 |  
114 |  
115 | #endif
```

# MVSDK API

Include >

## CameraZoomTool.h

```
1 #ifndef _MV_CAMERA_ZOOM_TOOL_H_
2 #define _MV_CAMERA_ZOOM_TOOL_H_
3
4 #include "CameraDefine.h"
5 #include "CameraStatus.h"
6
7
17 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_Create(
18     void** ZoomTool
19 );
20
30 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_Destroy(
31     void* ZoomTool
32 );
33
47 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_SetViewOrg(
48     void*           ZoomTool,
49     float            x,
50     float            y
51 );
52
66 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_GetViewOrg(
67     void*           ZoomTool,
68     float*          x,
69     float*          y
70 );
71
85 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_SetViewCenter(
86     void*           ZoomTool,
87     float            x,
88     float            y
89 );
90
104 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_GetViewCenter(
105    void*           ZoomTool,
106    float*          x,
107    float*          y
108 );
109
123 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_SetViewSize(
124    void*           ZoomTool,
125    int              w,
126    int              h
127 );
128
142 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_GetViewSize(
```

```
143     void*           ZoomTool,
144     int*            w,
145     int*            h
146 );
147
161 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_SetViewScrollPos(
162     void*           ZoomTool,
163     float            xPos,
164     float            yPos
165 );
166
180 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_GetViewScrollPos(
181     void*           ZoomTool,
182     float*          xPos,
183     float*          yPos
184 );
185
199 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_GetViewScrollRange(
200     void*           ZoomTool,
201     float*          xRange,
202     float*          yRange
203 );
204
218 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_SetFrameSize(
219     void*           ZoomTool,
220     int              w,
221     int              h
222 );
223
237 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_GetFrameSize(
238     void*           ZoomTool,
239     int*            w,
240     int*            h
241 );
242
254 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_SetScale(
255     void*           ZoomTool,
256     float            ratio
257 );
258
270 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_GetScale(
271     void*           ZoomTool,
272     float*          ratio
273 );
274
288 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_SetScaleAnchor(
289     void*           ZoomTool,
290     float            xAnchor,
291     float            yAnchor
292 );
293
307 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_GetScaleAnchor(
308     void*           ZoomTool,
```

```
309     float*           xAnchor,
310     float*           yAnchor
311 );
312
338 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_GetDrawRect(
339     void*            ZoomTool,
340     int*             FrameX,
341     int*             FrameY,
342     int*             FrameW,
343     int*             FrameH,
344     int*             ViewX,
345     int*             ViewY,
346     int*             ViewW,
347     int*             ViewH
348 );
349
365 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_Transform(
366     void*            ZoomTool,
367     int               Type,
368     float*           PointX,
369     float*           PointY
370 );
371
395 MVSDK_API CameraSdkStatus __stdcall CameraZoomTool_Draw(
396     void*            ZoomTool,
397     int               Algorithm,
398     void*            pFrameBuffer,
399     tSdkFrameHead*   pFrameHead,
400     HWND              hWnd,
401     int               xDst,
402     int               yDst,
403     HBRUSH            hBackBrush
404 );
405
406
407
408 #endif // _MV_CAMERA_ZOOM_TOOL_H_
```

# MVSDK API

## 示例

这里列出了所有示例:

- [grab.cpp](#)
- [grab\\_ex.cpp](#)
- [roi.cpp](#)
- [soft\\_trigger.cpp](#)
- [win\\_basic.cpp](#)
- [win\\_callback.cpp](#)
- [win\\_grabber.cpp](#)

# MVSDK API

## grab.cpp

```
#include <stdio.h>
#include <windows.h>
#include "CameraApi.h"

int main(int argc, char* argv[])
{
    CameraSdkStatus status;

    // 调用CameraEnumerateDevice前，先设置CameraNums = 16， 表示最
    // 多只读取16个设备。
    // 如果需要枚举更多的设备，请更改CameraList数组的大小和CameraNums
    // 的值
    // Before calling CameraEnumerateDevice, set CameraNums =
    // 16 to read only 16 devices at most.
    // If you need to enumerate more devices, change the size
    // of the CameraList array and CameraNums
    tSdkCameraDevInfo CameraList[16];
    int CameraNums = 16;

    // 枚举设备，获得设备列表
    // Enumerate devices to get a list of devices
    status = CameraEnumerateDevice(CameraList, &CameraNums);
    if (status != CAMERA_STATUS_SUCCESS)
    {
        printf("No camera was found!");
        return -1;
    }

    // 该示例中，我们只初始化第一个相机。
    // (-1,-1)表示加载上次退出前保存的参数，如果是第一次使用该相机，则加
    // 载默认参数。
    // In this example, we only initialize the first camera.
    // (-1,-1) means to load the parameters saved before the
    // last exit. If it is the first time to use the camera, then
    // load the default parameters.
    int hCamera = 0;
    status = CameraInit(&CameraList[0], -1, -1, &hCamera);
    if (status != CAMERA_STATUS_SUCCESS)
    {
```

```
    printf("Failed to init the camera! Error code is %d", st
    return -1;
}

// 获得该相机的特性描述
// Get the camera's feature description
tSdkCameraCapbility CameraInfo;
CameraGetCapability(hCamera, &CameraInfo);

// 判断是黑白相机还是彩色相机
// Judging whether it is a mono camera or a color camera
BOOL bMonoCamera = CameraInfo.sIspCapacity.bMonoSensor;

// 黑白相机让ISP直接输出MONO数据，而不是扩展成R=G=B的24位灰度
// Mono cameras allow the ISP to directly output MONO data
// instead of the 24-bit grayscale expanded to R=G=B
if (bMonoCamera)
{
    CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_MONO8);
}

// 相机模式切换成连续采集
// Switch camera mode to continuous acquisition
CameraSetTriggerMode(hCamera, 0);

// 手动曝光，曝光时间30ms
// Manual exposure, exposure time 30ms
CameraSetAeState(hCamera, FALSE);
CameraSetExposureTime(hCamera, 30 * 1000);

// 让SDK内部取图线程开始工作
// Let the SDK internal grab thread start working
CameraPlay(hCamera);

