工业相机开发手册

2.4 版本

目　录

简介

本文档提供给使用我公司工业相机进行二次开发的用户使用，针对系统开发中使用到的动态链接库函数进行详细说明，并提供了快速程序设计指导，以便在短时间内能够将相机集成到用户的系统中。

声明

1 概述

1.1 文件结构

在二次开发中，需要直接使用到的库文件位于安装目录的 SDK 文件夹中，分为 32 位和 64 位两个开发包。其中 64 位的 SDK 开发文件，位于 SDK/X64 文件夹中，相对于 32 位的 SDK 文件，64 位的 SDK 文件名均以_X64 结尾。

SDK 目录下的 MVCAMSDK.DLL 档是相机的 SDK 动态链接库，对外提供相机所有的接口函数，VC/C++、VB、VB.net、Delphi、C#的例程都引用了该库文件。同时，为了方便引用，MVCAMSDK.DLL 在安装时会被复制到系统 System32 目录下(WIN64 系统时复制在 syswow64 目录下)，您在二次开发时，您可以直接使用该文件名来访问而无需关心 MVCAMSDK.DLL 文件所在的路径，系统会自动在 System32 目录下找到该文件。

如果您需要发布自己的安装包，请参考第 7 章中的方法。

开发例程位于安装目录的以下子目录中：

- Demo/VC++。基于 VC++开发的例程，提供了 VC6 和 VS2010 的工程文件。
- Demo/VB6。基于 VB6 开发的例程。
- Demo/VB.net。基于 VB.net 开发的例程。
- Demo/Delphi6。基于 Delphi6 开发的例程。
- Demo/C#。基于 C#(VS2010)开发的例程。
- Demo/VC++/OpenCV。基于 VS2010 的 OpenCV 例子。

- Demo/LabView。基于 ActiveX 和 DLL 接口调用的开发例程。

其中 VC++提供了 Basic、BasicEx、Advanced、OCX、MultiCamera、MultiExposure 多例程，VB6、Delphi6、C#只提供了 Basic 一种例程。Basic 和 Advanced 的两个例程都囊括了绝大部分的相机功能操作，您可以根据开发需要来选择不同的例程来了解如何使用我们的 SDK。有关 Basic 和 Advanced 的例程各自的特点和适用范围，请参考第二章内容。

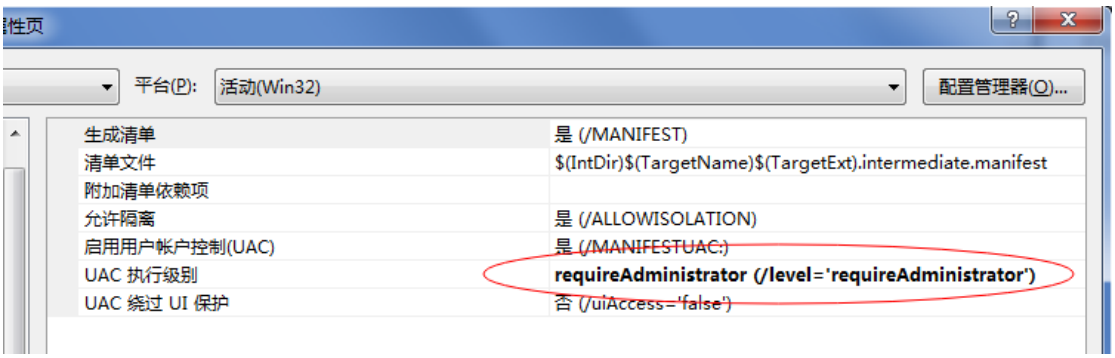
1.2 开发环境

SDK 是标准的 C 语言接口动态链接库，可以被 C、C++、C#、VB、Labview、Delphi 等开发工具载入。目前，我们提供了基于 VC++ 6.0、VS2010、VB6.0、VB.net、Delphi6、C#、Labview(8.6 9.0 2010 2011 2012)、Halcon(9、10、11、12)、OpenCV(2.4)的例程。

对于使用 VC6 以上版本的开发工具如 VS2003--VS2013 等,则可以直接打开 VC++6.0 的 DEMO 工程中 DSP 或者 DSW 文件，然后按照提示进行工程转换即可。

1.3 注意事项(常见问题)

- 由于 SDK 需要访问注册表，因此在 WIN7 和 WIN8 系统下，如果以非管理员用户登录，则需要以管理员权限运行相机的 DEMO 程序才能正常访问相机，否则相机初始化会报-13 之类的错误。常用的做法是以管理员方式登录系统，或者在您开发的程序里，申请管理员权限。对于 MFC 工程，在工程设置->链接器->清单文件中，进行如下设置：



，如果您使用 DELPHI、C#、VB6 之类的开发语言，也可以通过类似的方法获得管理员权限。

- 不同版本的安装包之间的切换。不同版本的 SDK，我们保持了 API 接口的兼容性，但是相机的内核驱动在不同软件版本之间，可能需要手动切换，如果安装新的版本后，相机初始化提示失败，则需要在设备管理器手动更新一次相机内核驱动。

1.4 工业相机常用词语解释

1.4.1 线阵相机和面阵相机

线阵相机，是采用线阵图像传感器的相机。线阵图像传感器以 CCD 为主，一行的数据可以到几 K 甚至几十 K，但是高度只有几个像素，行频很高，可以到每秒几万行，适合做非常高精度、宽画幅的扫描。

面阵相机，是采用面阵图像传感器的相机，CMOS 和 CCD 都有面阵相机，面阵相机的分辨率一行的宽度相比线阵相机会小很多，但是画面是整画幅的感应，一次成像的像素高度会比线阵相机大很多。同时程序开发上也相对简单，一次就可以获得一整副图像，不用进行每行数据的拼接。

目前，绝大部分的应用，还是使用面阵相机为主。价格方面，线阵相机价格昂贵，往往是面阵相机的几倍到几十倍。

1.4.2 滚动快门和全域快门

滚动快门和全域快门主要是针对面阵相机而言的。

滚动快门的相机，在感光时，是逐行进行的，从第一行开始，一直滚动到最后一行感光，每一行的曝光开始时间点不一样，边曝光边输出图像数据；全域快门的相机，在感光时，是整个面阵同时开始、同时结束，结束后整个一帧数据一次性读出。

由于滚动快门的原理上的限制，滚动快门的相机不适合拍摄高速运动的画面，相比全域快门的相机，滚动快门相机在拍摄运动画面时，每行图像会产生一定的位移偏差，最终造成图像扭曲变形。在这里要解释一下，这种扭曲变形，并非拖影，很多人误把这种现象理解为拖影，拖影是由于拍摄的物体运动速度太快，而相机的曝光时间又设置的太长造成，拖影会造成图像模糊，而滚动快门造成的扭曲变形，但是每行图像的清晰度并不受影响。

1.4.3 硬件触发(外触发)

正常模式下，相机开始工作后，就是一直连续的采集图像，采集完这一帧后，就马上开始下一帧的采集，如此循环。

在一些工业应用上，并不需要相机一直连续采集图像，而是等待特定的事件发生后，才希望相机采集一帧图像，并得到处理后的结果，这种情况下，就需要使用硬件（外）触发模式，支持这种工作方式的工业相机，都会在相机上留有专用的触发接头，一般是 4 芯到 12 芯不等的航空接头。进入触发模式后，相机会

等待有效的信号，信号的有效性可以通过软件来设置，例如高、低电平方式或者上、下边沿跳变方式，如果一直没有有效信号，则相机不会输出任何图像数据。当您发现程序抓图超时后，请检查一下是否在接口上设置了相机为触发工作模式。

1.4.4 光学尺寸(靶面)

光学尺寸是指相机感光区域的大小。常见的尺寸有 1/4"、1/3"、1/2.5"、1/2.3"、1/2"、2/3"等，需要根据这个尺寸来选择相应的镜头，以达到匹配的效果。

1.4.5 帧存和缓存

带缓存功能的工业相机，是指该相机具有一定的数据缓存图像数据能力，但是不具备缓存下整个一帧图像数据的能力，因此，当传输带宽不够、或者传输线路不够可靠和稳定时，就可能造成缓存溢出，而导致图像帧无法重建，造成丢帧或者完全不出图的现象。

带帧存功能的工业相机，是指该相机具有在相机内部保存下完整图像的帧的能力，因此，当传输带宽不够、或者传输线路不够可靠和稳定时，带帧缓存功能的相机仍然可以断点续传，在 PC 端可以重建图像帧。

工业相机是一定会带缓存功能的，但是不一定带帧存功能。一个典型的例子来说明帧存相机的好处：当同时连接 16 台相机用外触发模式拍照时，收到触发信号后，这 16 台相机开始同时曝光，并将图像先保存在相机内，随后 PC 端可以按照任意顺序去读出这 16 台相机的图像，无需担心读取时间长短和带宽是否足够的问题。而如果没有帧存功能，这 16 台相机的图像就可能全部丢失。但是

不带帧存的相机也有其优点，性价比高，结构简单，适合一个计算机接 1 到 2 个相机。

2 快速开发指南

2.1 相机操作流程概述

为了您快速而准确的进行开发，请您耐心阅读完本章节内容。

我们建议您按照如下的流程操作相机(其中有一些步骤是可选的，已经标明)：

一、载入 SDK 的动态链接库档 MVCAMSDK.DLL。您可以使用动态或者静态加载两种方式。

- 如果您使用 C/C++进行开发 ,在工程引用 CameraApi.h 头文件(位于安装目录的 SDK/DEMO/VC++/include 中)和 MVCAMSDK.lib 库文件(位于安装目录的 SDK 文件夹中),然后就可以直接在工程中引用 SDK 中的接口函数了,但是 MVCAMSDK.DLL 必须和您的应用程序放在同一目录下或者是系统的 system32 目录下，放置于其他目录时，必须设定系统的环境变量(PATH)。
- 如果您使用 VB 进行开发，可以通过我们 VB6 例程中类似的方法进行加载 ,直接定义 SDK 的接口函数并指明其引用的 DLL 文件即可。SDK\Demo\VB6\Module\CameraApi.bas 模块囊括了所有的 SDK 接口，将 MVCAMSDK.DLL 中导出的每一个函数接口都映像成了 VB 可以调用的函数。
- 如果您使用 Delphi 进行开发 ,可以通过我们 Delphi6 例程中类似的方法进行加载，直接定义 SDK 的接口函数并指明其引用的 DLL 文

件即可。SDK\Demo\Delphi6\Units\CameraApi.pas 单元囊括了所有的 SDK 接口，将 MVCAMSDK.DLL 中导出的每一个函数接口都映像成了 Delphi 可以调用的函数。

- 如果您使用 C#进行开发，可以通过我们 C#例程中类似的方法进行加载，C#加载 MVCAMSDK.DLL 的过程和 VB、Delphi 类似，为了方便使用，我们的 C# DEMO 中提供了 2 个工程，一个是 MVSDK，这个工程专门用来定义开发包数据结构和加载开发包的 API 函数；另一个是 Basic 工程，该工程中则实现了一个集预览、抓拍、相机设置为一体的例程。Basic 工程通过调用 MVSDK 工程中的代码来间接访问 MVCAMSDK.DLL。

二、初始化 SDK。完成了 SDK 的加载以后，在使用其他接口之前，请调用 CameraSdkInit 函数进行初始化。

三、枚举设备。调用 CameraEnumerateDevice 函数枚举设备，获得当前连接到 PC 上的相机的设备列表，列表中包括设备名(可自己修改)、版本号、唯一序列号、相机型号等信息。

四、初始化设备。根据第三步中获得的相机设备枚举信息，调用 CameraInit 函数初始化指定的相机，得到相机的句柄。如果需要同时打开多个相机，则利用多个相机的设备名多次调用 CameraInit 来获得多个相机的句柄，后续对相机的操作，都需要此时获得的相机句柄来指定操作的相机对象。

五、让 SDK 进入图像采集模式。调用 CameraPlay 函数，让相机进入工作模式，并且 SDK 开始接收来自相机的图像。

六、抓取图像。SDK 提供了两种获得图像数据的方式，这两种方式的效率都是一样的，底层都使用了零拷贝机制来提高效率，您可以根据您的开发习惯来选择其中一种。

- 主动调用 CameraGetImageBuffer 来获取一帧图像数据。该函数会获得一个 SDK 内部用来接收图像数据的缓冲区地址，以及帧头信息。同时，该函数可以设定超时时间，在指定的时间内没有获取到图像(线程会被挂起)，则返回超时。
- 在第三步中，初始化相机以后，调用 CameraSetCallbackFunction 来设定一个回调函数。这种方式是被动的，只有在 SDK 内部接收到有效的图像数据帧后，才会调用您设定的回调函数来传递收到的图像数据帧。

注意：您也可以同时使用以上两种方式来获取图像，但是不能在 CameraSetCallbackFunction 设定的回调函数中来调用 CameraGetImageBuffer 再次获取图像，这样会产生死锁问题。

七、处理图像。上一步获取的图像帧，是相机输出的原始格式，我公司大多数型号相机，原始输出都是 Bayer 格式或者 YUV 格式，这些格式信息会被自动添加到帧头信息中，调用 CameraImageProcess 来获得图像处理的效果，如颜色增益调整、白平衡校正、饱和度、LUT 变换、降噪等等，并将 YUV 或者 Bayer 格式的原始数据转换为 24BIT 的位图格式(RGB888)。

八、迭加十字线、自动曝光参考窗口、白平衡参考窗口等附加内容(如果您的开发中，不需要迭加信息，**这一步可以略过**)。调用

CameraImageOverlay 函数，被设置为可见状态的十字线、自动曝光参考窗口、白平衡参考窗口，将被迭加到输入的图像上。

CameraImageOverlay 的输入必须是位图格式，我们建议您在调用 CameraImageProcess 得到位图格式后，再调用 CameraImageOverlay 函数。。

九、将图像保存或者显示图像(如果您的开发中，对图像进行别的处理，而不需要将图像保存成文件或者进行显示，**这一步可以略过**)。

- 如果您需要保存图像到文件中，在第六步、第七步或者第八步后，调用 CameraSaveImage 函数来保存图片，SDK 支持 PNG、BMP、JPG 和原始数据四种方式。如果要保存原始数据，您应该在第六步以后就调用 CameraSaveImage 函数 ;如果保存成 BMP、PNG、JPG 格式，您应该在第七步后调用 CameraSaveImage 函数；如果保存成 BMP、PNG、JPG 格式的同时，您需要迭加十字线和自动曝光、白平衡参考窗口的位置，那么您可以在第八步后调用 CameraSaveImage 函数。
- 如果您需要显示图像，有以下两个方式：
    a. 自己根据开发环境来实现图像显示，例如利用 OpenGL、DirectDraw、Windows GDI 等方式来实现图像的显示。
    b. 利用我们的 SDK 里封装好的显示接口来显示图像。在第四步中初始化相机后，调用 CameraDisplayInit 函数来初始化显示接口，该函数需要传入显示控件的句柄(HWND 类型)，只适合 VC/C++、VS、VB、VB.NET、Delphi、C#等使用

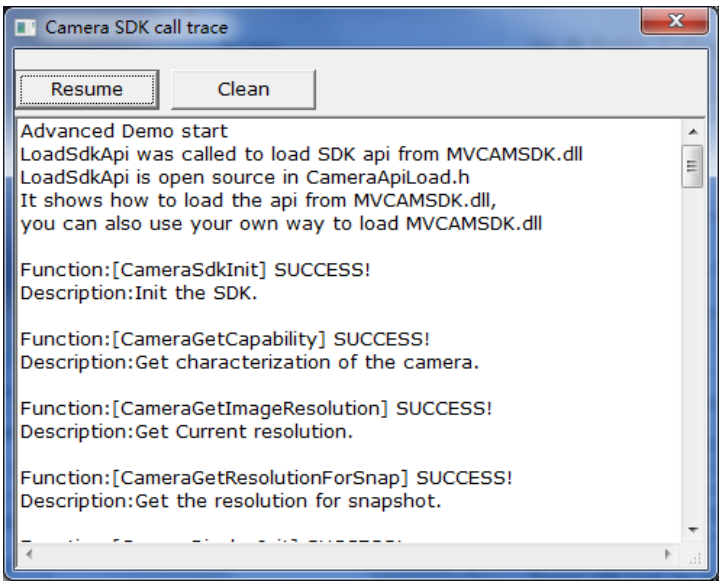
Windows GDI 开发接口的工具，是有一定的局限性，但如果您满足使用的条件，我们还是建议您使用我们封装好的显示接口。

十、在退出程序前关闭相机(反初始化，非常重要，如果直接关闭程序而不反初始化相机，程序有可能会报内存错误)。在关闭相机时，调用 CameraUnInit 函数。

2.2 开发例程

为了您更快速的开发和使用我们的 SDK，我们为您准备了多个例程。每个例程的特点不一样，适合开发的人群也不一样，您可以根据自己需求，选择不同的例程来入手。本文文件仅针对 VC 的例程进行说明，其余开发语言的例程说明和 VC 例程类似。

开发例程位于安装目录的 Demo 文件夹下,根据开发语言不通 ,分为 VC++、VB6、C#、VB.NET、Delphi6 等目录。目前，提供了 Basic、Advanced、MultiCamera、OCX、UserDataTest、TriggerAndStrobe、ImageFormat&Saving 等若干个例程，这些例程以 VC++为主，**C#/Delphi/VB6/VB.NET 等开发语言均可参考 VC++例子调用的 SDK 接口，实现所有功能，每种开发语言提供的 SDK 是相同的**。同时，为了您更加直观的看到例程中对相机 SDK 界面调用的情况,我们特意制作了一个相机 SDK 调用的跟踪信息面板，在例程运行后，该面板会自动显示，记录了整个过程中 SDK 接口函数的调用信息。如下图所示：



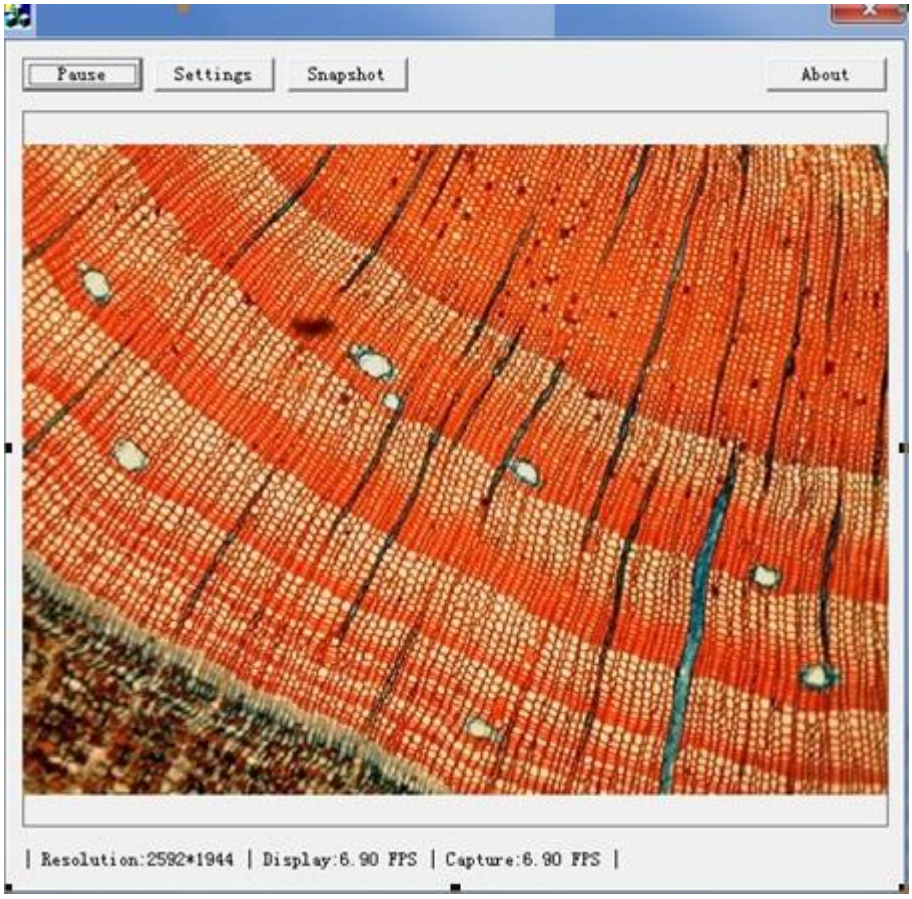
这些例程特点如下：

- Basic 例程使用了 SDK 的接口来创建相机的配置窗口 ,类似 DirectShow 接口中的设备属性页。使用这种方式，您基本上不需要自己开发 UI 接口，甚至也不用仔细去了解相机相关的 SDK 接口的使用方法，可以极大的节约您的开发和调试时间，同时，我们的相机参数是以二进制的文件形势保存的，在开发机器上在通过软件界面调试好参数后，将参数保存成文件，可一并发布到目标机器上，省去复杂的编程初始化参数的工作，强烈建议您使用这种方式进行开发。
- Advanced 例程涵盖了绝大部分的 SDK 接口 ,演示了这些接口函数的使用条件和方法 ,利用 VC++6.0 制作了相机属性的配置接口。如果 Basic 例程不能完全满足您的设计需求，您可以参考 Advanced 例程来进行。
- MultiCamera 例程演示了如何进行多相机开发 ,可连接 1 到 4 台同型号或者不同型号甚至是不同接口（USB2.0 USB3.0 GIGE）的相机进行同时预览。

- OCX 例程演示了如何通过我们提供的 ActiveX 控件进行相机开发 ,同样，用 OXC 控件方式，也是支持多相机的。
- UserDataTest 例程演示了如何向相机中写入或者读取自定义数据，这些数据是保存在相机内部的，掉电后仍然有效。可用作 ID、应用程序数据、捆绑数据的读写。
- TriggerAndStrobe 例程，演示了外触发和闪光灯接口的开发。
- ImageFormat&Saving 例程，演示了如何设置 SDK 抓图的数据格式和如何进行图片保存的操作。
- SnapshotOnPreview 例程，演示了如何在小分辨率下高速预览时，抓拍到一帧全分辨率的图像。
- ROI 例程，演示了当预设分辨率不能满足项目要求时，如何自定义相机图像分辨率。对于某些型号，相机可支持多区域 ROI 同时传输的功能，由相机硬件完成裁剪，软件端 SDK 自动拼接成整幅画面。去掉不需要的传输区域后，可有效提高帧率，例如 500 万像素 USB2.0 相机使用该技术后，在 500 万像素无压缩时，帧率可以提升到 15 帧。
- GPIO 例程，演示如何操作相机上自带的输入、输出 IO。

2.2.1 Basic 例程

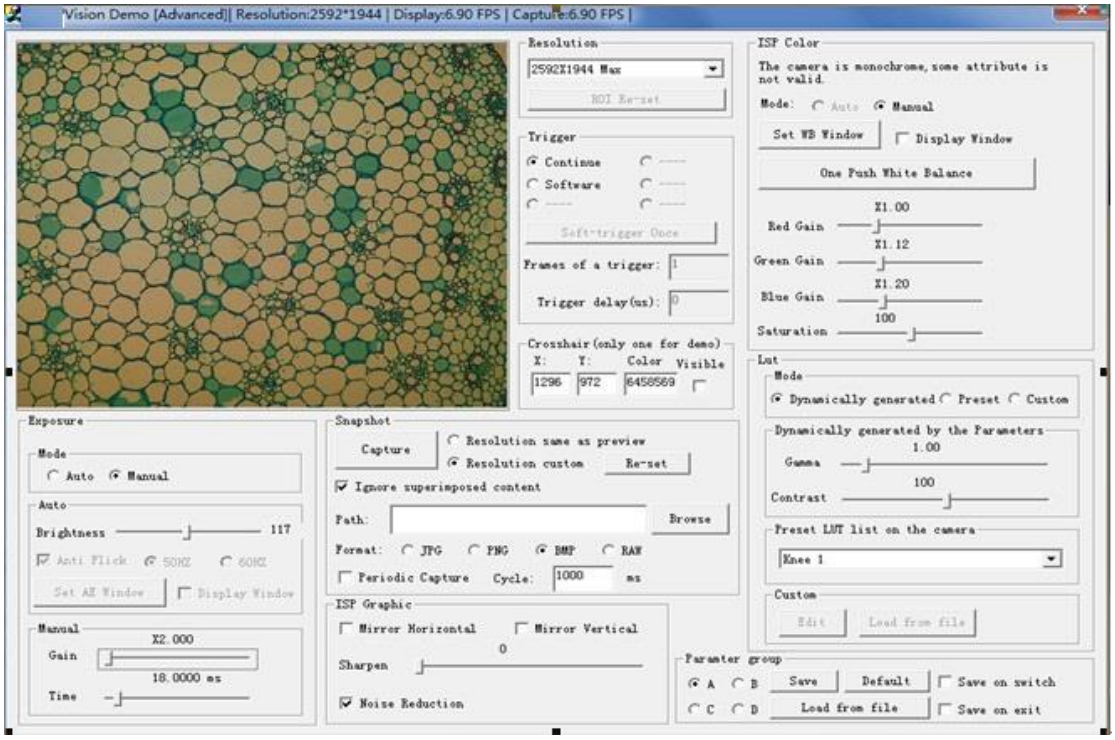
该例程位于安装目录的 Demo/VC++/Basic 文件夹中。例程运行后，接口如下图所示：



- Pause 按钮。用于暂停相机工作。
- Settings 按钮。点击后显示相机的配置窗口。配置窗口由 SDK 生成，并非常式中制作的。
- Snapshot 按钮。点击后使用 SDK 的抓拍接口来获得一张图片，并保存到本地档。
- About 按钮。点击后弹出该例程的说明信息。
- 最下方状态栏。从左到右依次显示了当前预览的分辨率、显示帧率、捕获帧率。

2.2.2 Advanced 例程

该例程位于安装目录的 Demo/VC++/Advanced 文件夹中。例程运行后，接口如下图所示：



- 接口的标题栏上，显示了分辨率、显示帧率和采集帧率。
- Exposure 模块，集合曝光相关的控制功能。
- Resolution 模块，集合分辨率相关的控制功能，包括自定义分辨率(ROI)。
- Trigger 模块，集合了触发模式的控制功能。Continue 表示连续输出模式，Software 表示软件触发模式。
- Crosshair 模块，集合十字线控制功能。
- Snapshot 模块，集合了抓拍模式的控制功能，同时演示了如何进行定时自动拍照。

- ISP Graphic 模块，集合图形相关的控制功能，包括镜像、锐化、降噪等功能。
- ISP Color 模块，集合了图像颜色相关的控制功能，包括白平衡、RGB 三信道数字增益、饱和度的控制灯。
- Lut 模块，集合查遍变换功的控制功能，SDK 支持三种查表变换方式，分别是通过调节参数生成 LUT 表、使用相机预设的 LUT 表和自定义的 LUT 表。

2.2.3 MultiCamera 例程(多相机同时使用)

该例程位于安装目录的 Demo/VC++/MultiCamera 文件夹中，演示了如何运用 SDK 开发出一台计算机同时接 1 到 4 个相机并显示 4 个相机的预览图像的方法。如果您需要接更多的相机，请参考本例子中的方法进行扩展，可以支持到 32 台以上的相机同时使用。

