

# 工业相机 SDK 开发手册

## ( C 篇 )

### Ver1.0

## 变更履历

编号	变更日期	版本号	变更内容概要	配套 SDK 版本
1	2019/02/28	Ver1.0	初版做成	

## 目录

1. SDK 概要说明.....	5
1.1. 下载 SDK 安装包.....	5
1.2. SDK 支持的操作系统.....	5
1.3. SDK 安装目录.....	6
1.3.1. Windows 版 SDK.....	6
1.3.2. Linux 版 SDK.....	7
1.4. SDK ( c 接口 ) 对应的资源.....	8
1.5. SDK 开发工程配置.....	9
1.5.1. VC6.0.....	9
1.5.2. VS2005 ~ VS2013.....	12
1.5.3. QT 工程 ( pro 文件 ) .....	16
1.5.4. Linux 下开发的工程配置方法.....	17
2. SDK 的整体调用流程 .....	19
3. SDK 的各功能接口调用说明 .....	19
3.1. 生成系统单例.....	19
3.2. 设备发现.....	20
3.3. 设备连接/断开.....	20
3.4. 设置属性.....	21
3.4.1. 使用常用属性节点.....	21
3.4.2. 使用通用属性.....	23
3.5. 采集图像.....	31

3.5.1.	创建/销毁流通道 .....	31
3.5.2.	设置图像接收缓存个数 .....	32
3.5.3.	开始/停止采集图像 .....	32
3.5.4.	获取图像 .....	33
3.6.	事件通知 .....	37
3.6.1.	设备连接状态事件 .....	37
3.6.2.	流事件 .....	39
3.6.3.	相机消息事件 .....	41
4.	第三方平台的图像对象转换方法 .....	43
4.1.	Halcon 对象 ( HObject ) .....	43
4.1.1.	相机图像格式为 Mono8 时 .....	43
4.1.2.	相机图像格式为 Mono8 以外时 ( 主要是彩色格式 ) .....	44
4.2.	OpenCV 对象 ( cv::Mat ) .....	45
4.2.1.	相机图像格式为 Mono8 时 .....	45
4.2.2.	相机图像格式为 Mono8 以外时 ( 主要是彩色格式 ) .....	45
4.3.	QT 对象 ( QImage ) .....	45
4.3.1.	相机图像格式为 Mono8 时 .....	45
4.3.2.	相机图像格式为 Mono8 以外时 ( 主要是彩色格式 ) .....	46
5.	配套例程说明 .....	46
6.	常见问题&注意点 .....	47
6.1.	读写相机属性处理存在内存泄露 .....	47
6.2.	客户的程序使用 C 接口采图 ,只能取到 8 张图像( 和默认的 SDK 缓存个数一致 ) .....	

---

.....	48
-------	----

## 1. SDK 概要说明

### 1.1. 下载 SDK 安装包

大华工业相机 SDK 可以通过华睿官网下载。

下载链接：<http://download.huaraytech.com/pub/sdk/>

该页面罗列了各个版本的 SDK。根据需要，下载相应的版本。



### 1.2. SDK 支持的操作系统

SDK 支持以下版本的操作系统

操作系统类型	操作系统版本号	备注
Windows	WinXP SP3	注意需要 SP3 补丁
	Win7 SP1 32/64bit	注意需要 SP1 补丁
	Win8 32/64bit	
	Win10 32/64bit	
Linux (x86)	Ubuntu 12.04 32/64bit	支持的内核版本范围参考*1

	Ubuntu 14.04 32/64bit	支持的内核版本范围参考*1
	Ubuntu 16.04 32/64bit	支持的内核版本范围参考*1
	Ubuntu 17.04 32/64bit	支持的内核版本范围参考*1
	Ubuntu 18.04 32/64bit	支持的内核版本范围参考*1
	CentOS 6.5~7.5 32/64bit	支持的内核版本范围参考*1

\*1. 目前支持的 Linux 内核版本号从 2.6.32 ( 含 ) 到 4.18.0 ( 含 )。

可以通过 `uname -a` 命令来确定系统内核版本号。

\*2. Linux ARM 版本的 SDK 需要定制化开发。

但针对 NVIDIA Jetson TK1、TX1、TX2 平台 ,SDK 已和 x86 平台一样标准发布。

## 1.3. SDK 安装目录

### 1.3.1. Windows 版 SDK

Windows 版本 SDK 的默认安装路径 ( \*1 ) 为 “c:\Program Files\DahuaTech\MV Viewer” , 安装目录结构如下 :

\*1. 从 Ver2.1.0 版本开始 , 支持用户自己选择安装路径。

文件夹	描述
Application	1. 相机客户端软件 ( MV Viewer ) , 包含 32 位和 64 位 2. 相机配套工具软件 ( CamTools ) , 包含 32 位和 64 位
Development	存放了 SDK 二次开发所需的各种资源文件 1. SDK 头文件、Lib 库





## 1.4. SDK ( c 接口 ) 对应的资源

大类别	小类别	文件位置
SDK 库	头文件	【SDK 安装目录】 \Development\Include\GenICam\CAPI\SDK.h
	Lib 库 ( 32 位 )	【SDK 安装目录】 \Development\Lib\win32\MVSDKmd.lib
	Lib 库 ( 64 位 )	【SDK 安装目录】 \Development\Lib\x64\MVSDKmd.lib
	运行库 ( 32 位 )	【SDK 安装目录】\Runtime \Win32\MVSDKmd.dll
	运行库 ( 64 位 )	【SDK 安装目录】 \Runtime \x64\MVSDKmd.dll
转码库	头文件	【SDK 安装目录】 \Development\Samples\MFC\SingleDisplay\Include\Media\ImageConvert.h
	Lib 库 ( 32 位 )	【SDK 安装目录】 \Development\Samples\MFC\SingleDisplay\Depends\win32\vs2013shared\ImageConvert.lib
	Lib 库 ( 64 位 )	【SDK 安装目录】 \Development\Samples\MFC\SingleDisplay\Depends\x64\vs2013shared\ImageConvert.lib
	运行库 ( 32 位 )	【SDK 安装目录】 \Development\Samples\MFC\SingleDisplay\Bin\win32\release\ImageConvert.dll
	运行库 ( 64 位 )	【SDK 安装目录】 \Development\Samples\MFC\SingleDisplay\Bin\x64\release\ImageConvert.dll
开发文档	中文	【SDK 安装目录】 \Documentations\ GenICam_API_C_CHS.chm

		【SDK 安装目录】\Documentations\工业相机 SDK 开发手册(C 篇).pdf
	英文	【SDK 安装目录】\Documentations\GenICam_API_C_ENG.chm

