

# 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 开发 (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).....	15
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.....	20
3.1.	生成系统单例.....	20
3.2.	设备发现.....	20
3.3.	设备连接/断开.....	20
3.4.	读写属性.....	21
3.4.1.	使用常用属性节点.....	21
3.4.2.	使用通用属性方法.....	24
3.5.	采集图像.....	30

3.5.1.	创建/销毁流对象 .....	30
3.5.2.	设置图像接收缓存个数 .....	31
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 .....	39
3.6.1.	设备连接状态事件 .....	39
3.6.2.	流事件 .....	41
3.6.3.	相机消息事件 .....	43
4.	第三方平台的图像对象的转换方法 .....	46
4.1.	Halcon 对象（HObject） .....	46
4.1.1.	相机图像格式为 Mono8 时 .....	46
4.1.2.	相机图像格式为 Mono8 以外时（主要是彩色格式） .....	47
4.2.	OpenCV 对象（cv::Mat） .....	48
4.2.1.	相机图像格式为 Mono8 时 .....	48
4.2.2.	相机图像格式为 Mono8 以外时（主要是彩色格式） .....	48
4.3.	QT 对象（QImage） .....	48
4.3.1.	相机图像格式为 Mono8 时 .....	48
4.3.2.	相机图像格式为 Mono8 以外时（主要是彩色格式） .....	49
5.	配套例程说明 .....	49
6.	常见问题&注意点 .....	50
6.1.	使用 GigE 相机时，进行断点调试开发时，发现操作相机失败。 .....	50
6.2.	只能采集到与图像接收缓存个数一样的图像数 .....	51

6.3. 注册回调函数为类的成员函数时，编译报错。 .....	51
6.4. 【MFC】用户编译程序时，提示“CString：不明确的符号“、”无法从 Dahua::Infra::CString 转换为“CString” .....	51

## 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 , 已和 x86 平台一样标准发布 SDK。

## 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

	<ul style="list-style-type: none"><li>2. dotNet SDK 库</li><li>3. 第三方算法平台 ( Halcon、Sherlock ) 插件库</li><li>4. SDK 例程( C、C++、MFC、C++ Builder、C#、VB.Net、QT、Delphi、VB6、Python )</li></ul>
Documentations	SDK 的各种开发文档 ( SDK 开发手册、API Doc 等 )
Drivers	<ul style="list-style-type: none"><li>1. GigE 驱动安装、卸载程序</li><li>2. U3V 驱动安装、卸载程序</li><li>3. x86 智能相机驱动安装、卸载程序</li><li>4. DShowFilter 安装、卸载程序</li></ul>
Runtime	<p>SDK 运行时库 ( 包含 GenICam V3.0 第三方库 )</p> <p>包含 32 位和 64 位</p>

### 1.3.2. Linux 版 SDK

Linux 版本 SDK 的安装路径为 “/opt/DahuaTech/MVviewer” ,文件( 夹 )结构如下 :





#### 1.4. SDK 开发（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
	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\Depend s\win32\vs2013shared\ImageConvert.lib

	Lib 库（64 位）	<b>【SDK 安装目录】</b> \Development\Samples\MFC\SingleDisplay\Depend s\x64\vs2013shared\ImageConvert.lib
	运行库（32 位）	<b>【SDK 安装目录】</b> \Development\Samples\MFC\SingleDisplay\Bin\win 32\release\ImageConvert.dll
	运行库（64 位）	<b>【SDK 安装目录】</b> \Development\Samples\MFC\SingleDisplay\Bin\x64 \release\ImageConvert.dll
开发文档	中文	<b>【SDK 安装目录】</b> \Documentations\GenlCam_API_C++_CHS.chm <b>【SDK 安装目录】</b> \Documentations\工业相机 SDK 开发手册(C++篇).pdf
	英文	<b>【SDK 安装目录】</b> \Documentations\GenlCam_API_C++_ENG.chm