2.2.4 OCX 例程(ActiveX)

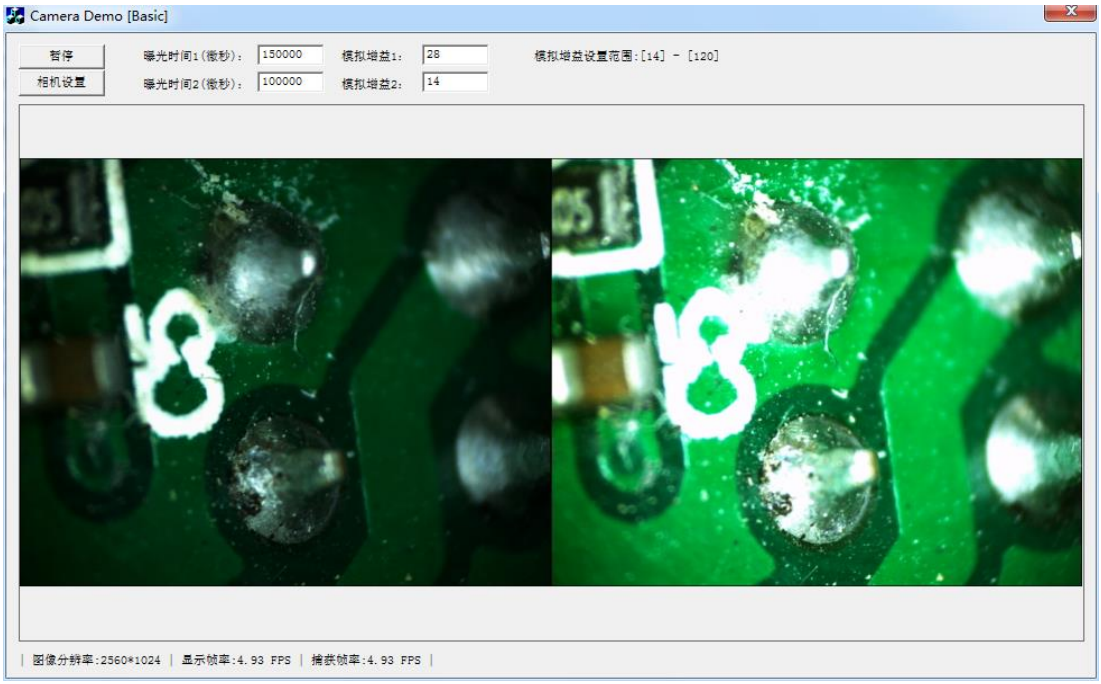
该例程位于安装目录的 Demo/VC++/OCX 文件夹中，演示了如何运用相机的 OCX 控件进行应用程序开发，OCX 方式也可以同时使用多相机，例子中可以接 1 到 2 台相机进行演示，需要接更多的相机同时使用，请参考例子中的方法进行扩展。

2.2.5 MultiExposure 例程(一个相机两个曝光信道预览，宽动态效果)

该例程位于安装目录的 Demo/VC++/MultiExposure 文件夹中，演示了如何运用一个相机实现 2 个通道下，不同曝光值和亮度增益的连续预览方式。运用同样的方式，您可以将其扩展为 4 个或者更多的预览通道，来获得不同曝光效果

的图像。下图是使用 USB2.0 接口 130 万彩色相机拍摄，这种双曝光模式下，每个信道最高可以到 10 帧。

典型应用：拍金属、玻璃等反光物质时，可用此方法获得不同亮度的图像进行分析。此方法进行扩展后，可获得 2 个以上的曝光效果。



2.2.6 ImageFormat&Saving 例程(Gray、RGB24、RGB32 格式设置)

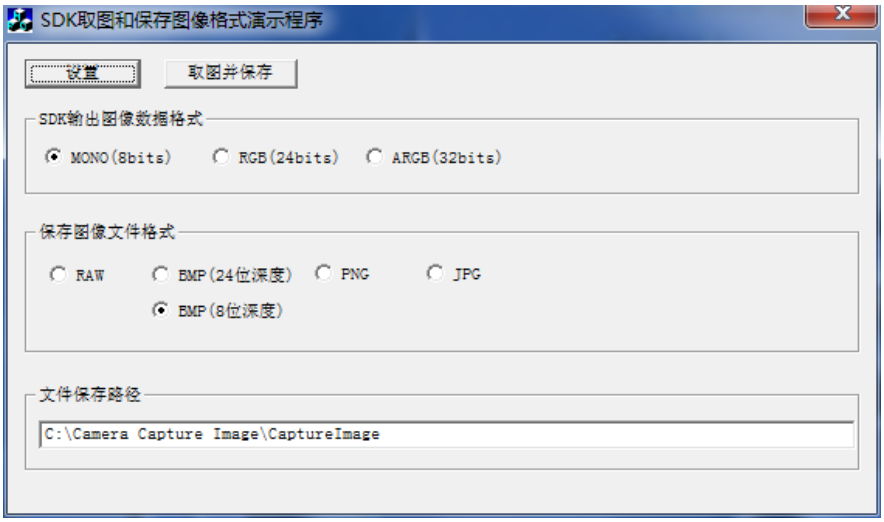
该例程位于安装目录的 Demo/VC++/ ImageFormat&Saving 文件夹中，演示了如何设置 SDK 图像处理输出格式和图像文件的保存操作方式。

目前，我们的 SDK 支持 8bit、24bit、32bit 图像格式输出，可以满足绝大多数的应用场合；提供 RAW 格式、BMP 8、24 位深度格式、PNG、JPG 共 5 种图像格式保存图像文件。其中 RAW 格式是相机原始输出的格式，可能是 8、10、12 和 16bit 的原始资料；BMP8 位深度的图像文件需要配置 SDK 输出格式

为 8bit 模式;BMP24 位深度、JPG、PNG 格式的保存，需要 SDK 输出格式为 24bit 模式。

典型应用：黑白相机输出 8 位灰度格式，会提升图像处理的速度；彩色相机输出 RGB32 格式，地址按 4 字节对齐，方便视觉库用汇编指令进行硬件加速。

该 DEMO 运行后的界面如下图所示。



2.2.7 TriggerAndStrobe 例程(触发和闪光灯信号控制)

该例程位于安装目录的 Demo/VC++/ TriggerAndStrobe 文件夹中，演示了如何设置相机的触发信号和闪光灯信号，注意，该例子对不支持外触发的型号无效。

目前，我们的相机支持 4 种信号触发方式，分别是上升沿触发、下降沿触发、高电平触发、低电平触发，如果是使用机械开关，则建议使用电平触发方式，同时设置去抖时间，来过滤掉机械开关量跳变时的干扰信号，如果是电子开关，则没有限制；闪光灯信号支持全自动和半自动两种方式，其中全自动是在相机曝光

时自动产生信号波形 ;半自动方式就是用程序接口去控制闪光灯信号端子上电平的高低转换。

典型应用：闪光灯同步输出，仅在相机曝光的瞬间启动闪光灯；手动程序设计控制闪关灯输出的引脚电平，来控制红外灯的开关(常亮或者常闭)；利用触发进行多相机同时拍照；利用触发在指定的时间拍照，例如按下按钮时。

该例程的运行接口如下图所示。

2.2.8 UserDataTest 例程(往相机中读写自定义数据)

该例程位于安装目录的 Demo/VC++/ UserDataTest 文件夹中，演示了如何向相机内写入自定义数据并读取和修改相机设备名的功能。

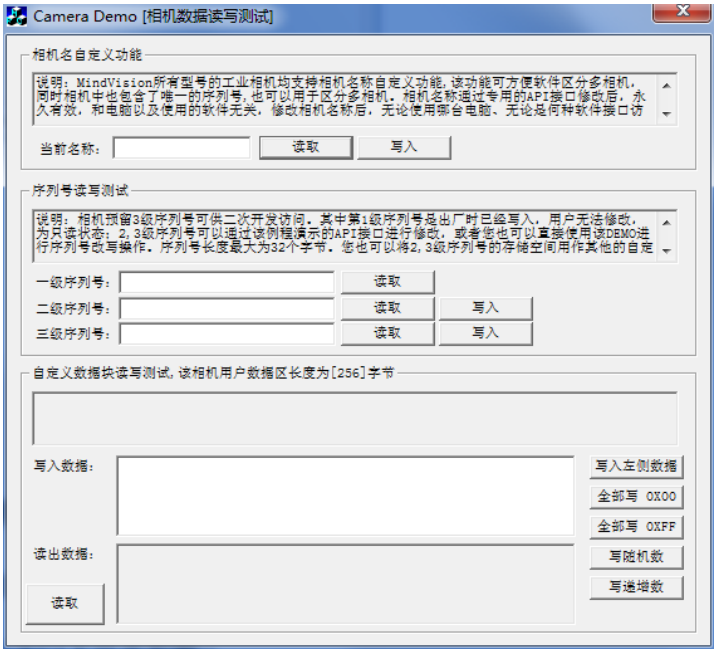
相机名称的修改，会直接影响到相机的枚举列表，名称修改后，Directshow、Twain、Halcon、Labview 等接口访问相机时，名称都会您是修改后的，固化在相机内，永久有效。该功能主要是为了同时接多个相机时便于区

分，修改好每个相机的名称后，就不用担心相机接在哪个接口上了，即使是更换了计算机主机也不会影响的相机的名称。

自定义数据的读写，则是为了方便二次开发时，在相机内保存程序数据；或者是写入一些特殊的数据以实现程序和相机的绑定。

典型应用：读写相机标定数据；读写特殊绑定数据来实现软件和硬件的关联。

该程序运行的接口如下图所示：



2.2.9 SnapshotOnPreview 例程(小分辨率高速预览、大分辨率拍照)

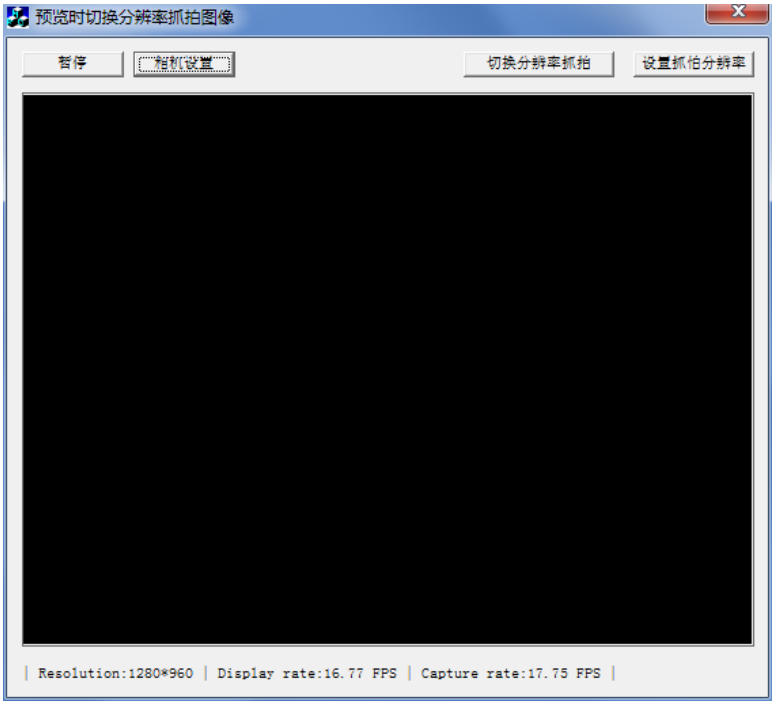
该例程位于安装目录的 Demo/VC++/ SnapshotOnPreview 文件夹中，演示了如何在连续预览时，利用抓拍函数抓取一张指定分辨率的图像，并保存成 BMP 文件(抓拍信道抓到的图像在内存中，可以直接处理也可以另外显示出来，本例中是抓到图像后再进行文件保存)。

抓拍分辨率、预览分辨率的设置以及抓拍图像的获取和预览图像的获取，都有各自独立的接口函数，互不影响；其它参数，如图像亮度、色彩等，预览和抓

拍通道共享。设置好抓拍图像分辨率后，在连续预览时抓拍，SDK 内部会自动进行切换操作，抓拍成功后，自动切换回预览时分辨率。

典型应用：小分辨率下，可高速预览，例如用 BIN 或者 SKIP 的分辨率预览，帧率可提高 4 倍以上，视频流畅性好；大分辨率抓拍，可获得全幅画面，进行高精度的处理。

该例程运行效果如下图，预览分辨率可在"相机设置"里设置：



2.2.10 RawTransTest 例程(离线 RAW 文件转换为 BMP、JPG 文件)

该例程位于安装目录的 Demo/VC++/ RawTransTest 文件夹中，演示了如何利用 SDK 将保存好的相机 RAW 照片文件，转换为 BMP 或者 JPG 文件。

RAW 文件在转换时，需要提供对应的相机配置文件(相机配置文件在安装目录的 Camera/Configs 文件夹里，该文件保存了相机运行时的参数信息)。通过

不同相机的配置文件，就可将不同相机拍摄的 RAW 照片文件准确还原成 BMP、JPG 文件。

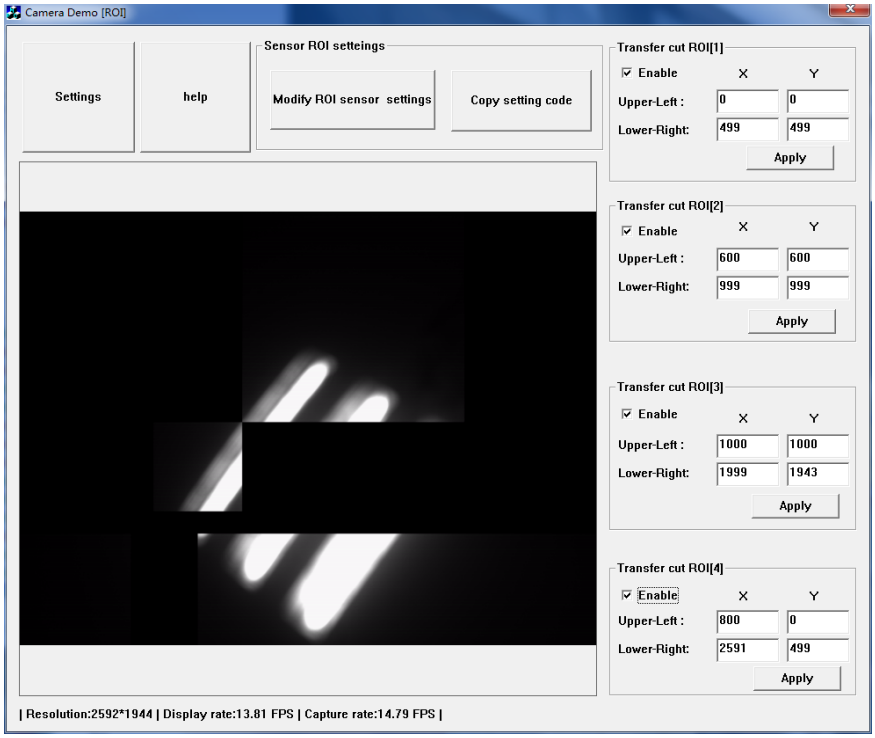
典型应用：单相机或者多相机高速保存 RAW 文件到 SSD 固态硬盘后，后期将 RAW 转换成 BMP、JPG 进行处理，如全景合成等应用。

2.2.11 LineScan 例程(使用线扫描模式)

对于某些型号的面阵相机，如 MV-U130M 和 MV-UB130M，（130 万像素黑白），支持输出一行的 ROI 分辨率，例如 1280X1，此时，可将该面阵相机当做一个性能较低的线阵相机来用，线速度大约 1700 线每秒。

该例程位于安装目录的 Demo/VC++/ LineScan 文件夹中，对于不支持线阵 ROI 分辨率输出的相机，该例程运行时会报错。该例程运行后，会按行读取图像数据，同时拼接成一整副画面后显示，因此该例程适合拍摄圆筒形滚动的物体。

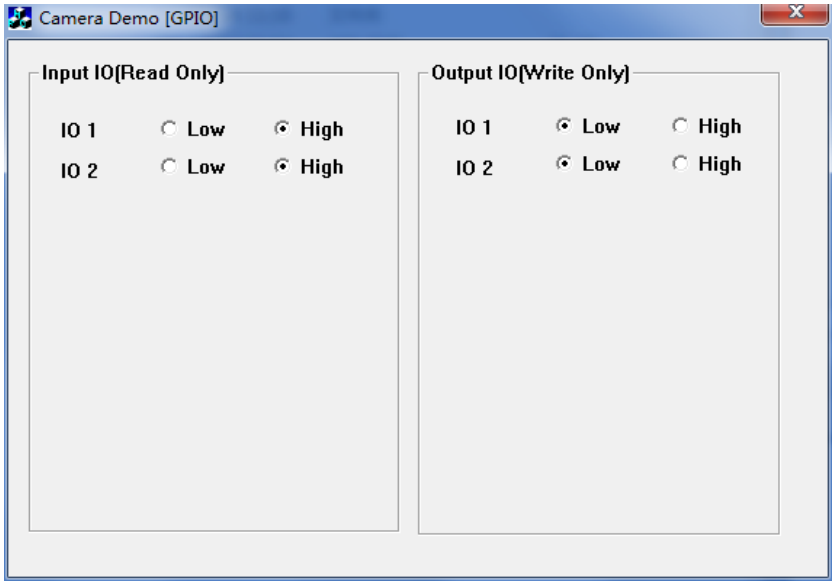
2.2.12 ROI 例程(如何自定义相机图像尺寸)



如上图所示，ROI 例程演示了 500 万像素 USB2.0 相机，在开启多区域 ROI 功能时，在 2592X1944 的图像尺寸下，通过 USB2.0 接口，可以达到每秒 14 帧的帧率。没有参与传输的部份被填充为黑色，四个可裁剪的 ROI 区域可动态修改。

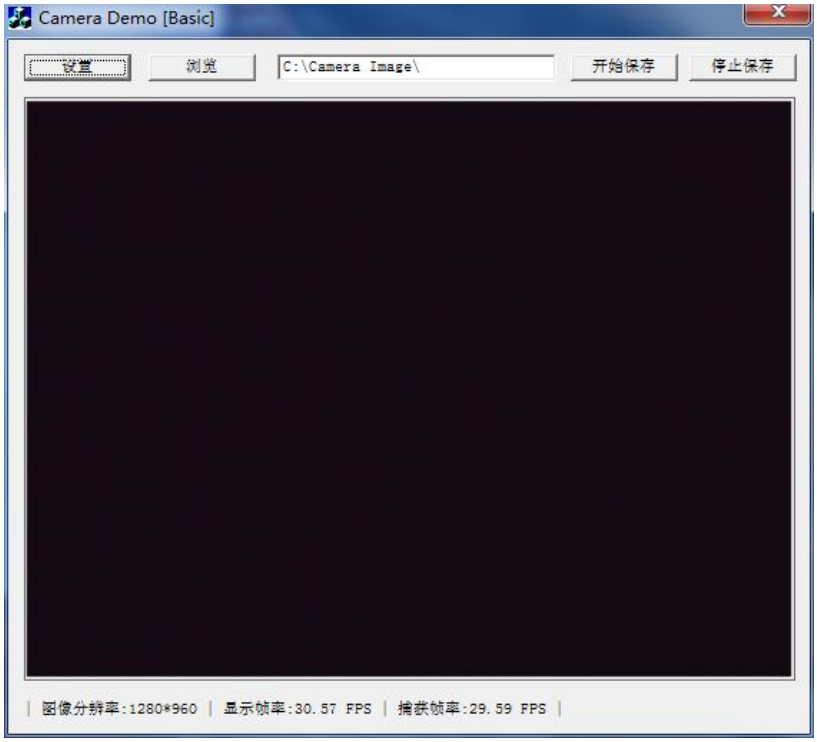
典型应用：当需要拍摄的视野非常大，并且分散为多个区域时，可考虑使用该功能，可有效节约传输带宽、降低 CPU 占用率，同时提高图像传输帧率。

2.2.13 GPIO 例程(仅针对带 GPIO 的型号有效)



如上图所示，GPIO 例程演示了如何操作相机自带的 GPIO 端口。当输入型 IO 状态发生改变时，界面上会自动更新；在界面上操作输出型 IO 时，相机输出 IO 的电平会发生相应的改变。

2.2.14 SaveFile 例程(连续保存每一张预览图像到磁盘中)



如上图所示，SaveFile 例程演示了如何进行高速图像保存的操作，通过该例程，可以直接将预览的每一帧图像都保存到磁盘中，多用于短时间内高速、连续拍照。使用该例程需要注意的是，系统的硬盘性能必须足够好，硬盘的写入带宽，必须大于预览图像的输出带宽。以 MV-GE130GM 相机为例，该相机每秒能输出 1280X960 的图像(120 万像素,1.2M)60 帧，带宽就是 60 X 1.2 = 72MB/s，如果保存图像选择为 RAW 格式，则磁盘写入带宽为 72MB 字节就能满足要求，相机输出的每一帧都能保存下来，如果图像选择为 BMP 格式，则磁盘写入带宽就需要 72MB X 3 = 216 MB/s ，才能将图像全部保存下来。当磁盘写入带宽低于要求时，就会出现部分图像文件无法被保存下来的现象。另外，不建议在高速保存时，选择 RAW 和 BMP 以外的格式，jpg、png 格式的图像，压缩编码需要大量的时间，会造成拥塞从而导致丢帧，RAW 格式完全没有经过任何处理的，需要转换时间是最小的，BMP 格式转换相比 RAW 格式，稍显复杂，但处

理时间一般都比较短，不影响性能。选择 RAW 格式保存，然后再配合 RawTransTest 例程，就可以将 RAW 数据离线转换成 BMP、jpg 或者 png 格式。

2.3 调试相机参数

2.3.1 如何设置曝光时间(抗频闪、无拖影、动态范围提升)

相机的曝光时间,是一个很重要的参数,在很多应用中,设置方法各有讲究,根据不用的应用,曝光时间的设置分为以下几个方面：

- **实现抗频闪效果。**当视觉系统中，用于照明的光源不是直流方式时，您在预览视频时，如果是滚动快门方式的相机，就有可能看到明暗相间的条纹；如果是全域快门方式的相机，就有可能出现前后两场图像亮度变化差异很大，看起来像闪烁的感觉。要解决这个问题，您首先需要知道光源的频率，一般来说，交流照明的光源是 50 赫兹或者 60 赫兹。如果是 50 赫兹的光源，您需要将曝光时间设置为 10 毫秒的整数倍(最小是 10 毫秒)；如果是 60 赫兹的光源，您需要将曝光时间设置为 8.3333 毫秒的整数倍。除此之外，您可以使用我们提供的自动曝光方式，并将抗频闪功能使能。
- **实现无拖影拍照。**在流水在线进行视觉检测时，往往是需要动态拍照，即被检测物体是在运动中抓拍的，如果曝光时间设置的不合理，则有可能造成很明显的拖影，导致画面模糊。要解决这个问题，您需要将曝光时间设置到一个很小的值，一般是需要低于 2 个毫秒，具体看物体运动的速度，不断降低曝光时间来匹配。由于曝光时间降低，图像亮度就会下降，需要外部提供更

强的补光，同时由于滚动快门方式的相机会在垂直方向产生扭曲形变，因此，需要选用全域快门方式的相机进行无拖影抓拍。

- **动态范围提升。**所谓动态范围就是图像所包含的从“最暗”至“最亮”的范围。动态范围越大，所能表示的层次越丰富。在实际成像中，往往会出现看清亮处时，暗处已经暗到无法识别；看清暗处时，亮出已经过曝，无法识别。目前解决这个问题的方法有两种，一是选择硬件提升宽动态范围的相机；二是使用手动设置曝光时间，然后配合伽马值和对比度调节来达到目的，这种方法是属软件提升动态范围，具体操作方法是，首先手动设置曝光时间，到能看清楚最亮的地方，然后，将伽马值往低缓慢调节，对比度往高缓慢调节，调节的过程中，观察图像的变化，直到获得比较好的效果。
- 自动曝光的参考区域设置。默认情况下，自动曝光算法的参考区域是图像正中间的 1/2,可以在软件接口上或者二次开发中调用相关的 SDK 函数，来修改自动曝光参考区域，修改后，当参考区域以外的图像部分发生改变时，自动曝光算法不会更新相机曝光值。

2.3.2 如何获得更好的图像色彩

大部分的工业自动化检测中对物体的色彩信息并不敏感，因此会使用黑白的相机，但是在显微镜成像、颜色分类识别等领域，相机的色彩还原就显得格外重要，在调节相机色彩方面的参数时，有以下几个方面需要注意。

- 白平衡/区域白平衡。大部分的光源本身并不是纯白色的,通过这些光源照明后,被测物体就有可能出现偏色的现象,例如黄色的光源下,相机成像后,图像就有可能是偏黄色的,这种情况下,就需要进行白平衡校正，其作

用就是在各种色温的灯光照明下，将图像中原本是真实白色的区域(但是受光源色温影响已经偏色)，还原成纯白色，即 R = G = B。所谓区域白平衡就是做白平衡操作时，可指定某一图像区域为参考分析对象，默认情况下，白平衡的参考范围是整副图像。

- 手动白平衡/自动白平衡。白平衡的操作上，分为手动和自动 2 种方式。手动白平衡又称一次性白平衡或一键白平衡，是指使用者在需要进行颜色校正时，手动点击软件上的按钮，进行一次白平衡校正，得到校正系数后，在未进行下一次白平衡时，一直使用同样的系数进行色彩校正；自动白平衡则是相机根据图像实时进行分析，动态修改校正系数。一般来说，工业应用上主要是以手动白平衡为主，以保证每次成像后的校正系数完全一样。
- 手动调节红绿蓝三个通道颜色增益。当进行完手动白平衡操作后，白色区域被还原为计算机上认为真实的白色，即白色区域的 R、G、B 分量完全相等，但是这并不意味着当前色彩会让人看起来很舒服，据调查显示，欧洲人偏好暖色调的图像(偏黄)，而亚洲人则偏好冷色调的图像(偏蓝)，这个时候，可以通过手动再修改 R、G、B 三信道增益来改善图像色彩。
- 饱和度调节。不同的人，对色彩浓艳的喜好程度是不一样的，可以通过饱和度调节来改善，喜欢艳丽色彩，则增大饱和度设定值；喜欢淡色调的，则减小饱和度设定值。

2.3.3 如何提高图像清晰度

总的来说，图像的清晰度由镜头的光学指针、聚焦、软件算法等几个方面决定。如果镜头的光学指针低于相机的要求或者聚焦不清晰，选用像素再高的相机也无济于事。

- 镜头与相机的匹配。在相机的参数表上，可以看到相机的光学尺寸，例如 1/2.5"靶面、2.2 X 2.2 um 像素尺寸，根据这些参数来选择适当的镜头，放能达到最佳的效果，具体选择方式在这里不做描述，请参考网络中的相关文文件。
- 锐度调节。可以在软件接口上增大锐度值来提高图像的清晰度。注意：锐度提高的同时，清晰度会明显上升，但是图像噪声也会被凸显出来。