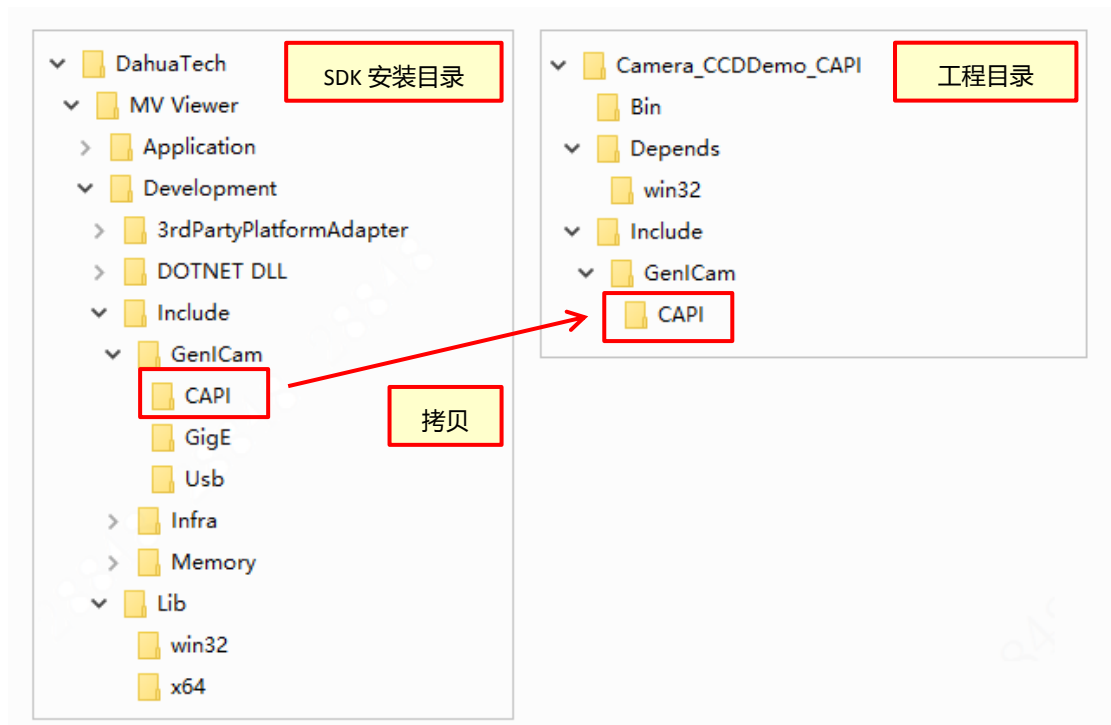
## 1.5. SDK 开发工程配置

### 1.5.1. VC6.0

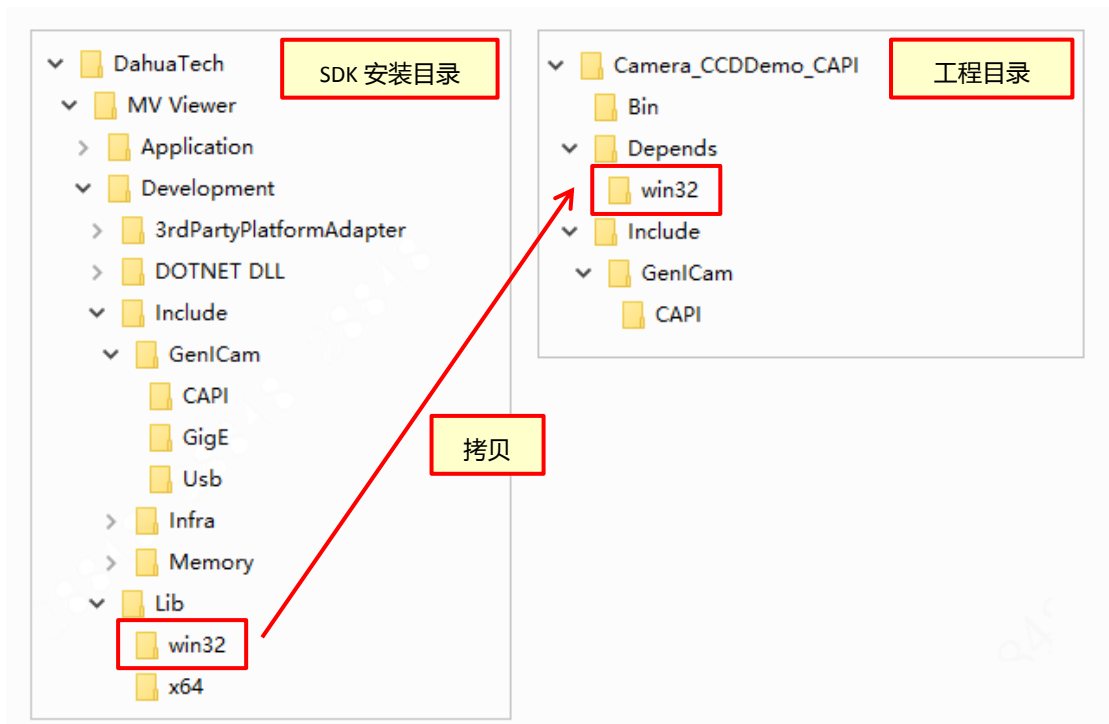
下面以 Win32 | Debug 组合为例，说明工程配置方法。

#### (1) 拷贝 SDK 相关资源文件

##### ① 拷贝头文件



##### ② 拷贝 Lib 库



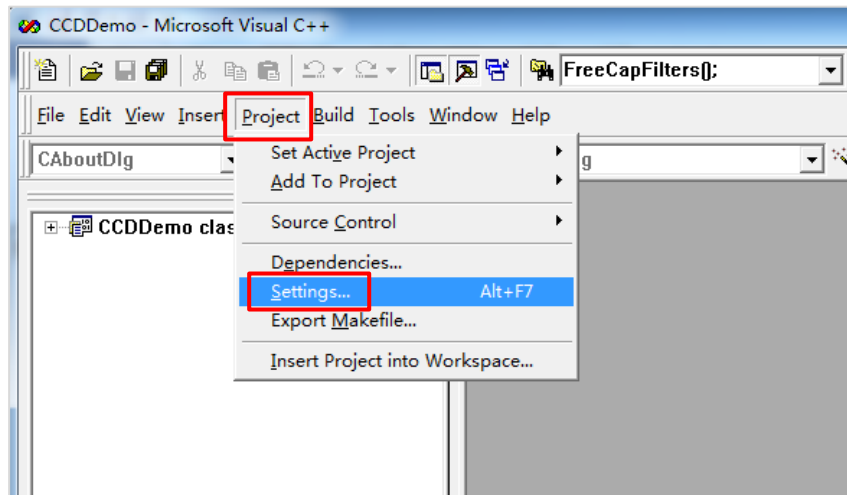
### ③ 拷贝运行时动态库

不需要拷贝 DLL 库。

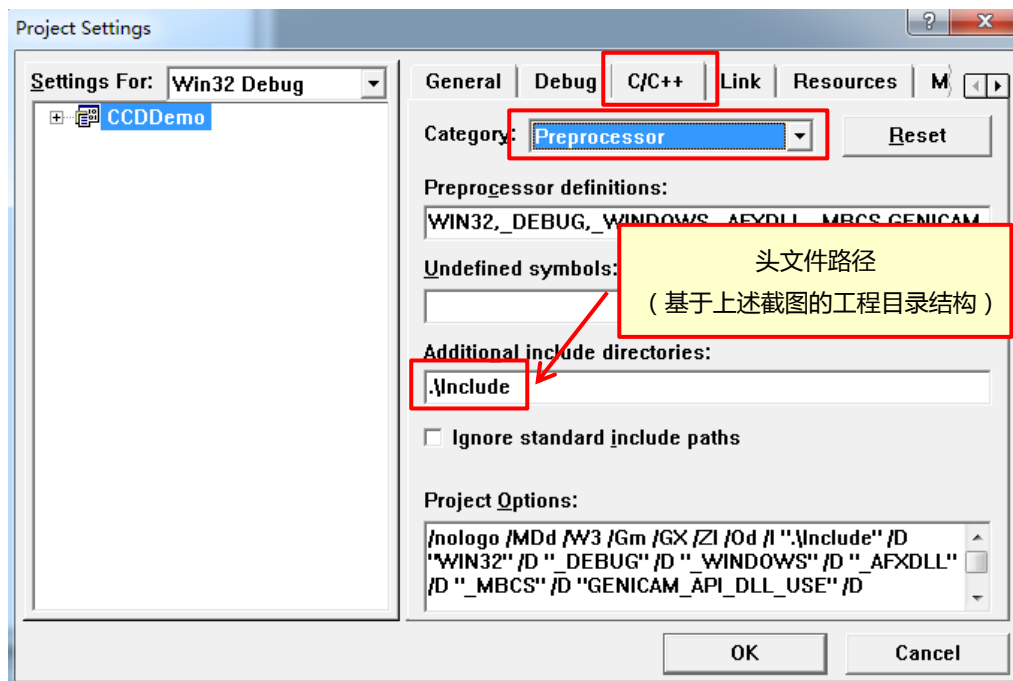
在安装 SDK 的环境里，运行程序时，会自动通过 PATH 环境变量找到 “【SDK 安装目录】\Runtime” 路径下，对应平台的运行时动态库。