// 计算RGB buffer所需的大小，这里直接按照相机的最大分辨率来分配
UINT FrameBufferSize =
    CameraInfo.sResolutionRange.iWidthMax *
    CameraInfo.sResolutionRange.iHeightMax * (bMonoCamera ? 1
                                                : 3);

// 分配RGB buffer，用来存放ISP输出的图像
// 备注：从相机传输到PC端的是RAW数据，在PC端通过软件ISP转为RGB数据
// （如果是黑白相机就不需要转换格式，但是ISP还有其它处理，所以也需要分配
// 这个buffer）
```

```
// allocate RGB buffer to store the image output by ISP
// Note: RAW data is transferred from the camera to the PC
// and converted to RGB data via the software ISP on the PC
// (if it is a Mono camera, there is no need to convert the
// format, but the ISP has other processing, so we also need
// to allocate this buffer)
BYTE* pFrameBuffer = (BYTE
*)CameraAlignMalloc(FrameBufferSize, 16);

// 从相机取一帧图片
// Take a frame from the camera
tSdkFrameHead FrameHead;
BYTE* pRawData;
status = CameraGetImageBuffer(hCamera, &FrameHead,
&pRawData, 2000);
if (status == CAMERA_STATUS_SUCCESS)
{
    CameraImageProcess(hCamera, pRawData, pFrameBuffer, &Fra
    CameraReleaseImageBuffer(hCamera, pRawData);

    // 此时图片已经存储在pFrameBuffer中, 对于彩色相机
    // pFrameBuffer=RGB数据, 黑白相机pFrameBuffer=8位灰度数据
    // 该示例中我们只是把图片保存到硬盘文件中
    // At this point, the picture has been stored in
    pFrameBuffer. For pFrameBuffer=RGB data in color camera,
    pFrameBuffer=8bit gray data in mono camera.
    // In this example we just saved the image to a hard
    disk file
    status = CameraSaveImage(hCamera, "z:\\grab.bmp",
    pFrameBuffer, &FrameHead, bMonoCamera ? FILE_BMP_8BIT :
    FILE_BMP, 100);
    if (status == CAMERA_STATUS_SUCCESS)
        printf("Save image successfully. image_size =
    %dx%d\n", FrameHead.iWidth, FrameHead.iHeight);
    else
        printf("Save image failed. err=%d\n", status);
}

// 关闭相机
// close camera
CameraUnInit(hCamera);

// release RGB buffer
CameraAlignFree(pFrameBuffer);
```

```
    return 0;  
}
```

# MVSDK API

## grab\_ex.cpp

```
#include <stdio.h>
#include <windows.h>
#include "CameraApi.h"

int main(int argc, char* argv[])
{
    CameraSdkStatus status;

    // 枚举设备，获得相机个数
    // Enumerate devices and get the number of cameras
    int CameraNums = CameraEnumerateDeviceEx();
    if (CameraNums < 1)
    {
        printf("No camera was found!");
        return -1;
    }

    // 该示例中，我们只初始化第一个相机。
    // (-1,-1)表示加载上次退出前保存的参数，如果是第一次使用该相机，则加载默认参数。
    // In this example, we only initialize the first camera.
    // (-1,-1) means to load the parameters saved before the
    // last exit. If it is the first time to use the camera, then
    // load the default parameters.
    int hCamera = 0;
    status = CameraInitEx(0, -1, -1, &hCamera);
    if (status != CAMERA_STATUS_SUCCESS)
    {
        printf("Failed to init the camera! Error code is %d", status);
        return -1;
    }

    // 由于前面并未获取设备列表，可以从相机获取设备枚举信息
    // You can get device enumeration information from the
    // camera because you did not get the device list before
    tSdkCameraDevInfo DevInfo;
    CameraGetEnumInfo(hCamera, &DevInfo);

    printf("Camera Name: %s\n", DevInfo.acFriendlyName);
```

```
printf("Camera SN: %s\n", DevInfo.acSn);

// 获得该相机的特性描述
// Get the camera's feature description
tSdkCameraCapbility CameraInfo;
CameraGetCapability(hCamera, &CameraInfo);

// 判断是黑白相机还是彩色相机
// Judging whether it is a mono camera or a color camera
BOOL bMonoCamera = CameraInfo.sIspCapacity.bMonoSensor;

// 黑白相机让ISP直接输出MONO数据，而不是扩展成R=G=B的24位灰度
// Mono cameras allow the ISP to directly output MONO data
// instead of the 24-bit grayscale expanded to R=G=B
if (bMonoCamera)
{
    CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_MONO8);
}

// 相机模式切换成连续采集
// Switch camera mode to continuous acquisition
CameraSetTriggerMode(hCamera, 0);

// 手动曝光，曝光时间30ms
// Manual exposure, exposure time 30ms
CameraSetAeState(hCamera, FALSE);
CameraSetExposureTime(hCamera, 30 * 1000);

// 让SDK内部取图线程开始工作
// Let the SDK internal grab thread start working
CameraPlay(hCamera);

// 从相机取一帧图片
// Take a frame from the camera
tSdkFrameHead FrameHead = { 0 };
BYTE* pFrameBuffer = CameraGetImageBufferEx(hCamera,
    &FrameHead.iWidth, &FrameHead.iHeight, 2000);
if (pFrameBuffer != NULL)
{
    FrameHead.uiMediaType = bMonoCamera ?
        CAMERA_MEDIA_TYPE_MONO8 : CAMERA_MEDIA_TYPE_BGR8;
    FrameHead.uBytes = FrameHead.iWidth * FrameHead.iHeight
        * (bMonoCamera ? 1 : 3);
}
```

```
// 此时图片已经存储在pFrameBuffer中，对于彩色相机  
pFrameBuffer=RGB数据，黑白相机pFrameBuffer=8位灰度数据  
// 该示例中我们只是把图片保存到硬盘文件中  
// At this point, the picture has been stored in  
pFrameBuffer. For pFrameBuffer=RGB data in color camera,  
pFrameBuffer=8bit gray data in mono camera.  
// In this example we just saved the image to a hard  
disk file  
status = CameraSaveImage(hCamera, "z:\\grab.bmp",  
pFrameBuffer, &FrameHead, bMonoCamera ? FILE_BMP_8BIT :  
FILE_BMP, 100);  
if (status == CAMERA_STATUS_SUCCESS)  
    printf("Save image successfully. image_size =  
%dx%d\\n", FrameHead.iWidth, FrameHead.iHeight);  
else  
    printf("Save image failed. err=%d\\n", status);  
}  
  