2.3.4 Bayer 还原算法的选择(有助于更好的提取图像轮廓)

对于彩色的工业相机，一般情况下，从相机传输到主机端的数据格式并不是 RGB 数据，而是一种叫做 Bayer 的阵列彩色数据格式，这个格式，需要通过一定的算法，还原成计算机视觉算法能识别的 RGB 彩色数据，不同的还原算法，在效果和性能上有所区别，我们的 SDK 提供了 A1 至 A5 一共五中 Bayer 换算算法可以选择：

- 默认算为 A2 算法，该算法优点，锐度还原较好，但是颜色交界的边缘处容易产生伪彩色。
- 对于使用彩色相机进行轮廓提取，推荐使用 A3、A4 算法，这 2 种算法锐度较 A2 算法有轻微下降，但是图像边缘会更加平滑、连续。

2.3.5 如何降低 CPU 占用率

可以通过以下几个方面来有效降低 CPU 占用率：

- 使用黑白相机，黑白相机无需色彩方面的转换，因此可以节约一部分 CPU 开销。同时设置 SDK 的图像处理格式为 8bit 模式，这样得到的数据是 8 位灰度的，可大幅度减少占用内存空间和内存复制的时间。
- 使用带硬件 ISP 功能的相机，例如 GIGE、USB3.0 相机。
- 调节曝光时间、LUT 曲线等参数，突出被检测物的特征的同时，尽量让不需要检测的部分很暗或者很亮，这样可以极大的降低视觉算法的运行时间。

2.3.6 相机参数保存与载入

SDK 中提供了四组参数供二次开发或者用户直接使用，用于保存或者加载相机参数。在我们的软件平台中，相机参数是以二进制文件保存在磁盘上的，保存路径是软件安装目录下/Camera/Configs，后缀名是.Config 文件。

- **三种不同的参数加载方法**

相机参数的存取方式可分为按型号、按昵称、按序号三种方式，预设是按型号加载，如下图所示：

以 MV-UB130M(130 万黑白帧存相机)为例:

1. 当选择按型号加载时四组参数的文件名分别是

MV-UB130M-Group0.Config, MV- UB130M-Group1.Config, MV- UB130M-Group2.Config, MV- UB130M-Group3.Config,分别对应软件设置接口上的 A，B, C ,D 四组参数,这种参数加载模式下,该计算机上连接的所有同型号的相机,都共享这四组参数;

当选择按昵称方式加载参数时，四组参数的文件名分别是 MV-UB130M#0--Group0.Config, MV- UB130M#0-Group1.Config, MV-UB130M#0-Group2.Config, MV- UB130M#0-Group3.Config, 分别对应软件设置接口上的 A，B, C ,D 四组参数,值得注意的是，相机的昵称是可以修改的，可以在软件接口上直接修改，也可以在 SDK 中通过软件接口修改，昵称修改后是永久有效的，固化在相机内，更换计算机、或者 USB 插槽后，其昵称不变，修改昵称后，相机对应的参数文件名也会变化，例如将相机改名为 Camera1, 那四组参数的文件名分别是 Camera1-Group0.Config, Camera1-Group1.Config, Camera1--Group2.Config, Camera1-Group3.Config,这种参数加载模式比较灵活,当计算机只连接一个相机时,如果未更改设备昵称,则每个同型号相机的昵称是一样的,大家共享这四组参数,如果想要其中某一台加载不同的参数组,则可以单独命名这一台相机,使其设备昵称与其他的同型号相机区分开来。

当选择按序号加载时，四组参数的文件名分别是 MVUB130M-XXXXX-XXXXX-Group0.Config, MVUB130M-XXXXX-XXXXX-Group1.Config, MVUB130M-XXXXX-XXXXX-Group2.Config,

MVUB130M-XXXXX-XXXXX-Group3.Config,其中(XXXXX-XXXXX 为每个相机唯一的序号)，因此该参数加载方式下，每个相机的参数都是分开的，互不影响。

在实际应用中，您可根据不同的需求来选择不同的加载方式，按型号加载可保证所有同型号相机使用一样的参数；按昵称加载方式，则可以让同一型号的相机使用相同的参数，也可以单独指定某一台使用独立的参数；按序号加载，则每台相机都使用独立的参数。

- **修改相机的默认参数**

当用户在软件配置接口上点击“恢复默认参数”按钮时，相机会进行参数复位操作，将参数复位到一个预设状态，以防止使用者在不了解相机参数的情况下，修改参数后出现的各种“异常”情况。

进行默认参数恢复时，参数会被恢复成出厂时的状态，但如果您是做二次开发，很可能希望客户在点击按钮时，恢复到你设定的默认参数，而不是出厂时的参数，这里，我们也提供了一种方法，可以修改预设参数，让终端客户在进行恢复预设参数时(或者是首次安装相机程序后，第一次打开相机)能够是您设定好的参数。方法很简单：使用我们的演示软件，将相机参数设定好后，保存成一个单独档，文件名为型号名+"-Default.Config"，以 MV-UB130M(130 万黑白帧存相机)为例，将其参数保存成档"MV-UB130M-Default.Config",您也可以直接将现有的 Config 档，更名为"MV-UB130M-Default.Config",然后将这个文件放在安装目录的/Camera/Configs 目录下。这样，终端使用者在进行预设参数恢复时就会恢复到"MV-UB130M-Default.Config"文件中的参数。

3 SDK 数据类型定义

3.1 结构体定义

➢ tSdkCameraDevInfo

原型：

```
typedef struct
{
    UINT    uVendorID;              // 厂商 ID
    UINT    uProductID;             // 产品 ID
    char acVendorName[32];          // 厂商名称
    char acProductSeries[32];       // 产品系列
    char acProductName[32];         // 产品名称
    char acFriendlyName[64];        // 初始化接口使用的设备昵称
    char acDevFileName[32];         // 驱动名称
    char acFirmwareVersion[32];     // 版本
    char acSensorType[32];          // sensor 类型
    char acPortType[32];            // 接口类型
} tSdkCameraDevInfo;
```

说明：相机的设备信息

➢ tSdkImageResolution

原型：

```
typedef struct
{
INT         iIndex;                 // 索引号，[0,N]表示默认的分辨率(N 为默认分辨率的最大个数，一般不超过 20),OXFF 表示自定义分辨率(ROI)
  char      acDescription[32];  // 该分辨率的描述信息。仅默认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 中 uBinSumModeMask
  UINT      uBinAverageMode;      // BIN(求均值)的模式,范围不能超过 tSdkResolutionRange 中 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 中 uSkipModeMask
    UINT        uResampleMask;          // 硬件重采样的屏蔽
    INT         iHOffsetFOV;            // 采集视场相对于 Sensor 最大视场左
上角的垂直偏移
    INT         iVOffsetFOV;            // 采集视场相对于 Sensor 最大视场左
上角的水平偏移
    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.
} tSdkImageResolution;
```

说明：相机的分辨率描述

➢ tSdkMediaType

原型：

```
typedef struct
{
    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
} tSdkMediaType;
```

说明：相机输出的图像数据格式

➢ tSdkIspCapacity

原型：

```
typedef struct
{
        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)。
    } tSdkIspCapacity;
```

说明：ISP 模块的使能信息

➢ tSdkCameraCapbility

原型：

```
typedef struct
{
        tSdkTrigger     *pTriggerDesc;              //  触发模式
        INT             iTriggerDesc;               //  触发模式的个数，即 pTriggerDesc 数组的大小

        tSdkImageResolution     *pImageSizeDesc;//  默认分辨率选择
        INT                     iImageSizeDesc; //  默认分辨率的个数，即 pImageSizeDesc 数组的大小
```

```
    tSdkColorTemperatureDes *pClrTempDesc;// 默认色温模式，用于白平衡
    INT                     iClrTempDesc;

    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，对应接口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              iPackLenDesc;       // 可供选择的传输分包长度的个数，即 pPackLenDesc 数组的大小。

    INT              iOutputIoCounts;    // 可程序设计输出 IO 的个数
    INT              iInputIoCounts;     // 可程序设计输入 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

    /* 图像参数的调节范围定义,用于动态构建 UI*/
    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
} tSdkCameraCapbility;
```

说明：定义整合的设备描述信息，这些信息可以用于动态构建 UI。

➢ tSdkFrameHead

原型：

```
typedef struct
{
    UINT    uiMediaType;        // 图像格式,Image Format
    UINT    uBytes;             // 图像数据字节数,Total bytes
    UINT    iHeight;            // 高度 Image width
    UINT    iWidth;             // 宽度 Image height
    BOOL    bIsTrigger;         // 指示是否为触发帧 is trigger
    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
}tSdkFrameHead;
```

说明：图像帧头信息

➢ tSdkFrame

原型：

```
typedef struct sCameraFrame
{
    tSdkFrameHead head;         //帧头
    BYTE *      pBuffer;        //数据区
}tSdkFrame;
```

说明：图像帧描述

3.2 参数类型定义

➢ emSdkLutMode

原型：

```
typedef enum
{
    LUTMODE_PARAM_GEN=0,        //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
}emSdkLutMode;
```

说明：图像查表变换的方式

➢ emSdkParamTarget

原型：

```
typedef enum
{
    PARAM_ON_PC = 0,            //保存到 PC 上
    PARAM_ON_DEVICE = 1,        //保存到相机中
}emSdkParamTarget;
```

说明：相机参数保存的对象

➢ emSdkRunMode

原型：

```
typedef enum
{
    RUNMODE_PLAY=0,         //正常预览，捕获到图像就显示。
    RUNMODE_PAUSE,          //暂停
    RUNMODE_STOP,           //停止相机工作。
}emSdkRunMode;
```

说明：相机的视频流控制

➢ emSdkDisplayMode

原型：

```
typedef enum
{
    DISPLAYMODE_SCALE=0,            //缩放显示模式
    DISPLAYMODE_REAL                //1:1 显示模式
}emSdkDisplayMode;
```

说明：SDK 内部显示接口的显示方式

➢ emSdkRecordMode

原型：

```
typedef enum
{
    RECORD_STOP = 0,                //停止
    RECORD_START,                   //录像中
    RECORD_PAUSE,                   //暂停
}emSdkRecordMode;
```

说明：录像状态

➢ emSdkMirrorDirection

原型：

```
typedef enum
{
    MIRROR_DIRECTION_HORIZONTAL = 0,        //水平镜像
    MIRROR_DIRECTION_VERTICAL ,             //垂直镜像
}emSdkMirrorDirection;
```

说明：图像的镜像操作

➢ emSdkFrameSpeed

原型：

```
typedef enum
{
    FRAME_SPEED_LOW = 0,            //低速模式
    FRAME_SPEED_NORMAL,             //普通模式
```

```
    FRAME_SPEED_HIGH,           //高速模式
    FRAME_SPEED_SUPER,              //超高速模式
}emSdkFrameSpeed;
```

说明：相机视频的帧率

➢ emSdkFileType

原型：

```
typedef enum
{
    FILE_JPG = 1,               //JPG
    FILE_BMP = 2,               //BMP
    FILE_RAW = 4,                   //RAW：相机输出的 bayer 格式档
    FILE_PNG = 8                //PNG
}emSdkFileType;
```

说明：保存档的格式类型

➢ emSdkLightFrequency

原型：

```
typedef enum
{
    LIGHT_FREQUENCY_50HZ = 0,       //50HZ
    LIGHT_FREQUENCY_60HZ ,          //60HZ
}emSdkLightFrequency;
```

说明：自动曝光时抗频闪的频闪

➢ emSdkParameterMode

原型：

```
typedef enum
{
   PARAM_MODE_BY_MODEL = 0,  //根据相机型号名从档中载入参数，例如 MV-U300
  PARAM_MODE_BY_NAME,          //根据设备昵称(tSdkCameraDevInfo.acFriendlyName)从档中载入参数 ,例如 MV-U300,该昵称可自定义
```

```
    PARAM_MODE_BY_SN,               //根据设备的唯一序号从档中加载参数，序号在出厂时已经写入设备，每台相机拥有不同的序号。
    PARAM_MODE_IN_DEVICE            //从设备的固态内存中加载参数。不是所有的型号都支持从相机中读写参数组，由 tSdkCameraCapbility.bParamInDevice 决定
}emSdkParameterMode;
```

说明：PARAM_MODE_BY_MODEL:所有同型号的相机共享 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
相机参数载入。

PARAM_MODE_BY_NAME:所有设备名相同的相机，共享 ABCD 四组参数文件。
默认情况下，当计算机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
不同的参数档，则可以通过修改其设备名的方式
来让其加载指定的参数档。

PARAM_MODE_BY_SN:相机按照自己的唯一序号来加载 ABCD 四组参数档，
序号在出厂时已经固化在相机内，每台相机的序号
都不相同，通过这种方式，每台相机的参数档都是独立的。

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计算机上都共享 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
使用自己参数档而其余的 MV-U300 又要使用相同的参数档，那么使用 PARAM_MODE_BY_NAME 方式;如果您希望每台 MV-U300 都使用不同的参数档，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 为后缀名的档。

➢ emSdkParameterTeam

原型：

```
typedef enum
{
    PARAMETER_TEAM_DEFAULT = 0xff,
    PARAMETER_TEAM_A = 0,
    PARAMETER_TEAM_B = 1,
    PARAMETER_TEAM_C = 2,
    PARAMETER_TEAM_D = 3
}emSdkParameterTeam;
```

说明：相机的配置参数，分为 A,B,C,D 4 组进行保存。

➢ emSdkPropSheetMsg

原型：

```
typedef enum
{
    SHEET_MSG_LOAD_PARAM_DEFAULT = 0,       //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,          //从指定档加载参数后，触发该消息
    SHEET_MSG_SAVE_PARAM_GROUP               //当前参数组被保存时，触发该消息
}emSdkPropSheetMsg;
```

说明：SDK 生成的相机配置页面的回调消息类型

➢ tSdkWhiteBalanceDes

原型：

```
typedef struct
{
    INT    iIndex;                  // 模式索引号
    char acDescription[32];         // 描述信息
} tSdkWhiteBalanceDes;
```

说明：相机白平衡模式描述信息

➢ tSdkFrameSpeed

原型：

```
typedef struct
{
    INT    iIndex;                    // 帧率索引号
    char acDescription[32];           // 描述信息
} tSdkFrameSpeed;
```

说明：相机帧率描述信息

➢ tSdkExpose

原型：

```
typedef struct
{
    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，
    UINT    uiAnalogGainMax;          //模拟增益的最大值
    float    fAnalogGainStep;         //模拟增益步进。
    UINT    uiExposeTimeMin;          //手动模式下，曝光的最小行数
    UINT    uiExposeTimeMax;          //手动模式下，曝光的最大行数
} tSdkExpose;
```

说明：相机曝光功能范围定义

➢ tSdkTrigger

原型：

```
typedef struct
{
    INT      iIndex;                  //模式索引号
    char     acDescription[32];       //该模式的描述信息
} tSdkTrigger;
```

说明：触发模式描述

➢ tSdkPackLength

原型：

```
typedef struct
{
    INT     iIndex;                    //分包大小索引号
    char    acDescription[32];         //对应的描述信息
} tSdkPackLength;
```

说明：传输分包大小描述

➢ tSdkHardwareIO

原型：

```
typedef struct
{
    INT     iIndex;                    //IO 的编号
    BOOL    bOutPut;                   //IO 的属性，输入或者是输出
} tSdkHardwareIO;
```

说明：相机上可程序设计 IO 的描述

➢ tSdkPresetLut

原型：

```
typedef struct
{
    INT   iIndex;                      //编号
    char acDescription[32];            //描述信息
} tSdkPresetLut;
```

说明：预设的 LUT 表描述

➢ tSdkFrameStatistic

原型：

```
typedef struct
{
    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
```

```
} tSdkFrameStatistic;
```

说明：帧率统计信息

➢ tGammaRange

原型：

```
typedef struct
{
    INT     iMin;                   //最小值
    INT     iMax;                   //最大值
} tGammaRange;
```

说明：伽马的设定范围

➢ tContrastRange

原型：

```
typedef struct
{
    INT iMin;                       //最小值
    INT iMax;                       //最大值
} tContrastRange;
```

说明：对比度的设定范围

➢ tRgbGainRange

原型：

```
typedef struct
{
    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
} tRgbGainRange;
```

说明：RGB 三信道数字增益的设定范围

➢ tSaturationRange

原型：

```
typedef struct
{
    INT iMin;       //最小值
    INT iMax;       //最大值
} tSaturationRange;
```

说明：饱和度设定的范围

➢ tSdkResolutionRange

原型：

```
typedef struct
{
  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 模式屏蔽，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)模式屏蔽，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)模式屏蔽，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;             //硬件重采样的屏蔽
} tSdkResolutionRange;
```

说明：相机的分辨率设定范围

➢ emSdkRefWinType

原型：

```
typedef enum
{
    REF_WIN_AUTO_EXPOSURE = 0,
    REF_WIN_WHITE_BALANCE,
}emSdkRefWinType;
```

说明：可视化选择参考窗口的类型

➢ tSharpnessRange

原型：

```
typedef struct
{
    INT iMin;       //最小值
    INT iMax;       //最大值
} tSharpnessRange;
```

说明：锐化的设定范围

➢ SKIP_MASK_xxx 宏定义

原型：

```
#define MASK_2X2_HD       (1<<0)     //硬件 SKIP、BIN、重采样  2X2
#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)    //硬件 SKIP、BIN、重采样  2X2
#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)
```

说明：tSdkResolutionRange 结构体中 SKIP、 BIN、RESAMPLE 模式的屏蔽值

➢ CAMERA_MEDIA_TYPE_MONOxx 宏定义

原型：

```
#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)
```

说明：图像格式定义

➢ CAMERA_MEDIA_TYPE_BAYxxx 宏定义

原型：

```
#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)
```

说明：Bayer 格式编码定义

➢ CAMERA_MEDIA_TYPE_RGB8xx 宏定义

原型：

```
#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_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)
```

说明：RGB 格式编码定义

➢ CAMERA_MEDIA_TYPE_YUV411_8_xxxx 宏定义

原型：

```
#define CAMERA_MEDIA_TYPE_YUV411_8_UYYVYY
(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_CBYYCRYY
(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_CBYYCRYY
(CAMERA_MEDIA_TYPE_COLOR |
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_CBYYCRYY
(CAMERA_MEDIA_TYPE_COLOR |
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0042)
```

说明：YUV 格式编码定义。

➢ CAMERA_MEDIA_TYPE_RGBXX_PLANAR 宏定义

原型：

```
#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)
```

说明：RGB planar 格式编码定义。

3.3 接口返回值定义(错误码)

```
#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，则回返回该错误。
#define CAMERA_STATUS_NO_MEMORY                 -21
// 没有足够系统内存
#define CAMERA_STATUS_FILE_CREATE_FAILED        -22
// 创建档失败
#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_MOUDLE_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_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
//0x8008-0x0800B   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 x0814
//由于通道带宽问题，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                x0816
//命令执行已经超过有效的指定时间

#define CAMERA_AIA_ERROR                      0x8FFF
//错误
```

4 SDK 接口函数说明(C/C++ VB Delphi C#通用)

本章节描述 SDK 中各接口函数的原型及说明，对于 C/C++、VB、Delphi、可直接按本手册中的函数名进行调用，C#中则使用封装好的转接类 MVSDK.MvApi 进行调用。例如，MVSDK.MvApi.CameraSdkInit(1)进行初始化，其他接口的函数的调用，也只需要加上 MVSDK.MvApi 前缀即可。

4.1 CameraSdkInit

原型：

```
MVSDK_API CameraSdkStatus
CameraSdkInit
(
    int            iLanguageSel
);
```

功能说明：SDK 初始化

参数说明：iLanguageSel：接口语言选择，1：中文，0：英文。其他暂不支持。

返回值：成功时，返回 CAMERA_STATUS_SUCCESS =0;否则返回非 0 值的错误码,请参考 CameraStatus.h 中错误码的定义。

注意：在调用任何 SDK 其他接口前，必须先调用该接口进行初始化。该函数在整个进程运行期间只需要调用一次。

Example：

```
/* 初始化 SDK 为中文接口 */
CameraSdkInit(1);
```

4.2 CameraInit

原型：

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

功能说明：相机初始化。

参数说明：

pCameraInfo：该相机的枚举描述信息，由 CameraEnumerateDevice 获得。

emParamLoadMode: 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。其余值参考 emSdkParameterMode 类型定义。

emTeam：初始化时使用的参数组。-1 表示载入上次退出时的参数组。

*pCameraHandle：相机的句柄指标，初始化成功后，该指标返回该相机的有效句柄，在调用其他相机相关的操作接口时，都需要传入该句柄，主要用于多相机之间的区分。

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

注意：初始化成功后，才能调用任何其他相机相关的操作接口。

Example：

```
/* 初始化相机，载入上次退出时的参数设置 */
tSdkCameraDevInfo   sCameraList[10];
INT                 iCameraNums = 10;
INT                 status;
```

```
if (CameraEnumerateDevice(sCameraList,&iCameraNums) ==
CAMERA_STATUS_SUCCESS && iCameraNums > 0)
{
    status  =  CameraInit(&sCameraList[0],-1,-1,&m_hCamera);
     if (status  ==  CAMERA_STATUS_SUCCESS)
    {
    //相机初始化成功
    }
}
```

4.3 CameraInitEx

原型：

```
MVSDK_API CameraSdkStatus
CameraInitEx
(
    int                 iDeviceIndex,
    int                 iParamLoadMode,
    int                 emTeam,
    CameraHandle*     pCameraHandle
);
```

功能说明：相机初始化。

参数说明：

iDeviceIndex　　相机的索引号，从 0 开始。

iParamLoadMode　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。

为 PARAM_MODE_BY_MODEL 表示按型号加载

为 PARAM_MODE_BY_SN 表示按序列号加载

为 PARAM_MODE_BY_NAME 表示按昵称加载

详细请参开 CameraDefine.h 中 emSdkParameterMode 定义。

emTeam　　初始化时使用的参数组。-1 表示加载上次退出时的参数组。

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，在调用其他相机相关的操作接口时，都需要传入该句柄，主要用于多相机之间的区分。

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

注意：初始化成功后，才能调用任何其他相机相关的操作接口。

Example：

```
/* 初始化相机，载入上次退出时的参数设置 */
tSdkCameraDevInfo    sCameraList[10];
INT                  iCameraNums = 10;
INT                  status;
if ((iCameraNums    =
CameraEnumerateDeviceEx(sCameraList,&iCameraNums) )> 0)
{
    status    =    CameraInitEx(0,-1,-1,&m_hCamera);
     if (status    ==    CAMERA_STATUS_SUCCESS)
     {
     //相机初始化成功
     }
}
```

4.4 CameraInitEx2

原型：

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

功能说明：相机初始化。

参数说明：

CameraName　　相机的名字，字符串，以 0 字符结尾。
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，在调用其他相机相关的操作接口时，都需要传入该句柄，主要用于多相机之间的区分。

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

注意：初始化成功后，才能调用任何其他相机相关的操作接口。

Example：

```
/* 初始化名字为 Camera1 的相机，需要事先用我们软件将相机名字改为 Camera1 才能初始化成功 */
tSdkCameraDevInfo  sCameraList[10];
INT                iCameraNums = 10;
INT                status;
if ((iCameraNums  = CameraEnumerateDeviceEx(sCameraList,&iCameraNums) )> 0)
{
    status  =  CameraInitEx2("Camera1",-1,-1,&m_hCamera);
    if (status  ==  CAMERA_STATUS_SUCCESS)
    {
    //相机初始化成功
    }
}
```

4.5 CameraDisplayInit

原型：

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

功能说明：初始化 SDK 内部的显示模块

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

hWndDisplay：显示窗口的句柄，一般为窗口的 m_hWnd 成员。

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

注意：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。

Example：

```
/* 初始化显示接口，使用 SDK 封装好的显示接口*/
CameraDisplayInit(m_hCamera,m_cPreview.GetSafeHwnd())
```

4.6 CameraSetDisplayMode

原型：

```
MVSDK_API CameraSdkStatus
CameraSetDisplayMode
(
    CameraHandle    hCamera,
    int             mode
);
```

功能说明：设置显示的模式。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

mode：显示模式，DISPLAYMODE_SCALE 或者 DISPLAYMODE_REAL，具体参见 CameraDefine.h 中 emSdkDisplayMode 的定义

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

注意：必须调用过 CameraDisplayInit 进行初始化才能调用本函数。

Example：

4.7 CameraSetDisplayOffset

原型：

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

功能说明：设置显示的起始偏移值。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iOffsetX：偏移的 X 坐标。

iOffsetY：偏移的 Y 坐标。

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

注意：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。必须调用过 CameraDisplayInit 进行初始化才能调用本函数。

Example：

4.8 CameraSetCallbackFunction

原型：

```
MVSDK_API CameraSdkStatus
CameraSetCallbackFunction
(
    CameraHandle            hCamera,
    CAMERA_SNAP_PROC        pCallbackFunction,
    PVOID                   pContext,
    CAMERA_SNAP_PROC        *pCallbackOld
);
```

功能说明：设置图像捕获的回调函数。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

pCallbackFunction：回调函数指针。

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，可以为 NULL。多用于多个相机时携带附加信息。

*pCallbackOld：用于保存当前的回调函数。可以为 NULL。

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

注意：

Example：

```
/* */
CameraSetCallbackFunction(m_hCamera,GrabImageCallback,(PVOID
)this,NULL)
```

4.9 CameraUnInit

原型：

```
MVSDK_API CameraSdkStatus
```

```
CameraUnInit
(
      CameraHandle     hCamera
);
```

功能说明：相机反初始化。释放资源。
参数说明： hCamera 相机的句柄，由 CameraInit 函数获得。
返回值：成功时，返回 CAMERA_STATUS_SUCCESS (0);否则返回非 0 值的错误码,请参考 CameraStatus.h 中错误码的定义。

Example：

```
/* 反初始化相机 */
CameraUnInit(m_hCamera)
```

4.10 CameraPlay

原型：

```
MVSDK_API CameraSdkStatus
CameraPlay
(
    CameraHandle          hCamera
);
```

功能说明：让 SDK 进入工作模式，开始接收来自相机发送的图像。
参数说明：hCamera：相机的句柄，由 CameraInit 函数获得。
返回值：成功时，返回 CAMERA_STATUS_SUCCESS (0);否则返回非 0 值的错误码,请参考 CameraStatus.h 中错误码的定义。
注意：如果当前相机是触发模式，则需要接收到触发帧以后才会更新图像。

Example：

```
/* 相机开始采集图像 */
Cameraplay(m_hCamera);
```

4.11 CameraPause

原型：

```
MVSDK_API CameraSdkStatus
CameraPause
(
    CameraHandle          hCamera
);
```

功能说明：让 SDK 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。

参数说明：hCamera：相机的句柄，由 CameraInit 函数获得。

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

注意：暂停模式下，可以对相机的参数进行配置，并立即生效。

Example：

```
/*  相机开始采集图像  */
CameraPause (m_hCamera);
```

4.12 CameraStop

原型：

```
MVSDK_API CameraSdkStatus
CameraStop
(
    CameraHandle          hCamera
);
```