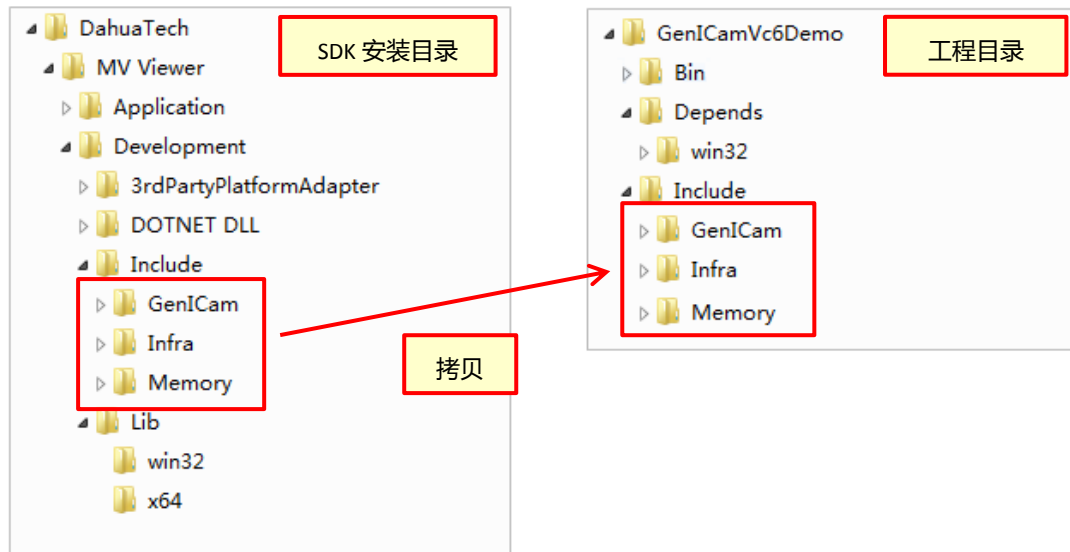
## 1.5. SDK 开发工程配置

### 1.5.1. VC6.0

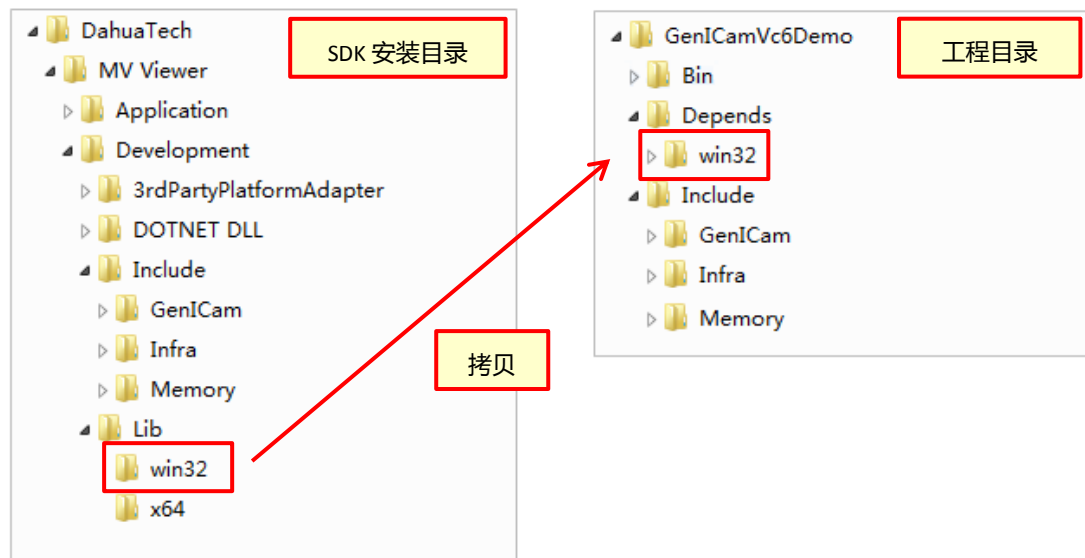
下面以 Win32 | Release 组合为例，说明工程配置方法。

#### (1) 拷贝 SDK 相关资源文件

##### ① 拷贝头文件



## ② 拷贝 Lib 库



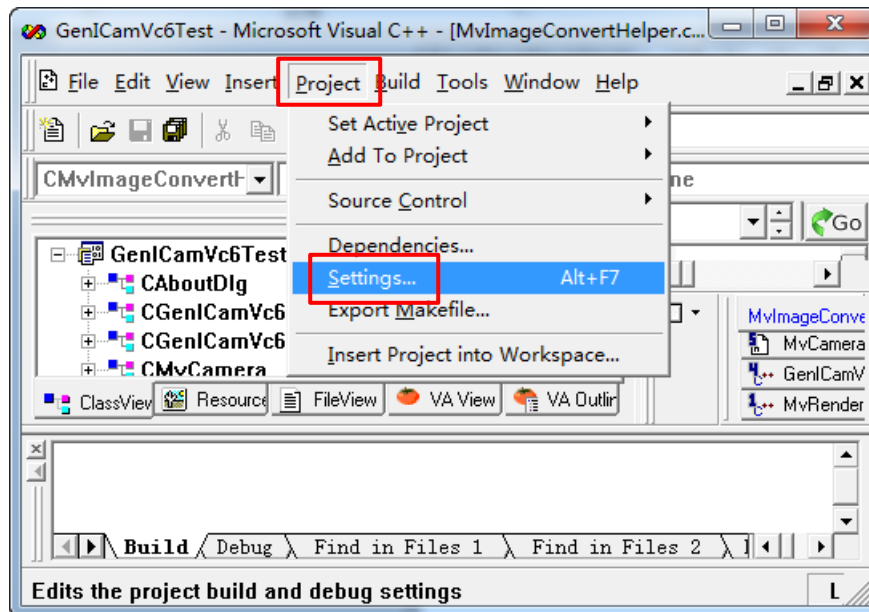
## ③ 拷贝运行时动态库

不需要拷贝 DLL 库。

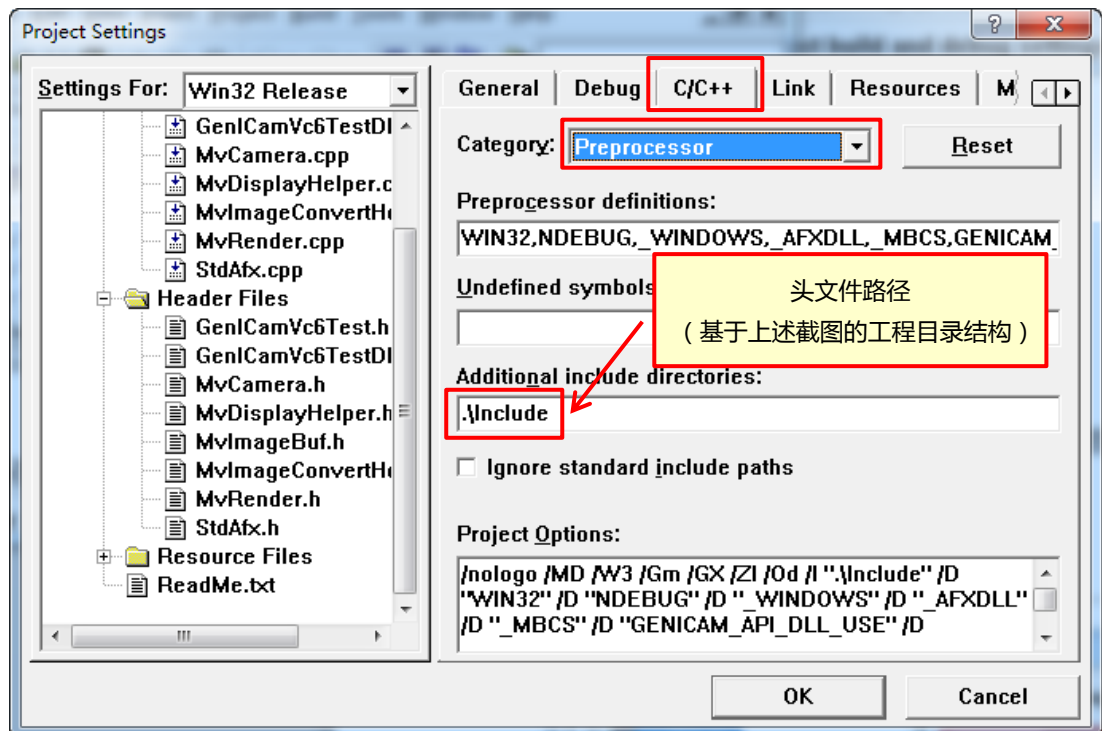
在安装 SDK 的环境里，运行程序时，会自动通过 PATH 环境变量找到“【SDK 安装目录】\Runtime”路径下，对应平台的运行时动态库。