// 关闭相机  
// close camera  
CameraUnInit(hCamera);  
  
return 0;  
}
```

# MVSDK API

## roi.cpp

```
#include <stdio.h>
#include <windows.h>
#include "CameraApi.h"

// offsetx, offsey, width, height: 偏移宽高都选取为16的倍数兼容性
// 最好 (不同的相机对这个的要求不同, 有的只需要2的倍数, 有的可能需要16的
// 倍数)
// offsetx, offsey, width, height: offset width and height
// are all chosen to be 16 times the best compatibility
// (different cameras have different requirements for this,
// some need only a multiple of 2, some may require a
// multiple of 16)
int SetCameraResolution(int hCamera, int offsetx, int offsey,
    int width, int height)
{
    tSdkImageResolution sRoiResolution = { 0 };

    // 设置成0xff表示自定义分辨率, 设置成0到N表示选择预设分辨率
    // Set to 0xff for custom resolution, set to 0 to N for
    // select preset resolution
    sRoiResolution.iIndex = 0xff;

    // iWidthFOV表示相机的视场宽度, iWidth表示相机实际输出宽度
    // 大部分情况下iWidthFOV=iWidth。有些特殊的分辨率模式如BIN2X2:
    // iWidthFOV=2*iWidth, 表示视场是实际输出宽度的2倍
    // iWidthFOV represents the camera's field of view width,
    // iWidth represents the camera's actual output width
    // In most cases iWidthFOV=iWidth. Some special resolution
    // modes such as BIN2X2:iWidthFOV=2*iWidth indicate that the
    // field of view is twice the actual output width
    sRoiResolution.iWidth = width;
    sRoiResolution.iWidthFOV = width;

    // 高度, 参考上面宽度的说明
    // height, refer to the description of the width above
    sRoiResolution.iHeight = height;
    sRoiResolution.iHeightFOV = height;

    // 视场偏移
```

```
// Field of view offset
sRoiResolution.iHOffsetFOV = offsetx;
sRoiResolution.iVOffsetFOV = offsety;

// ISP软件缩放宽高, 都为0则表示不缩放
// ISP software zoom width and height, all 0 means not zoom
sRoiResolution.iWidthZoomSw = 0;
sRoiResolution.iHeightZoomSw = 0;

// BIN SKIP 模式设置 (需要相机硬件支持)
// BIN SKIP mode setting (requires camera hardware support)
sRoiResolution.uBinAverageMode = 0;
sRoiResolution.uBinSumMode = 0;
sRoiResolution.uResampleMask = 0;
sRoiResolution.uSkipMode = 0;

return CameraSetImageResolution(hCamera, &sRoiResolution);
}

int main(int argc, char* argv[])
{
    CameraSdkStatus status;

    // 调用CameraEnumerateDevice前, 先设置CameraNums = 16, 表示最多只读取16个设备。
    // 如果需要枚举更多的设备, 请更改CameraList数组的大小和CameraNums的值
    // Before calling CameraEnumerateDevice, set CameraNums = 16 to read only 16 devices at most.
    // If you need to enumerate more devices, change the size of the CameraList array and CameraNums
    tSdkCameraDevInfo CameraList[16];
    int CameraNums = 16;

    // 枚举设备, 获得设备列表
    // Enumerate devices to get a list of devices
    status = CameraEnumerateDevice(CameraList, &CameraNums);
    if (status != CAMERA_STATUS_SUCCESS)
    {
        printf("No camera was found!");
        return -1;
    }

    // 该示例中, 我们只初始化第一个相机。
```

```
// (-1,-1)表示加载上次退出前保存的参数，如果是第一次使用该相机，则加载默认参数。  
// In this example, we only initialize the first camera.  
// (-1,-1) means to load the parameters saved before the  
last exit. If it is the first time to use the camera, then  
load the default parameters.  
int hCamera = 0;  
status = CameraInit(&CameraList[0], -1, -1, &hCamera);  
if (status != CAMERA_STATUS_SUCCESS)  
{  
    printf("Failed to init the camera! Error code is %d", st  
    return -1;  
}  
  
// 获得该相机的特性描述  
// Get the camera's feature description  
tSdkCameraCapbility CameraInfo;  
CameraGetCapability(hCamera, &CameraInfo);  
  
// 判断是黑白相机还是彩色相机  
// Judging whether it is a mono camera or a color camera  
BOOL bMonoCamera = CameraInfo.sIspCapacity.bMonoSensor;  
  
// 黑白相机让ISP直接输出MONO数据，而不是扩展成R=G=B的24位灰度  
// Mono cameras allow the ISP to directly output MONO data  
instead of the 24-bit grayscale expanded to R=G=B  
if (bMonoCamera)  
{  
    CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_MONO8);  
}  
  
// 相机模式切换成连续采集  
// Switch camera mode to continuous acquisition  
CameraSetTriggerMode(hCamera, 0);  
  
// 手动曝光，曝光时间30ms  
// Manual exposure, exposure time 30ms  
CameraSetAeState(hCamera, FALSE);  
CameraSetExposureTime(hCamera, 30 * 1000);  
  
// 设置ROI分辨率  
// Set ROI resolution  
SetCameraResolution(hCamera, 0, 0, 320, 240);
```

```
// 让SDK内部取图线程开始工作
// Let the SDK internal grab thread start working
CameraPlay(hCamera);

// 计算RGB buffer所需的大小，这里直接按照相机的最大分辨率来分配
UINT FrameBufferSize =
    CameraInfo.sResolutionRange.iWidthMax *
    CameraInfo.sResolutionRange.iHeightMax * (bMonoCamera ? 1
                                                : 3);

// 分配RGB buffer，用来存放ISP输出的图像
// 备注：从相机传输到PC端的是RAW数据，在PC端通过软件ISP转为RGB数据
// （如果是黑白相机就不需要转换格式，但是ISP还有其它处理，所以也需要分配
// 这个buffer）
// allocate RGB buffer to store the image output by ISP
// Note: RAW data is transferred from the camera to the PC
// and converted to RGB data via the software ISP on the PC
// (if it is a Mono camera, there is no need to convert the
// format, but the ISP has other processing, so we also need
// to allocate this buffer)
BYTE* pFrameBuffer = (BYTE
    *)CameraAlignMalloc(FrameBufferSize, 16);