## (2) 配置工程

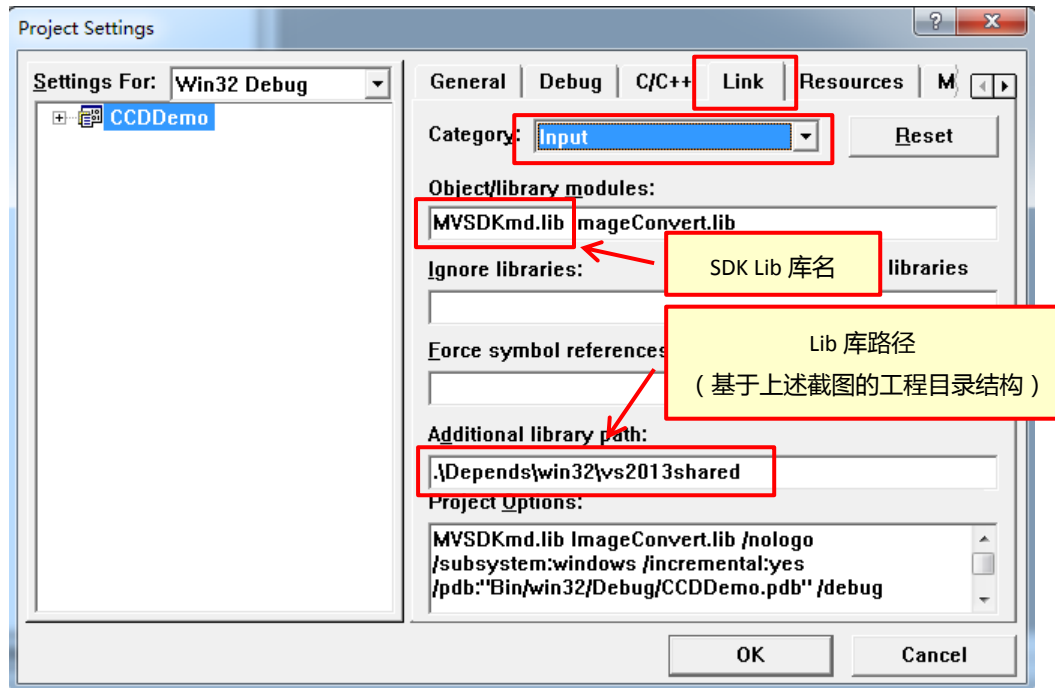
### ① 打开工程配置对话框



## ② 配置头文件



## ③ 配置 Lib 库



### 1.5.2. VS2005 ~ VS2013

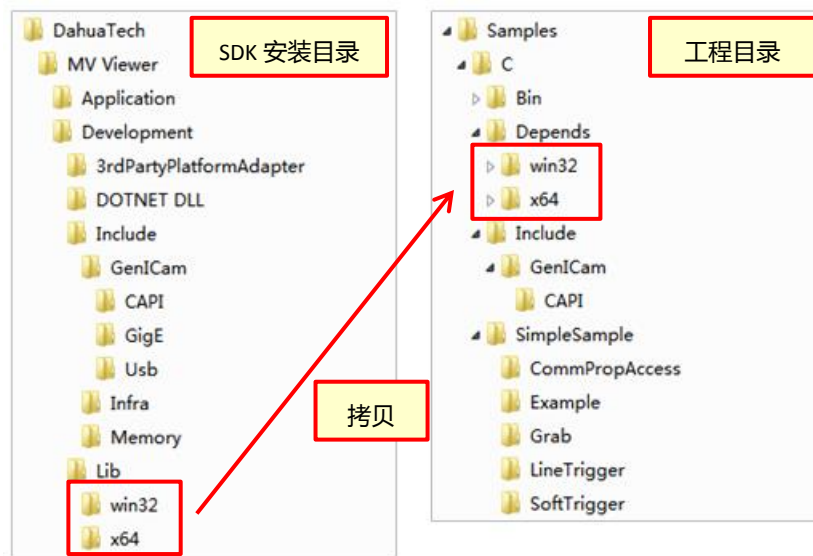
以 Debug | Win32 为例，其它组合参考该配置。

(1) 拷贝 SDK 相关资源文件

① 拷贝头文件



## ② 拷贝 Lib 库



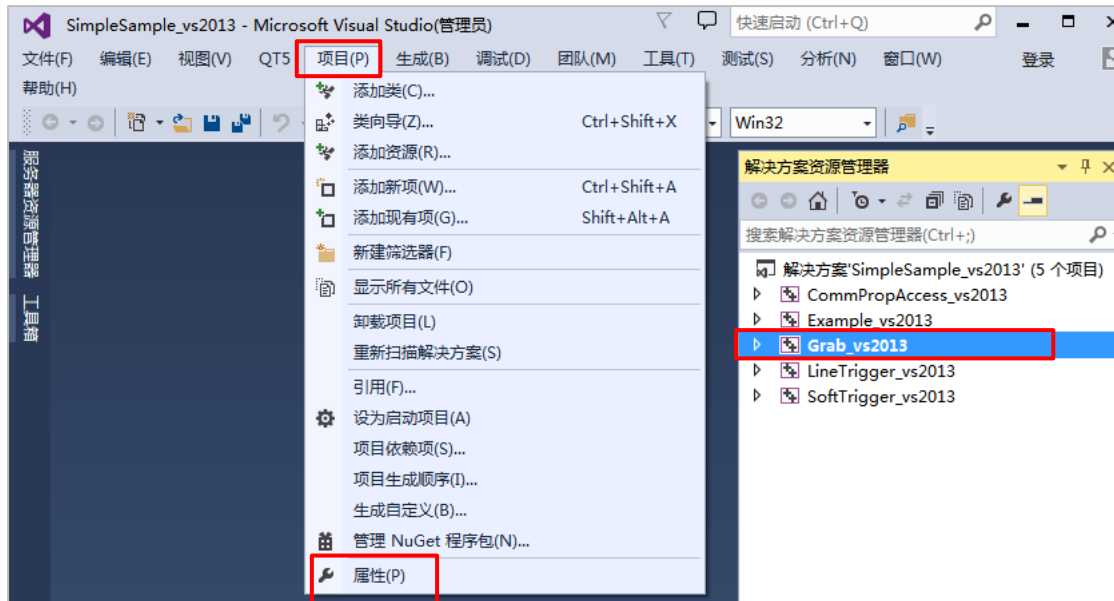
## ③ 拷贝运行时动态库

不需要拷贝 DLL 库。

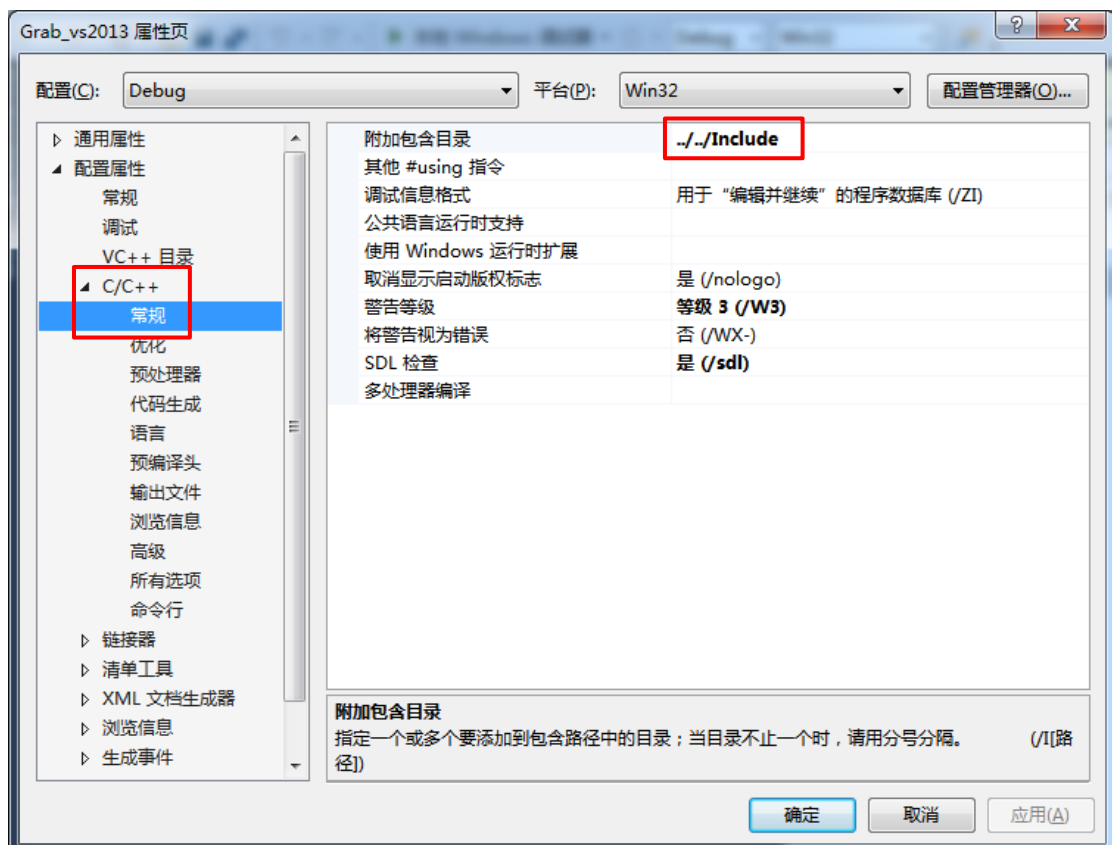
在安装 SDK 的环境里，运行程序时，会自动通过 PATH 环境变量找到 “【SDK 安装目录】\Runtime” 路径下，对应平台的运行时动态库。

## (2) 配置工程

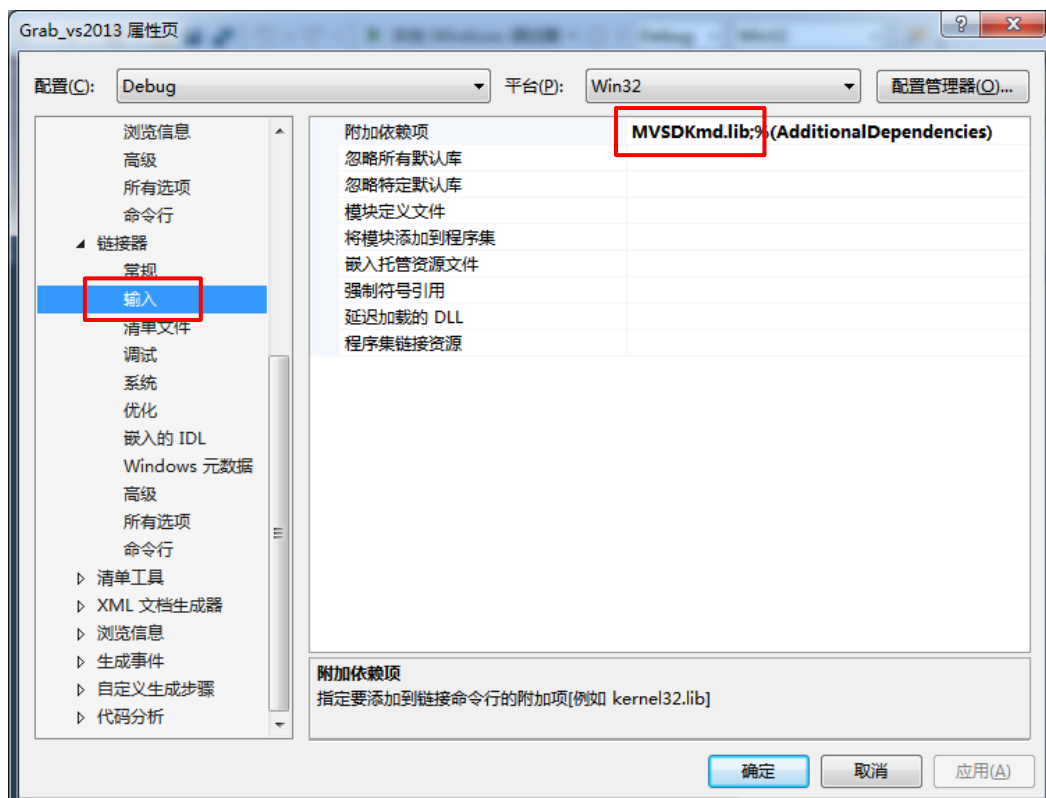
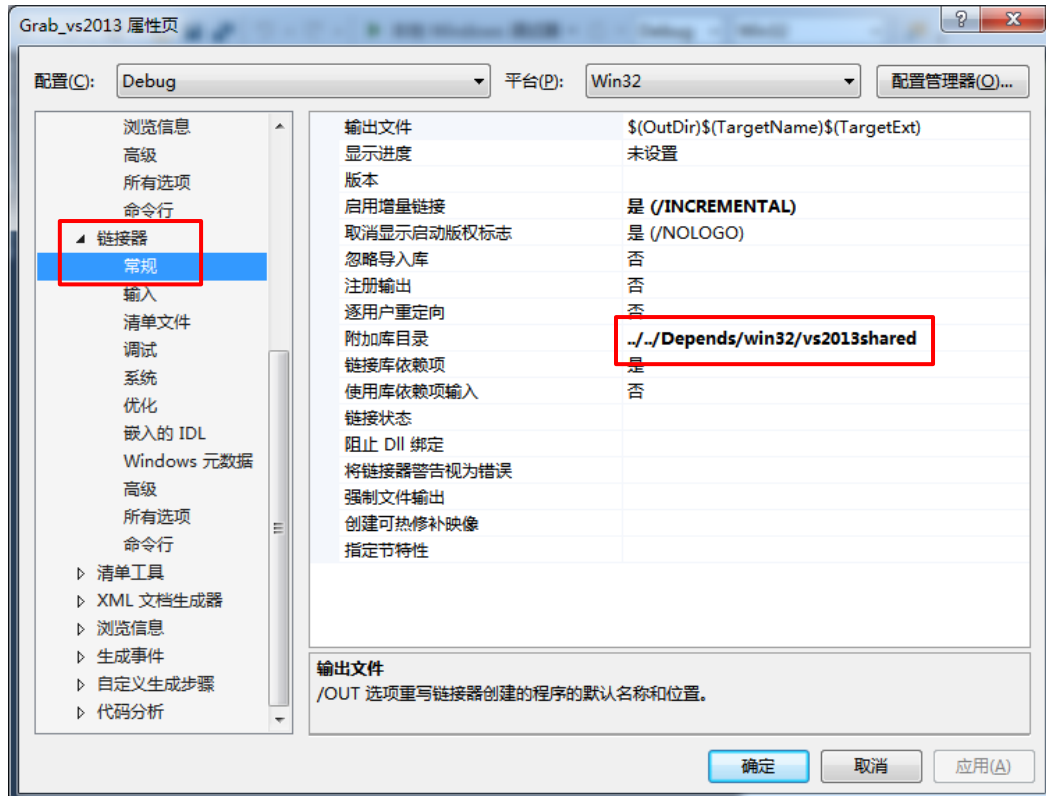
### ① 打开工程配置对话框