## (2) 配置工程

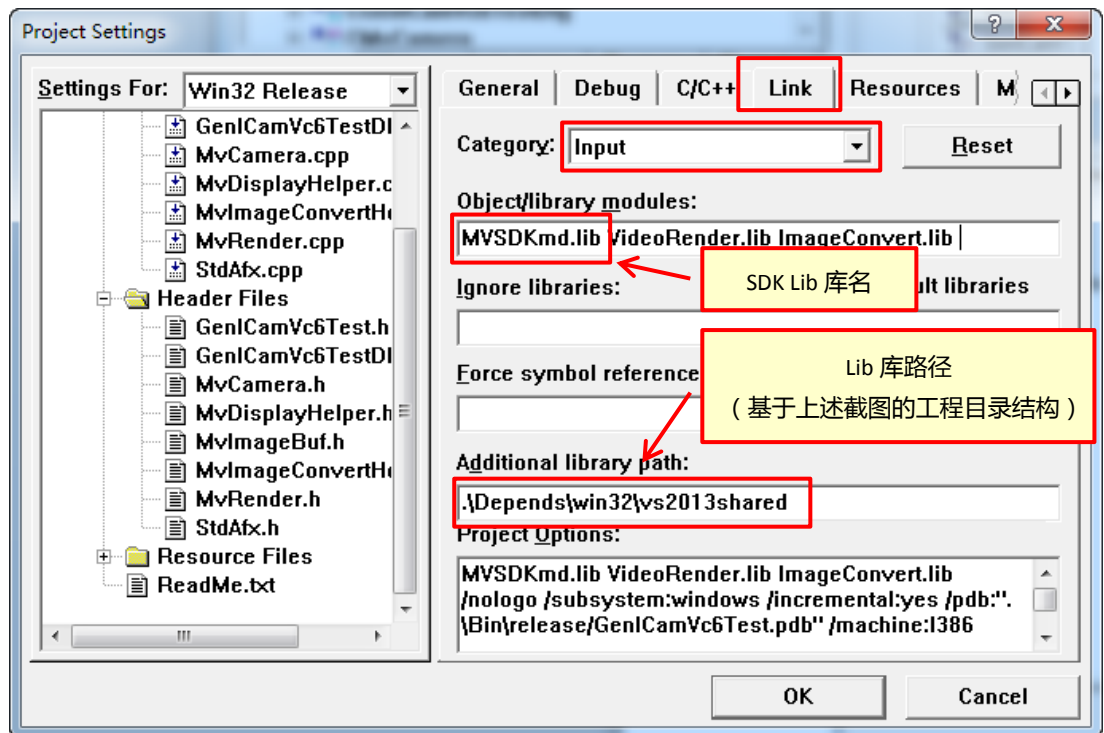
### ① 打开工程配置对话框



### ② 配置头文件



### ③ 配置 Lib 库

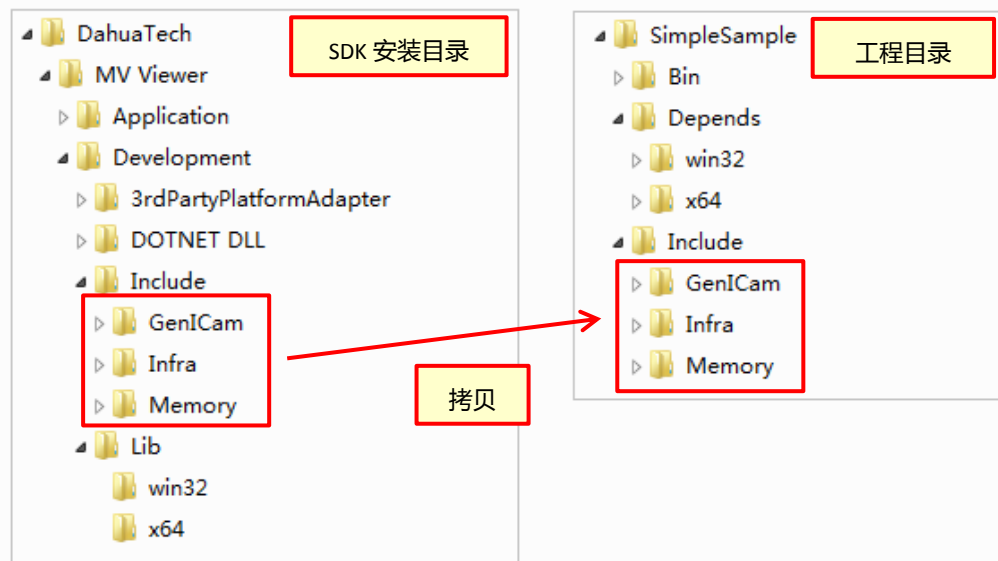


### 1.5.2. VS2005 ~ VS2013

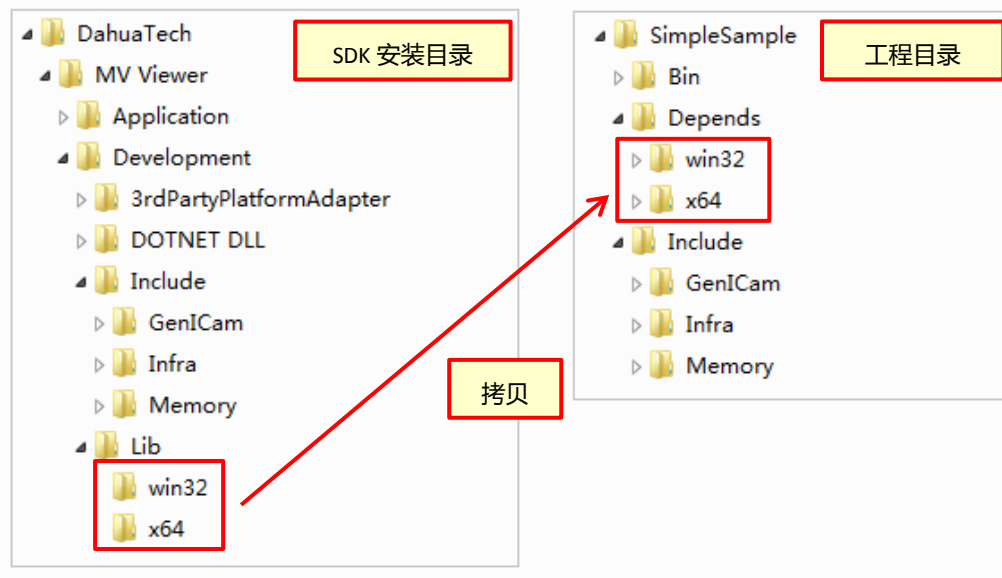
以 Debug | Win32 为例，其它组合参考该配置。