// 从相机取一帧图片
// Take a frame from the camera
tSdkFrameHead FrameHead;
BYTE* pRawData;
status = CameraGetImageBuffer(hCamera, &FrameHead,
    &pRawData, 2000);
if (status == CAMERA_STATUS_SUCCESS)
{
    CameraImageProcess(hCamera, pRawData, pFrameBuffer, &Fra
    CameraReleaseImageBuffer(hCamera, pRawData);

    // 此时图片已经存储在pFrameBuffer中，对于彩色相机
    // pFrameBuffer=RGB数据，黑白相机pFrameBuffer=8位灰度数据
    // 该示例中我们只是把图片保存到硬盘文件中
    // At this point, the picture has been stored in
    pFrameBuffer. For pFrameBuffer=RGB data in color camera,
    pFrameBuffer=8bit gray data in mono camera.
    // In this example we just saved the image to a hard
    disk file
    status = CameraSaveImage(hCamera, "z:\\grab.bmp",
    pFrameBuffer, &FrameHead, bMonoCamera ? FILE_BMP_8BIT :
```

```
FILE_BMP, 100);
    if (status == CAMERA_STATUS_SUCCESS)
        printf("Save image successfully. image_size =
%dx%d\n", FrameHead.iWidth, FrameHead.iHeight);
    else
        printf("Save image failed. err=%d\n", status);
}

// 关闭相机
// close camera
CameraUnInit(hCamera);

// release RGB buffer
CameraAlignFree(pFrameBuffer);

return 0;
}
```

# MVSDK API

## soft\_trigger.cpp

```
#include <stdio.h>
#include <windows.h>
#include "CameraApi.h"

int main(int argc, char* argv[])
{
    CameraSdkStatus status;

    // 调用CameraEnumerateDevice前，先设置CameraNums = 16， 表示最
    // 多只读取16个设备。
    // 如果需要枚举更多的设备，请更改CameraList数组的大小和CameraNums
    // 的值
    // Before calling CameraEnumerateDevice, set CameraNums =
    // 16 to read only 16 devices at most.
    // If you need to enumerate more devices, change the size
    // of the CameraList array and CameraNums
    tSdkCameraDevInfo CameraList[16];
    int CameraNums = 16;

    // 枚举设备，获得设备列表
    // Enumerate devices to get a list of devices
    status = CameraEnumerateDevice(CameraList, &CameraNums);
    if (status != CAMERA_STATUS_SUCCESS)
    {
        printf("No camera was found!");
        return -1;
    }

    // 该示例中，我们只初始化第一个相机。
    // (-1,-1)表示加载上次退出前保存的参数，如果是第一次使用该相机，则加
    // 载默认参数。
    // In this example, we only initialize the first camera.
    // (-1,-1) means to load the parameters saved before the
    // last exit. If it is the first time to use the camera, then
    // load the default parameters.
    int hCamera = 0;
    status = CameraInit(&CameraList[0], -1, -1, &hCamera);
    if (status != CAMERA_STATUS_SUCCESS)
    {
```

```
    printf("Failed to init the camera! Error code is %d", st
    return -1;
}

// 获得该相机的特性描述
// Get the camera's feature description
tSdkCameraCapbility CameraInfo;
CameraGetCapability(hCamera, &CameraInfo);

// 判断是黑白相机还是彩色相机
// Judging whether it is a mono camera or a color camera
BOOL bMonoCamera = CameraInfo.sIspCapacity.bMonoSensor;

// 黑白相机让ISP直接输出MONO数据，而不是扩展成R=G=B的24位灰度
// Mono cameras allow the ISP to directly output MONO data
// instead of the 24-bit grayscale expanded to R=G=B
if (bMonoCamera)
{
    CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_MONO8);
}

// 设置相机为软触发模式，并且把一次触发的帧数固定为1
// Set the camera to soft trigger mode and fix the number
// of frames triggered at one time to 1
CameraSetTriggerMode(hCamera, 1);
CameraSetTriggerCount(hCamera, 1);

// 手动曝光，曝光时间30ms
// Manual exposure, exposure time 30ms
CameraSetAeState(hCamera, FALSE);
CameraSetExposureTime(hCamera, 30 * 1000);

// 让SDK内部取图线程开始工作
// Let the SDK internal grab thread start working
CameraPlay(hCamera);

// 计算RGB buffer所需的大小，这里直接按照相机的最大分辨率来分配
UINT FrameBufferSize =
    CameraInfo.sResolutionRange.iWidthMax *
    CameraInfo.sResolutionRange.iHeightMax * (bMonoCamera ? 1
    : 3);

// 分配RGB buffer，用来存放ISP输出的图像
```

```
// 备注：从相机传输到PC端的是RAW数据，在PC端通过软件ISP转为RGB数据  
// (如果是黑白相机就不需要转换格式，但是ISP还有其它处理，所以也需要分配  
// 这个buffer)  
// allocate RGB buffer to store the image output by ISP  
// Note: RAW data is transferred from the camera to the PC  
and converted to RGB data via the software ISP on the PC  
(if it is a Mono camera, there is no need to convert the  
format, but the ISP has other processing, so we also need  
to allocate this buffer)  
BYTE* pFrameBuffer = (BYTE  
*)CameraAlignMalloc(FrameBufferSize, 16);  
  
// 从相机取一帧图片  
// Take a frame from the camera  
tSdkFrameHead FrameHead;  
BYTE* pRawData;  
  
// 由于相机目前处于软触发模式，需要软件发送指令通知相机拍照（为了避免  
意外取到相机缓存中的旧图片，在给触发指令前先清空了缓存）  
// Since the camera is currently in soft trigger mode,  
software is required to send a command to inform the  
camera to take pictures (to avoid accidentally fetching  
old pictures in the camera cache, the cache is cleared  
before the trigger command)  
CameraClearBuffer(hCamera);  
CameraSoftTrigger(hCamera);  
  
status = CameraGetImageBuffer(hCamera, &FrameHead,  
&pRawData, 2000);  
if (status == CAMERA_STATUS_SUCCESS)  
{  
    CameraImageProcess(hCamera, pRawData, pFrameBuffer, &Fra  
    CameraReleaseImageBuffer(hCamera, pRawData);  
  