功能说明：让 SDK 进入停止状态，一般是反初始化时调用该函数。

参数说明：hCamera：相机的句柄，由 CameraInit 函数获得。

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

注意：该函数被调用，不能再对相机的参数进行配置。

Example：

```
/* 相机开始采集图像 */
CameraStop (m_hCamera);
```

4.13 CameraCreateSettingPage

原型：

```
MVSDK_API CameraSdkStatus
CameraCreateSettingPage
(
    CameraHandle            hCamera,
    HWND                    hParent,
    Char                    *pWinText,
    CAMERA_PAGE_MSG_PROC    pCallbackFunc,
    PVOID                   pCallbackCtx,
    UINT                    uReserved
);
```

功能说明：创建该相机的属性配置窗口。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

hParent：应用程序主窗口的句柄。可以为 NULL。

*pWinText：字符串指针，窗口显示的标题栏。

pCallbackFunc：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，例如切换了参数之类的操作时，pCallbackFunc 被回调时，在入口参数处指明了消息类型。这样可以方便您自己开发的接口和我们生成的 UI 之间进行同步。该参数可以为 NULL。

pCallbackCtx：回调函数的附加参数。可以为 NULL。pCallbackCtx 会在 pCallbackFunc 被回调时，做

为参数之一传入。您可以使用该参数来做一些灵活的判断。

uReserved：预留。必须设置为 0。

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

注意：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接口的时间。强烈建议使用您使用该函数让 SDK 为您创建好配置窗口。

Example：

```
```

4.14 CameraCustomizeResolution

原型：

```
MVSDK_API CameraSdkStatus
CameraCustomizeResolution
(
    CameraHandle            hCamera,
    tSdkImageResolution     *pImageCustom
);
```

功能说明：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。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。相机名

*pImageCustom：指针，返回自定义的分辨率。

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

Example：

```
/*  用户自定义分辨率  */
```

```
CameraCustomizeResolution(m_hCamera,&sImageSize)
```

4.15 CameraShowSettingPage

原型：

```
MVSDK_API CameraSdkStatus
CameraShowSettingPage
(
    CameraHandle        hCamera,
    BOOL                bShow
);
```

功能说明：设置相机属性配置窗口显示状态。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

bShow：TRUE，显示。FALSE，隐藏。

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

注意：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。

Example：

```

```

4.16 CameraSetActiveSettingSubPage

原型：

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

```
    INT                 index
);
```

功能说明：设置相机配置窗口的启动页面。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

index：子页面的索引号。

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

注意：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启动状态，显示在最前端。

Example：

4.17 CameraGetInformation

原型：

```
MVSDK_API CameraSdkStatus
CameraGetInformation
(
    CameraHandle     hCamera,
    char             **pbuffer
);
```

功能说明：获得相机的描述信息

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

**pbuffer：指向相机描述信息指针的指针。

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

Example：

```
```

4.18 CameraImageProcess

原型：

```
MVSDK_API CameraSdkStatus
CameraImageProcess
(
    CameraHandle        hCamera,
    BYTE                *pbyIn,
    BYTE                *pbyOut,
    tSdkFrameHead       *pFrInfo
);
```

功能说明：将获得的相机原始输出图像数据进行处理，迭加饱和度、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。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pbyIn：输入图像数据的缓冲区地址，不能为 NULL。

*pbyOut：处理后图像输出的缓冲区地址，不能为 NULL。

*pFrInfo：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。

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

Example：

```
/* */
CameraImageProcess(m_hCamera,
pbyBuffer,m_pbImgBuffer,&tFrameHead);
```

4.19 CameraImageProcessEx

原型：

```
MVSDK_API CameraSdkStatus
CameraImageProcessEx
(
    CameraHandle        hCamera,
    BYTE*               pbyIn,
    BYTE*               pbyOut,
    tSdkFrameHead*      pFrInfo,
    UINT                uOutFormat,
    UINT                uReserved
);
```

功能说明：将获得的相机原始输出图像数据进行处理，迭加饱和度、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。

参数说明：

hCamera 相机的句柄，由 CameraInit 函数获得。

pbyIn 输入图像数据的缓冲区地址，不能为 NULL。

pbyOut 处理后图像输出的缓冲区地址，不能为 NULL。

pFrInfo 输入图像的帧头信息，处理完成后，帧头信息

uOutFormat 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_BGR、CAMERA_MEDIA_TYPE_RGBA8 CAMERA_MEDIA_TYPE_RGBA8 的其中一种。

pbyIn 对应的缓冲区大小，必须和 uOutFormat 指定的格式相匹配。

uReserved 预留参数，必须设置为 0

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

4.20 CameraDisplayRGB24

原型：

```
MVSDK_API CameraSdkStatus
CameraDisplayRGB24
(
    CameraHandle        hCamera,
    BYTE                *pbyRGB24,
    tSdkFrameHead       *pFrInfo
);
```

功能说明：显示图像。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pbyRGB24：图像的数据缓冲区，RGB888 格式。

*pFrInfo：图像的帧头信息。

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

注意：必须调用过 CameraDisplayInit 进行初始化才能调用本函数。

Example：

```
/*调用显示函数显示图像*/
CameraDisplayRGB24(pThis->m_hCamera, pThis->m_pFrameBuffer,
pFrameHead);
```

4.21 CameraSetDisplaySize

原型：

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

```
);
```

功能说明：设置显示控件的尺寸。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iWidth：宽度

iHeight：高度

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

注意：必须调用过 CameraDisplayInit 进行初始化才能调用本函数。

Example：

```
/* */
CameraSetDisplaySize(m_hCamera,rect.right - rect.left,rect.bottom - rect.top)
```

4.22 CameraGetImageBuffer

原型：

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

功能说明：获得一帧图像数据。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pFrameInfo：图像的帧头信息指针。

**pbyBuffer：指向图像的数据的缓冲区指针。由于采用了零拷贝机制来提高效率 ,因此这里使用了一个指向指标的指标。

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信息。

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

注意：为了提高效率，SDK 在图像抓取时采用了零拷贝机制，CameraGetImageBuffer 实际获得是内核中的一个缓冲区地址，该函数成功调用后，必须调用 CameraReleaseImageBuffer 释放由 CameraGetImageBuffer 得到的缓冲区,以便让内核继续使用该缓冲区。

Example：

```
/* */
CameraGetImageBuffer(pThis->m_hCamera,&sFrameInfo,&pbyBuffer,1000);
```

4.23 CameraGetImageBufferEx

原型：

```
MVSDK_API CameraSdkStatus
CameraGetImageBufferEx
(
    CameraHandle        hCamera,
    INT*                piWidth,
    INT*                piHeight,
    UINT                wTimes
);
```

功能说明：获得一帧图像数据。区别于 CameraGetImageBuffer 函数，该函数得到的 RGB 格式的图像数据。后续不需要再调用 CameraImageProcess 函数和 CameraReleaseImageBuffer 函数。

参数说明：

hCamera　相机的句柄，由 CameraInit 函数获得。

piWidth　整形指针，返回图像的宽度

piHeight　整形指针，返回图像的高度 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信息。

返回值：成功时，**返回图像数据缓冲区的首地址**失败时，返回 NULL 或者 0。

4.24 CameraReleaseImageBuffer

原型：

```
MVSDK_API CameraSdkStatus
CameraReleaseImageBuffer
(
    CameraHandle          hCamera,
    BYTE                  *pbyBuffer
);
```

功能说明：释放由 CameraGetImageBuffer 获得的缓冲区。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pbyBuffer：由 CameraGetImageBuffer 获得的缓冲区地址。

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

Example：

```
/* */
CameraReleaseImageBuffer(pThis->m_hCamera,pbyBuffer);
```

4.25 CameraGetFrameStatistic

原型：

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

功能说明：获得相机接收帧率的统计信息，包括错误帧和丢帧的情况。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*psFrameStatistic：指针，返回统计信息。

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

Example：

```

```

4.26 CameraSetNoiseFilter

原型：

```
MVSDK_API CameraSdkStatus
CameraSetNoiseFilter
(
    CameraHandle          hCamera,
    BOOL                  bEnable
);
```

功能说明：设置图像降噪模块的使能状态。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

bEnable：TRUE，使能；FALSE，禁止。

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

Example：

```
```

4.27 CameraGetNoiseFilterState

原型：

```
MVSDK_API CameraSdkStatus
CameraGetNoiseFilterState
(
    CameraHandle        hCamera,
    BOOL                *pEnable
);
```

功能说明：获得图像降噪模块的使能状态。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pEnable：指标，返回状态。TRUE，为使能。

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

注意：

Example：

```
```

4.28 CameraInitRecord

原型：

```
MVSDK_API CameraSdkStatus
CameraInitRecord
(
    CameraHandle        hCamera,
    int                 iFormat,
    char                *pcSavePath,
    BOOL                b2GLimit,
    DWORD               dwQuality,
    int                 iFrameRate
);
```

功能说明：初始化一次录像。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iFormat：录像的格式，当前只支持不压缩和 MSCV 两种方式。0:不压缩；1:MSCV 方式压缩。

*pcSavePath：录像文件保存的路径。

b2GLimit：如果为 TRUE,则档大于 2G 时自动分割。

dwQuality：录像的质量因子，越大，则品质越好。范围 1 到 100.

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。

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

Example：

4.29 CameraStopRecord

原型：

```
MVSDK_API CameraSdkStatus
CameraStopRecord
(
    CameraHandle      hCamera
);
```

功能说明：结束本次录像。

参数说明：hCamera：相机的句柄，由 CameraInit 函数获得。

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

Example：

```

```

4.30 CameraPushFrame

原型：

```
MVSDK_API CameraSdkStatus
CameraPushFrame
(
    CameraHandle          hCamera,
    BYTE                  *pbyImageBuffer,
    tSdkFrameHead         *pFrInfo
);
```

功能说明：将一帧数据存入录像流中。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pbyImageBuffer：图像的数据缓冲区，必须是 RGB 格式。

*pFrInfo：图像的帧头信息。

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

注意：必须调用 CameraInitRecord 才能调用该函数。

CameraStopRecord 调用后，不能再调用该函数。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。

Example：

4.31 CameraSaveImage

原型：

```
MVSDK_API CameraSdkStatus
CameraSaveImage
(
    CameraHandle        hCamera,
    LPCTSTR             lpszFileName,
    BYTE                *pbyImageBuffer,
    tSdkFrameHead       *pFrInfo,
    BYTE                byFileType,
    BYTE                byQuality
);
```

功能说明：将图像缓冲区的数据保存成图片文件。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

lpszFileName：图片保存文件完整路径。

*pbyImageBuffer：图像的数据缓冲区。

*pFrInfo：图像的帧头信息。

byFileType：图像保存的格式。取值范围参见 CameraDefine.h 中 emSdkFileType 的类型定义。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 pbyImageBuffer 和 pFrInfo 是由 CameraGetImageBuffer 获得的数据，而且未经 CameraImageProcess 转换成 BMP 格式；反之，如果要保存成 BMP、JPG 或者 PNG 格式，则 pbyImageBuffer 和 pFrInfo 是由 CameraImageProcess 处理后的 RGB 格式数据。具体用法可以参考 Advanced 的例程。

byQuality：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。

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

注意：

Example：

```
```

4.32 CameraSetMirror

原型：

```
MVSDK_API CameraSdkStatus
CameraSetMirror
(
    CameraHandle        hCamera,
    INT                 iDir,
    BOOL                bEnable
);
```

功能说明：设置图像镜像操作。镜像操作分为水平和垂直两个方向。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iDir：表示要获得的镜像方向。0：水平方向；1：垂直方向。

bEnable：TRUE，使能镜像;FALSE，禁止镜像

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

Example：

```
```

4.33 CameraGetMirror

原型：

```
MVSDK_API CameraSdkStatus
CameraGetMirror
(
    CameraHandle     hCamera,
    INT              iDir,
    BOOL             *pbEnable
);
```

功能说明：获得图像的镜像状态。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iDir：表示要获得的镜像方向。0：水平方向；1：垂直方向。

*pbEnable：

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

注意：

Example：

4.34 CameraSetMonochrome

原型：

```
MVSDK_API CameraSdkStatus
CameraSetMonochrome
(
    CameraHandle        hCamera,
    BOOL                bEnable
);
```

功能说明：设置彩色转为黑白功能的使能。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

bEnable：TRUE，表示将彩色图像转为黑白。

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

Example：

4.35 CameraGetMonochrome

原型：

```
MVSDK_API CameraSdkStatus
CameraGetMonochrome
(
    CameraHandle        hCamera,
    BOOL                *pbEnable
```

```
);
```

功能说明：获得彩色转换黑白功能的使能状况。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*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。

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

Example：

```

```

4.36 CameraSetInverse

原型：

```
MVSDK_API CameraSdkStatus
CameraSetInverse
(
    CameraHandle        hCamera,
    BOOL                bEnable
);
```

功能说明：设置彩图像颜色翻转功能的使能。（负片）

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

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。

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

注意：

Example：

```

```

4.37 CameraGetInverse

原型：

```
MVSDK_API CameraSdkStatus
CameraGetInverse
(
    CameraHandle        hCamera,
    BOOL                *pbEnable
);
```

功能说明：获得图像颜色反转功能的使能状态。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pbEnable：指针，返回该功能使能状态。

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

注意：

Example：

```

```

4.38 CameraSetImageResolution

原型：

```
MVSDK_API CameraSdkStatus
CameraSetImageResolution
(
```

```
    CameraHandle            hCamera,
    tSdkImageResolution     *pImageResolution
);
```

功能说明：设置预览的分辨率。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pImageResolution：结构体指针，用于设置当前的分辨率。

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

Example：

4.39 CameraGetImageResolution

原型：

```
MVSDK_API CameraSdkStatus
CameraGetImageResolution
(
    CameraHandle            hCamera,
    tSdkImageResolution     *psCurImageResolution
);
```

功能说明：获得当前预览的分辨率。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*psCurImageResolution：结构体指针，用于返回当前的分辨率。

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

Example：

```

```

4.40 CameraGetMediaType

原型：

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

功能说明：获得相机当前输出原始数据的格式索引号。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piMediaType：指针，用于返回当前格式类型的索引号。由 CameraGetCapability 获得相机的属性，在 tSdkCameraCapbility 结构体中的 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。pMediaTypeDesc[*piMediaType].iMediaType 则表示当前格式的编码。该编码请参见 CameraDefine.h 中[图像格式定义]部分。

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

Example：

```

```

4.41 CameraSetMediaType

原型：

```
MVSDK_API CameraSdkStatus
CameraSetMediaType
(
    CameraHandle          hCamera,
    INT                   iMediaType
);
```

功能说明：设置相机当前输出原始数据的格式索引号。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piMediaType：指针，用于返回当前格式类型的索引号。由 CameraGetCapability 获得相机的属性，在 tSdkCameraCapbility 结构体中的 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。pMediaTypeDesc[*piMediaType].iMediaType 则表示当前格式的编码。该编码请参见 CameraDefine.h 中[图像格式定义]部分。

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

Example：

4.42 CameraSetAeState

原型：

```
MVSDK_API CameraSdkStatus
CameraSetAeState
```

```
(
    CameraHandle      hCamera,
    BOOL              bState
);
```

功能说明：设置相机曝光的模式。自动或者手动。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

bState：TRUE，使能自动曝光；FALSE，停止自动曝光。

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

Example：

```

```

4.43 CameraGetAeState

原型：

```
MVSDK_API CameraSdkStatus
CameraGetAeState
(
    CameraHandle          hCamera,
    BOOL                  *pbAeState
);
```

功能说明：获得相机当前的曝光模式。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pbAeState：指标，用于返回自动曝光的使能状态。

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

Example：

```

```

4.44 CameraSetAeTarget

原型：

```
MVSDK_API CameraSdkStatus
CameraSetAeTarget
(
    CameraHandle        hCamera,
    INT                 iAeTarget
);
```

功能说明：设定自动曝光的亮度目标值。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iAeTarget：亮度目标值。

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

注意：设定范围由 CameraGetCapability 函数获得。

Example：

```

```

4.45 CameraGetAeTarget

原型：

```
MVSDK_API CameraSdkStatus
CameraGetAeTarget
(
```

```
    CameraHandle        hCamera,
    INT                 *piAeTarget
);
```

功能说明：获得自动曝光的亮度目标值。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piAeTarget：指针，返回目标值。

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

Example：

```

```

4.46 CameraSetExposureTime

原型：

```
MVSDK_API CameraSdkStatus
CameraSetExposureTime
(
    CameraHandle        hCamera,
    double              fExposureTime
);
```

功能说明：设置曝光时间。单位为微秒。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

fExposureTime：曝光时间，单位微秒。

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

注意：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
CameraGetExposureTime 来获得实际设定的值。

Example：

4.47 CameraGetExposureTime

原型：

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

功能说明：获得曝光时间。单位为微秒。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

pfExposureTime：指针，返回曝光时间，单位微秒。

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

4.48 CameraGetExposureLineTime

原型：

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

功能说明：获得一行的曝光时间。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*puiLineTime：行曝光时间，单位微秒。

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

Example：

```
```

4.49 CameraSetAnalogGain

原型：

```
MVSDK_API CameraSdkStatus
CameraSetAnalogGain
(
    CameraHandle        hCamera,
    INT                 usAnalogGain
);
```

功能说明：设置相机的图像模拟增益值。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

usAnalogGain：设定的模拟增益值。

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

注意：该值乘以 CameraGetCapability 获得的相机属性结构体中 sExposeDesc.fAnalogGainStep ,就得到实际的图像信号放大倍数。

Example：

```
```

4.50 CameraGetAnalogGain

原型：

```
MVSDK_API CameraSdkStatus
CameraGetAnalogGain
(
    CameraHandle      hCamera,
    INT               *pusAnalogGain
);
```

功能说明：获得图像信号的模拟增益值。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pusAnalogGai：指针，返回当前的模拟增益值。

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

注意：

Example：

4.51 CameraSetAeWindow

原型：

```
MVSDK_API CameraSdkStatus
CameraSetAEWindow
(
    CameraHandle          hCamera,
     int                  iHOff,
```

```
    int             iVOff,
    int             iWidth,
    int             iHeight
);
```

功能说明：设置自动曝光的参考窗口

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iHOff：窗口左上角的横坐标

iVOff：窗口左上角的纵坐标

iWidth：窗口的宽度

iHeight：窗口的高度

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

注意：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

Example：

4.52 CameraGetAEWindow

原型：

```
MVSDK_API CameraSdkStatus
CameraGetAEWindow
(
    CameraHandle    hCamera,
    INT*            piHOff,
    INT*            piVOff,
```

```
    INT*                piWidth,
    INT*                piHeight
);
```

功能说明：获得自动曝光参考窗口的位置。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

piHOff：指针，返回窗口位置左上角横坐标值。

piVOff：指针，返回窗口位置左上角纵坐标值。

piWidth：指标，返回窗口的宽度。

piHeight：指标，返回窗口的高度。

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

注意：

Example：

```

```

4.53 CameraSetSharpness

原型：

```
MVSDK_API CameraSdkStatus
CameraSetSharpness
(
    CameraHandle        hCamera,
    INT                 iSharpness
);
```

功能说明：设置图像的处理的锐化参数。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iSharpness：锐化参数。范围由 CameraGetCapability 获得，一般是[0,100]，0 表示关闭锐化处理。

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

Example：

```
```

4.54 CameraGetSharpness

原型：

```
MVSDK_API CameraSdkStatus
CameraGetSharpness
(
    CameraHandle          hCamera,
    INT                   *piSharpness
);
```

功能说明：获取当前锐化设定值。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

* piSharpness：指针，返回当前设定的锐化的设定值。

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

Example：

```
```

4.55 CameraSetOnceWB

原型：

```
MVSDK_API CameraSdkStatus
CameraSetOnceWB
(
    CameraHandle        hCamera
);
```

功能说明：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。

参数说明：hCamera：相机的句柄，由 CameraInit 函数获得。

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

Example：

```

```

4.56 CameraSetOnceBB

原型：

```
MVSDK_API CameraSdkStatus
CameraSetOnceBB
(
    CameraHandle        hCamera
);
```

功能说明：执行一次黑平衡操作。

参数说明：hCamera：相机的句柄，由 CameraInit 函数获得。

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

Example：

4.57 CameraSetLutMode

原型：

```
MVSDK_API CameraSdkStatus
CameraSetLutMode
(
    CameraHandle          hCamera,
    INT                   emLutMode
);
```

功能说明：设置相机的查表变换模式 LUT 模式。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

emLutMode :LUTMODE_PARAM_GEN 表示由伽马和对比度参数动态生成 LUT 表。

LUTMODE_PRESET 表示使用预设的 LUT 表。

LUTMODE_USER_DEF 表示使用用户自定的 LUT 表。

LUTMODE_PARAM_GEN 的定义参考 CameraDefine.h 中 emSdkLutMode 类型。

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

Example：

4.58 CameraGetLutMode

原型：

```
MVSDK_API CameraSdkStatus
CameraGetLutMode
(
    CameraHandle        hCamera,
    INT                 *pemLutMode
);
```

功能说明：获得相机的查表变换模式 LUT 模式

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pemLutMode：指针，返回当前 LUT 模式。意义与 CameraSetLutMode 中 emLutMode 参数相同。

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

Example：

```

```

4.59 CameraSelectLutPreset

原型：

```
MVSDK_API CameraSdkStatus
CameraSelectLutPreset
(
    CameraHandle        hCamera,
    int                 iSel
);
```

功能说明：选择默认 LUT 模式下的 LUT 表。必须先使用 CameraSetLutMode

将 LUT 模式设置为默认模式。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iSel：表的索引号。表的个数由 CameraGetCapability 获得。

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

Example：

```
```

4.60 CameraGetLutPresetSel

原型：

```
MVSDK_API CameraSdkStatus
CameraGetLutPresetSel
(
    CameraHandle        hCamera,
    int*                piSel
);
```

功能说明：获得默认 LUT 模式下的 LUT 表索引号。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

piSel：指针，返回表的索引号。

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

Example：

```
```

4.61 CameraSetCustomLut

原型：

```
MVSDK_API CameraSdkStatus
CameraSetCustomLut
(
    CameraHandle        hCamera,
    BYTE*               pLut
);
```

功能说明：设置自定义的 LUT 表。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

pLut：指针，指向 LUT 表的地址。LUT 表为 256 字节大小，分别代码颜色通道从 0 到 256 对应的映射值。

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

注意：必须先使用 CameraSetLutMode 将 LUT 模式设置为自定义模式。

Example：

4.62 CameraGetCustomLut

原型：

```
MVSDK_API CameraSdkStatus
CameraGetCustomLut
(
    CameraHandle        hCamera,
    BYTE*               pLut
);
```

功能说明：获得当前使用的自定义 LUT 表。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

pLut :指针 ,用于返回当前使用的自定义 LUT 表。LUT 表为 256 字节大小 ,分别代码颜色通道从 0 到 256 对应的映射值。自定义 LUT 表的模式值是 0 到 255。表示 1:1 线性映像，不改变原有的值。

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

注意：

Example：

4.63 CameraGetCurrentLut

原型：

```
MVSDK_API CameraSdkStatus
CameraGetCurrentLut
(
    CameraHandle        hCamera,
    BYTE*               pLut
);
```

功能说明：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。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

pLut：指针，用于返回相机当前 LUT 表。

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

Example：

```

```

4.64 CameraSetWbMode

原型：

```
MVSDK_API CameraSdkStatus
CameraSetWbMode
(
    CameraHandle        hCamera,
    BOOL                bAuto
);
```

功能说明：设置相机白平衡模式。分为手动和自动两种方式。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

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。

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

Example：

```

```

4.65 CameraGetWbMode

原型：

```
MVSDK_API CameraSdkStatus
CameraGetWbMode
```

```
(
    CameraHandle      hCamera,
    BOOL              *pbAuto
);
```

功能说明：获得当期的白平衡模式。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pbAuto：指针，返回 TRUE 表示自动模式，FALSE 为手动模式。

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

Example：

4.66 CameraIsWbWinVisible

原型：

```
MVSDK_API CameraSdkStatus
CameraIsWbWinVisible
(
    CameraHandle          hCamera,
    BOOL                  *pbShow
);
```

功能说明：获得白平衡窗口的显示状态。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pbShow：指针，返回 TRUE，则表示窗口是可见的。

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

Example：

4.67 CameraGetWbWindow

原型：

```
MVSDK_API CameraSdkStatus
CameraGetWbWindow
(
    CameraHandle     hCamera,
    INT*             PiHOff,
    INT*             PiVOff,
    INT*             PiWidth,
    INT*             PiHeight
);
```

功能说明：获得白平衡参考窗口的位置。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

PiHOff：指标，返回参考窗口的左上角横坐标。

PiVOff：指标，返回参考窗口的左上角纵坐标。

PiWidth：指标，返回参考窗口的宽度。

PiHeight：指标，返回参考窗口的高度。

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

Example：

4.68 CameraSetWbWindow

原型：

```
MVSDK_API CameraSdkStatus
CameraSetWbWindow
(
    CameraHandle     hCamera,
    INT              iHOff,
    INT              iVOff,
    INT              iWidth,
    INT              iHeight
);
```

功能说明：设置白平衡参考窗口的位置。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iHOff：参考窗口的左上角横坐标

iVOff：参考窗口的左上角纵坐标

iWidth：参考窗口的宽度

iHeight：参考窗口的高度

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

注意：

Example：

4.69 CameraImageOverlay

原型：

```
MVSDK_API CameraSdkStatus
```

```
CameraImageOverlay
(
    CameraHandle      hCamera,
    BYTE              *pRgbBuffer,
    tSdkFrameHead     *pFrInfo
);
```

功能说明：将输入的图像数据上迭加十字线、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迭加上。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pRgbBuffer：图像数据缓冲区。

*pFrInfo：图像的帧头信息。

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

Example：

```

```

4.70 CameraSetCrossLine

原型：

```
MVSDK_API CameraSdkStatus
CameraSetCrossLine
(
    CameraHandle      hCamera,
    int               iLine,
    INT               x,
    INT               y,
    UINT              color,
    BOOL              bVisible
```

```
);
```

功能说明：设置指定十字线的参数。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iLine：表示要设置第几条十字线的状态。范围为[0,8]，共 9 条。

x：十字线中心位置的横坐标值。

y：十字线中心位置的纵坐标值。

color：十字线的颜色，格式为(R|(G<<8)|(B<<16))

bVisible：十字线的显示状态。TRUE，表示显示。

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

注意：只有设置为显示状态的十字线，在调用 CameraImageOverlay 后才会被迭加到图像上。

Example：

```

```

4.71 CameraGetCrossLine

原型：

```
MVSDK_API CameraSdkStatus
CameraGetCrossLine
(
    CameraHandle     hCamera,
    int              iLine,
    INT              *px,
    INT              *py,
    UINT             *pcolor,
    BOOL             *pbVisible
);
```