(1) 拷贝 SDK 相关资源文件

① 拷贝头文件



## ② 拷贝 Lib 库



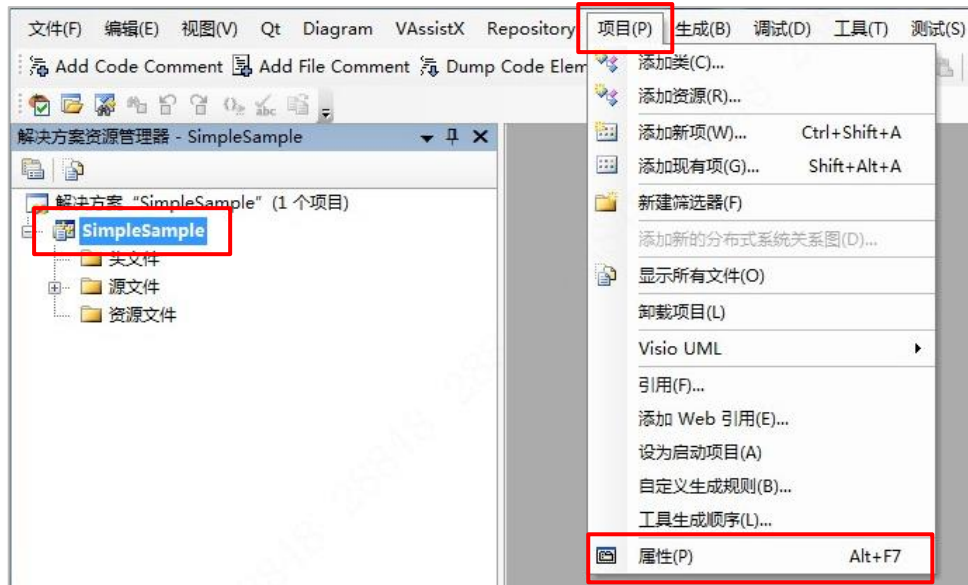
## ③ 拷贝运行时动态库

不需要拷贝 DLL 库。

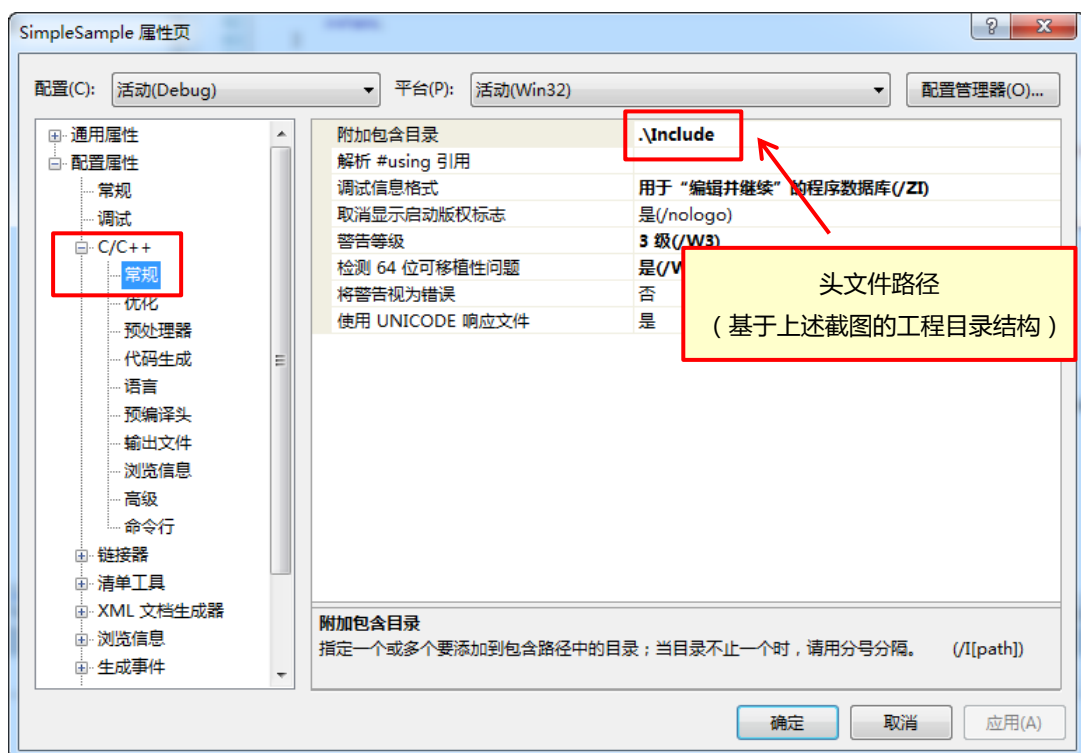
在安装 SDK 的环境里，运行程序时，会自动通过 PATH 环境变量找到 “【SDK 安装目录】\Runtime” 路径下，对应平台的运行时动态库。