    // 此时图片已经存储在pFrameBuffer中，对于彩色相机  
    pFrameBuffer=RGB数据，黑白相机pFrameBuffer=8位灰度数据  
    // 该示例中我们只是把图片保存到硬盘文件中  
    // At this point, the picture has been stored in  
    pFrameBuffer. For pFrameBuffer=RGB data in color camera,  
    pFrameBuffer=8bit gray data in mono camera.  
    // In this example we just saved the image to a hard  
disk file  
    status = CameraSaveImage(hCamera, "z:\\grab.bmp",  
pFrameBuffer, &FrameHead, bMonoCamera ? FILE_BMP_8BIT :
```

```
FILE_BMP, 100);
    if (status == CAMERA_STATUS_SUCCESS)
        printf("Save image successfully. image_size =
%dx%d\n", FrameHead.iWidth, FrameHead.iHeight);
    else
        printf("Save image failed. err=%d\n", status);
}

// 关闭相机
// close camera
CameraUnInit(hCamera);

// release RGB buffer
CameraAlignFree(pFrameBuffer);

return 0;
}
```

# MVSDK API

## win\_basic.cpp

```
#include <windows.h>
#include <process.h>
#include "CameraApi.h"

struct Camera
{
    int hCamera;
    BYTE* pFrameBuffer;
    HANDLE hGrabThread;
    BOOL bExit;
};

// 图像抓取线程，主动调用SDK接口函数获取图像
// The image capture thread actively calls the SDK interface
// function to get the image
UINT WINAPI GrabThread(LPVOID lpParam)
{
    Camera* pCam = (Camera*)lpParam;

    while (!pCam->bExit)
    {
        int status;
        tSdkFrameHead FrameHead;
        BYTE* pRaw;

        status = CameraGetImageBufferPriority(pCam->hCamera,
&FrameHead, &pRaw, 2000, CAMERA_GET_IMAGE_PRIORITY_NEWEST);
        if (status == CAMERA_STATUS_SUCCESS)
        {
            CameraImageProcess(pCam->hCamera, pRaw, pCam-
>pFrameBuffer, &FrameHead);
            CameraReleaseImageBuffer(pCam->hCamera, pRaw);

            // 可以在此处处理抓取到的图片数据
            // The captured image data can be processed here

            // 调用SDK封装好的显示接口来显示图像
            // Call the SDK packaged display interface to
display the image
        }
    }
}
```

```
        CameraImageOverlay(pCam->hCamera, pCam-
>pFrameBuffer, &FrameHead);
        CameraDisplayRGB24(pCam->hCamera, pCam-
>pFrameBuffer, &FrameHead);
    }
}

return 0;
}

HWND CreateAppWindow(HINSTANCE hInstance);
int InitCamera(Camera* pCam, HWND hWnd);
void CloseCamera(Camera* pCam);

int WINAPI WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,
                    int nShowCmd)
{
    HWND hAppWnd = CreateAppWindow(hInstance);
    ShowWindow(hAppWnd, nShowCmd);
    UpdateWindow(hAppWnd);

    // 打开相机
    // Open the camera
    Camera cam;
    if (InitCamera(&cam, hAppWnd) != CAMERA_STATUS_SUCCESS)
        return 0;

    // 消息循环
    // message loop
    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    CloseCamera(&cam);
    return msg.wParam;
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam,
                        LPARAM lParam)
```

```
{  
    HDC         hdc;  
    PAINTSTRUCT ps;  
    RECT        rect;  
  
    switch (message)  
    {  
        case WM_DESTROY:  
            PostQuitMessage(0);  
            return 0;  
    }  
  
    return DefWindowProc(hwnd, message, wParam, lParam);  
}  
  
HWND CreateAppWindow(HINSTANCE hInstance)  
{  
    static TCHAR lpszAppName[] = TEXT("HelloWin");  
    HWND      hwnd;  
    MSG       msg;  
    WNDCLASS wc;  
  
    wc.style      = CS_HREDRAW | CS_VREDRAW;  
    wc.lpfnWndProc = WndProc;  
    wc.cbClsExtra = 0;  
    wc.cbWndExtra = 0;  
    wc.hInstance   = hInstance;  
    wc.hIcon       = LoadIcon(NULL, IDI_APPLICATION);  
    wc.hCursor     = LoadCursor(NULL, IDC_ARROW);  
    wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);  
    wc.lpszMenuName = NULL;  
    wc.lpszClassName = lpszAppName;  
  
    // 注册窗口类  
    // Registration window class  
    if (!RegisterClass(&wc))  
    {  
        MessageBox(NULL, TEXT("This program requires Windows NT!  
                           lpszAppName, MB_ICONERROR);  
        return NULL;  
    }  
  
    // 创建应用程序主窗口  
    // Create application main window
```

```
hwnd = CreateWindow(lpszAppName,
    TEXT("The Hello Program"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    NULL,
    NULL,
    hInstance,
    NULL);
return hwnd;
}

int InitCamera(Camera* pCam, HWND hWnd)
{
    CameraSdkStatus status;

    // 调用CameraEnumerateDevice前，先设置CameraNums = 16，表示最多只读取16个设备。
    // 如果需要枚举更多的设备，请更改CameraList数组的大小和CameraNums的值
    // Before calling CameraEnumerateDevice, set CameraNums = 16 to read only 16 devices at most.
    // If you need to enumerate more devices, change the size of the CameraList array and CameraNums
    tSdkCameraDevInfo CameraList[16];
    int CameraNums = 16;

    // 枚举设备，获得设备列表
    // Enumerate devices to get a list of devices
    status = CameraEnumerateDevice(CameraList, &CameraNums);
    if (status != CAMERA_STATUS_SUCCESS)
    {
        MessageBox(NULL, TEXT("No camera was found!!"),
            NULL, MB_ICONERROR);
        return status;
    }