功能说明：获得指定十字线的状态。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iLine：表示要获取的第几条十字线的状态。范围为[0,8]，共 9 条。

*px：指针，返回该十字线中心位置的横坐标。

*py：指针，返回该十字线中心位置的横坐标。

*pcolor：指针，返回该十字线的颜色，格式为(R|(G<<8)|(B<<16))。

*pbVisible：指针，返回 TRUE，则表示该十字线可见。

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

Example：

```

```

4.72 CameraSetGain

原型：

```
MVSDK_API CameraSdkStatus
CameraSetGain
(
    CameraHandle    hCamera,
    int             iRGain,
    int             iGGain,
    int             iBGain
);
```

功能说明：设置图像的数字增益。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iRGain：红色通道的增益值。

iGGain：绿色通道的增益值。

iBGain：蓝色通道的增益值。

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

注意：设定范围由 CameraGetCapability 获得的相机属性结构体中 sRgbGainRange 成员表述。实际的放大倍数是设定值/100。

Example：

4.73 CameraGetGain

原型：

```
MVSDK_API CameraSdkStatus
CameraGetGain
(
    CameraHandle    hCamera,
    int             *piRGain,
    int             *piGGain,
    int             *piBGain
);
```

功能说明：获得图像处理的数字增益

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piRGain：指针，返回红色信道的数字增益值。

*piGGain：指针，返回绿色信道的数字增益值。

*piBGain：指针，返回蓝色信道的数字增益值。

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

Example：

4.74 CameraSetGamma

原型：

```
MVSDK_API CameraSdkStatus
CameraSetGamma
(
    CameraHandle      hCamera,
    int               iGamma
);
```

功能说明：设定 LUT 动态生成模式下的 Gamma 值。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iGamma：要设定的 Gamma 值。

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

注意：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。

Example：

4.75 CameraGetGamma

原型：

```
MVSDK_API CameraSdkStatus
CameraGetGamma
```

```
(
    CameraHandle      hCamera,
    int               *piGamma
);
```

功能说明：获得 LUT 动态生成模式下的 Gamma 值。请参考 CameraSetGamma 函数的功能描述。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piGamma：指针，返回当前的 Gamma 值。

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

注意：

Example：

```

```

4.76 CameraSetSaturation

原型：

```
MVSDK_API CameraSdkStatus
CameraSetSaturation
(
    CameraHandle      hCamera,
    INT               iSaturation
);
```

功能说明：设定图像处理的饱和度。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iSaturation：设定的饱和度值。

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

注意：对黑白相机无效。

Example：

```

```

4.77 CameraGetSaturation

原型：

```
MVSDK_API CameraSdkStatus
CameraGetSaturation
(
    CameraHandle        hCamera,
    INT                 *piSaturation
);
```

功能说明：获得图像处理的饱和度。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piSaturation：指针，返回当前图像处理的饱和度值。

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

Example：

```

```

4.78 CameraSetContrast

原型：

```
MVSDK_API CameraSdkStatus
CameraSetContrast
(
    CameraHandle      hCamera,
    INT               iContrast
);
```

功能说明：设定 LUT 动态生成模式下的对比度值。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iContrast：设定的对比度值。

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

注意：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。

Example：

```

```

4.79 CameraGetContrast

原型：

```
MVSDK_API CameraSdkStatus
CameraGetContrast
(
    CameraHandle      hCamera,
    INT               *piContrast
);
```

功能说明：获得 LUT 动态生成模式下的对比度值。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piContrast：指针，返回当前的对比度值。

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

Example：

```
```

4.80 CameraSetFrameSpeed

原型：

```
MVSDK_API CameraSdkStatus
CameraSetFrameSpeed
(
    CameraHandle      hCamera,
    INT               iFrameSpeedSel
);
```

功能说明：设定相机输出图像的帧率。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iFrameSpeedSel：选择的帧率模式索引号，范围从 0 到 CameraGetCapability 获得的信息结构体中 iFrameSpeedDesc - 1

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

Example：

```
```

4.81 CameraGetFrameSpeed

原型：

```
MVSDK_API CameraSdkStatus
CameraGetFrameSpeed
(
    CameraHandle        hCamera,
    INT                 *piFrameSpeedSel
);
```

功能说明：获得相机输出图像的帧率选择索引号。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piFrameSpeedSel：指针，返回选择的帧率模式索引号。

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

注意：

Example：

4.82 CameraSetAntiFlick

原型：

```
MVSDK_API CameraSdkStatus
CameraSetAntiFlick
(
    CameraHandle        hCamera,
    BOOL                bEnable
);
```

功能说明：设置自动曝光时抗频闪功能的使能状态。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

bEnable：TRUE，开启抗频闪功能;FALSE，关闭该功能。

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

注意：对于手动曝光模式下无效。

Example：

4.83 CameraGetAntiFlick

原型：

```
MVSDK_API CameraSdkStatus
CameraGetAntiFlick
(
    CameraHandle      hCamera,
    BOOL              *pbEnable
);
```

功能说明：获得自动曝光时抗频闪功能的使能状态。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pbEnable：指针，返回该功能的使能状态。

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

Example：

4.84 CameraGetLightFrequency

原型：

```
MVSDK_API CameraSdkStatus
CameraGetLightFrequency
(
    CameraHandle        hCamera,
    int                 *piFrequencySel
);
```

功能说明：获得自动曝光时，消频闪的频率选择。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piFrequencySel：指针，返回选择的索引号。0:50HZ 1:60HZ

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

Example：

```

```

4.85 CameraSetLightFrequency

原型：

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

功能说明：设置自动曝光时消频闪的频率。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iFrequencySel：0:50HZ , 1:60HZ

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

Example：

4.86 CameraSetTransPackLen

原型：

```
MVSDK_API CameraSdkStatus
CameraSetTransPackLen
(
    CameraHandle        hCamera,
    INT                 iPackSel
);
```

功能说明：设置相机传输图像数据的分包大小。(已经废弃该方式，SDK 中已经改为自动协商分包方式，使用 GIGE 相机时，请使能网卡的巨帧传输功能，使能后，SDK 将自动启用最大分包来优化性能)

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

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 pPackLenDesc 成员表述，iPackLenDesc 成员则表示最大可选的分包模式个数。

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

Example：

4.87 CameraGetTransPackLen

原型：

```
MVSDK_API CameraSdkStatus
CameraGetTransPackLen
(
    CameraHandle        hCamera,
    INT                 *piPackSel
);
```

功能说明：获得相机当前传输分包大小的选择索引号。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piPackSel：指针，返回当前选择的分包大小索引号。

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

Example：

4.88 CameraWriteSN

原型：

```
MVSDK_API CameraSdkStatus
CameraWriteSN
(
    CameraHandle        hCamera,
    BYTE                *pcSN,
    INT                 iLevel
);
```

功能说明：设置相机的序号。我公司相机序号分为 3 级。0 级出厂默认的相机序号，1 级和 2 级留给二次开发使用。每级序号长度都是 32 个字节。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pcSN：序号的缓冲区。

iLevel：要设定的序号级别，只能是 1 或者 2。

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

注意：

Example：

```
```

4.89 CameraReadSN

原型：

```
MVSDK_API CameraSdkStatus
CameraReadSN
(
    CameraHandle     hCamera,
    BYTE             *pcSN,
    INT              iLevel
);
```

功能说明：读取相机指定级别的序号

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pcSN：序号的缓冲区。

iLevel：要读取的序号级别。只能是 1 和 2。

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

Example：

4.90 CameraSaveParameter

原型：

```
MVSDK_API CameraSdkStatus
CameraSaveParameter
(
    CameraHandle          hCamera,
    INT                   iTeam
);
```

功能说明：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。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iTeam：PARAMETER_TEAM_A 保存到 A 组中,
PARAMETER_TEAM_B 保存到 B 组中,
PARAMETER_TEAM_C 保存到 C 组中,
PARAMETER_TEAM_D 保存到 D 组中

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

Example：

4.91 CameraLoadParameter

原型：

```
MVSDK_API CameraSdkStatus
CameraLoadParameter
(
    CameraHandle        hCamera,
    INT                 iTeam
);
```

功能说明：加载指定组的参数到相机中。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iTeam：PARAMETER_TEAM_A 保存到 A 组中,
PARAMETER_TEAM_B 保存到 B 组中,
PARAMETER_TEAM_C 保存到 C 组中,
PARAMETER_TEAM_D 保存到 D 组中

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

Example：

```

```

4.92 CameraReadParameterFromFile

原型：

```
MVSDK_API CameraSdkStatus
CameraReadParameterFromFile
(
    CameraHandle        hCamera,
    char                *sFileName
);
```

功能说明：从 PC 上指定的参数档中加载参数。我公司相机参数，保存在 PC 上为.config 后缀的档，位于安装下的 Camera\Configs 文件夹中。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*sFileName：参数文件的完整路径。

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

Example：

```
```

4.93 CameraGetCurrentParameterGroup

原型：

```
MVSDK_API CameraSdkStatus
CameraGetCurrentParameterGroup
(
    CameraHandle          hCamera,
    INT                   *piTeam
);
```

功能说明：获得当前选择的参数组。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piTeam：指针，返回当前选择的参数组。

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

Example：

```
```

4.94 CameraEnumerateDevice

原型：

```
MVSDK_API CameraSdkStatus
CameraEnumerateDevice
(
    tSdkCameraDevInfo        *pDSCameraList,
    INT                      *piNums
);
```

功能说明：枚举设备，并建立设备清单。

参数说明：

* pDSCameraList：设备列表数组指针。

*piNums ：设备的个数指标，调用时传入 pDSCameraList 数组的元素个数，函数返回时，保存实际找到的设备个数。

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

注意：在调用 CameraInit 之前，必须调用该函数来获得设备的信息。piNums 指向的值必须初始化，且不超过 pDSCameraList 数组元素个数，否则有可能造成内存溢出。

Example：

4.95 CameraGetCapability

原型：

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

功能说明：获得相机的特性描述结构体。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pCameraInfo：指针，返回该相机特性描述的结构体。tSdkCameraCapbility 在 CameraDefine.h 中定义。

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

Example：

```

```

4.96 CameraSetTriggerCount

原型：

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

功能说明：设置触发模式下的触发帧数。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iCount：一次触发采集的帧数。

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

注意：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。

Example：

4.97 CameraGetTriggerCount

原型：

```
MVSDK_API CameraSdkStatus
CameraGetTriggerCount
(
    CameraHandle        hCamera,
    INT                 *piCount
);
```

功能说明：获得一次触发的帧数。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piCount：指针，返回一次触发采集的帧数。

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

Example：

4.98 CameraGetTriggerDelayTime

原型：

```
MVSDK_API CameraSdkStatus
CameraGetTriggerDelayTime
(
    CameraHandle        hCamera,
    UINT                *puDelayTimeUs
);
```

功能说明：获得当前设定的硬触发延时时间。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*puDelayTimeUs：指针，返回延时时间，单位微秒。

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

注意：仅部分型号的相机支持该功能。

Example：

4.99 CameraSetTriggerDelayTime

原型：

```
MVSDK_API CameraSdkStatus
CameraSetTriggerDelayTime
(
    CameraHandle        hCamera,
    UINT                uDelayTimeUs
);
```

功能说明：设置硬件触发模式下的触发延时时间，单位微秒。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

uDelayTimeUs：触发延时。单位微秒。

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

注意：仅部分型号的相机支持该功能。

Example：

```
```

4.100 CameraSoftTriggerEx

原型：

```
MVSDK_API CameraSdkStatus
CameraSoftTriggerEx
(
    CameraHandle     hCamera,
    int              iParam
);
```

功能说明：执行一次软触发，之行前，会自动清除历史缓存图像，让相机重新拍照，获得一张最新的图像。

参数说明： hCamera：相机的句柄，由 CameraInit 函数获得。

iParam： 可选参数，目前版本中，固定填 1 即可。

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

Example：

```
```

4.101 CameraSetTriggerMode

原型：

```
MVSDK_API CameraSdkStatus
CameraSetTriggerMode
(
    CameraHandle      hCamera,
    INT               iTriggerModeSel
);
```

功能说明：设置相机的触发模式。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iTriggerModeSel：模式选择索引号。可设定的模式由 CameraGetCapability 函数获取。请参考 CameraDefine.h 中 tSdkCameraCapbility 的定义。一般情况，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。

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

Example：

```

```

4.102 CameraGetTriggerMode

原型：

```
MVSDK_API CameraSdkStatus
CameraGetTriggerMode
(
    CameraHandle          hCamera,
    INT                   *piTriggerMode
);
```

功能说明：获得相机的触发模式

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piTriggerMode：指针，返回当前选择的相机触发模式的索引号。

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

Example：

```

```

4.103 CameraSnapToBuffer

原型：

```
MVSDK_API CameraSdkStatus
CameraSnapToBuffer
(
    CameraHandle        hCamera,
    tSdkFrameHead       *pFrameInfo,
    BYTE                **pbyBuffer,
    UINT                uWaitTimeMs
);
```

功能说明：抓拍一张图像到缓冲区中。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pFrameInfo：指针，返回图像的帧头信息。

**pbyBuffer：指向指针的指针，用来返回图像缓冲区的地址。

uWaitTimeMs：超时时间，单位毫秒。

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

Example：

```

```

4.104 CameraGetResolutionForSnap

原型：

```
MVSDK_API CameraSdkStatus
CameraGetResolutionForSnap
(
    CameraHandle            hCamera,
    tSdkImageResolution     *pImageResolution
);
```

功能说明：获得抓拍模式下的分辨率选择索引号。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pImageResolution：指针，返回抓拍模式的分辨率。

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

Example：

```

```

4.105 CameraSetResolutionForSnap

原型：

```
MVSDK_API CameraSdkStatus
CameraSetResolutionForSnap
(
```

```
    CameraHandle            hCamera,
    tSdkImageResolution     *pImageResolution
);
```

功能说明：设置抓拍模式下相机输出图像的分辨率。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*pImageResolution：如果 pImageResolution->iWidth 和 pImageResolution->iHeight 都为 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。

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

Example：

```

```

4.106 CameraSetParameterTarget

原型：

```
MVSDK_API CameraSdkStatus
CameraSetParameterTarget
(
    CameraHandle        hCamera,
    int                 iTarget
);
```

功能说明：设定参数存取的目标对象。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

iTarget：参数存取的对象。PARAM_ON_PC 表示从 PC 上读写参数保存档，PARAM_ON_DEVICE从相机中读写参数档。参考 CameraDefine.h 中 emSdkParamTarget 的类型定义。

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

Example：

4.107 CameraGetParameterTarget

原型：

```
MVSDK_API CameraSdkStatus
CameraGetParameterTarget
(
    CameraHandle     hCamera,
    int              *piTarget
);
```

功能说明：获取参数存取的目标对象。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

*piTarget：指针，返回参数存取的对象。

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

注意：

Example：

4.108 CameraSetParameterMask

原型：

```
MVSDK_API CameraSdkStatus
CameraSetParameterMask
(
    CameraHandle        hCamera,
    UINT                uMask
);
```

功能说明：设置参数存取的屏蔽。参数加载和保存时会根据该屏蔽来决定各个模块参数的是否加载或者保存。

参数说明：

hCamera：相机的句柄，由 CameraInit 函数获得。

uMask：屏蔽。参考 CameraDefine.h 中 emSdkPropSheetMask 类型定义。

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

Example：

```

```

4.109 CameraRstTimeStamp

原型：

```
MVSDK_API CameraSdkStatus
CameraRstTimeStamp
(
    CameraHandle        hCamera
);
```

功能说明：复位图像采集的时间戳，从 0 开始。

参数说明： hCamera：相机的句柄，由 CameraInit 函数获得。

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

Example：

4.110 CameraSaveUserData

原型：

```
MVSDK_API CameraSdkStatus
CameraSaveUserData
(
    CameraHandle      hCamera,
    UINT              uStartAddr,
    BYTE              *pbData,
    int               ilen
);
```

功能说明：将用户自定义的数据保存到相机的非易性内存中。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。

参数说明：hCamera 相机的句柄，由 CameraInit 函数获得。
uStartAddr 起始地址，从 0 开始，注意，地址必须 64 字节对齐。
pbData 数据缓冲区指针，返回读到的数据。
ilen 读取数据的长度，ilen + uStartAddr 必须小于用户区最大长度

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

Example：

4.111 CameraLoadUserData

原型：

```
MVSDK_API CameraSdkStatus
CameraLoadUserData
(
    CameraHandle      hCamera,
    UINT              uStartAddr,
    BYTE              *pbData,
    int               ilen
);
```

功能说明：从相机的非易性内存中读取用户自定义的数据。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。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。
uStartAddr　起始地址，从 0 开始，注意，地址必须 64 字节对齐。
pbData　　数据缓冲区指针，返回读到的数据。
ilen　　读取数据的长度，ilen + uStartAddr 必须小于用户区最大长度

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

Example：

```

```

4.112 CameraGetFriendlyName

原型：

```
MVSDK_API CameraSdkStatus
CameraGetFriendlyName
(
    CameraHandle      hCamera,
    char *            pName
);
```

功能说明：读取相机的昵称，相机昵称默认是自动生成的，以型号名+#0,1,2 的方式，例如 Camera MV-U500#0 和 Camera MV-U500#1，表示同一个电脑上接的 2 个 500 万像素相机，如需单独设置某个相机的昵称，可以调用 CameraSetFriendlyName 函数,例如将其中第一个

相机改为 My Camera 1,那么 2 个 MV-U500 的名字就分别是 My Camera 1 和 Camera MV-U500#1。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。

pName　　返回相机昵称的字符串缓冲区首地址，缓冲区大小需要大于 32 个字节

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

Example：

```
```

4.113 CameraSetFriendlyName

原型：

```
MVSDK_API CameraSdkStatus
CameraSetFriendlyName
(
    CameraHandle      hCamera,
    char*             pName
);
```

功能说明：设置某个相机的昵称，设置成功后，昵称固化在相机内，该功能可方便的区分多相机。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。

pName　　指向要设置的相机昵称的字符串缓冲区首地址，昵称必须小于 31 个字符，设置成字符串“auto”,或者"自动生成"则表示使用默认的昵称。

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

Example：

```
```

4.114 CameraSdkGetVersionString

原型：

```
MVSDK_API CameraSdkStatus
CameraSdkGetVersionString
(
    char*           pVersionString
);
```

功能说明：从相机的非易性内存中读取用户自定义的数据。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。

参数说明：pVersionString　返回 SDK 版本字符串，缓冲区需长度大于 32 个字。

返回值：返回 SDK 版本描述字符串。

Example：

```

```

4.115 CameraGetEnumInfo

原型：

```
MVSDK_API CameraSdkStatus
CameraGetEnumInfo
(
    CameraHandle            hCamera,
    tSdkCameraDevInfo*      pCameraInfo
);
```

功能说明：获得指定设备的枚举信息。

参数说明：hCamera　相机的句柄，由 CameraInit 函数获得。
pCameraInfo 指针，返回设备的枚举信息。

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

Example：

4.116 CameraSetIOState

原型：

```
MVSDK_API CameraSdkStatus
CameraSetIOState
(
    CameraHandle      hCamera,
    INT               iOutputIOIndex,
    UINT              uState
);
```

功能说明：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 中 iOutputIoCounts 决定。

参数说明：hCamera 相机的句柄，由 CameraInit 函数获得。
iOutputIOIndex IO 的索引号，从 0 开始。
uState 要设定的状态，1 为高，0 为低

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

Example：

4.117 CameraGetIOState

原型：

```
MVSDK_API CameraSdkStatus
CameraGetIOState
(
    CameraHandle      hCamera,
    INT               iInputIOIndex,
    UINT*             puState
);
```

功能说明：设置指定 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 中 iInputIoCounts 决定。

参数说明：hCamera 相机的句柄，由 CameraInit 函数获得。
iInputIOIndex IO 的索引号，从 0 开始。
puState 指针，返回 IO 状态,1 为高，0 为低。

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

Example：

4.118 CameraSetIspOutFormat

原型：

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

功能说明：设置 CameraImageProcess 函数的图像处理的输出格式，支持 CAMERA_MEDIA_TYPE_MONO8 和 CAMERA_MEDIA_TYPE_RGB8、CAMERA_MEDIA_TYPE_RGBA8、CAMERA_MEDIA_TYPE_BGR8、CAMERA_MEDIA_TYPE_BGRA8 (在 CameraDefine.h 中定义)五种，分别对应 8 位灰度图像和 24 位 RGB、32 位 RGBA、24 位 BGR、32 位 BGRA 格式，二次开发时可根据视觉库的需要来选择输出格式 ,例如 OPENCV 下需要 BGR 格式，而用 QT 开发，则需要 RGB 格式。

参数说明：hCamera 相机的句柄，由 CameraInit 函数获得。
uFormat 要设定格式，可以是 CAMERA_MEDIA_TYPE_MONO8 和 CAMERA_MEDIA_TYPE_RGB8、CAMERA_MEDIA_TYPE_RGBA8、CAMERA_MEDIA_TYPE_BGR8、

CAMERA_MEDIA_TYPE_BGRA8 其中之一。

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

Example：

```
```

4.119 CameraGetIspOutFormat

原型：

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

功能说明：获得 CameraGetImageBuffer 函数图像处理的输出格式，该格式可由 CameraSetIspOutFormat 进行设置。

参数说明：hCamera 相机的句柄，由 CameraInit 函数获得。

puFormat 返回当前设定的格式，其值是 CAMERA_MEDIA_TYPE_MONO8 和 CAMERA_MEDIA_TYPE_RGB8、CAMERA_MEDIA_TYPE_RGBA8、CAMERA_MEDIA_TYPE_BGR8、CAMERA_MEDIA_TYPE_BGRA8 其中之一。

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

Example：

```
```

4.120 CameraGetErrorString

原型：

```
MVSDK_API    char*
CameraGetErrorString
(
CameraSdkStatus        iStatusCode
);
```

功能说明：获得错误码对应的描述字符串。

参数说明：iStatusCode　　错误码。(定义于 CameraStatus.h 中)

返回值：成功时，输入错误码对应的字符串首地址。

Example：

```

```

4.121 CameraReConnect

原型：

```
MVSDK_API CameraSdkStatus
CameraReConnect
(
    CameraHandle      hCamera,
);
```

功能说明：重新连接设备，用于 USB、GIGE 设备意外掉线后重连。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。

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

Example：

```

```

4.122 CameraConnectTest

原型：

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

```
);
```

功能说明：测试相机的连接状态，用于检测相机是否掉线。

参数说明：hCamera 相机的句柄，由 CameraInit 函数获得。

返回值：成功时，返回 CAMERA_STATUS_SUCCESS (0)，表示未掉线;否则返回非 0 值的错误码,请参考 CameraStatus.h 中错误码的定义。

Example：

```

```

4.123 CameraSetTriggerDelayTime

原型：

```
MVSDK_API CameraSdkStatus
CameraSetTriggerDelayTime
(
    CameraHandle      hCamera,
    UINT              uDelayTimeUs
);
```

功能说明：设置硬件触发模式下的触发延时时间，单位微秒。当硬触发信号来临后，经过指定的延时，再开始采集图像。仅部分型号的相机支持该功能。具体请查看产品说明书。

参数说明：hCamera 相机的句柄，由 CameraInit 函数获得。
uDelayTimeUs 硬触发延时。单位微秒。

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

Example：

```

```

4.124 CameraGetTriggerDelayTime

原型：

```
MVSDK_API CameraSdkStatus
CameraGetTriggerDelayTime
```

```
(
    CameraHandle      hCamera,
    UINT*             puDelayTimeUs
);
```

功能说明：获得当前设定的硬触发延时时间。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。
puDelayTimeUs　指针，返回延时时间，单位微秒。

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

Example：

```

```

4.125 CameraSetStrobeMode

原型：

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

功能说明：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。

参数说明：hCamera　相机的句柄，由 CameraInit 函数获得。
iMode　当为 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 (CameraSetStrobePolarity)。
当为 STROBE_SYNC_WITH_TRIG_MANUAL 时，和触发信号同步，触发后，STROBE 延时指定的时间后 (CameraSetStrobeDelayTime)，再持续指定时间的脉冲 (CameraSetStrobePulseWidth),有效极性可设置(CameraSetStrobePolarity)。
当为 STROBE_ALWAYS_HIGH 时，STROBE 信号恒

为高,忽略其他设置

当为 STROBE_ALWAYS_LOW 时，STROBE 信号恒为低,忽略其他设置

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

Example：

4.126 CameraGetStrobeMode

原型：

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

功能说明：或者当前 STROBE 信号设置的模式。

参数说明：hCamera 相机的句柄，由 CameraInit 函数获得。

piMode 指针，返回 STROBE_SYNC_WITH_TRIG_AUTO, STROBE_SYNC_WITH_TRIG_MANUAL、STROBE_ALWAYS_HIGH 或者 STROBE_ALWAYS_LOW。

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

Example：

4.127 CameraSetStrobeDelayTime

原型：

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

功能说明：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。。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。

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。

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

Example：

```

```

4.128 CameraGetStrobeDelayTime

原型：

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

功能说明：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。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。

upDelayTimeUs　　指针，返回延时时间，单位 us。

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

Example：

4.129 CameraSetStrobePulseWidth

原型：

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

功能说明：当 STROBE 信号处于 STROBE_SYNC_WITH_TRIG 时，通过该函数设置其脉冲宽度。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。
uTimeUs　　脉冲的宽度，单位为时间 us。

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

Example：

4.130 CameraGetStrobePulseWidth

原型：

```
MVSDK_API CameraSdkStatus
CameraGetStrobePulseWidth
(
    CameraHandle      hCamera,
    UINT*             upTimeUs
);
```

功能说明：当 STROBE 信号处于 STROBE_SYNC_WITH_TRIG 时，通过该函数获得其脉冲宽度。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。
upTimeUs　指针，返回脉冲宽度。单位为时间 us。

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

Example：

```
```

4.131 CameraSetStrobePolarity

原型：

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

功能说明：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。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。

iPolarity　　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。

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

Example：

```
```

4.132 CameraGetStrobePolarity

原型：

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

```
    INT*                    upPolarity
);
```

功能说明：获得相机当前 STROBE 信号的有效极性。默认为高电平有效。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。

ipPolarity　　指针，返回 STROBE 信号当前的有效极性。

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

Example：

4.133 CameraSetExtTrigSignalType

原型：

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

功能说明：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。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。

iType　　外触发信号种类，返回值参考 CameraDefine.h 中 emExtTrigSignal 类型定义。

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

Example：

4.134 CameraGetExtTrigSignalType

原型：

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

功能说明：获得相机当前外触发信号的种类。。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。
ipType　　指针，返回外触发信号种类，返回值参考 CameraDefine.h 中 emExtTrigSignal 类型定义。

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

Example：

```

```

4.135 CameraSetExtTrigDelayTime

原型：

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

功能说明：设置外触发信号延时时间，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。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。
uDelayTimeUs　　延时时间，单位为微秒，默认为 0.