## (2) 配置工程

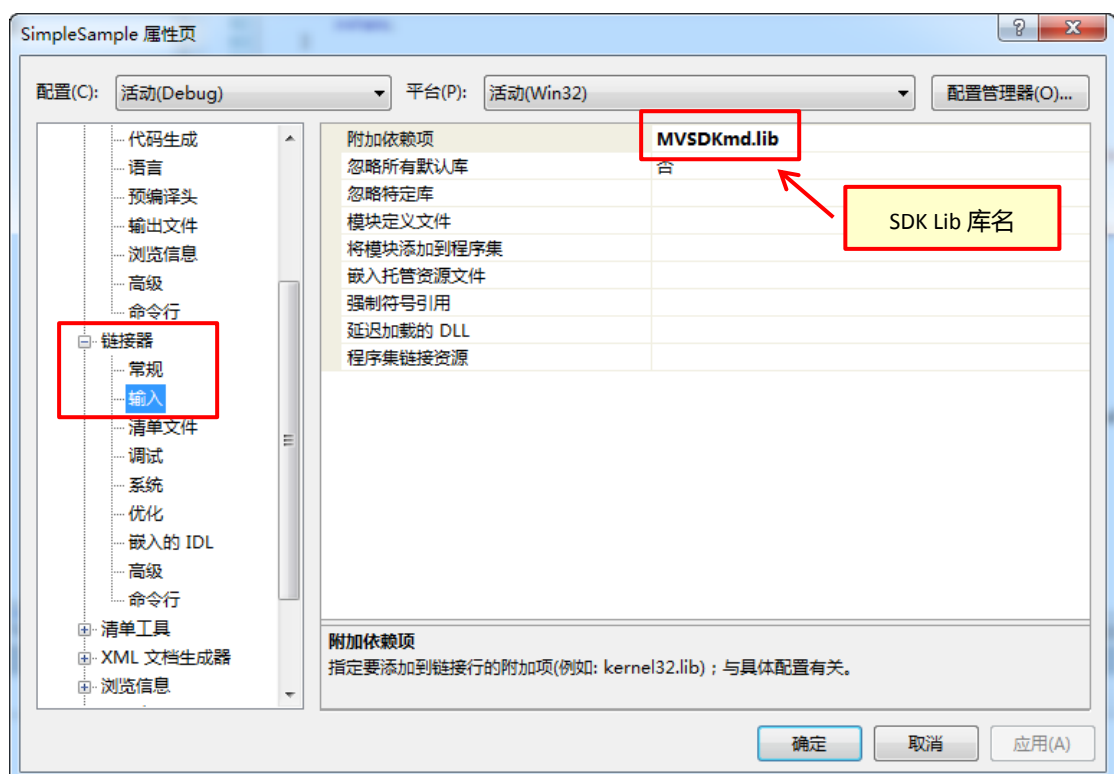
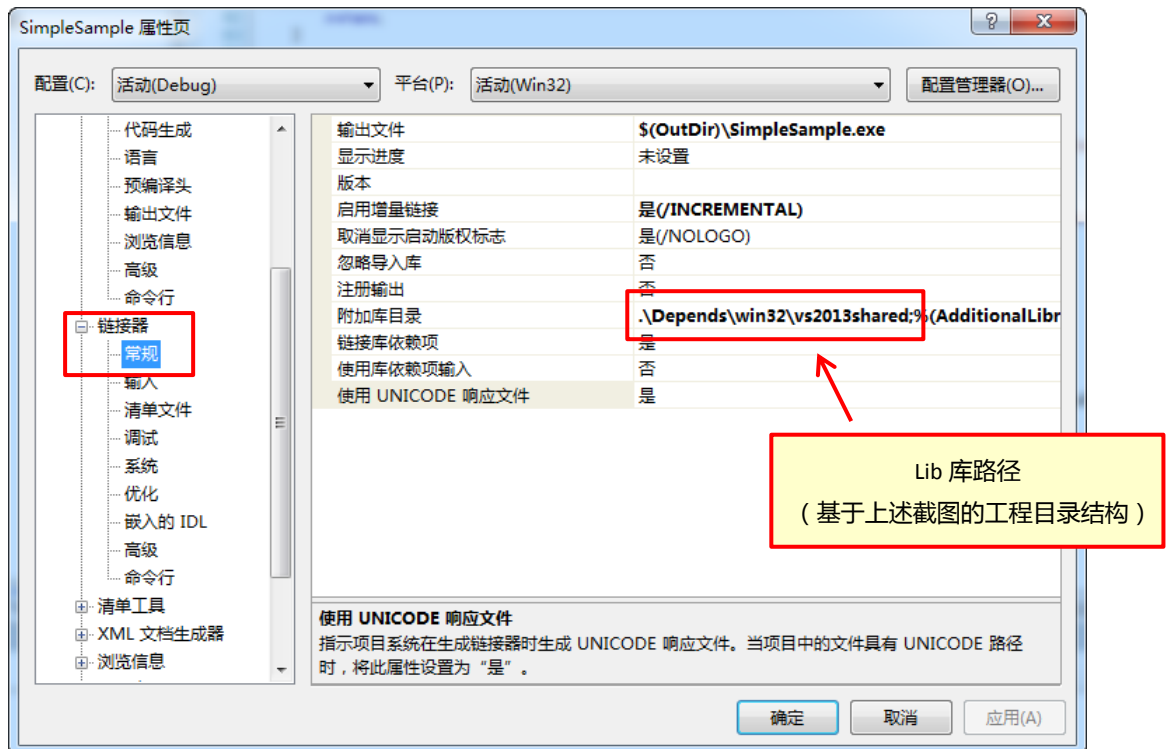
### ① 打开工程配置对话框