    // 该示例中，我们只初始化第一个相机。
    // (-1,-1)表示加载上次退出前保存的参数，如果是第一次使用该相机，则加载默认参数。
    // In this example, we only initialize the first camera.
```

```
// (-1,-1) means to load the parameters saved before the
// last exit. If it is the first time to use the camera, then
// load the default parameters.
int hCamera = 0;
status = CameraInit(&CameraList[0], -1, -1, &hCamera);
if (status != CAMERA_STATUS_SUCCESS)
{
    MessageBox(NULL, TEXT("Failed to init the camera!!"),
               NULL, MB_ICONERROR);
    return status;
}

// 获得该相机的特性描述
// Get the camera's feature description
tSdkCameraCapbility CameraInfo;
CameraGetCapability(hCamera, &CameraInfo);

// 判断是黑白相机还是彩色相机
// Judging whether it is a mono camera or a color camera
BOOL bMonoCamera = CameraInfo.sIspCapacity.bMonoSensor;

// 黑白相机让ISP直接输出MONO数据，而不是扩展成R=G=B的24位灰度
// Mono cameras allow the ISP to directly output MONO data
// instead of the 24-bit grayscale expanded to R=G=B
if (bMonoCamera)
{
    CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_MONO8);
}

// 使用SDK封装好的显示接口
// Use SDK to display camera images.
CameraDisplayInit(hCamera, hWnd);
// Set display window size
RECT rect;
GetClientRect(hWnd, &rect);
CameraSetDisplaySize(hCamera, rect.right - rect.left,
                     rect.bottom - rect.top);

// 打开相机设置页面
// Open camera settings page
CameraCreateSettingPage(hCamera, hWnd,
                        CameraList[0].acFriendlyName, NULL, NULL, 0);
CameraShowSettingPage(hCamera, TRUE);
```

```
// 相机模式切换成连续采集
// Switch camera mode to continuous acquisition
CameraSetTriggerMode(hCamera, 0);

// 手动曝光，曝光时间30ms
// Manual exposure, exposure time 30ms
CameraSetAeState(hCamera, FALSE);
CameraSetExposureTime(hCamera, 30 * 1000);

// 让SDK内部取图线程开始工作
// Let the SDK internal grab thread start working
CameraPlay(hCamera);

// 计算RGB buffer所需的大小，这里直接按照相机的最大分辨率来分配
UINT FrameBufferSize =
    CameraInfo.sResolutionRange.iWidthMax *
    CameraInfo.sResolutionRange.iHeightMax * (bMonoCamera ? 1
    : 3);

// 分配RGB buffer，用来存放ISP输出的图像
// 备注：从相机传输到PC端的是RAW数据，在PC端通过软件ISP转为RGB数据
// （如果是黑白相机就不需要转换格式，但是ISP还有其它处理，所以也需要分配
// 这个buffer）
// allocate RGB buffer to store the image output by ISP
// Note: RAW data is transferred from the camera to the PC
// and converted to RGB data via the software ISP on the PC
// (if it is a Mono camera, there is no need to convert the
// format, but the ISP has other processing, so we also need
// to allocate this buffer)
BYTE* pFrameBuffer = (BYTE
*)CameraAlignMalloc(FrameBufferSize, 16);

// 准备启动抓图线程
// Prepare to start the capture thread
memset(pCam, 0, sizeof(*pCam) );
pCam->hCamera = hCamera;
pCam->pFrameBuffer = pFrameBuffer;
pCam->bExit = FALSE;

// 启动抓图线程
// Start the capture thread
pCam->hGrabThread = (HANDLE)_beginthreadex(NULL, 0,
GrabThread, pCam, 0, NULL);
```

```
    return CAMERA_STATUS_SUCCESS;
}

void CloseCamera(Camera* pCam)
{
    // 结束抓图线程
    // End the capture thread
    pCam->bExit = TRUE;
    WaitForSingleObject(pCam->hGrabThread, INFINITE);
    CloseHandle(pCam->hGrabThread);

    // 关闭相机
    // close camera
    CameraUnInit(pCam->hCamera);

    // release RGB buffer
    CameraAlignFree(pCam->pFrameBuffer);
}
```

# MVSDK API

## win\_callback.cpp

```
#include <windows.h>
#include <process.h>
#include "CameraApi.h"

struct Camera
{
    int      hCamera;
    BYTE*   pFrameBuffer;
};

// 取图回调
// Grab callback
void WINAPI GrabImageCallback(CameraHandle hCamera, BYTE
    *pRaw, tSdkFrameHead* pFrameHead, PVOID pContext)
{
    Camera* pCam = (Camera*)pContext;

    CameraImageProcess(pCam->hCamera, pRaw, pCam-
        >pFrameBuffer, pFrameHead);

    // 可以在此处处理抓取到的图片数据
    // The captured image data can be processed here

    // 调用SDK封装好的显示接口来显示图像
    // Call the SDK packaged display interface to display the im
    CameraImageOverlay(pCam->hCamera, pCam->pFrameBuffer, pFrame
    CameraDisplayRGB24(pCam->hCamera, pCam->pFrameBuffer, pFrame
}

HWND CreateAppWindow(HINSTANCE hInstance);
int InitCamera(Camera* pCam, HWND hWnd);
void CloseCamera(Camera* pCam);

int WINAPI WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,
                    int nShowCmd)
{
    HWND hAppWnd = CreateAppWindow(hInstance);
```

```
ShowWindow(hAppWnd, nShowCmd);
UpdateWindow(hAppWnd);

// 打开相机
// Open the camera
Camera cam;
if (InitCamera(&cam, hAppWnd) != CAMERA_STATUS_SUCCESS)
    return 0;

// 消息循环
// message loop
MSG msg;
while (GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}

CloseCamera(&cam);
return msg.wParam;
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    HDC         hdc;
    PAINTSTRUCT ps;
    RECT        rect;

    switch (message)
    {
    case WM_DESTROY:
        PostQuitMessage(0);
        return 0;
    }

    return DefWindowProc(hwnd, message, wParam, lParam);
}

HWND CreateAppWindow(HINSTANCE hInstance)
{
    static TCHAR lpszAppName[] = TEXT("HelloWin");
    HWND      hwnd;
    MSG      msg;
```