## ② 配置头文件



## ③ 配置 Lib 库

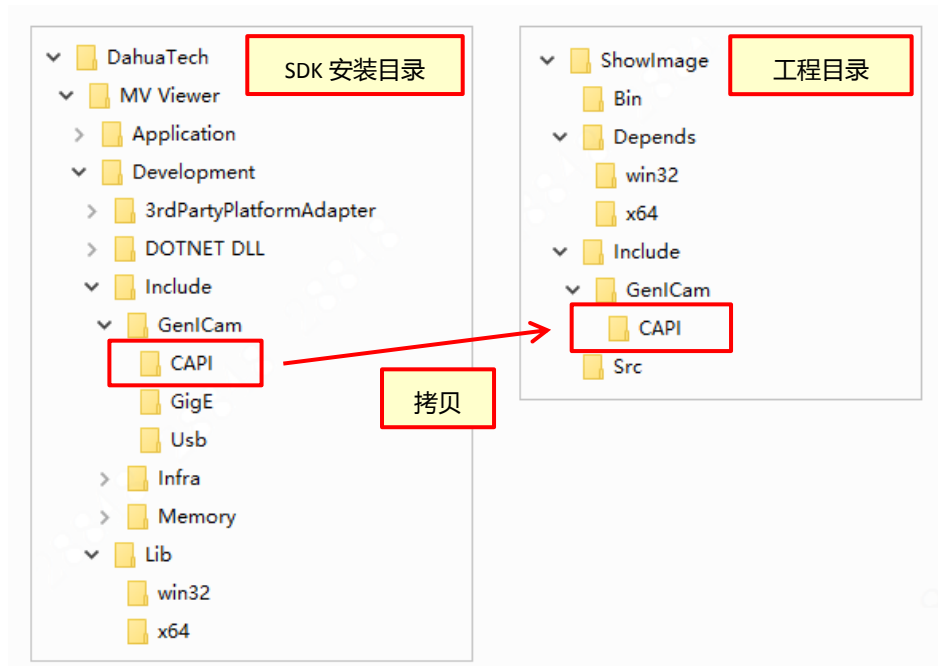




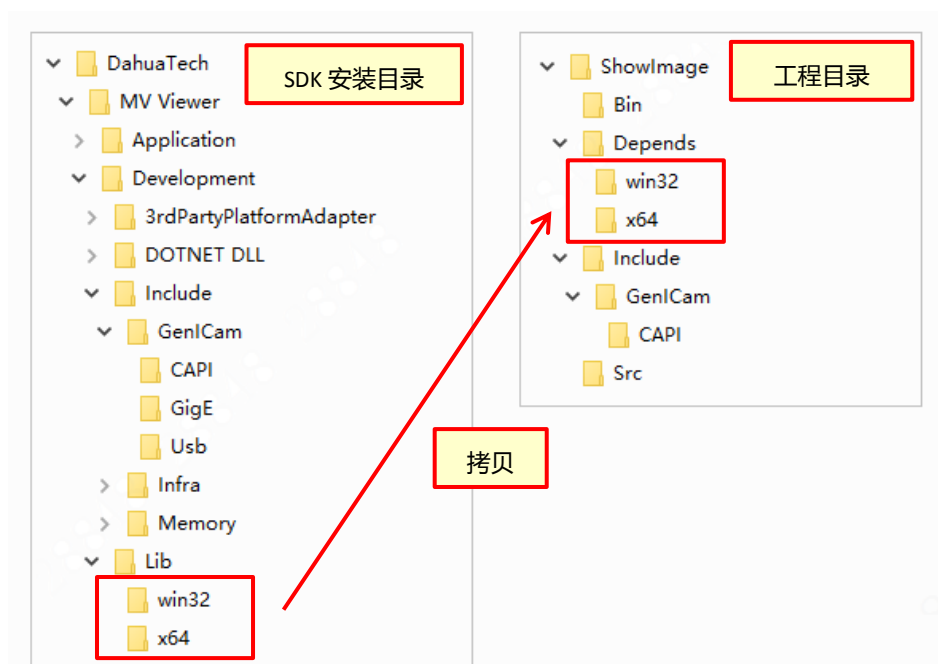
### 1.5.3. QT 工程 ( pro 文件 )

#### (1) 拷贝 SDK 相关资源文件

##### ① 拷贝头文件



##### ② 拷贝 Lib 库



### ③ 拷贝运行时动态库

不需要拷贝 DLL 库。

在安装 SDK 的环境里，运行程序时，会自动通过 PATH 环境变量找到 “【SDK 安装目录】\Runtime” 路径下，对应平台的运行时动态库。

## (2) 配置 QT 工程 pro 文件

在 pro 文件中加入下列配置，配置依赖的头文件和 lib 库：

```
Debug {
    contains(QMAKE_COMPILER_DEFINES, _WIN64) {
        LIBS += -L./Depends/x64/vs2013shared      -IMVSDKmd
    }
    else {
        LIBS += -L./Depends/win32/vs2013shared    -IMVSDKmd
    }
}
else {
    contains(QMAKE_COMPILER_DEFINES, _WIN64) {
        LIBS += -L./Depends/x64/vs2013shared      -IMVSDKmd
    }
    else {
        LIBS += -L./Depends/win32/vs2013shared    -IMVSDKmd
    }
}
INCLUDEPATH += ./Include
```

## 1.5.4. Linux 下开发的工程配置方法

### (1) 拷贝 SDK 相关资源文件

#### ① 拷贝头文件



## ② 拷贝动态库



## ③ 配置 Makefile 文件

在 Makefile 中, 修改 INCLUDES 和 LINKLIBS 部分。

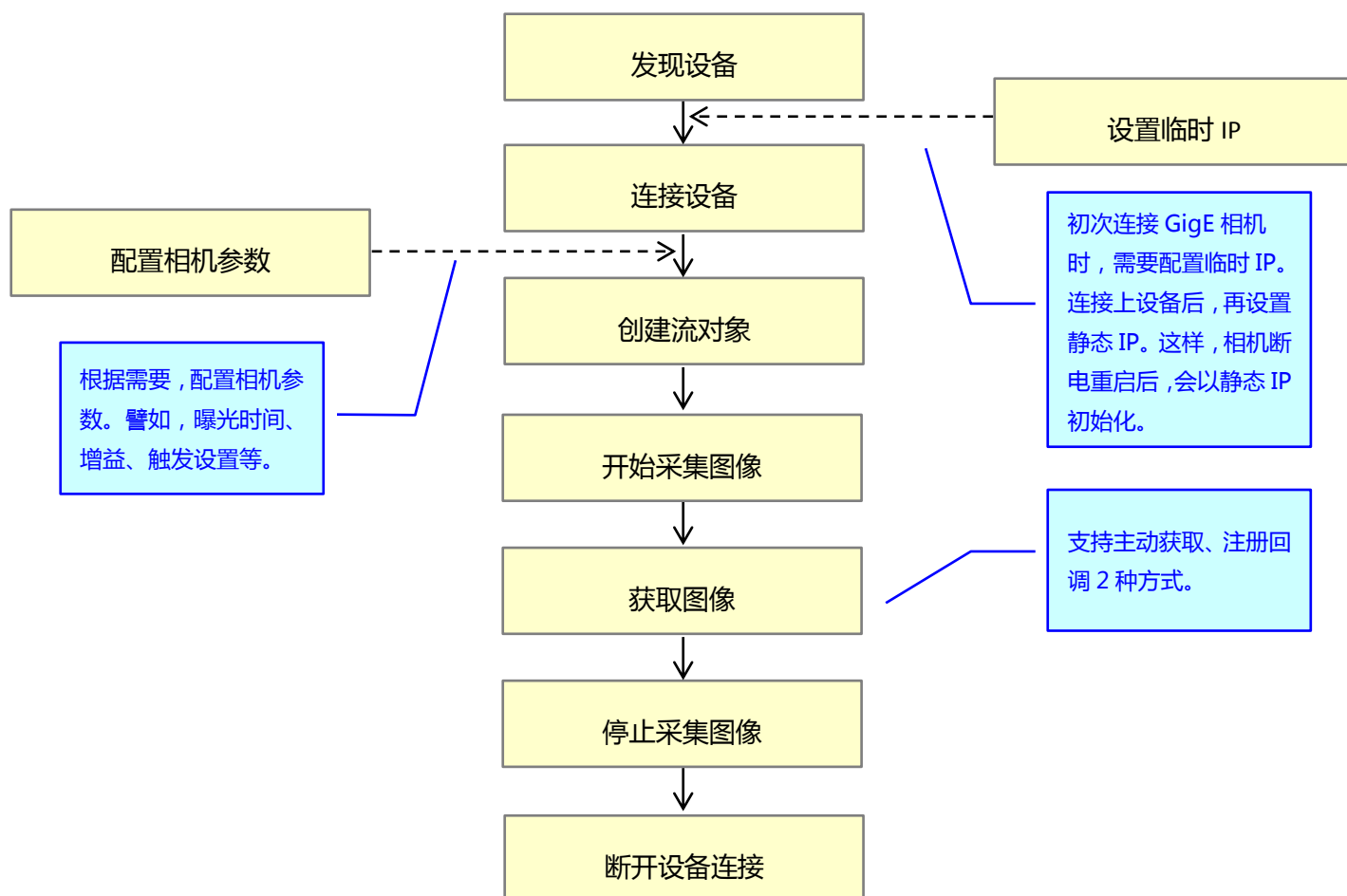
将上述拷贝的头文件路径、动态库路径及名称, 按以下示例追加进去。

```
... (省略)

INCLUDES = -I./include
LINKLIBS = -L./depends/ -lMVSDK

... (省略)
```