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

Example：

4.136 CameraGetExtTrigDelayTime

原型：

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

功能说明：获得设置的外触发信号延时时间，默认为 0，单位为微秒。

参数说明：hCamera 相机的句柄，由 CameraInit 函数获得。

upDelayTimeUs 指针，返回延时时间。

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

Example：

4.137 CameraSetExtTrigJitterTime

原型：

```
MVSDK_API CameraSdkStatus
CameraSetExtTrigJitterTime
(
    CameraHandle      hCamera,
    UINT              uTimeUs
```

```
);
```

功能说明：设置相机外触发信号的消抖时间，只有当外触发信号模式选择高电平或者低电平触发时，去抖时间才会生效。默认为 0，单位为微秒，最大 150 毫秒。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。
uTimeUs　　单位为微秒。

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

Example：

```

```

4.138 CameraGetExtTrigJitterTime

原型：

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

功能说明：获得设置的相机外触发消抖时间，默认为 0.单位为微秒。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。
upTimeUs　　指针，返回设置的消抖时间。

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

Example：

```

```

4.139 CameraGetExtTrigCapability

原型：

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

功能说明：获得相机外触发的属性掩码。

参数说明：hCamera　　相机的句柄，由 CameraInit 函数获得。
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。

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

4.140 CameraAlignMalloc

原型：

```
MVSDK_API BYTE*
CameraAlignMalloc
(
    int               size,
    int               align
);
```

功能说明：申请地址对齐的内存块。用于配合硬件加速算法，需要使用对齐的内存块。

参数说明：size　内存大小
align　地址对齐，如果需要 16 字节对齐，则设置为 16

返回值：成功时，返回申请到的内存首地址。失败时，返回 NULL。

4.141 CameraAlignFree

原型：

```
MVSDK_API VOID
CameraAlignFree
(
    BYTE*          membuffer
);
```

功能说明：释放由 CameraAlignMalloc 申请地址对齐的内存块。

参数说明：membuffer 由 **CameraAlignMalloc 返回的内存地址。**

返回值：无。

5 SDK 接口函数按功能分类解释

为了方便用户快速的找到某个功能的开发方式 ,我们对一些常用的方法进行了总结。

5.1 相机初始化与反初始化

对于相机的工作流程，我们采用 先枚举、初始化，然后退出程序前反初始化相机。

- 相机枚举我们提供了 2 个函数，CameraEnumerateDevice 和 CameraEnumerateDeviceEx。CameraEnumerateDevice 会返回相机的个数和详细的描述信息，其中描述信息以结构体数组的方式返回，比如相机的名称、型号、序列号、接口类型等都会详细列出；CameraEnumerateDeviceEx 则仅仅只返回相机的个数。
- 相机的初始化我们提供了 CameraInit、CameraInitEx、CameraInitEx2 这三个函数。CameraInit 需要和 CameraEnumerateDevice 配套使用，而 CameraInitEx 和 CameraInitEx2 则需要和 CameraEnumerateDeviceEx 配套使用。CameraInit 需要传入由 CameraEnumerateDevice 得到的相机结构体信息；CameraInitEx 只要传入相机的序号 ID 即可，例如初始化第 1 个相机，传入 0，第二个相机传入 1，以此类推；CameraInitEx2 则是传入相机的名称，例如可以通过我们提供的工具，事先将相机名称改为"Camera1"，CameraInitEx2 调用时，传入"Camera1"即可，对于多相机同时工作的

案例，这个方法可以有效的建立一一对应的关系，但是要注意，每个相机的名称必须改成不同的，确保唯一性。

- 反初始化函数， CameraUnInit。无论采取哪种初始化方式，反初始化调用 CameraUnInit 即可。

以下是采用 CameraEnumerateDevice 和 CameraInit 典型的程序流程：

```
tSdkCameraDevInfo sCameraList[10];
int    iCameraNums;
int    status;
iCameraNums = 10;
//枚举相机，最多返回10个相机的描述信息
    if (CameraEnumerateDevice(sCameraList,&iCameraNums) !=
CAMERA_STATUS_SUCCESS || iCameraNums == 0)
    {
        return FALSE;
    }
//如果只有一个相机，iCameraNums会被CameraEnumerateDevice内部修改为1。
    if ((status = CameraInit(&sCameraList[0],-1,-1,&m_hCamera)) !=
CAMERA_STATUS_SUCCESS)
    {
        return FALSE;
```

```
    }
```

//程序退出前调用

CameraUnInit(m_hCamera);

以下是采用 CameraEnumerateDeviceEx 和 CameraInitEx 典型的程序流程：

```
int    iCameraNums;
iCameraNums = CameraEnumerateDeviceEx();
if (iCameraNums == 0) return FALSE;

    if ((status = CameraInitEx(0,-1,-1,&m_hCamera)) != CAMERA_STATUS_SUCCESS)
    {
        return FALSE;
    }
```

//程序退出前调用

CameraUnInit(m_hCamera);

以下是采用 CameraEnumerateDeviceEx 和 CameraInitEx2 典型的程序流程：

```
int    iCameraNums;
```

```
iCameraNums = CameraEnumerateDeviceEx();
if (iCameraNums == 0) return FALSE;

//本例程中，使用"Camera1"名字进行初始化，因此相机必须先用工具更名为"Camera1"
    if ((status = CameraInitEx2("Camera1",&m_hCamera)) != CAMERA_STATUS_SUCCESS)
    {
        return FALSE;
    }

//程序退出前调用
CameraUnInit(m_hCamera);
```

5.2 相机参数的保存与加载

相机的参数是保存在电脑里的， 以一个二进制文件的形势存在。界面上分为 A、B、C、D 四组。加载方式也分为按型号、按昵称和按序列号三种方式，如下图所示：

- CameraSetParameterMode,用于设置参数保存的方式。详情参考第四章函数解释。
- CameraSaveParameter，保存参数到指定的参数组。详情参考第四章函数解释。
- CameraLoadParameter，加载指定组的参数。详情参考第四章函数解释。
- CameraSaveParameterToFile，保存当前相机参数到指定的文件中。详情参考第四章函数解释。

- CameraReadParameterFromFile，从指定的文件中加载相机参数。详情参考第四章函数解释。

5.3 相机取图（主动取图或者回调函数方式）

初始化完成之后，就可以开始相机取图工作了。 按照取图方式，我们的 SDK 分为主动和被动（回调函数）两种。

- **主动读图方式：**

在完成相机初始化后工作，可以使用 CameraGetImageBuffer 或者 CameraGetImageBufferEx3 函数主动读取图像，二者的区别是，CameraGetImageBuffer 函数得到的是原始的 RAW 数据缓冲区，需要使用 CameraImageProcess 函数将 RAW 数据转换为指定的数据格式，在使用完得到的缓冲区指针后，要调用 CameraReleaseImageBuffer 释放缓冲区使用权，这里需要解释的是 CameraReleaseImageBuffer 只是释放由 CameraGetImageBuffer 得到的数据缓冲区的使用权，并不会反复申请和释放内存，无需担心效率问题；CameraGetImageBufferEx3 函数则是直接得到指定的格式的图像数据，具体的格式由输入的参数决定，可以是 8 位、16 位灰度，也可以是 24 位、32 位彩色格式数据，并且 CameraGetImageBufferEx3 函数调用后，不需要调用 CameraReleaseImageBuffer。两个读图函数的共同点是都可以设置超时时间，例如设置 1000 毫秒的超时时间，那么在 1000 毫秒内，

如果没有获取到有效图像，则该函数会阻塞，线程会被挂起，直到超过 1000 毫秒或者读到了有效图像，所以在软件结构的设计上，用户可以创建一个专门采集图像的线程，然后一直进行图像采集，只需要设定一个合理的超时时间即可，或者在需要采集图像的时候调用一次函数，获取一张图像。

以下是使用 CameraGetImageBuffer 的代码示例：

```
tSdkFrameHead    sFrameInfo;
BYTE*            pbyBuffer;
CameraSdkStatus      status;
/* pbyBuffer  由SDK内部自动申请好了内存，存放了原始RAW数据*/
if(CameraGetImageBuffer(m_hCamera,&sFrameInfo,&pbyBuffer,1000) ==
CAMERA_STATUS_SUCCESS)
{
/* m_pFrameBuffer需要先申请好内存pbyBuffer  转换后的图像数据保存在m_pFrameBuffer  中，
默认会得到BRG24bit的图像数据*/
        CameraImageProcess(m_hCamera, pbyBuffer, m_pFrameBuffer,&sFrameInfo);
        CameraReleaseImageBuffer(m_hCamera,pbyBuffer);

}
```

以下是使用 CameraGetImageBufferEx3 的代码示例：

```
int width,height;//图像宽、高
```

```
unsigned int uTimeStamp;//图像的时间戳
BYTE*pImageData;
pImageData = CameraAlignMalloc(1280*1024*3,16);//假定使用130万像素彩色相机

if(CameraGetImageBufferEx3(m_hCamera,
pImageData ,1,&width,&height,&uTimeStamp,1000) == CAMERA_STATUS_SUCCESS)
{
    /*成功采集到图像，进行视觉处理..... */
    /* pImageData 存放了RGB24格式的彩色图像数据,width和height会返回图像的宽和高，
uTimeStamp会返回图像的时间戳，1000为超时时间*
}

CameraAlignFree(pImageData);  //实际应用中 ,在程序初始化时为pImageData分配一次内
存，在最后退出时再释放内存，中间过程可以反复使用该缓冲区。
```

- 被动读图方式（回调函数）

  如果需要使用回调函数进行读图处理，则需要在相机初始化以后设置好读图的回调函数

  在 CameraInit 后，调用如下函数设置回调。

```
m_pFrameBuffer = CameraAlignMalloc(1280*1024*3,16);//假定使用130万像素彩色相机


CameraSetCallbackFunction(m_hCamera,GrabImageCallback,(PVOID)0,NULL);
```

其中 GrabImageCallback 定义如下：

```
void _stdcall GrabImageCallback(CameraHandle hCamera, BYTE *pFrameBuffer,
tSdkFrameHead* pFrameHead,PVOID pContext)
    {
        CameraSdkStatus status;
        //将RAW数据转换成指定格式的图像数据。 默认转换为BGR24格式的图像数据。
        status = CameraImageProcess(hCamera, pFrameBuffer,
m_pFrameBuffer,pFrameHead);
        //转换成功后，使用m_pFrameBuffer中的数据进行后续处理。
    }
```

5.4 利用显示控件预览图像

为了简化用户二次开发的流程，我们 SDK 内部封装了一些接口函数可以方便可以进行图像预览，前提是必须使用 VS 相关的工具进行界面开发，例如 C#、VB、Delphi、MFC 等。

在相机初始化后，调用如下代码完成显示部分的初始化工作。

CameraDisplayInit 传入窗口的 hWnd 句柄。 不同的开发工具，有不同的方式获得显示控件的 hWnd 句柄。

CameraSetDisplaySize 设置显示区域的大小，单位是像素。 一般设置为显示控件的实际大小即可。

CameraSetDisplayMode 设置显示的模式，有平铺和缩放两种显示模式。默认是缩放显示。图像会缩放到合适的大小显示到显示控件上。

CameraSetDisplayOffset 设置显示坐标的偏移值。当显示控件的尺寸小于图像的尺寸并且 CameraSetDisplayMode 设置为平铺显示模式时，可以通过该函数设置偏移值显示图像的某个区域。

以上几个函数的详细用法，我们提供了专门的例程，请参考 MindVision\Demo\VC++\CameraDisplay 例程。

5.5 调整相机图像亮度（设置曝光时间）

相机出厂时，为了方便演示效果，默认是自动曝光的，也就是说，相机会根据环境光的亮度，自动调节相机的曝光时间和模拟增益值来获取最佳的图像亮度。但是这种自动调节模式，并不适合工业应用，因此需要调用一些函数，来手动设置曝光和增益，来稳定图像亮度，以适合软件算法的需求，例如一些情况下，用户可能希望图像大部分区域过曝光以减少干扰图像。

- CameraSetAeState，设置相机曝光模式，分为手动和自动模式。
- CameraSetExposureTime ，设置相机的曝光时间，单位为微秒。注意，曝光时间越大，相机的帧率越低，例如曝光时间设置到 500 毫秒，那么一秒相机最多成像 2 次，图像看上去就会比较卡顿。因此为了减少取图时间，曝

光时间应该是要越低越好，当然，曝光时间低，就需要外部光源加大光照，否则图像就会很暗。

- CameraSetAnalogGain，设置相机的模拟增益值，这个值的大小只会影响图像亮度，但不会影响图像帧率，因为只是一个电路放大系数，不过该值增大后会提升图像背景噪声，对于追求画质的应用场合，模拟增益应该设置到最小。

注意，CameraSetExposureTime 和 CameraSetAnalogGain 进行手动设置的前提是通过 CameraSetAeState 将相机设置为手动曝光模式，否则设置无效！

5.6 切换不同的分辨率和自定义分辨率(ROI 功能)

ROI 的步进值说明一下

5.7 设置相机的对比度、伽马、饱和度、锐度等 ISP 参数

- CameraSetGamma，设置相机的 gamma 值，gamma 值越小，会提升亮度值较小的像素点，使得整体亮度正大；gamma 值越大，会减小高亮度的像素点的灰度值，使得整体亮度变小。gamma 值默认是 1.0。
- CameraSetContrast，设置相机的对比度值，对比度越大，会使图像黑的区域越黑，白的区域越白，使得图像看起来黑白很分明，在一些视觉处理上可以有效的捕捉轮廓；反之，如果对比度越小，会是黑白不分明，看起来比较朦胧。对比度默认是 100，最小可以到 1，最大到 200。

- CameraSetSaturation，设置饱和度。饱和度越大色彩越浓；反之越淡，饱和度如果设置为 0 ,图像就完全没有色彩了 ,等效于黑白相机。默认值是 100。调节范围是 0 到 200。
- CameraSetGain，设置 R、G、B 三个颜色通道的增益值。
- CameraSetSharpness，设置图像的锐化级别。锐化度越高，图像清晰度越好，但是噪声也会越大；反之锐化度越低，图像朦胧感就强，但是噪声很低，显得很平滑。默认值是 0，就是没有锐化增强的效果。

5.8 彩色相机转成黑白相机使用

CameraSetMonochrome 函数可以将彩色图像转为黑白使用 ,效果等效于将图像的饱和度设置为 0 ，因此也可以用 CameraSetSaturation 函数设置饱和度为 0 来获取黑白灰度图像。

注意事项：无论是黑白还是彩色相机，出厂默认都是输出 BGR24 的格式的图像数据，对于黑白相机的 BGR24，B=G=R。如果只需要处理 8 位的灰度图像，在初始化以后调用以下代码后，CameraImageProcesss 函数输出的图像就是 8 位灰度了。

```
//初始化相机
CameraInit....
//设置 CameraImageProcesss 函数的输出格式为 8 位灰度
CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_MONO8);
```

//后续的取图部分

```
    if(CameraGetImageBuffer(m_hCamera,&sFrameInfo,&pbyBuffer,1000) ==
CAMERA_STATUS_SUCCESS)
        {
            CameraImageProcess(m_hCamera, pbyBuffer,
m_pFrameBuffer,&sFrameInfo);// m_pFrameBuffer,指向的输出缓冲区地址中，就保存
的是8位的灰度图像格式了。
}
```

或者直接用 CameraGetImageBufferEx 函数，就可以取到 8 位灰度图像。

5.9 多个相机同时使用，如何建立相机对应关系

当视觉系统中同时使用多个相机时，在软件上面就需要确认确认好相机和检测工位之间的一一对应关系，但是相机的扫描顺序往往是不固定的，和相机的上电顺序有关系，因此需要在软件端进行一些特殊的识别。我们推荐采用以下 2 种方式进行绑定：

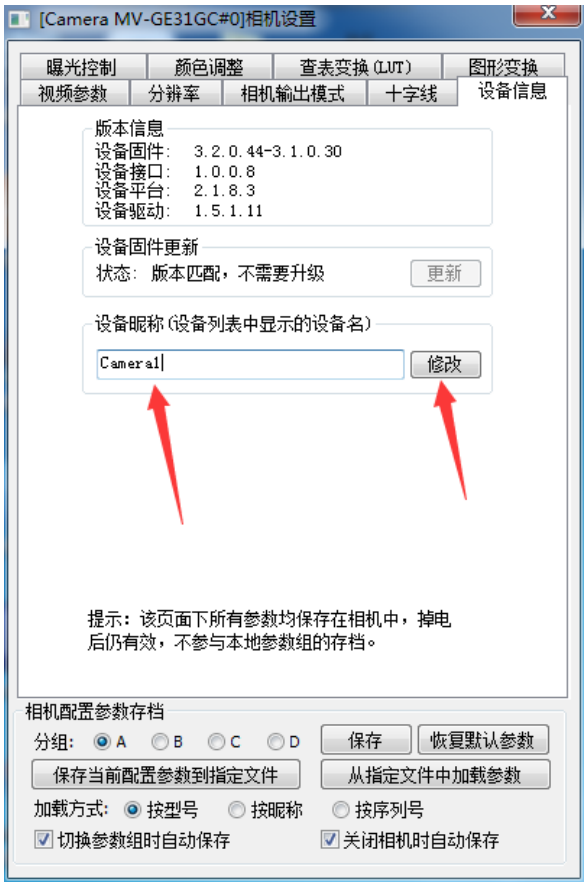
- 按照相机的序列号。相机出厂前，都有一个唯一的序列号编码。这个序列号，在相机的枚举阶段就可以读到，通过CameraEnumerateDevice函数可以扫描到相机的个数和详细的描述信息结构体数组，类型为tSdkCameraDevInfo，tSdkCameraDevInfo中的acSn成员就保存了最大32个字节的序列号字符串。可以通过软件判断序列号，来选择相应的

tSdkCameraDevInfo数组进行后面的CameraInit调用。

- 按照相机的自定义名称。默认情况下，相机的名称是随着相机上电顺序变化的，例如第一个上电的相机，以型号名加#0结尾，例如MV-GE500M#0和MV-GE500M#1分别表示系统中接入的2个相机名称，如果相机的上电顺序改变了，那么原来的0号相机就有可能变成了1号相机。因此我们提供了修改工具，可以手动修改每个相机的自定义名称。该自定义名称和序列号一样，是可以在枚举阶段就被枚举出来的，也是通过CameraEnumerateDevice函数可以扫描到相机的个数和详细的描述信息结构体数组，类型为tSdkCameraDevInfo，tSdkCameraDevInfo中的acFriendlyName成员就保存了最大32个字节的设备自定义名称。后续根据名称选择好tSdkCameraDevInfo数组来进行CameraInit调用。

  相机自定义名称的修改方式如下，输入设备昵称后，点修改按钮，然后关断一次相机电源进行重启，自定义名称就生效了，会永久固化到相机内部。

  特别注意：CameraEnumerateDevice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，CameraEnumerateDevice返回的列表，名字为“Camera1”的相机会排前面，名为“Camera2”的相机排后面，这个方法是比较简单有效的,代码上就可以按顺序初始化了,不用进行特殊绑定了。



修改相机的自定义名称

相机名称修改好以后，也可以通过CameraInitEx2函数进行快速初始化。假设将2个相机改名为"Camera1"和"Camera2"代码如下：

```
//扫描相机，如果有2个，iCameraCounts = 2
int iCameraCouts = CameraEnumerateDeviceEx();
//初始化自定义名称为"Camera1"的相机，如果没找到对应名字的相机，则初始化会失败。
CameraInitEx2 ("Camera1",&hCamera1);
//初始化自定义名称为"Camera2"的相机，如果没找到对应名字的相机，则初始化会失败。
CameraInitEx2 ("Camera2",&hCamera2);
```

5.10 图像的翻转镜像与旋转功能

- 使能图像水平翻转，CameraSetMirror(hCamera,0,1);
- 禁止图像水平翻转（默认）, CameraSetMirror(hCamera,0,0);
- 使能图像垂直翻转, CameraSetMirror(hCamera,1,1);
- 禁止图像垂直翻转（默认）, CameraSetMirror(hCamera,1,0);
- 图像旋转 0 度（默认），CameraSetRotate(hCamera,0);
- 图像旋转 90 度（默认）, CameraSetRotate(hCamera,1);
- 图像旋转 180 度（默认）, CameraSetRotate(hCamera,2);
- 图像旋转 270 度（默认）, CameraSetRotate(hCamera,3);

注意，图像旋转以后，图像的宽高尺寸会掉换，例如原来是 800X600，旋转 90 度以后，就是 600X800 的尺寸，处理的时候要注意，否则会造成显示错误，图像看起来很混乱。

5.11 在图像上叠加文字功能

为了方便客户在图像上叠加文字进行信息提示，我们封装了一个函数 CameraDrawText 进行快速调用。有关该函数的用法，我们专门提供了一个例程，位于\Demo\VC++\DrawText 下。 需要注意的是，CameraDrawText 函数叠加文字，是直接在图像内容上叠加的， 为了不影响视觉处理，应该放在视觉处理的最后流程再调用（图像显示函数之前调用）。

5.12 保存图片和录像功能

- 图像保存功能。使用 CameraSaveImage，可以将图像保存为 RAW、PNG、JPG、BMP24bit、BMP8bit 的其中一种。一个典型的流程为：

  1，使用 CameraGetImageBuffer 函数取到 RAW 图像数据；

  2，使用 CameraImageProcess 函数将 RAW 转换成 BGR24 或者 8 位灰度格式。如果要保存 RAW 格式数据，这一步跳过。

  3，CameraReleaseImageBuffer，释放 CameraGetImageBuffer 的到的 RAW 数据缓冲区的使用权。

  4，CameraSaveImage 函数将 CameraImageProcess 函数得到的图像数据进行保存成 JPG、BMP 或者 PNG 其中的一种。如果是 RAW 数据保存则使用 CameraGetImageBuffer 得到的 RAW 数据 buffer。

  图像保存的例子位于\Demo\VC++\ImageFormat&Saving

- 录像功能。CameraInitRecord 函数初始化一次录像。CameraPushFrame 编码一帧图像。CameraStopRecord 停止录像，结束写文件操作。

  录像的例子位于\Demo\VC++\Record。

5.13 设置相机输出的图像位深度

默认情况下，工业相机传输的一般是 8bit 的 raw 格式数据，在一些宽动态的应用上行，用户可以通过 CameraSetMediaType 函数来改变相机输出的 raw 数据位深度。目前我们支持的位深度有 8、12、16 位 3 种可选。

CameraSetMediaType 是选择相机支持的位深度种类。

例如 CameraSetMediaType (hCamera,0);选择第 1 种，就是 8bit 格式的，CameraSetMediaType(hCamera,1);选择第 2 种，是 12bit，CameraSetMediaType(hCamera,2);选择第 3 种，是 16bit。

注意 :不是所有型号都支持 12 和 16bit 输出模式。如果 CameraSetMediaType 返回非 0 值，就表示设置失败。

5.14 设置图像的像素格式（8 位灰度，24、32、48 位彩色）

不同于 5.13，这里说的像素格式，是最终通过 CameraImageProcess 函数的得到的图像像素格式，而并非相机输出的原始 RAW 的像素格式。

大部分型号的工业相机都可以输出 12bit 甚至 16bit 的原始 RAW 图像，对于彩色相机，如果一个颜色分量用 8bit 就会丢失掉图像低位的细节，因此我们提供了 RBG48bit 的彩色图像格式来满足用户做高动态、高精度的视觉分析。红、绿、蓝三个颜色通道，每个通道都使用 16bit 来表示，一个像素需要 6 个字节。当相机支持 12bit 或者 16bit 的原始 RAW 格式时，都可以使用 48bit 的彩色图像格式。12bit 格式的 RAW 数据，在转成 48bit 的彩色图像时，每个通道的低 4bit 会填充 0 。

在相机初始化以后，通过调用 CameraSetIspOutFormat 来设置了。

例如：

```
CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_MONO8);
```

//设置后，CameraImageProcess 输出的图像就是 8bit 的灰度图像了，1 个像素占用 1 个字节，依次排列。

CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_MONO16);

//设置后，CameraImageProcess 输出的图像就是 16bit 的灰度图像了，1 个像素占用 2 个字节，依次排列。

CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_BGR8);

//设置后，CameraImageProcess 输出的图像就是 24bit BGR 的彩色图像了，1 个像素占用 3 个字节，依次是蓝色、绿色、红色这样排列。

CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_BGR16);

//设置后，CameraImageProcess 输出的图像就是 48bit BGR 的彩色图像了，1 个像素占用 6 个字节，依次是蓝色 16bit、绿色 16bit、红色 16bit 这样排列。

CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_BGRA8);

//设置后，CameraImageProcess 输出的图像就是 32bit BGRA 的彩色图像了，1 个像素占用 3 个字节，依次是蓝色、绿色、红色、Alpha 这样排列。

CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_BGRA16);

//设置后，CameraImageProcess 输出的图像就是 64bit BGRA 的彩色图像了，1 个像素占用 8 个字节 ,依次是蓝色 16bit、绿色 16bit、红色 16bit、Alpha16bit 这样排列。

CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_RGB8);

//设置后，CameraImageProcess 输出的图像就是 24bit RBG 的彩色图像了，1 个像素占用 3 个字节，依次是红色、绿色、蓝色这样排列。

CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_RGB16);

//设置后，CameraImageProcess 输出的图像就是 48bit RBG 的彩色图像了，1 个像素占用 6 个字节，依次是红色 16bit、绿色 16bit、蓝色 16bit 这样排列。

CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_RGBA8);

//设置后，CameraImageProcess 输出的图像就是 32bit RBGA 的彩色图像了，1 个像素占用 4 个字节，依次是红色 8bit、绿色 8bit、蓝色 8bit、Alpha8bit 这样排列。

CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_RGBA16);

//设置后，CameraImageProcess 输出的图像就是 64bit RBGA 的彩色图像了，1 个像素占用 8 个字节 ,依次是红色 16bit、绿色 16bit、蓝色 16bit、Alpha16bit 这样排列。

对于 16bit（1 个通道 2 字节）的数据，我们采用 little-endian 方式存储，字节 1 是低 8bit，字节 2 是高 8bit。

该函数的用法，可以参考该例程：\Demo\VC++\ImageFormat&Saving

5.15 对原始的 RAW 图像进行处理

一般情况下，工业相机传输的都是原始的 RAW 数据，对于黑白相机，就是灰度图像数据，对于彩色相机，就是 Bayer 格式的图像数据。

CameraGetImageBuffer 函数，得到的就是原始的 RAW 数据，默认是 8bit 的，如果需要 12、16bit 格式的，请参考 5.13 中的方法。 注意 CameraGetImageBuffer 需要和 CameraReleaseImageBuffer 配套使用。

5.16 网口相机使用 API 动态设置 IP、网关、子网掩码

在使用网络相机时，必须先将相机设置到和连接的网卡同一个子网网段，这样才能进行正常的通信。我们提供一个专用的千兆网 IP 设置工具进行 IP 设置（位于软件安装目录的 TOOLS 文件夹内）。

同时用户也可以通过 CameraGigeSetIp 函数，在程序里动态修改相机的 IP 参数。具体用法请参考例程：Demo\VC++\GigeConfig。

5.17 设置相机的帧率

通过 CameraSetFrameSpeed 函数，可以动态设置相机的输出帧率。CameraSetFrameSpeed 是设置速度的档位，我们将其分为高速、中速、低速等几种模式，对不不同型号的相机，高中低速度模式下对应具体的帧率是不一样的。