```
WNDCLASS    wc;

wc.style      = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc = WndProc;
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;
wc.hInstance   = hInstance;
wc.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
wc.hCursor     = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
wc.lpszMenuName = NULL;
wc.lpszClassName = lpszAppName;

// 注册窗口类
// Registration window class
if (!RegisterClass(&wc))
{
    MessageBox(NULL, TEXT("This program requires Windows NT!
                           lpszAppName, MB_ICONERROR);
    return NULL;
}

// 创建应用程序主窗口
// Create application main window
hwnd = CreateWindow(lpszAppName,
                    TEXT("The Hello Program"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    NULL,
                    NULL,
                    hInstance,
                    NULL);
return hwnd;
}

int InitCamera(Camera* pCam, HWND hWnd)
{
    CameraSdkStatus status;

    // 调用CameraEnumerateDevice前，先设置CameraNums = 16， 表示最
    // 多只读取16个设备。
```

```
// 如果需要枚举更多的设备, 请更改CameraList数组的大小和CameraNums  
// 的值  
// Before calling CameraEnumerateDevice, set CameraNums =  
16 to read only 16 devices at most.  
// If you need to enumerate more devices, change the size  
of the CameraList array and CameraNums  
tSdkCameraDeviceInfo CameraList[16];  
int CameraNums = 16;  
  
// 枚举设备, 获得设备列表  
// Enumerate devices to get a list of devices  
status = CameraEnumerateDevice(CameraList, &CameraNums);  
if (status != CAMERA_STATUS_SUCCESS)  
{  
    MessageBox(NULL, TEXT("No camera was found!!"),  
              NULL, MB_ICONERROR);  
    return status;  
}  
  
// 该示例中, 我们只初始化第一个相机。  
// (-1,-1)表示加载上次退出前保存的参数, 如果是第一次使用该相机, 则加  
载默认参数。  
// In this example, we only initialize the first camera.  
// (-1,-1) means to load the parameters saved before the  
last exit. If it is the first time to use the camera, then  
load the default parameters.  
int hCamera = 0;  
status = CameraInit(&CameraList[0], -1, -1, &hCamera);  
if (status != CAMERA_STATUS_SUCCESS)  
{  
    MessageBox(NULL, TEXT("Failed to init the camera!!"),  
              NULL, MB_ICONERROR);  
    return status;  
}  
  
// 获得该相机的特性描述  
// Get the camera's feature description  
tSdkCameraCapbility CameraInfo;  
CameraGetCapability(hCamera, &CameraInfo);  
  
// 判断是黑白相机还是彩色相机  
// Judging whether it is a mono camera or a color camera  
BOOL bMonoCamera = CameraInfo.sIspCapacity.bMonoSensor;
```

```
// 黑白相机让ISP直接输出MONO数据，而不是扩展成R=G=B的24位灰度
// Mono cameras allow the ISP to directly output MONO data
// instead of the 24-bit grayscale expanded to R=G=B
if (bMonoCamera)
{
    CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_MONO8);
}

// 使用SDK封装好的显示接口
// Use SDK to display camera images.
CameraDisplayInit(hCamera, hWnd);
// Set display window size
RECT rect;
GetClientRect(hWnd, &rect);
CameraSetDisplaySize(hCamera, rect.right - rect.left,
rect.bottom - rect.top);

// 打开相机设置页面
// Open camera settings page
CameraCreateSettingPage(hCamera, hWnd,
CameraList[0].acFriendlyName, NULL, NULL, 0);
CameraShowSettingPage(hCamera, TRUE);

// 相机模式切换成连续采集
// Switch camera mode to continuous acquisition
CameraSetTriggerMode(hCamera, 0);

// 手动曝光，曝光时间30ms
// Manual exposure, exposure time 30ms
CameraSetAeState(hCamera, FALSE);
CameraSetExposureTime(hCamera, 30 * 1000);

// 让SDK内部取图线程开始工作
// Let the SDK internal grab thread start working
CameraPlay(hCamera);

// 计算RGB buffer所需的大小，这里直接按照相机的最大分辨率来分配
UINT FrameBufferSize =
    CameraInfo.sResolutionRange.iWidthMax *
    CameraInfo.sResolutionRange.iHeightMax * (bMonoCamera ? 1
    : 3);

// 分配RGB buffer，用来存放ISP输出的图像
```

```
// 备注：从相机传输到PC端的是RAW数据，在PC端通过软件ISP转为RGB数据  
// (如果是黑白相机就不需要转换格式，但是ISP还有其它处理，所以也需要分配  
// 这个buffer)  
// allocate RGB buffer to store the image output by ISP  
// Note: RAW data is transferred from the camera to the PC  
and converted to RGB data via the software ISP on the PC  
(if it is a Mono camera, there is no need to convert the  
format, but the ISP has other processing, so we also need  
to allocate this buffer)  
BYTE* pFrameBuffer = (BYTE  
*)CameraAlignMalloc(FrameBufferSize, 16);  
  
// 准备启动回调  
// Prepare to start callback  
memset(pCam, 0, sizeof(*pCam) );  
pCam->hCamera = hCamera;  
pCam->pFrameBuffer = pFrameBuffer;  
  
// 启动回调  
// Start callback  
CameraSetCallbackFunction(hCamera, GrabImageCallback,  
pCam, NULL);  
  
return CAMERA_STATUS_SUCCESS;  
}  
  
void CloseCamera(Camera* pCam)  
{  
    // 关闭相机  
    // close camera  
    CameraUnInit(pCam->hCamera);  
  
    // release RGB buffer  
    CameraAlignFree(pCam->pFrameBuffer);  
}
```

# MVSDK API

## win\_grabber.cpp

```
#include <windows.h>
#include <process.h>
#include "CameraApi.h"
#include "CameraGrabber.h"

void WINAPI FrameRGBCallback(void* Grabber, BYTE
    *pFrameBuffer, tSdkFrameHead* pFrameHead, void* Context)
{
    // 数据处理回调
    // Data processing callback

    // 由于黑白相机在相机打开后设置了ISP输出灰度图像
    // 因此此处pFrameBuffer=8位灰度数据
    // 否则会和彩色相机一样输出BGR24数据
    // As the mono camera set the ISP output grayscale image
    // after the camera is open
    // So here pFrameBuffer=8-bit grayscale data
    // otherwise it will output BGR24 data just like a color cam