## 2. SDK 的整体调用流程



## 3. SDK 的各功能接口调用说明

SDK 接口头文件：GenICam/CAPI/SDK.h

转码库接口头文件：ImageConvert.h

### 3.1. 生成系统单例

接口：int32\_t GENICAM\_getSystemInstance(GENICAM\_System\*\* ppSystem);

功能说明：创建系统单例对象，全局唯一。执行成功时返回值为 0，失败时返回值小于 0。

[out] ppSystem 系统单例指针

代码范例：

```
GENICAM_System *pSystem = NULL;
int32_t ret = GENICAM_getSystemInstance(&pSystem);
```

## 3.2. 设备发现

接口：int32\_t (\*discovery)(struct GENICAM\_System \*thiz, GENICAM\_Camera \*\*ppCameraList, uint32\_t \*pCameraCnt, GENICAM\_EProtocolType interfaceType);

功能说明：发现相机。执行成功时返回值为 0，失败时返回值小于 0。

[in] thiz 系统单例指针

[out] ppCameraList 发现的相机列表

[out] pCameraCnt 相机个数

[in] interfaceType 需要发现的相机类型

代码范例（先调用 3.1 生成系统单例）：

```
GENICAM_Camera *pCameraList = NULL;
uint32_t cameraCnt = 0;
ret = pSystem->discovery(pSystem, &pCameraList, &cameraCnt, typeAll);
```

## 3.3. 设备连接/断开

接口：int32\_t (\*connect)(struct GENICAM\_Camera \*thiz, GENICAM\_ECameraAccessPermission accessPermission);

功能说明：连接相机。执行成功时返回值为 0，失败时返回值小于 0。

[in] thiz 相机对象指针

[in] accessPermissionk 以何种权限连接相机。目前只支持 accessPermissionControl 权限。

代码范例：

```
pCamera = &pCameraList[0];  
ret = pGetCamera->connect(pGetCamera, accessPermissionControl);
```

接口：int32\_t (\*disconnect)(struct GENICAM\_Camera \*thiz);

功能说明：断开连接。执行成功时返回值为 0，失败时返回值小于 0。

[in] thiz 相机对象指针

代码范例：

```
ret = pGetCamera->disconnect(pGetCamera);
```

## 3.4. 设置属性

注意，读写属性之前必须先连接相机。

SDK 提供了 2 种方式读写属性：

- 通过预先给定的常用属性节点
- 通过通用属性方法

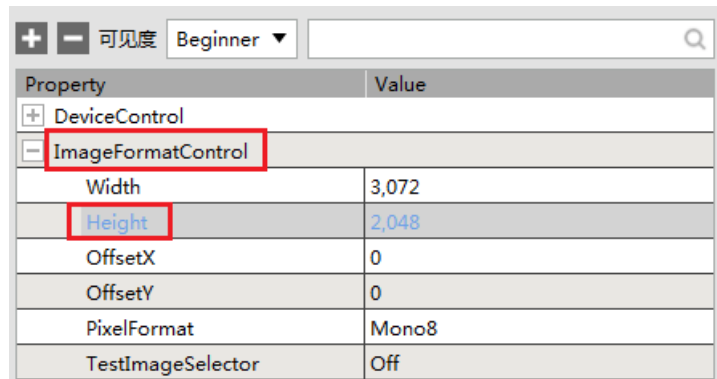
### 3.4.1. 使用常用属性节点

#### (1) 确定属性节点

打开相机客户端，找到要对应的属性，查看该属性属于哪一个属性类别。

例如，要获取相机的像素高度信息，找到 Height 属性，它属于 ImageFormatControl，

则创建 ImageFormatControl 节点。



Property	Value
DeviceControl	
ImageFormatControl	
Width	3,072
Height	2,048
OffsetX	0
OffsetY	0
PixelFormat	Mono8
TestImageSelector	Off

## (2) 代码范例 ( 以 Height 属性为例 )

// 创建 ImageFormatControl 属性节点的信息

```
GENICAM_ImageFormatControllInfo imageFormatControllInfo = { 0 };
```

```
imageFormatControllInfo.pCamera = pCamera;    // pCamera 是已经连接的相机
```

// 创建 ImageFormatControl 属性节点

```
GENICAM_ImageFormatControl *pImageFormatCtrl = NULL;
```

```
if (0 != GENICAM_createImageFormatControl(&imageFormatControllInfo, &pImageFormatCtrl))
{
```

```
    // 注意：需要调用 release 释放内存
```

```
    pImageFormatCtrl->release(pImageFormatCtrl);
```

```
    return -1;
```

```
}
```

// 获取属性操作对象。

// 如果头文件中没有该属性对应的获取属性操作对象的接口，则说明没有提供。

// 那么需要通过 3.4.2 的通用属性方式来设置。

```
GENICAM_IntNode intNode = pImageFormatCtrl->height(pImageFormatCtrl);
```

```
if (0 != intNode.isValid(&intNode))
{
    // 注意：需要调用 release 释放内存

    plImageFormatCtrl->release(plImageFormatCtrl);
    intNode.release(&intNode);
    return -1;
}

// 获取属性值
int64_t nHeight;
if (0 != intNode.getValue(&intNode, &nHeight))
{
    // 注意：需要调用 release 释放内存

    plImageFormatCtrl->release(plImageFormatCtrl);
    intNode.release(&intNode);
    return -1;
}

// 设置属性值
if (0 != intNode.setValue(&intNode, nHeight))
{
    // 注意：需要调用 release 释放内存

    plImageFormatCtrl->release(plImageFormatCtrl);
    intNode.release(&intNode);
    return -1;
}

// 注意：需要调用 release 释放内存

plImageFormatCtrl->release(plImageFormatCtrl);
intNode.release(&intNode);
```

### 3.4.2. 使用通用属性

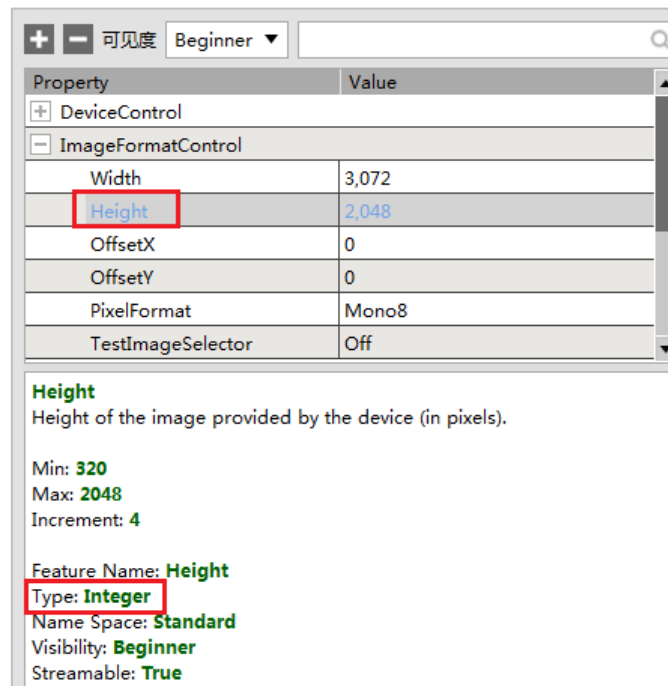
使用3.4.1的属性节点方式只能设置现有提供的属性。



对于其他属性，可以通过通用属性的方式来设置。

打开相机客户端，找到要对应的属性，选中该属性，在属性列表下方会展示该属性的相关信息，其中Type即为该属性的类型。

例如，选中Height属性，Type为Integer，说明Height是整数类型。



### ① 整型属性

以Width属性为例说明

// 创建整数节点类型信息

```
GENICAM_IntNodeInfo intNodeInfo = { 0 };
```

```
intNodeInfo.pCamera = pCamera; // pCamera 是已经连接的相机
```

```
memcpy(intNodeInfo.attrName, "Width", sizeof("Width")); //属性名字
```

// 创建整数节点类型对象

```
GENICAM_IntNode *pIntNode = NULL;
```

```
if (0 != GENICAM_createIntNode(&intNodeInfo, &pIntNode))
{
    return -1;
}
```

// 获取属性值

```
int64_t nWidthValue = 0;
if (0 != pIntNode->getValue(pIntNode, &nWidthValue))
{
    // 注意：需要释放 pIntNode 内部对象内存
    pIntNode->release(pIntNode);
    return -1;
}
```