CameraSetFrameSpeed（hCamera,0）;//设置为低速模式。

CameraSetFrameSpeed（hCamera,1）;//设置为中速模式。

CameraSetFrameSpeed（hCamera,2）;//设置为高速模式。

一般相机出厂时，默认就是最高速度的，如果需要降速运行，就可以调用CameraSetFrameSpeed函数进行降速。 另外，有些型号的相机，只支持2个速度模式，有些支持4个，具体支持模式的数量，可以通过CameraGetCapability函数，得到相机的描述信息tSdkCameraCapbility结构体，tSdkCameraCapbility结构体中的iFrameSpeedDesc成员，表明了当前型号的相机支持的速度模式的个数。

5.18 使用软触发或者硬(外)触发功能

- 设置相机为连续取图模式。相机出厂默认就处于连续取图模式，也可以通过CameraSetTriggerMode(hCamera,0);切换为连续取图模式。
- 设置相机为软触发取图模式。通过 CameraSetTriggerMode(hCamera,1);切换为软触发取图模式。进入该模式后，相机停止图像采集和发送。 只有当用户调用 CameraSoftTriggerEx(hCamera,1)一次，相机就采集一次图像发送上来。发完之后相机又会进入等待状态，直到下次用户调用CameraSoftTriggerEx(hCamera,1)。
- 设置相机为硬触发取图模式。通过 CameraSetTriggerMode(hCamera,2);切换为硬触发取图模式。进入该模式后，相机停止图像采集和发送。 只有当用户在相机外壳上的触发端子上输入一个脉冲时，相机就采集一次图像发送上来。发完之后相机又会进入等待状态，直到下次收到触发脉冲。

如果用户需要一次触发多帧图像，可以通过 CameraSetTriggerCount 函数来设定，默认是一次触发得到一帧图像。如果需要一次触发得到 N 个图像，则可以调用 CameraSetTriggerCount(hCamera,N) 来设定。

5.19 外触发模式下给信号去抖

为了避免干扰信号对外触发产生影响 ,特别是当使用机械信号给相机进行外触发时，信号会存在很大的干扰和抖动，容易造成误判。因此可以使用 CameraSetExtTrigJitterTime 函数，设置去抖时间，来增加系统的扛干扰能力。默认时间是 0，就表示不去抖，在电子开关信号很干净的时候不进行去抖工作。设置去抖时间，会引入触发延时，例如 10 毫秒的去抖时间，则触发延时至少会延后 10 个毫秒。

5.20 外触发模式设置触发延时时间

如果触发信号发出来后，需要相机延时一段时间再拍照，则可以使用触发延时的方式。典型的应用，比如使用接近开关做为触发信号源，但是接近开关发出信号的时候，物体还没有到位或者还处于抖动等不稳定模式，这个时候就需要延迟一段时间再触发拍照。 CameraSetTriggerDelayTime 函数用于设置外触发延时，单位微秒。

5.21 检测相机掉线与自动重连

在一些极端环境下，相机可能存在掉线的风险，例如 USB 相机在电脑主机供电不稳定的时候，或者震动比较大，USB 口松动等等。

默认情况下，我司的 USB 和千兆网相机都有自动掉线重连的功能，SDK 内部也已经集成了该功能。 如果用户需要实时监测掉线情况，可以周期性的调用 CameraConnectTest 函数进行查询。

5.22 相机序列号的读取和写入

相机自带三级序列号，其中一级序列号是只读的，不可以修改，出厂的时候已经固定好了。二级和三级序列号可自由读写。每级序列号都是 32 个字节。

CameraReadSN，读取序列号。

CameraWriteSN，写入序列号。

该函数的用法请参考 安装目录下\Demo\VC++\UserDataTest 例程。

5.23 在相机中读取和写入自定义数据

我司所有型号的相机，都带一段数据存储空间，用户可以自由读写。并且掉电后也会保存，不会丢失数据。每个型号的相机，存储空间的大小是不一样的，具体的大小可以通过 CameraGetCapability 函数得到，tSdkCameraCapbility 结构体的 iUserDataMaxLen 表示相机的最大存储字节数。

CameraSaveUserData，写一段数据到相机中。掉电后仍然保存的。

CameraLoadUserData，从相机中读取一段数据。

该函数的用法请参考 安装目录下\Demo\VC++\UserDataTest 例程。

5.24 获取错误码对应的字符串描述信息

CameraGetErrorString

例如 CameraInit 函数返回错误码-45，然后

char* s = CameraGetErrorString(-45) 就会得到一个字符串的指针，如果用 printf 函数打印出来的话，就会显示：

"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)"；

6 GigE Vision 和 USB3 Vision 的 XML 文件定义解释

对于使用标准 Vision 协议接口开发的客户，在设备枚举阶段，需要解析相机对应的 XML 配置文件，这个部分我们提供了专门的文件进行解释，请参考安装目录下 Document 文件夹里《USB3、GigE Vision 接口开发及 XML 解析.pdf》

7 Halcon 开发指导

Halcon 是目前世界上最全能的机器视觉软件。世界各地的用户从 Halcon 为快速开发图像分析和机器视觉程序的灵活架构获益匪浅。Halcon 提供了超过 1100 多种具备突出性能控制器的库,如模糊分析,形态,模式匹配,3D 校正等。Halcon 支持多个操作系统。

我们的相机接口专门为 Halcon 进行了优化设置,能够良好的支持 Halcon。在已经安装了 Halcon 的系统中，再运行我们的相机软件安装包 MindVision Platform Setup(x.x.x.x).exe 安装程序，安装程序会自动检测 Halcon 的安装路径,并对其进行相应的设置。目前,我们的相机开发包支持 Halcon8、Halcon9、Halcon10，Halcon11，Halcon12 的 32 位和 64 位版本。

用数据线将相机和 PC 连接后，便可开始开发了，以下说明以 MV-U300 型号的相机为例进行说明，其他型号的相机开发时，只需要修改型号名称即可。

7.1 HDevelop 中进行开发

1. 启动 Halcon 后，点击”助手”菜单，选择“打开新的 Image Acquisition” ，如图 7.1 所示。

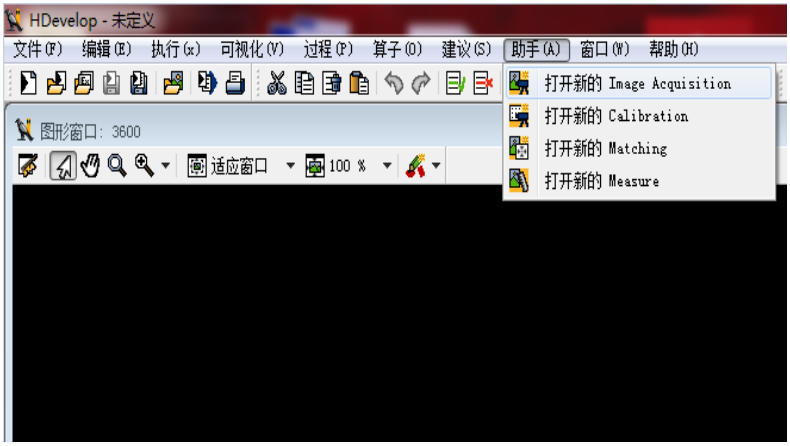


图 7.1 Halcon 图像采集接口

2. 点击“图像获取接口(I)”,在下拉列表中，找到 MindVision。如图 7.2 所示。

3. 点击图 7.2 中的“连接”标签页，出现如图 7.3 所示的窗口。在图 7.3 中，我们可以看到我们的 MV-U300 相机。如果同时连接了多台 MindVision 相机，请在设备清单中选择您想要访问的相机。

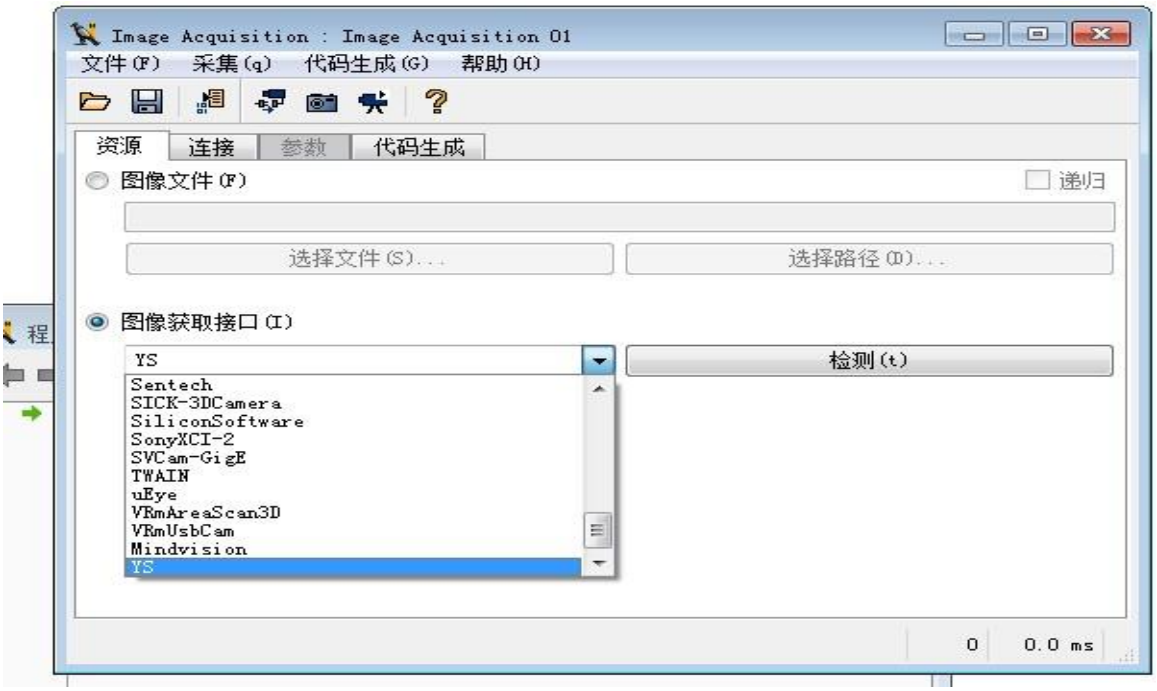


图 7.2 在 Halcon 图像采集接口列表中找到 Mindvision

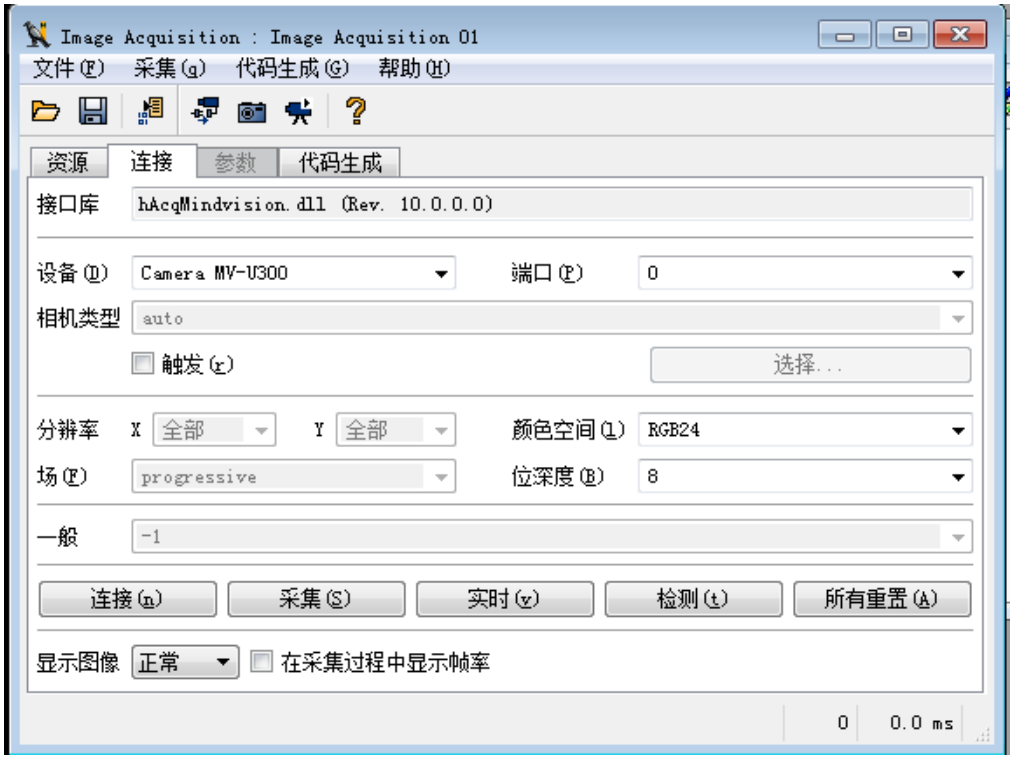


图 7.3 Halcon 中相机采集接口窗口

4. 点击图 7.3 中的”连接(n)”按钮，成功连接后，“参数”标签页被启动，如图 7.4 所示。此时可以进行颜色空间和位深度的选择。

- 我们的接口支持 RGB24 和 Gray 两种颜色空间，当选择 RGB24 时，输出的图像是 RGB888 格式，适合彩色相机，当选择 Gray 时，输出的图像格式是 8 位灰度，适合黑白相机。

- 位深度支持 8、10、12 位三种(仅部分相机支持 10、12 比特的位深度，无帧存系列只支持 8 位位深度)。

图 7.4 Halcon 中成功连接 MV-U300

5. 参数设置。点击“参数”卷标页按钮，切换到参数配置页面，如图 7.5 所示。

   - bits_per_channel 中可以选择每通道下的位深度；
   - grab_timeout 为抓图超时时间 ,单位为毫秒,默认是 200 毫秒超时; 注意，在超时时间内，如果图像没有到来，采集线程将会被阻塞，直到超时时间结束或者读取到有效图像帧。可以通过 set_framegrabber_param (AcqHandle, 'grab_timeout', 200) 设置采图的超时时间，单位为毫秒。
   - camera_play 点击后，开始或者继续图像采集功能;
   - camer_pause 点击后，暂停图像采集功能;
   - show_camera_settings 点击后 ,显示 MV-U300 相机的配置页面，如图 5.6 所示。在 halcon 程序中，可以通过 set_framegrabber_param (AcqHandle, 'show_camera_settings', 1)来显示相机的配置页面。

- software_trig 点击后，如果相机处于软触发模式，则触发相机采集一帧图像，并发往主机。如果相机没有工作在软触发模式，该按钮点击无效。在 halcon 程序中，可以通过 set_framegrabber_param (AcqHandle, 'software_trig', 1) 向相机发送一次软触发命令。
- shutter,设置相机的曝光时间，单位为微秒。
- contrast,设置相机的对比度。范围为 0 到 200，默认是 150。
- gamma,设置相机的 gamma 值，范围为 0 到 1000，默认为 50，对应 gamma 系数为 0.5。
- exposure_mode,选择曝光模式。0 表示手动模式；1 表示自动模式。只有当 exposure_mode 设置为 0 时，shutter 和 gain 参数的调节才会生效。
- trigger_mode，选择相机工作模式。0 表示连续采集模式；1 表示软触发模式；2 表示外部硬件触发模式。在程序中，可以通过 set_framegrabber_param (AcqHandle, 'trigger_mode', 2)让相机进入外触发模式，此时只要外部输入信号到相机触发引脚上，相机就会开始采集一帧图像传输到 PC，没有触发信号时，grab_image_async (Image, AcqHandle, -1)函数会返回超时。
- frame_speed，选择相机的采集速度，0 表示最慢，1 表示中等，2 表示最高。部分相机最高只有中速模式。采集速度越慢，相机的最大曝光时间越长，如果需要长时间曝光，请将 frame_speed 设置到 0。halcon 程序中，可以通过 set_framegrabber_param (AcqHandle, 'frame_speed', 0) 调用来改变相机的采集速度。
- resolution,选择默认分辨率。0 表示选择第一个默认分辨率，1 表示选择第二个默认分辨率，依次类推。可动态切换。
- color_temperature,选择色温模式(对于黑白相机改参数无效)。0 表示色温 2700K 左右，1 表示色温 4200K，2 表示色温 5500K 左右，3 表示色温 6500K 左右。

- red_gain,设置红色的信道的数字增益。范围为 0 到 4，用于颜色的微调。对于黑白相机，该参数无意义。
- green_gain,设置绿色的信道的数字增益。范围为 0 到 4，用于颜色的微调。对于黑白相机，该参数无意义。
- blue_gain,设置蓝色的信道的数字增益。范围为 0 到 4，用于颜色的微调。对于黑白相机，该参数无意义。
- saturation,设置相机颜色的饱和度。仅对彩色相机有效，范围为 0 到 200，设置为 0 时，图像完全失去色彩，变为黑白图像，设置为 200 时，颜色最艳丽，默认为 100。
- gain,设置相机的模拟增益，该参数和 shutter 参数一起，决定图像的亮度。仅当 exposure_mode 设置为 0 时，gain 和 shutter 参数才能手动调节。
- color_space,设置图像输出的格式。选择 Gray 是，输出 8 会灰度图像，选择 RGB24 时，输出 24 位的彩色图像。黑白相机也可以选择 RGB24 格式，但是 R=G=B；彩色相机也可以选择 8 位灰度图像进行输出。
- color_space,设置图像输出的格式。选择 Gray 是，输出 8 会灰度图像，选择 RGB24 时，输出 24 位的彩色图像。黑白相机也可以选择 RGB24 格式，但是 R=G=B；彩色相机也可以选择 8 位灰度图像进行输出。
- GPO0,GPO1,GPO2,GPO3 这 4 个对应相机上的 4 个输出型 IO 状态的控制，set_framegrabber_param (AcqHandle, ' GPO0', 1) 设置编号为 0 的 OUTPUT IO 状态为高电平，set_framegrabber_param (AcqHandle, ' GPO0', 0) 设置编号为 0 的 OUTPUT IO 状态为低电平；set_framegrabber_param (AcqHandle, ' GPO1', 1) 设置编号为 1 的 OUTPUT IO 状态为高电平，set_framegrabber_param (AcqHandle, ' GPO1', 0) 设置编号为 1 的 OUTPUT IO 状态为低电平 ;其他编号的 IO 以此类推。

- GPI0,GPI1，这 2 个对应相机上的 2 个输入型 IO 状态的读取，get_framegrabber_param (AcqHandle, ' GPI0')读取编号为 0 的 IO 的状态，返回 0 表示低电平，1 表示高电平；get_framegrabber_param (AcqHandle, ' GPI1')读取编号为 1 的 IO 的状态，返回 0 表示低电平，1 表示高电平。

图 7.5 Halcon 中相机参数设置页面

6. 图像采集。Halcon 中图像采集分为单次采集和实时采集两种方式。点击图 7.4 中“采集(S)”按钮，进行一次单次采集，采集后图像现在 Halcon 窗口内，如图 7.7 所示；点击图 7.4 中的”实时(v)”按钮，进行连续采集，Halcon 中将连续显示采集到得图像。

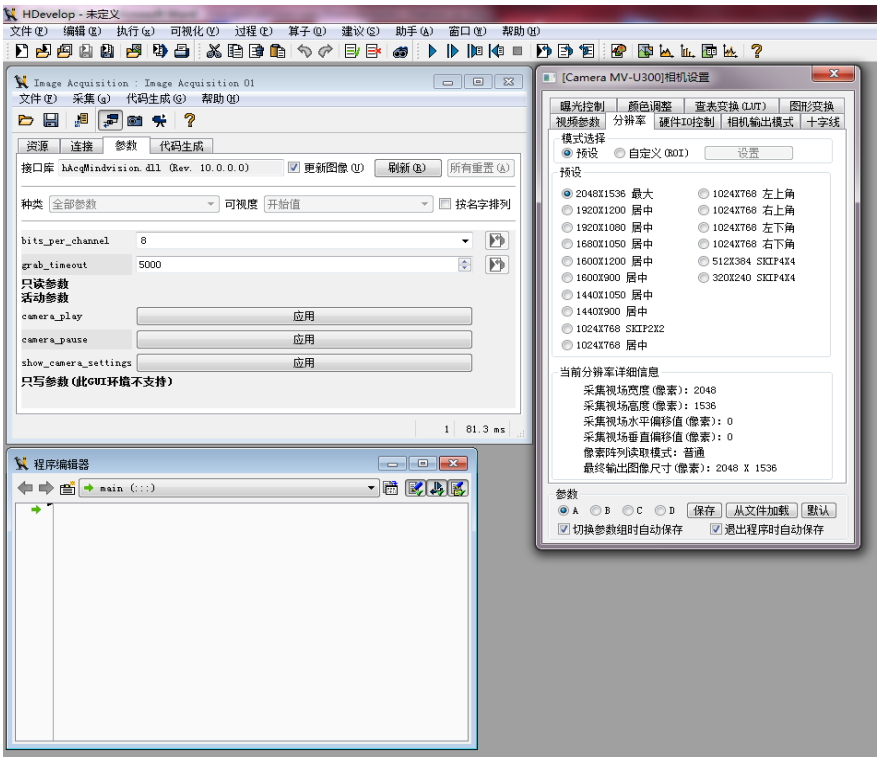


图 7.6 Halcon 中显示相机的设置窗口

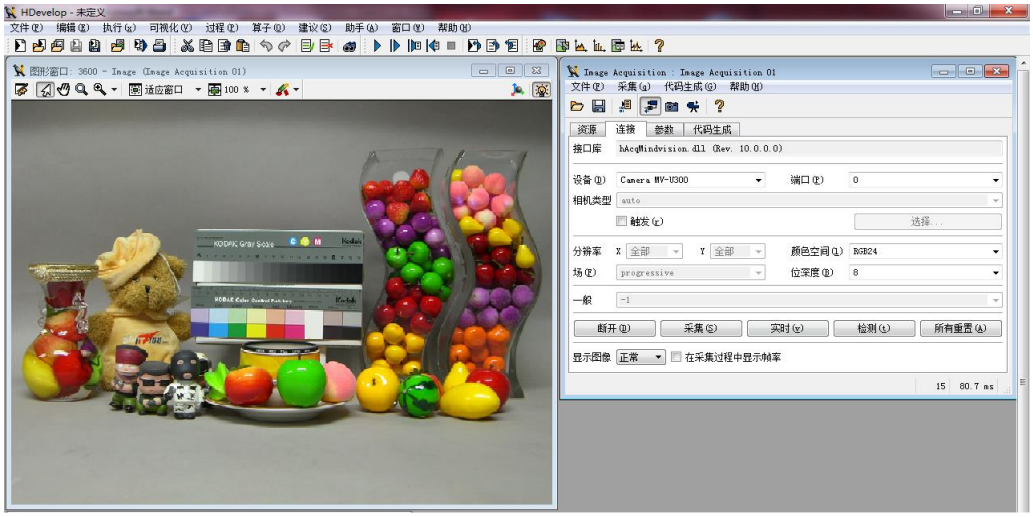


图 7.7 Halcon 中采集到的图像

7.2 C/C++、 VB、 C#中进行 Halcon 开发

如果您需要使用 C/C++、VB、C#等其他语言进行 Halcon 开发，只需要在 HDevelop 中生成相应的工程和代码即可。具体的做法是：在图 7.5 中，点击”代码生成”标签后，如图 7.8 所示：

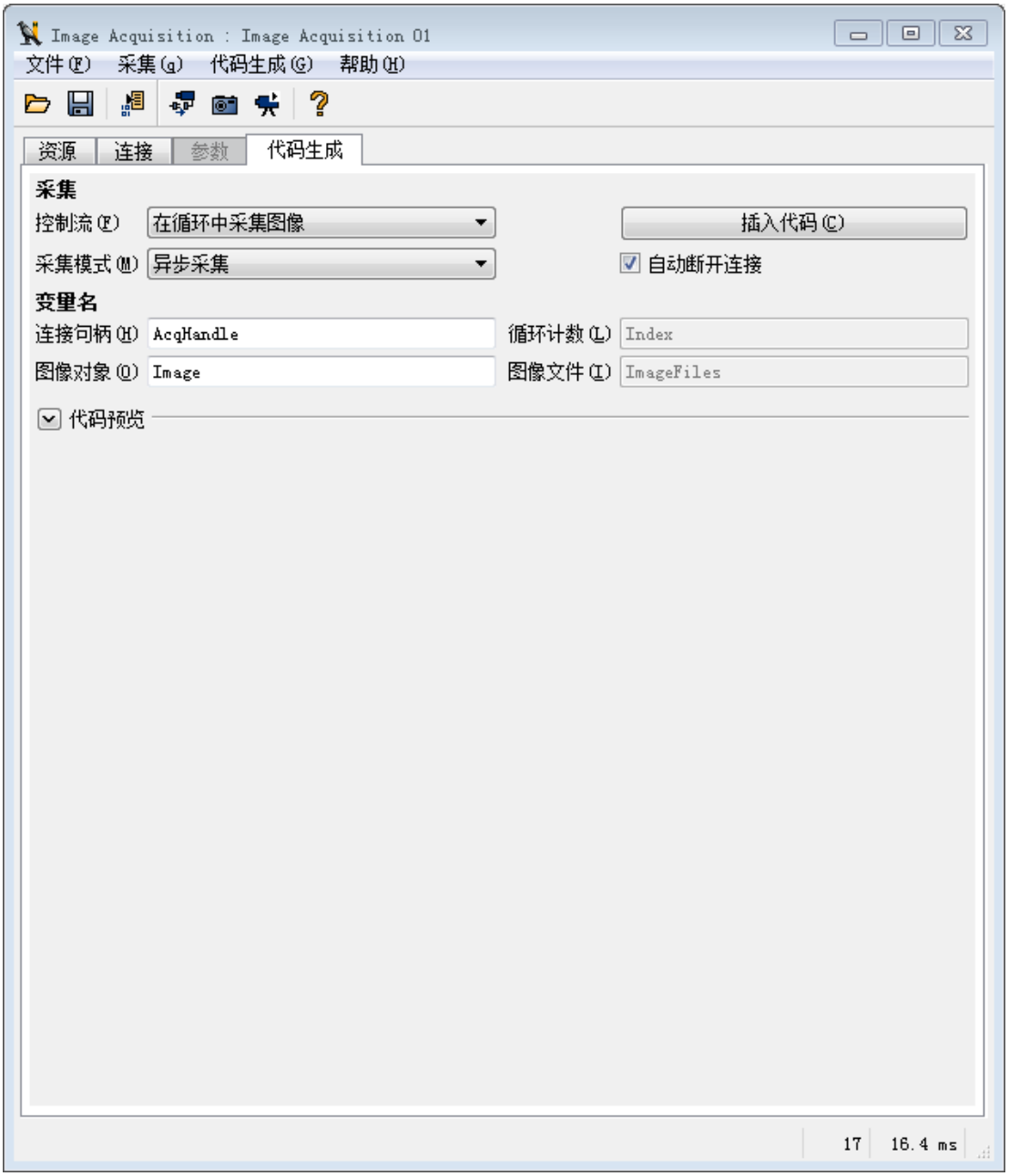


图 7.8 Halcon 中代码生成配置

在 7.8 中，您可以进行相应的配置，或者直接点击”插入代码“按钮，就可以得到 HDevelop 中可以编译和运行的代码，如图 7.9 所示。

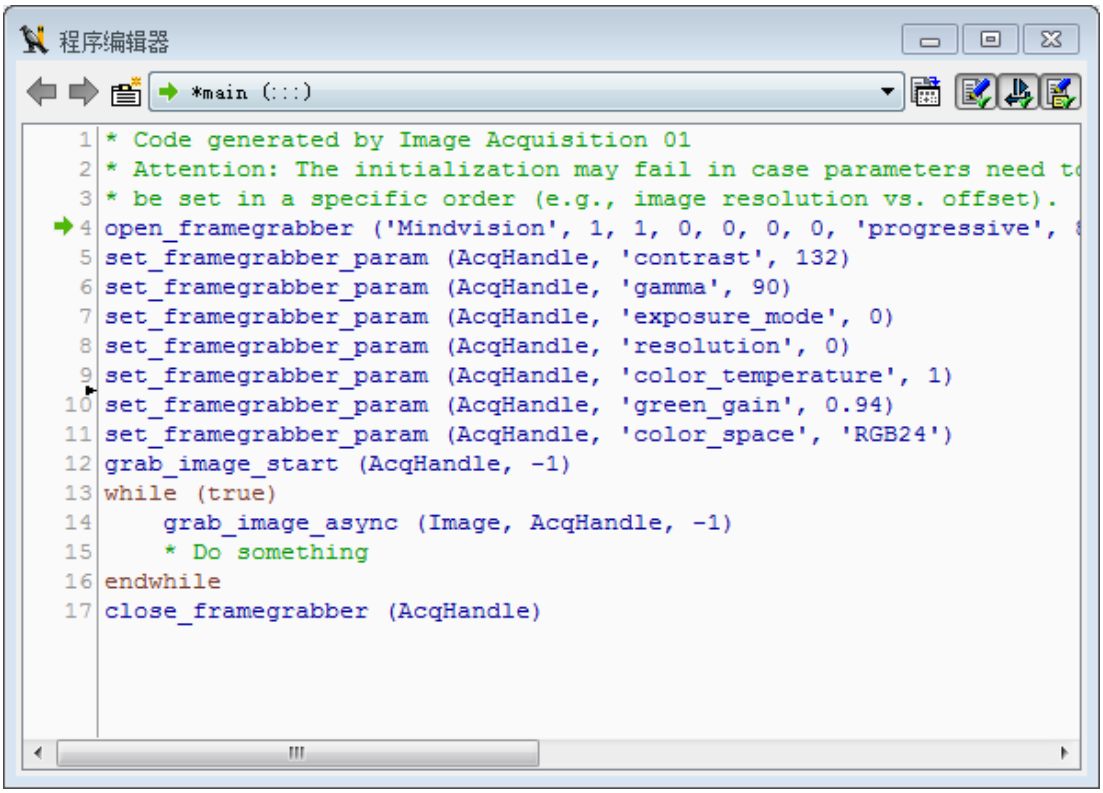


图 7.9 自动生成的 HDevelop 代码

接下来，就是将图 7.9 所示的代码转换为 C/C++、VB、Delphi、C#工程。点击“文件”菜单，选择“导出”，如图 7.10 所示。

然后在弹出的导出对话框中，进行设置，如图 7.11 所示，在下拉列表中，选择您要导出的语言类型，然后点“导出”按钮即可。汇出完成后，您就可以在您设置的路径下，找到对应语言的工程文件，直接打开编译就可以运行了。