    // 彩色相机ISP默认会输出BGR24图像
    // pFrameBuffer=BGR24数据
    // Color Camera ISP outputs BGR24 image by default
    // pFrameBuffer=BGR24 data

    // 注意：在回调中无需再调用CameraDisplayRGB24来显示图像，只要调用
    // CameraGrabber_SetHWnd给Grabber指定了窗口便会自动被绘制到窗口上
    // Note: CameraDisplayRGB24 does not need to call in
    // callback. Just call CameraGrabber_SetHWnd to specify the
    // window to Grabber will be automatically drawn to the window.
}

HWND CreateAppWindow(HINSTANCE hInstance);
void* InitGrabber(HWND hWnd);
void CloseGrabber(void* Grabber);

int WINAPI WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpCmdLine,
    int nShowCmd)
```

```
{  
    HWND hAppWnd = CreateAppWindow(hInstance);  
    ShowWindow(hAppWnd, nShowCmd);  
    UpdateWindow(hAppWnd);  
  
    // 打开采集器  
    // Open the Grabber  
    void *Grabber = InitGrabber(hAppWnd);  
    if (Grabber == NULL)  
        return 0;  
  
    // 消息循环  
    // message loop  
    MSG msg;  
    while (GetMessage(&msg, NULL, 0, 0))  
    {  
        TranslateMessage(&msg);  
        DispatchMessage(&msg);  
    }  
  
    CloseGrabber(Grabber);  
    return msg.wParam;  
}  
  
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam,  
    LPARAM lParam)  
{  
    HDC hdc;  
    PAINTSTRUCT ps;  
    RECT rect;  
  
    switch (message)  
    {  
        case WM_DESTROY:  
            PostQuitMessage(0);  
            return 0;  
    }  
  
    return DefWindowProc(hwnd, message, wParam, lParam);  
}  
  
HWND CreateAppWindow(HINSTANCE hInstance)  
{  
    static TCHAR lpszAppName[] = TEXT("HelloWin");
```

```
HWND      hwnd;
MSG       msg;
WNDCLASS  wc;

wc.style      = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc = WndProc;
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;
wc.hInstance   = hInstance;
wc.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
wc.hCursor     = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
wc.lpszMenuName = NULL;
wc.lpszClassName = lpszAppName;

// 注册窗口类
// Registration window class
if (!RegisterClass(&wc))
{
    MessageBox(NULL, TEXT("This program requires Windows NT!
                           lpszAppName, MB_ICONERROR);
    return NULL;
}

// 创建应用程序主窗口
// Create application main window
hwnd = CreateWindow(lpszAppName,
                    TEXT("The Hello Program"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    NULL,
                    NULL,
                    hInstance,
                    NULL);
return hwnd;
}

void* InitGrabber(HWND hWnd)
{
    CameraSdkStatus status;
```

```
// 调用CameraEnumerateDevice前，先设置CameraNums = 16，表示最
多只读取16个设备。
// 如果需要枚举更多的设备，请更改CameraList数组的大小和CameraNums
// 的值
// Before calling CameraEnumerateDevice, set CameraNums =
16 to read only 16 devices at most.
// If you need to enumerate more devices, change the size
of the CameraList array and CameraNums
tSdkCameraDeviceInfo CameraList[16];
int CameraNums = 16;

// 枚举设备，获得设备列表
// Enumerate devices to get a list of devices
status = CameraEnumerateDevice(CameraList, &CameraNums);
if (status != CAMERA_STATUS_SUCCESS)
{
    MessageBox(NULL, TEXT("No camera was found!!"),
               NULL, MB_ICONERROR);
    return NULL;
}

// 该示例中，我们只初始化第一个相机。
// In this example, we only initialize the first camera.
void* Grabber = NULL;
status = CameraGrabber_Create(&Grabber, &CameraList[0]);
if (status != CAMERA_STATUS_SUCCESS)
{
    MessageBox(NULL, TEXT("Failed to init the camera!!"),
               NULL, MB_ICONERROR);
    return NULL;
}

// Grabber底层会使用CameraInit来打开相机，因此可以获取原始句柄来调
用其他SDK函数
// Grabber will use CameraInit to open the camera, so you
can get the original handle to call other SDK functions
int hCamera = 0;
CameraGrabber_GetCameraHandle(Grabber, &hCamera);

// 获得该相机的特性描述
// Get the camera's feature description
tSdkCameraCapbility CameraInfo;
CameraGetCapability(hCamera, &CameraInfo);
```

```
// 判断是黑白相机还是彩色相机
// Judging whether it is a mono camera or a color camera
BOOL bMonoCamera = CameraInfo.sIspCapacity.bMonoSensor;

// 黑白相机让ISP直接输出MONO数据，而不是扩展成R=G=B的24位灰度
// Mono cameras allow the ISP to directly output MONO data
// instead of the 24-bit grayscale expanded to R=G=B
if (bMonoCamera)
{
    CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_MONO8);
}

// 打开相机设置页面
// Open camera settings page
CameraCreateSettingPage(hCamera, hWnd,
    CameraList[0].acFriendlyName, NULL, NULL, 0);
CameraShowSettingPage(hCamera, TRUE);

// 相机模式切换成连续采集
// Switch camera mode to continuous acquisition
CameraSetTriggerMode(hCamera, 0);

// 手动曝光，曝光时间30ms
// Manual exposure, exposure time 30ms
CameraSetAeState(hCamera, FALSE);
CameraSetExposureTime(hCamera, 30 * 1000);

// 设置取图回调
// Set the callback
CameraGrabber_SetRGBCallback(Grabber, FrameRGBCallback, NULL);

// 设置显示窗口
// Set the display window
CameraGrabber_SetHWnd(Grabber, hWnd);

// 让Grabber内部取图线程开始工作
// Let the Grabber internal grab thread start working
CameraGrabber_StartLive(Grabber);

return Grabber;
}

void CloseGrabber(void* Grabber)
{
```

```
// 关闭采集器（内部会调用CameraUninit关闭相机）
// Close Grabber(internal CameraUninit will be called to
// close the camera)
CameraGrabber_Destroy(Grabber);
}
```

# MVSDK API

Include >

## Include 目录参考