// 设置属性值

```
if (0 != pIntNode->setValue(pIntNode, nWidthValue))
{
    // 注意：需要释放 pIntNode 内部对象内存
    pIntNode->release(pIntNode);
    return -1;
}
```

// 注意：需要释放 pIntNode 内部对象内存

```
pIntNode->release(pIntNode);
```

## ② 浮点型属性

以 ExposureTime 属性为例说明

// 创建浮点数节点类型信息

```
GENICAM_DoubleNodeInfo doubleNodeInfo = {0};
```

```
doubleNodeInfo.pCamera = pCamera //pCamera 是已经连接的相机
```

```
memcpy(doubleNodeInfo.attrName, "ExposureTime", sizeof("ExposureTime")); //属性名字
```

```
// 创建浮点数节点类型对象

GENICAM_DoubleNode *pDoubleNode = NULL;
if (0 != GENICAM_createDoubleNode(&doubleNodeInfo, &pDoubleNode))
{
    return -1;
}

// 获取属性值

double dExposureTimeValue = 0.0;
if (0 != pDoubleNode->getValue(pDoubleNode, &dExposureTimeValue))
{
    // 注意：需要释放 pDoubleNode 内部对象内存

    pDoubleNode->release(pDoubleNode);
    return -1;
}

// 设置属性值

if (0 != pDoubleNode->setValue(pDoubleNode, dExposureTimeValue))
{
    // 注意：需要释放 pDoubleNode 内部对象内存

    pDoubleNode->release(pDoubleNode);
    return -1;
}

// 注意：需要释放 pDoubleNode 内部对象内存

pDoubleNode->release(pDoubleNode);
```

### ③ 布尔型属性

以ReverseX属性为例说明

```
// 创建布尔节点类型信息
```

```
GENICAM_BoolNodeInfo boolNodeInfo = {0};
```

```
boolNodeInfo.pCamera = pCamera; // pCamera 是已经连接的相机
```

```
memcpy(boolNodeInfo.attrName, "ReverseX", sizeof("ReverseX")) // 属性名字
```

```
// 创建布尔节点类型对象
```

```
GENICAM_BoolNode *pBoolNode = NULL;
```

```
if (0 != GENICAM_createBoolNode(&boolNodeInfo, &pBoolNode))
```

```
{
```

```
    return -1;
```

```
}
```

```
// 获取属性值
```

```
uint32_t nReverseXValue = 0;
```

```
if (0 != pBoolNode->getValue(pBoolNode, &nReverseXValue))
```

```
{
```

```
    // 注意：需要释放 pBoolNode 内部对象内存
```

```
    pBoolNode->release(pBoolNode);
```

```
    return -1;
```

```
}
```

```
// 设置属性值
```

```
if (0 != pBoolNode->setValue(pBoolNode, 1))
```

```
{
```

```
    // 注意：需要释放 pBoolNode 内部对象内存
```

```
    pBoolNode->release(pBoolNode);
```

```
    return -1;
```

```
}
```

```
// 注意：需要释放 pBoolNode 内部对象内存
```

```
pBoolNode->release(pBoolNode);
```

#### ④ 枚举型属性

以 TriggerSelector 属性为例说明

```
// 创建枚举节点类型信息
GENICAM_EnumNodeInfo enumNodeInfo = {0};

enumNodeInfo.pCamera = pCamera; // pCamera 是已经连接的相机

memcpy(enumNodeInfo.attrName, "TriggerSelector", sizeof("TriggerSelector")); // 属性名字

GENICAM_EnumNode *pEnumNode = NULL;
if (0 != GENICAM_createEnumNode(&enumNodeInfo, &pEnumNode))
{
    return -1;
}

// 获取枚举属性 symbol 值
char triggerSelectorValue[256] = { 0 };
uint32_t maxCnt;
if (0 != pEnumNode->getValueSymbol(pEnumNode, triggerSelectorValue, &maxCnt))
{
    // 注意：需要释放 pEnumNode 内部对象内存

    pEnumNode->release(pEnumNode);
    return -1;
}

// 设置枚举属性 symbol 值
if (0 != pEnumNode->setValueBySymbol(pEnumNode, "FrameStart"))
{
    // 注意：需要释放 pEnumNode 内部对象内存

    pEnumNode->release(pEnumNode);
    return -1;
}
```

```
// 注意：需要释放 pEnumNode 内部对象内存
```

```
pEnumNode->release(pEnumNode);
```

### ⑤ 字符型属性

以 DeviceUserID 属性为例说明

```
// 创建字符串节点类型信息
```

```
GENICAM_StringNodeInfo stringNodeInfo = {0};
```

```
stringNodeInfo.pCamera = pCamera; // pCamera 是已经连接的相机
```

```
memcpy(stringNodeInfo.attrName, "DeviceUserID", sizeof("DeviceUserID")); // 属性名字
```

```
// 创建字符串节点类型对象
```

```
GENICAM_StringNode *pStringNode = NULL;
```

```
if (0 != GENICAM_createStringNode(&stringNodeInfo, &pStringNode))
```

```
{
```

```
    return -1;
```

```
}
```

```
// 获取属性值
```

```
char szDeviceUserID[256] = { 0 };
```

```
uint32_t maxCnt;
```

```
if (0 != pStringNode->getValue(pStringNode, &szDeviceUserID, &maxCnt))
```

```
{
```

```
    // 注意：需要释放 pStringNode 内部对象内存
```

```
    pStringNode->release(pStringNode);
```

```
    return -1;
```

```
}
```

```
//设置属性值
```

```
if (0 != pStringNode->setValue(pStringNode, "camera"))
```

```
{
```

```
// 注意：需要释放 pStringNode 内部对象内存  
pStringNode->release(pStringNode);  
return -1;  
}  
  
// 注意：需要释放 pStringNode 内部对象内存  
pStringNode->release(pStringNode);
```

## ⑥ 命令型属性

以 TriggerSoftware 属性为例说明

```
// 创建命令节点类型信息  
GENICAM_CmdNodeInfo cmdNodeInfo = {0};  
  
cmdNodeInfo.pCamera = pCamera; // pCamera 是已经连接的相机  
  
memcpy(cmdNodeInfo.attrName, "TriggerSoftware", sizeof("TriggerSoftware")); // 属性名  
  
// 创建命令节点类型对象  
GENICAM_CmdNode *pCmdNode = NULL;  
if (0 != GENICAM_createCmdNode(&cmdNodeInfo, &pCmdNode))  
{  
    return -1;  
}  
  
// 执行命令类型属性  
if (0 != pCmdNode->execute(pCmdNode))  
{  
    // 注意：需要释放 pCmdNode 内部对象内存  
    pCmdNode->release(pCmdNode);  
    return -1;  
}
```

```
// 注意：需要释放 pCmdNode 内部对象内存  
pCmdNode->release(pCmdNode);
```

## 3.5. 采集图像

### 3.5.1. 创建/销毁流通道

#### (1) 创建流对象

```
// 流对象参数  
GENICAM_StreamSourceInfo streamSourceInfo;  
streamSourceInfo.pCamera = pCamera;           // pCamera 是已经连接的相机  
streamSourceInfo.channelId = 0;                // 通道号（固定为 0）  
  
// 创建流对象  
GENICAM_StreamSource *pStreamSource = NULL;  
if (0 != GENICAM_createStreamSource(&streamSourceInfo, &pStreamSource))  
{  
    return -1;  
}
```

#### (2) 销毁流对象

```
// 销毁流对象  
pStreamSource->release(pStreamSource);
```



### 3.5.2. 设置图像接收缓存个数

接口 : `int32_t(*setBufferCount)(struct GENICAM_StreamSource *thiz, uint32_t nSize);`

功能说明 : 连接相机。执行成功时返回值为 0 , 失败时返回值为-1.

缓存个数一般使用默认值(8) , 只有在图像分辨率较大时才使用该接口设置一个较小的缓存个数 , 并且要在开始拉流之前调用。

[in] thiz 流对象指针

[in] nSize 缓存数量

### 3.5.3. 开始/停止采集图像

接口 : `int32_t (*startGrabbing)(struct GENICAM_StreamSource *thiz, uint64_t maxImagesGrabbed, GENICAM_EGrabStrategy strategy);`

功能说明 : 开始抓图。执行成功时返回值为 0 , 失败时返回值为-1.

[in] thiz 流对象指针

[in] maxImagesGrabbed 允许最多的抓图数 , 固定设置为 0.

[in] strategy 抓图策略。

grabStrategySequential 表示按到达顺序处理图片

grabStrategyLatestImage 表示获取获取最新的图片

接口 : `int32_t (*stopGrabbing)(struct GENICAM_StreamSource *thiz);`

功能说明 : 停止抓图。执行成功时返回值为 0 , 失败时返回值为-1.

[in] thiz 流对象指针

### 3.5.4. 获取图像

#### (1) 主动采图

接口 : `int32_t (*getFrame)(struct GENICAM_StreamSource *thiz, GENICAM_Frame **ppFrame, uint32_t timeoutMS);`

功能说明：获取一帧图像。执行成功时返回值为 0，失败时返回值为-1.

[in] thiz 流对象指针

[out] ppFrame 一帧图像,内存由函数内部分配，用完该帧后需要调用 release 接口显示释放。

[in] timeoutMS 获取一帧图像的超时时长,单位 MS。当值为 INFINITE 时表示无限等待。

#### (2) 注册回调采图

##### ① 注册数据帧回调函数

接口 : `int32_t (*attachGrabbing)(struct GENICAM_StreamSource *thiz, callbackFun proc);`

功能说明：注册数据帧回调函数。执行成功时返回值为 0，失败时返回值为-1.