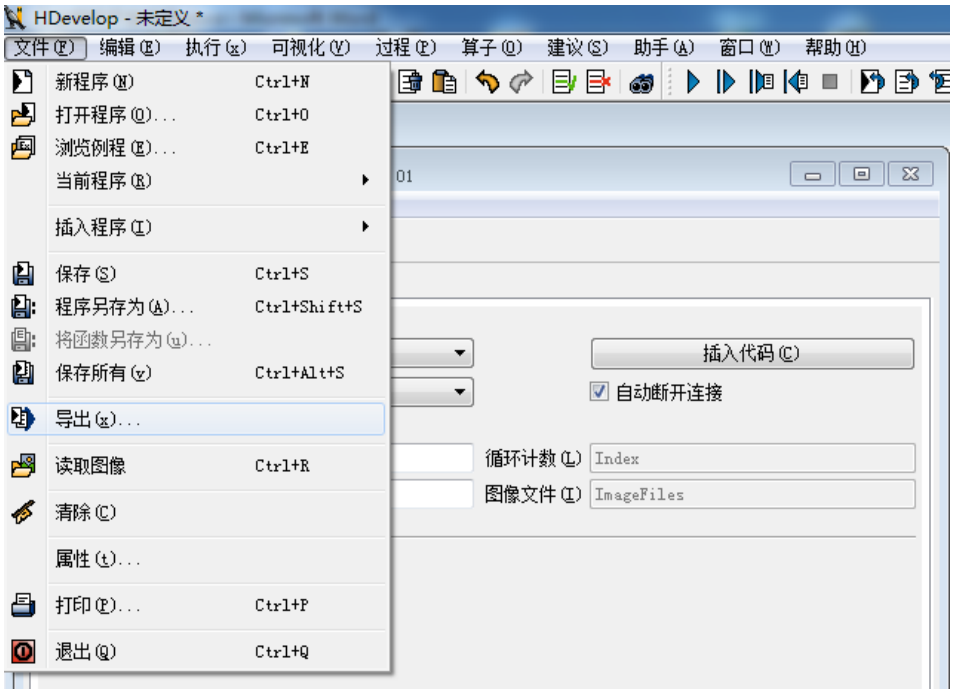
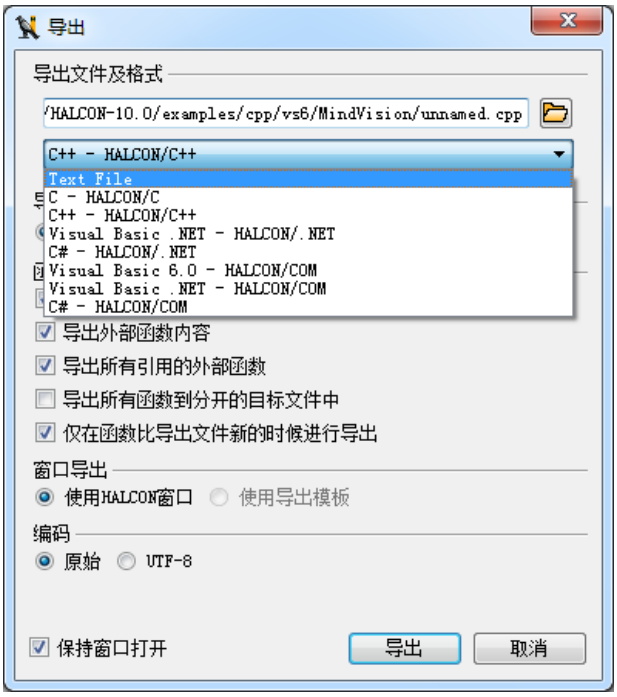


图 7.10 导出其他开发语言



7.11 汇出设置

7.3 同时使用多个相机开发

当同时连接 2 个相机时，为了更好的区分每个相机的对应关系，建议采用如下的流程进行：

1，使用我们提供的演示软件，修改相机的设备昵称，例如，第一台相机的名称修改为 Camera1,第二台相机的名称修改为 Camera2,修改完成后，无论这 2 台相机接在哪个 USB 或者网络接口上，其名称都会是 Camera1 和 Camera2，更换计算机后仍然有效。(相机设备名的修改见图 5.12)

2 ,初始化 Halcon 设备时 ,分别填入设备名"Camera1"和"Camera2",如下所示：

open_framegrabber ('MindVision', 1, 1, 0, 0, 0, 0, 'progressive', 8, 'Gray', -1, 'false', 'auto', 'Camera1', 0, -1, AcqHandle)

open_framegrabber ('MindVision', 1, 1, 0, 0, 0, 0, 'progressive', 8, 'Gray', -1, 'false', 'auto', 'Camera2', 0, -1, AcqHandle)。

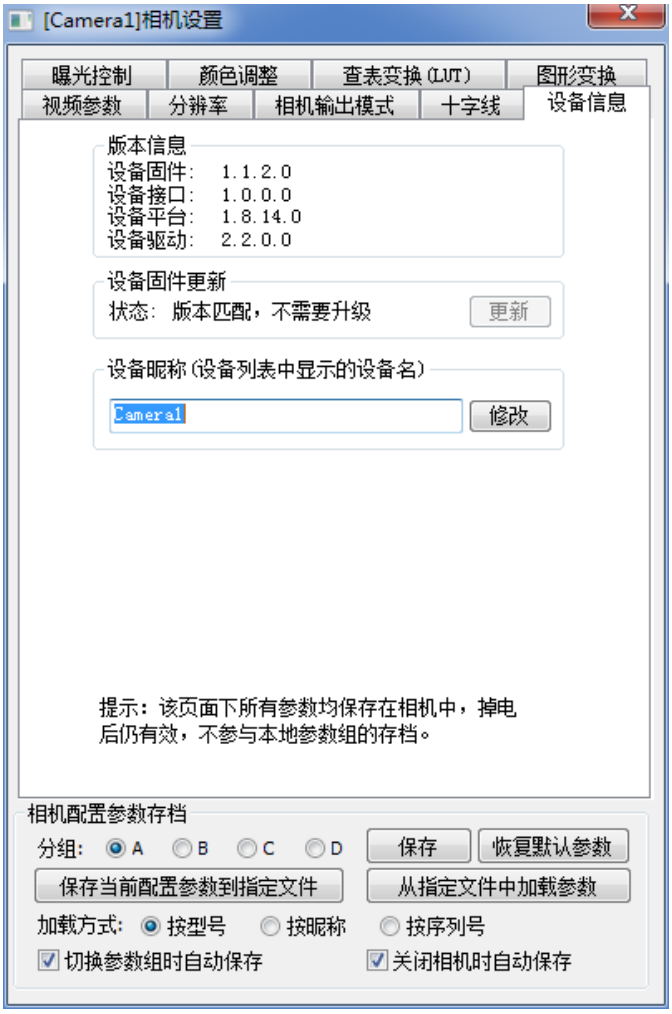

[Camera1]相机设置
曝光控制
颜色调整
查表变换 (LUT)
图形变换
视频参数
分辨率
相机输出模式
十字线
设备信息
版本信息
设备固件: 1.1.2.0
设备接口: 1.0.0.0
设备平台: 1.8.14.0
设备驱动: 2.2.0.0
设备固件更新
状态: 版本匹配，不需要升级
更新
设备昵称(设备列表中显示的设备名)
Camera1
修改
提示：该页面下所有参数均保存在相机中，掉电后仍有效，不参与本地参数组的存档。
相机配置参数存档
分组: A B C D
保存
恢复默认参数
保存当前配置参数到指定文件
从指定文件中加载参数
加载方式: 按型号 按昵称 按序列号
切换参数组时自动保存
关闭相机时自动保存

图 7.12 修改相机名称

8 LabView 开发指导

在 Labview 中可以通过 NI MAX 工具和我们提供的基于 DLL 调用的例程两种方式进行开发。

8.1 使用 NI MAX 开发

1. 先连接上相机，然后运行 NI MAX 后。可以在”设备和接口”的 NI-IMAQdx Devices 下，找到我们的相机，如图 8.1 所示。双击后，即可得到如图 8.2 所示的相机预览画面(需要点击图 8.2 上方的 Grab 按钮后才能预览)。
2. 在图 8.2 中，可以进行分辨率的切换。
3. 点击图 8.2 中的 Camera Attributers 标签后，可以进行相机其他参数的设置。如图 8.3 所示。该接口下，由于受协议的限制，只允许设置相机的部分参数，其他参数的设置，您可以通过我们的演示软件，调整好后，保存下来(相机参数可以保存成一个档，无论您使用哪种方式开发该参数档都可以被有效的自动加载)，再用 NI MAX 打开，同样有效果。

图 8.1 NI MAX 设备清单

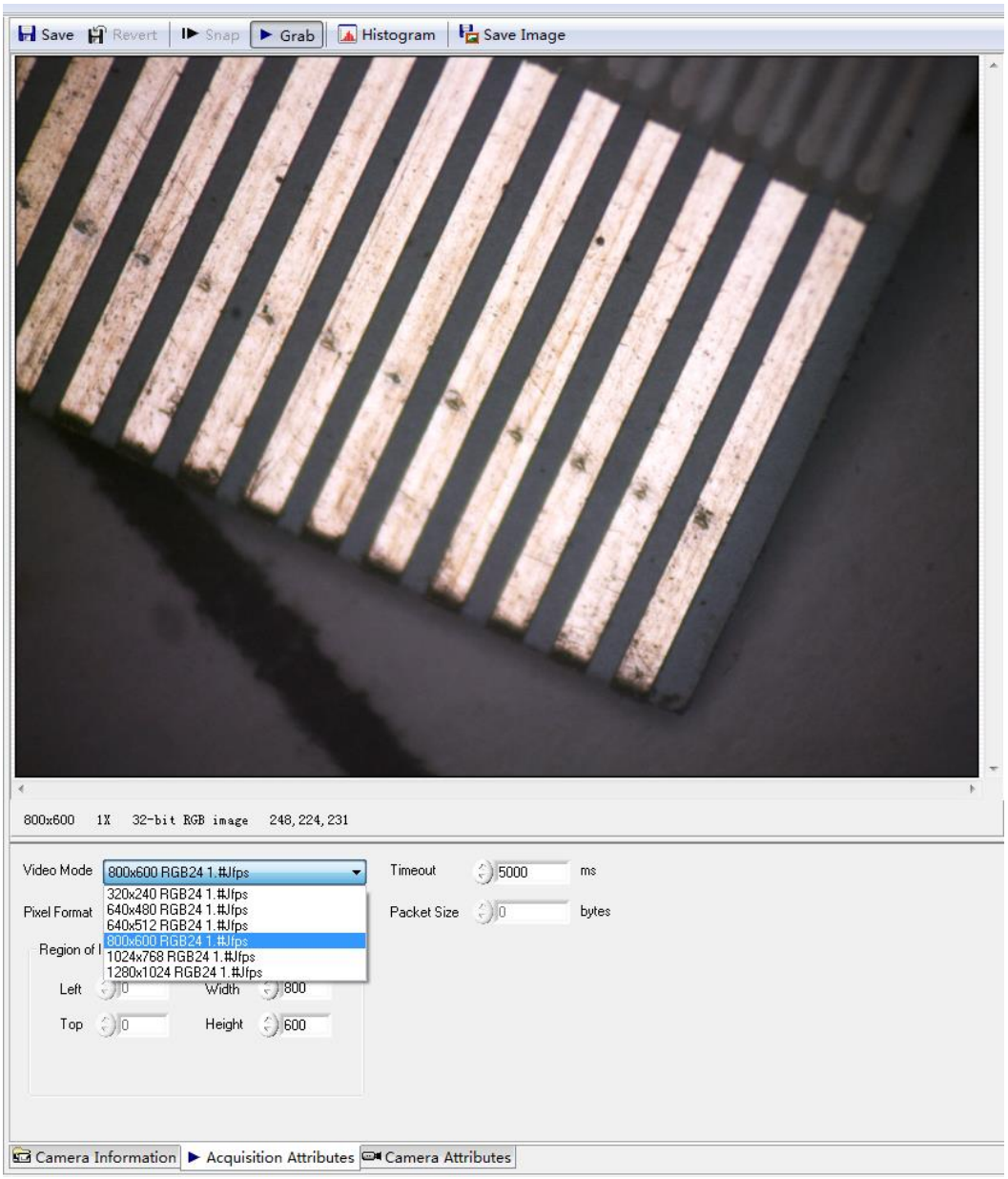

Save
Revert
Snap
Grab
Histogram
Save Image
800x600 1X 32-bit RGB image 248, 224, 231
Video Mode
800x600 RGB24 1.#Jfps
320x240 RGB24 1.#Jfps
640x480 RGB24 1.#Jfps
640x512 RGB24 1.#Jfps
800x600 RGB24 1.#Jfps
1024x768 RGB24 1.#Jfps
1280x1024 RGB24 1.#Jfps
Timeout
5000
ms
Pixel Format
Packet Size
0
bytes
Region of I
Left
0
Width
800
Top
0
Height
600
Camera Information
Acquisition Attributes
Camera Attributes

图 8.2 预览界面

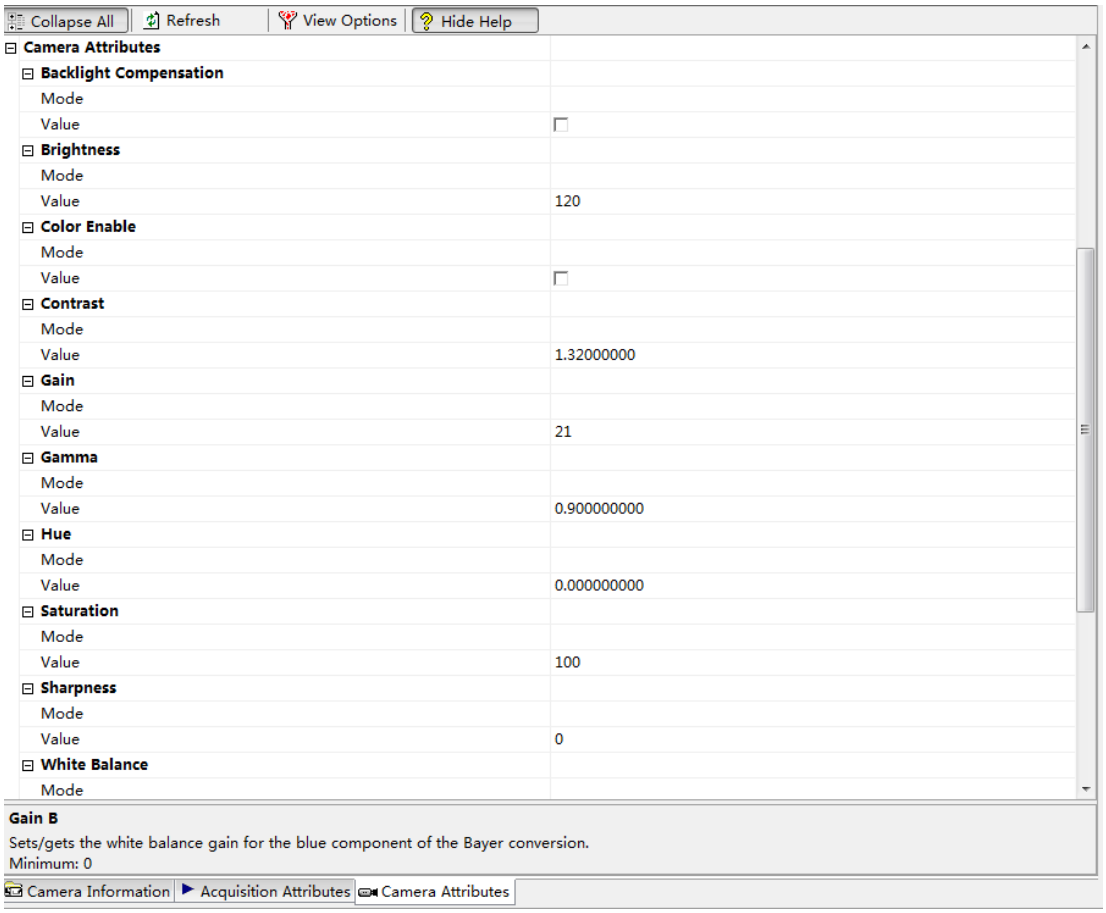


图 8.3 NI MAX 中相机的参数配置

8.2 基于 DLL 档调用方式进行开发

该方式的例程位于安装目录的 DEMO/Labview/useDLL 文件夹下，演示了如何通过调用 SDK 的 DLL 档(MVCAMSDK.dll)进行开发。该例程有如下特点：

1. 支持黑白和彩色的相机自动识别 ,黑白相机使用 U8 格式的灰度图像进行显示；彩色相机使用 U32 格式的彩色图像进行显示。
2. 支持多相机同时开发使用。只需要将 CameraInitEx 接口调用中第一个参数递增即可。输入 0 表示初始化第一个相机，输入 1 表示初始化第二个相机，其余 VI 源码可全部直接复制。

3. 该例程可适用于我司所有型号的相机，无论分辨率大小、触发模式、传输接口如何，该例程均可自动识别，无需手动修改任何变量。
4. 相机的参数支持保存和加载(档方式)。通过其他软件修改好的相机参数档，也可以被该例程加载，减少手工输入代码的工作量。

8.3 Labview 中使用多个相机

该方式的例程位于安装目录的 DEMO/Labview/TwoCameras 目录下，这个例程也是基于 DLL 方式的，演示了在 Labview 中如何同时使用多个相机。例程中给出的是 2 个相机同时使用的方法，可以是相同型号的 2 个相机，也可以是不同型号的，甚至是不同接口的多个相机，例如 1 个 USB2.0，1 个 USB3.0，1 个 GIGE 相机，都可以使用这种方式进行多相机开发。使用 2 个以上的相机同时开发时，请参考本例程中的方式进行扩展，目前 SDK 的多相机数量限制为 64 个，当需要同时使用 64 个以上的相机时，请与我们技术支持取得联系，可进一步扩展多相机支持数量。

8.4 Labview 多相机的区分

当使用多相机时开发时，往往需要每台相机的对应关系，不同的相机，将用来完成不同的任务。区分多相机的方法有很多种，可以通过相机内唯一序号、相机名称、相机内自定义数据等多种方式。在 Labview 中，我们提供以下 2 种方式来区分多相机：

- 使用自定义数据的方式。我们提供了接口，可以在相机中读写自定义的数据，您可以根据这些数据，来区分不同的相机。但是这种方法必须是已经在初始化相机后，才能读取到自定义的数据，因此 Labview 的程序设计中，必须在

相机初始化完成后，再通过接口获得自定义数据后，判断是哪一个相机，再进行相应的分支处理。（例程中已经提供了相机自定义数据的读写方式）

- 使用自定义设备名的功能。该过程和 5.3 章节 Halcon 中区分多相机类似。第一步，如图 6.4 中所示，分别将 2 台相机的设备名改成 Camera1 和 Camera2。第二步，在 Labview 中，调用 CameraInitEx2 接口来进行相机的初始化，第一个参数分别传入字符串"Camera1"和"Camera2"。修改后，名字被固化到相机内部，永久有效，不受接口、计算机更换的影响。该例程位于安装目录的 DEMO/Labview/TwoCamerasEx 目录下。

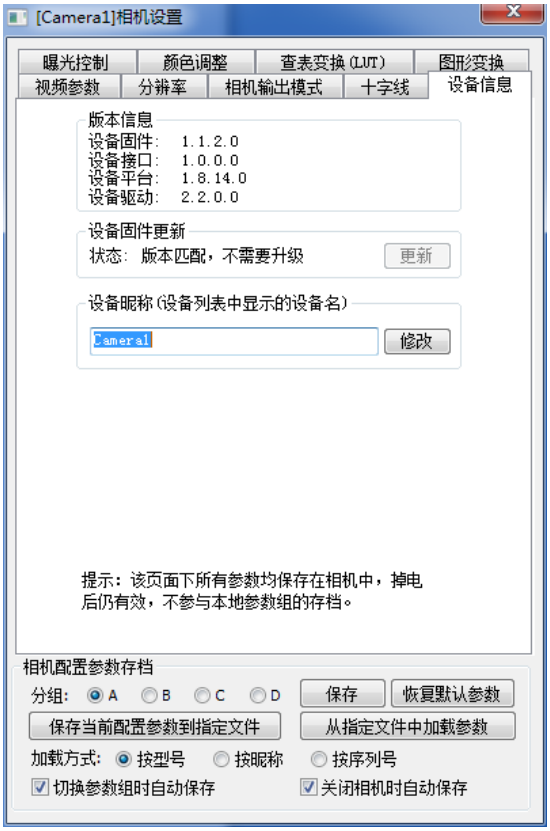


图 8.4 修改相机昵称

9 集成后发布相机安装文件

当您集成我们的相机后，需要将相机SDK文件一并打包发布时，可遵循以下的方式：

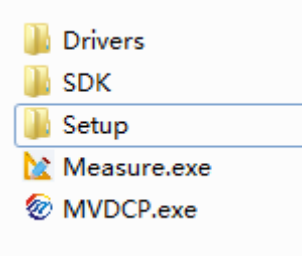
1， 需要打包的文件在我们的相机程序安装目录下，有以下文件夹：

- ./SDK目录，包含了所有的SDK文件。
- ./Drivers目录，包含了相机的内核驱动文件。
- ./Demo目录，包含了相机的开发DEMO，您可根据需要进行发布，或者完全不发布Demo程序。
- ./Camera/Configs目录下，对应的相机配置文件。该目录下保存了您当前计算机上对相机的配置参数，您可以将其一起打包发布，参数文件会被自动加载。如果不发布相机配置文件，相机第一次使用时，会使用默认参数，并在Camera/Configs目录下生成一个文件。请将以上文件和您开发的exe程序放到同一个目录下。

2， 发布方式。您可以选择以下3种方式发布。

- 使用我们的平台安装包直接安装，MindVision Platform setup(版本号).exe，但是该安装包会包含一些公司信息。
- 使用我们提供的工具。在Setup文件夹里，我们提供了一个名为“Installer.exe”程序，其作用是将SDK和Drivers目录下的所有文件进行安装。将SDK、Drivers、Setup文件夹复制到目标机器上，然后运行Setup文件夹里的" Installer.exe "即可自动完成安

装，如下图所示(MVDCP.exe和Measure.exe是我们提供的演示软件和测量软件，可以不发布，运行Installer.exe后，MVDCP和Measure软件就可以正常运行了)：



- 手动方式或者自己写安装程序进行安装。首先将Drivers目录复制到目标机器，然后手动安装设备内核驱动(USB相机在相机插入电脑时，会提示安装驱动，选择Drivers目录下对应的INF文件即可，Gige相机需要手动运行Drivers\Gige下的MvDriverInstall.exe程序就可完成内核驱动的安装)；然后将SDK(如果是X64系统，请复制SDK/X64)文件夹里所有文件(**注意：是文件夹下的文件，并不是文件夹**)复制到目标机器的任意目录（假设为C:\TEST目录），然后将您二次开发好的可执行文件，放到C:\TEST目录下，就可以直接运行了，无需进其他安装。

10技术支持

您在使用和开发的过程中遇到任何疑问，请尽快与我们的技术支持取得联系。我们建议您尽量用电子邮件、传真等的书面形式与我们交流，同时附加上问题的详细描述、截图等信息，这样有利于我们快速解决您的问题。

您的成功就是我们的成功，您的满意就是我们的满意。您回馈的任何建议都是对我们的肯定和鼓励。我们期待您的回馈，我们会努力做的更好！