## ② 配置头文件



## ③ 配置 Lib 库

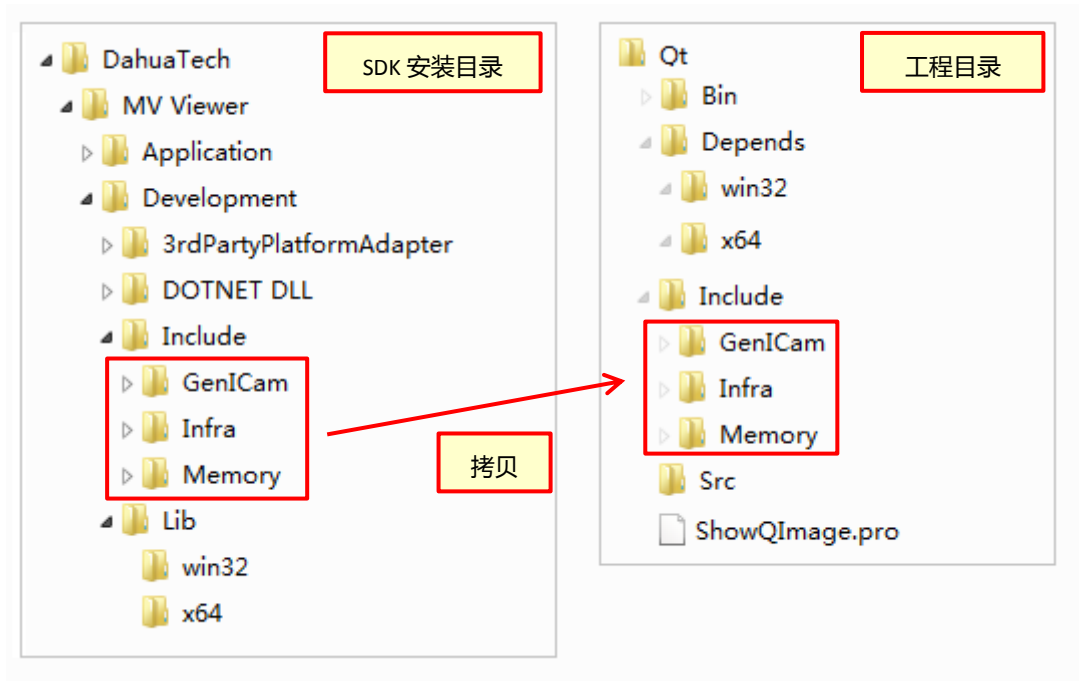


### 1.5.3. QT 工程 ( pro 文件 )

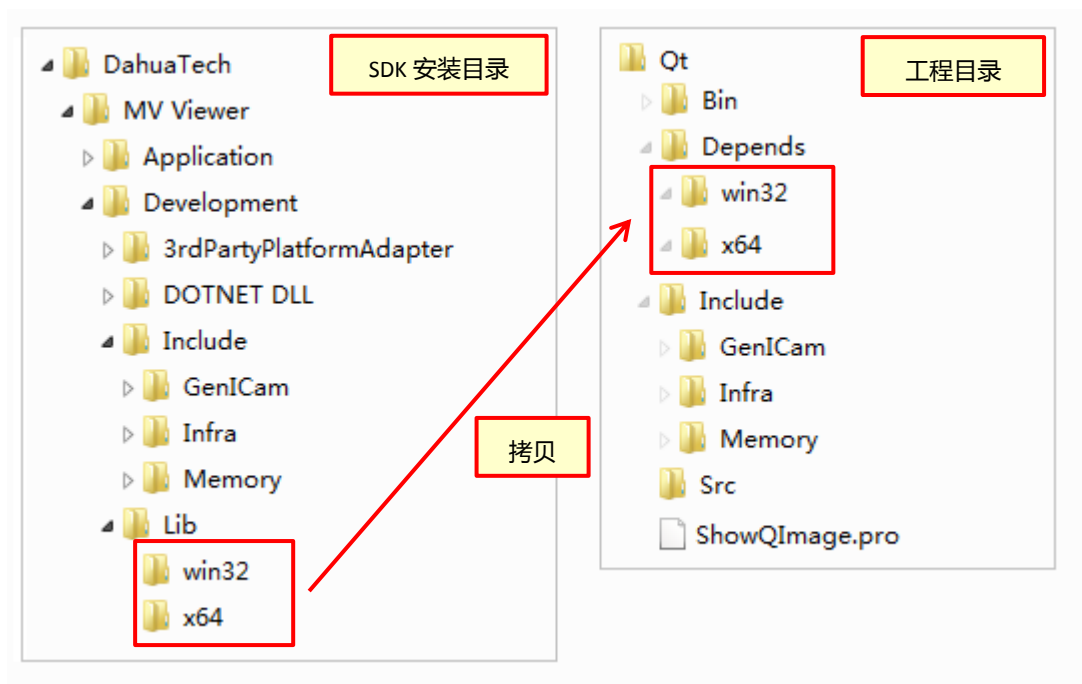
#### (1) 拷贝 SDK 相关资源文件



### 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



## (2) 配置 Makefile 文件

在 Makefile 中，修改 INCLUDES 和 LINKLIBS 部分。

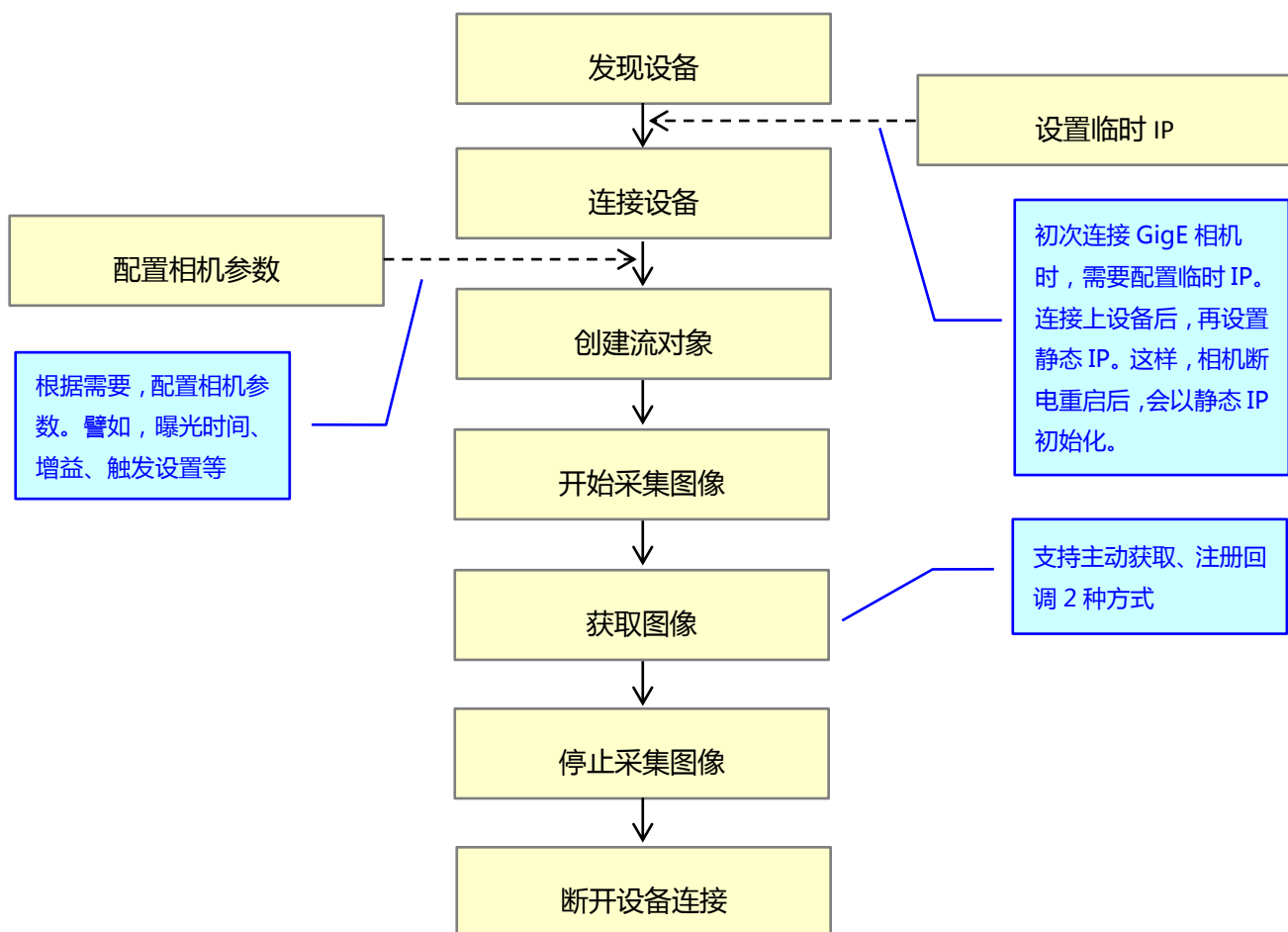
将上述拷贝的头文件路径、动态库路径及名称，按以下示例追加进去。

```
... ( 省略 )

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

... ( 省略 )
```

## 2. SDK 的整体调用流程



## 3. SDK 的各功能接口调用说明

### 3.1. 生成系统单例

头文件：GenICam\System.h

接口： static CSystem& getInstance();

功能说明：CSystem 单例获取接口，返回 CSystem 单例对象的引用

代码范例：

```
CSystem &systemObj = CSystem::getInstance();
```

### 3.2. 设备发现

头文件：GenICam\System.h

接口： bool discovery(Infra::TVector<ICameraPtr>& vCameraPtrList,  
EInterfaceType interfaceType = typeAll);

功能说明：发现指定接口类型可达的设备。成功返回 true，失败返回 false。

[out] vCameraPtrList 指定接口类型所有在线设备对象列表

[in] interfaceType 需要发现的相机类型,默认为发现全部类型

代码范例 (先调用 3.1 生成系统单例)：

```
TVector<ICameraPtr> vCameraPtrList;  
bool isDiscoverySuccess = systemObj.discovery(vCameraPtrList);
```

### 3.3. 设备连接/断开

头文件：GenICam\Camera.h

接口： virtual bool connect(ECameraAccessPermission accessPermission =  
accessPermissionControl) = 0;

功能说明：连接设备

[in] accessPermission 打开相机控制通道时，默认为 control access 权限，不要做  
修改

代码范例:

```
ICameraPtr cameraSptr = vCameraPtrList[0];  
cameraSptr.connect();
```

接口： virtual bool disConnect() = 0;