注意：该接口需要在开始拉流之前调用。

[in] thiz 流对象指针

[out] proc 数据帧回调函数。建议不要在该函数中处理耗时的操作，否则会阻塞后续数据帧的实时性。

接口 : `int32_t(*attachGrabbingEx)(struct GENICAM_StreamSource *thiz, callbackFunEx`

```
proc, void* pUser);
```

功能说明：注册数据帧回调函数，可传入用户数据用以区分相机。

执行成功时返回值为 0，失败时返回值为-1.

注意：该接口需要在开始拉流之前调用。

[in] thiz 流对象指针

[out] proc 数据帧回调函数。建议不要在该函数中处理耗时的操作，否则会阻塞后续数据帧的实时性。

[in] pUser 用户数据

## ② 反注册数据帧回调函数

接口： `int32_t(*detachGrabbing)(struct GENICAM_StreamSource *thiz, callbackFun proc);`

功能说明：反注册数据帧回调函数。执行成功时返回值为 0，失败时返回值为-1.

注意：该接口需要在停止拉流之前调用。

[in] thiz 流对象指针

[out] proc 数据帧回调函数。

接口： `int32_t(*detachGrabbingEx)(struct GENICAM_StreamSource *thiz, callbackFunEx proc, void* pUser);`

功能说明：反注册数据帧回调函数，可传入用户数据用以区分相机。

执行成功时返回值为 0，失败时返回值为-1.

注意：该接口需要在停止拉流之前调用。

[in] thiz 流对象指针

[out] proc 数据帧回调函数。

[in] pUser 用户数据

### (3) 获取图像裸数据

使用主动采图或回调取图方式，可以获取到相机采集的一帧图像数据 ( GENICAM\_Frame 对象 )。GENICAM\_Frame 对象提供了获取图像的裸数据、裸数据长度、图像宽高等信息的接口。此处仅描述获取图像裸数据接口。

接口： `const void* (*getImage)(struct GENICAM_Frame *thiz);`

功能说明：返回该帧图片裸数据的内存首地址。

[in] thiz 图像结构指针

补充说明：可以通过 `getImageSize` 接口，获取图像裸数据 ( RAW Data ) 的长度。这样从首地址开始，偏移相应的长度，就可以获得整个图像裸数据。

### (4) 图像格式转换

当相机出图的图像格式为 Mono8 或者 RGB8 时，通过 3.5.4 - (3) 获得到的裸数据 ( RAW Data ) 可以直接使用。

当相机出图的图像格式为其他格式时，通过 3.5.4 - (3) 获得到的裸数据 ( RAW Data ) 则要进一步使用 `ImageConvert.dll` 转换为所需的对应格式才能使用 ( BGR24、RGB24、Mono8 )。

头文件： `ImageConvert.h`，路径参考 1.4 SDK ( c 接口 ) 对应的资源

接口：参照下表

函数名	功能
IMGCNV_ConvertToBGR24	转换为 BGR24 的转换函数
IMGCNV_ConvertToRGB24	转换为 RGB24 的转换函数
IMGCNV_ConvertToMono8	转换为 Mono8 的转换函数
IMGCNV_ConvertToBGR24_Ex	转换为 BGR24 的转换函数，并支持选择转换 Bayer 格式所用的算法
IMGCNV_ConvertToRGB24_Ex	转换为 RGB24 的转换函数，并支持选择转换 Bayer 格式所用的算法
IMGCNV_ConvertToMono8_Ex	转换为 Mono8 的转换函数，并支持选择转换 Bayer 格式所用的算法

转换到其他目标图像格式时，参考以下例子，调用相应的图像格式转码接口。

```
// 分配转换所需 buffer
```

```
// 目标格式为 BGR24、RGB24 时，需要缓存大小为 图像宽*图像高*3
```

```
// 目标格式为 Mono8 时，需要缓存大小为 图像宽*图像高
```

```
int nBGRBufferSize = pFrame->getImageWidth(pFrame) * pFrame->getImageHeight(pFrame) * 3;
uint8_t *pBGRbuffer = (uint8_t *)malloc(nBGRBufferSize);
```

```
// 设置转换配置
```

```
IMGCNV_SOpenParam openParam;
openParam.width      = pFrame->getImageWidth(pFrame);
openParam.height     = pFrame->getImageHeight(pFrame);
openParam.paddingX   = pFrame->getImagePaddingX(pFrame);
openParam.paddingY   = pFrame->getImagePaddingY(pFrame);
openParam.dataSize   = pFrame->getImageSize(pFrame);
openParam.pixelFormat = pFrame->getImagePixelFormat(pFrame);
```

```
// 转换
IMGCNV_EErr eStatus = IMGCNV_ConvertToBGR24(pFrame->getImage(pFrame),
                                             &openParam,
                                             pBGRbuffer,
                                             &nBGRBufferSize);

if (eStatus != IMGCNV_SUCCESS)
{
    printf("image convert failed!");
    free(pBGRbuffer);
}
```

## 3.6. 事件通知

注意，注册事件通知之前必须先连接相机。

### 3.6.1. 设备连接状态事件

#### (1) 设备连接状态事件回调函数定义

```
// 设备连接状态事件回调函数
void onDeviceLinkNotify(const GENICAM_SConnectArg* pConnectArg, void *pUser)
{
    if (pConnectArg->m_event == offLine)
    {
        // 相机掉线

        // 此处一般要销毁流对象、事件订阅对象，

        // 反注册流事件回调、相机事件回调、拉流回调，并停止拉流。
    }
}
```

```

else if (pConnectArg->m_event == onLine)
{
    //相机上线

    //此处一般要断开相机连接，重新连接相机，

    //重新创建流对象、事件订阅对象，

    //重新注册流事件回调、相机事件回调、拉流回调，

    //重新开始拉流。
}
}

```

## (2) 注册设备连接状态事件回调

```

// 事件订阅对象参数
GENICAM_EventSubscribeInfo eventSubscribeInfo = { 0 };

eventSubscribeInfo.pCamera = pCamera;    // pCamera 是已经连接的相机

// 创建事件订阅对象
GENICAM_EventSubscribe *pEventSubscribe = NULL;
if (0 != GENICAM_createEventSubscribe(&eventSubscribeInfo, &pEventSubscribe))
{
    return -1;
}

// 注册设备连接状态事件回调
if (0 != pEventSubscribe->subscribeConnectArgsEx(pEventSubscribe, onDeviceLinkNotify,
"Camera1"))
{
    // 注意：需要调用 release 释放内存

    pEventSubscribe->release(pEventSubscribe);
    return -1;
}

```

### (3) 反注册设备连接状态事件回调

```
// 取消注册设备连接状态事件回调

if (0 != pEventSubscribe->unsubscribeConnectArgsEx(pEventSubscribe, onDeviceLinkNotify,
"Camera1"))
{
    // 注意：需要调用 release 释放内存

    pEventSubscribe->release(pEventSubscribe);
    return -1;
}

// 注意：需要调用 release 释放内存
pEventSubscribe->release(pEventSubscribe);
```

## 3.6.2. 流事件

相机提供了以下流事件通知。

streamEventLostPacket	传输层丢包引起的丢帧
streamEventStreamChannelError	U3V 通信链路异常

### (1) 流通道事件回调函数定义

```
// 流通道事件回调函数

void onStreamEvent(const GENICAM_SStreamArg* pStreamArg, void *pUser)
{
    if (pStreamArg->eStreamEventStatus == streamEventLostPacket)
    {
        // GigE 相机传输层丢包导致丢帧
    }
}
```



```
else if (pStreamArg->eStreamEventStatus == streamEventStreamChannelError)
{
    // USB 相机链路错误

    // 此处一般要停止拉流，并重新开始拉流。
}
}
```

## (2) 注册流通道事件回调

```
// 事件订阅对象参数
GENICAM_EventSubscribeInfo eventSubscribeInfo = { 0 };

eventSubscribeInfo.pCamera = pCamera;           // pCamera 是已经连接的相机

// 创建事件订阅对象
GENICAM_EventSubscribe *pEventSubscribe = NULL;
if (0 != GENICAM_createEventSubscribe(&eventSubscribeInfo, &pEventSubscribe))
{
    return -1;
}

// 注册流通道事件回调
if (0 != pEventSubscribe->subscribeStreamArgEx(pEventSubscribe, onStreamEvent,
"Camera1"))
{
    // 注意：需要调用 release 释放内存

    pEventSubscribe->release(pEventSubscribe);
    return -1;
}
```

## (3) 反注册流通道事件回调并销毁事件订阅对象

```
// 取消注册流通道事件回调
```

```
if (0 != pEventSubscribe->unsubscribeStreamArgEx(pEventSubscribe, onStreamEvent,
"Camera1"))
{
    // 注意：需要调用 release 释放内存

    pEventSubscribe->release(pEventSubscribe);
    return -1;
}
```

```
// 注意：需要调用 release 释放内存
```

```
pEventSubscribe->release(pEventSubscribe);
```

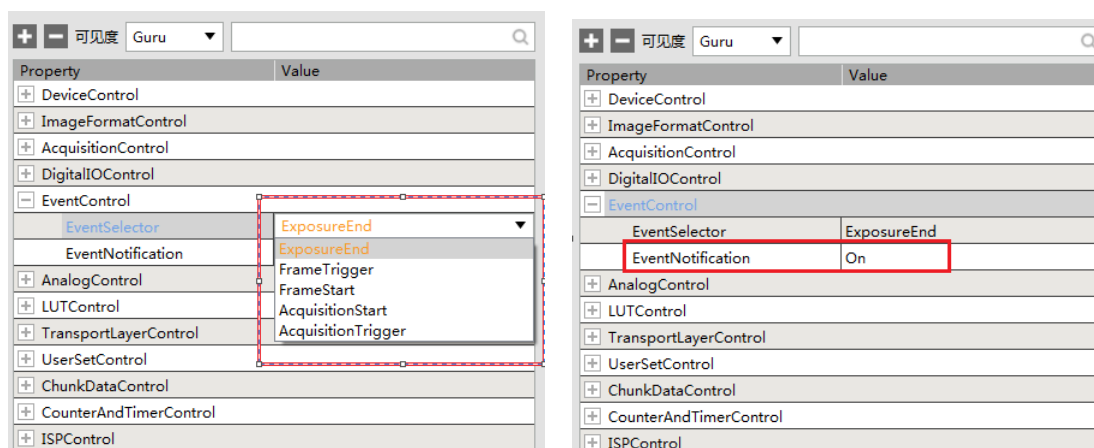
### 3.6.3. 相机消息事件

相机提供了一些内部消息事件，例如曝光结束、帧开始等事件通知。

使用相机消息事件之前，需要将对应的相机事件开关设置为 On。

例如要获取相机曝光结束的事件通知，则设置 EventSelector 为 ExposureEnd，并设

置 EventNotification 为 On，如下图：



#### (1) 消息通道事件回调定义

```
// 消息通道事件回调
```

```
void onMsgChannel(const GENICAM_SMsgChannelArg* pMsgChannelArg, void *pUser)
{
    if (pMsgChannelArg->eventID == MSG_EVENT_ID_EXPOSURE_END)
    {
        // 曝光结束事件通知

    }
}
```

## (2) 注册消息通道事件回调

```
// 事件订阅对象参数
```

```
GENICAM_EventSubscribeInfo eventSubscribeInfo = { 0 };
```

```
eventSubscribeInfo.pCamera = pCamera; // pCamera 是已经连接的相机
```

```
// 创建事件订阅对象
```

```
GENICAM_EventSubscribe *pEventSubscribe = NULL;
if (0 != GENICAM_createEventSubscribe(&eventSubscribeInfo, &pEventSubscribe))
{
    return -1;
}
```

```
// 注册消息通道事件回调
```

```
if (0 != pEventSubscribe->subscribeMsgChannelEx(pEventSubscribe, onMsgChannel,
"Camera1"))
{
    // 注意：需要调用 release 释放内存

    pEventSubscribe->release(pEventSubscribe);
    return -1;
}
```

## (3) 反注册消息通道事件回调并销毁事件订阅对象

```
// 取消注册消息通道事件回调
if (0 != pEventSubscribe->unsubscribeMsgChannelEx(pEventSubscribe, onMsgChannel, "Camera1"))
{
    // 注意：需要调用 release 释放内存
    pEventSubscribe->release(pEventSubscribe);
    return -1;
}

// 注意：需要调用 release 释放内存
pEventSubscribe->release(pEventSubscribe);
```

## 4. 第三方平台的图像对象转换方法

当相机图像格式为 Mono8 时，可以直接转换为三方平台的图像对象。

当相机图像格式为 Mono8 以外时，先将图像转换为 BGR24，或者 RGB24 格式，再转换为三方平台的图像对象。

### 4.1. Halcon 对象 (HObject)

#### 4.1.1. 相机图像格式为 Mono8 时

相机图像格式为 Mono8 时，直接用采集到的图像数据构建 Halcon 对象。

此处，假设采集到的图像帧结构体 (GENICAM\_Frame) 为 pFrame。

```
Hobject grabImg;

gen_image1_extern(&(grabImg),
                  "byte",
                  (Hlong) pFrame->getImageWidth(pFrame),
                  (Hlong) pFrame->getImageHeight(pFrame),
                  (Hlong)(pFrame->getImage(pFrame)),
                  0);
```

#### 4.1.2. 相机图像格式为 Mono8 以外时 (主要是彩色格式)

- (1) 将采集到的相机图像数据, 转换成 BGR24 格式

转换方法参考 3.5.4 - (4)的图像格式转换的说明。

此处, 假设采集到的图像帧结构体 (GENICAM\_Frame) 为 pFrame,转换后的

BGR24 格式图像数据保存在 char\* pBGRbuffer 里。

- (2) 使用转换后的 BGR24 格式图像数据, 构建 Halcon 图像对象 (HObject)

```
Hobject grabImg;

gen_image_interleaved(&grabImg,
                      (Hlong) pBGRbuffer,
                      "bgr",
                      pFrame->getImageWidth(pFrame),
                      pFrame->getImageHeight(pFrame),
                      0,
                      "byte",
                      pFrame->getImageWidth(pFrame),
                      pFrame->getImageHeight(pFrame),
                      0,0,8,0);
```

## 4.2. OpenCV 对象 ( cv::Mat )

### 4.2.1. 相机图像格式为 Mono8 时

相机图像格式为 Mono8 时，直接用采集到的图像数据构建 cv::Mat 对象。

此处，假设采集到的图像帧结构体 ( GENICAM\_Frame ) 为 pFrame。

```
cv::Mat image = cv::Mat(pFrame->getImageHeight(pFrame),
                        pFrame->getImageWidth(pFrame),
                        CV_8U,
                        (uint8_t*)(pFrame->getImage(pFrame)));
```

### 4.2.2. 相机图像格式为 Mono8 以外时 ( 主要是彩色格式 )

#### (1) 将采集到的相机图像数据，转换成 BGR24 格式

转换方法参考 3.5.4 - (4)的图像格式转换的说明。

此处，假设采集到的图像帧结构体 ( GENICAM\_Frame ) 为 pFrame,转换后的

BGR24 格式图像数据保存在 char\* pBGRbuffer 里。

#### (2) 使用转换后的 BGR24 格式图像数据，构建 cv::Mat 对象

```
cv::Mat image = cv::Mat(pFrame->getImageHeight(pFrame),
                        pFrame->getImageWidth(pFrame),
                        CV_8UC3,
                        (uint8_t*) pBGRbuffer);
```

## 4.3. QT 对象 ( QImage )

### 4.3.1. 相机图像格式为 Mono8 时

相机图像格式为 Mono8 时，直接用采集到的图像数据构建 QImage 对象。

此处，假设采集到的图像帧结构体（GENICAM\_Frame）为 pFrame。

```
QImage image = QImage((uint8_t*) pFrame->getImage(pFrame),
    pFrame->getImageWidth(pFrame),
    pFrame->getImageHeight(pFrame),
    QImage::Format_Grayscale8);
```

### 4.3.2. 相机图像格式为 Mono8 以外时（主要是彩色格式）

- (1) 将采集到的相机图像数据，转换成 BGR24 格式

转换方法参考 3.5.4 - (4)的图像格式转换的说明。

此处，假设采集到的图像帧结构体（GENICAM\_Frame）为 pFrame,转换后的

BGR24 格式图像数据保存在 char\* pBGRbuffer 里。

- (2) 使用转换后的 BGR24 格式图像数据，构建 QImage 对象

```
QImage image = QImage((uint8_t*)pBGRbuffer,
    pFrame->getImageWidth(pFrame),
    pFrame->getImageHeight(pFrame),
    QImage::Format_RGB888);
```

## 5.配套例程说明

C 例程路径：【SDK 安装目录】\Development\Samples\C 。具体如下：

例程名	例程说明
CommPropAccess	通用属性例程
Example	常用属性节点使用例程
Grab	自由模式采图（主动取流）

LineTrigger	外触发方式采图（回调取流）
SoftTrigger	软触发方式采图（回调取流）

## 6. 常见问题&注意点

### 6.1. 读写相机属性处理存在内存泄露

原因：用户创建的 SDK 对象使用结束后，没有调用对象的 release 接口释放资源。

解决方法：在使用完 SDK 类型的对象后，调用相应对象的 release 接口释放资源。

如下图所示，pAcquisitionCtrl，doubleNode 都需要调用 release 进行释放。

```
int32_t isExposureTimeSuccess;
GENICAM_DoubleNode doubleNode;
double exposureTimeValue;
GENICAM_AcquisitionControl *pAcquisitionCtrl = NULL;
GENICAM_AcquisitionControlInfo acquisitionControlInfo = {0};

acquisitionControlInfo.pCamera = pGetCamera;
isExposureTimeSuccess = GENICAM_createAcquisitionControl(&acquisitionControlInfo);
if( isExposureTimeSuccess != 0)
{
    printf("ExposureTime fail.\n");
    return -1;
}

exposureTimeValue = 0.0;
doubleNode = pAcquisitionCtrl->exposureTime(pAcquisitionCtrl);
isExposureTimeSuccess = doubleNode.getValue(&doubleNode, &exposureTimeValue);
if( isExposureTimeSuccess != 0)
{
    printf("get exposureTime fail.\n");

    //注意：需要释放pAcquisitionCtrl内部对象内存
    pAcquisitionCtrl->release(pAcquisitionCtrl);

    //注意：需要释放doubleNode内部对象内存
    doubleNode.release(&doubleNode);
    return -1;
}
else
{
    //注意：需要释放pAcquisitionCtrl内部对象内存
    pAcquisitionCtrl->release(pAcquisitionCtrl);
    printf("before change ,exposureTime is %f\n",exposureTimeValue);
}
```



## 6.2. 客户的程序使用 C 接口采图，只能取到 8 张图像 (和默认的 SDK 缓存个数一致)。

原因：C 接口取到图像对象后，没有及时释放 GENICAM\_Frame 对象，导致 SDK 驱动缓存被占满，无法再取到流。

解决方法：如下图所示，取到帧 pFrame 使用结束后，都需要调用 release 函数进行释放。

```
ret = pStreamSource->getFrame(pStreamSource, &pFrame, 100);
if (ret < 0)
{
    printf("getFrame fail.\n");
    continue;
}

ret = pFrame->valid(pFrame);
if (ret < 0)
{
    printf("frame is invalid!\n");

    //Caution: release the frame after using it
    //注意: 使用该帧后需要显示释放
    pFrame->release(pFrame);

    continue;
}

printf("get frame id = [%u] successfully!\n", pFrame->getBlockId(pFrame));

//Caution: release the frame after using it
//注意: 使用该帧后需要显示释放
pFrame->release(pFrame);
```

- END -