功能说明：断开连接

代码范例:

```
ICameraPtr cameraSptr = vCameraPtrList[0];  
cameraSptr.disConnect();
```

### 3.4. 读写属性

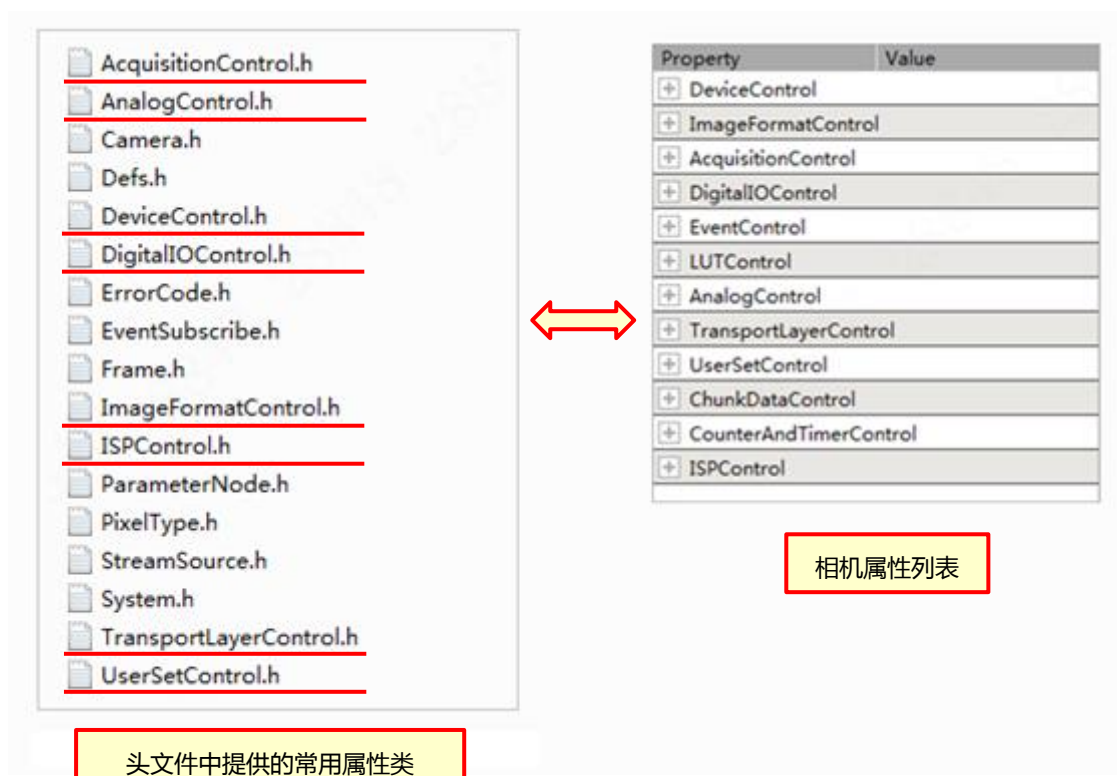
注意，读写属性之前必须先连接相机。

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

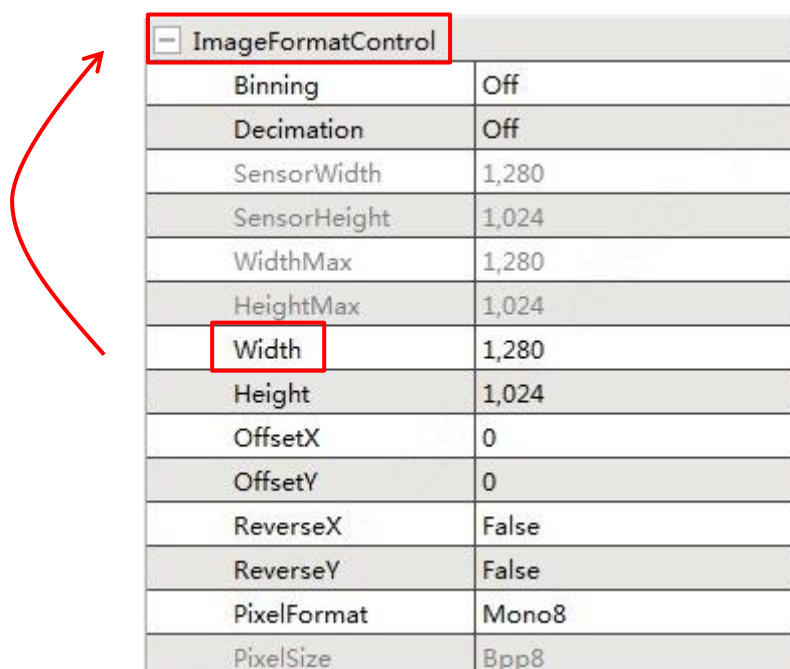
SDK 头文件中提供的常用属性与相机属性列表对应关系如下：



以下，通过属性 Width 为例，讲解常用属性节点的使用方法

(1) 在相机客户端软件的相机属性列表中查找 Width 属性

通过下图，可以看出 Width 属性属于 ImageFormatControl 分类。



Property	Value
ImageFormatControl	
Binning	Off
Decimation	Off
SensorWidth	1,280
SensorHeight	1,024
WidthMax	1,280
HeightMax	1,024
Width	1,280
Height	1,024
OffsetX	0
OffsetY	0
ReverseX	False
ReverseY	False
PixelFormat	Mono8
PixelSize	Bpp8

(2) ImageFormatControl 分类的常用属性节点,对应的头文件为 ImageFormatControl.h

在该头文件内,有 Width 属性对应对象的获取函数。

```
class GENICAM_API IImageFormatControl
{
public:
    /// \~chinese
    /// \brief 获取Height属性操作对象
    /// \return 返回属性操作对象
    /// \~english
    /// \brief get Height property operation object
    /// \return property's object
    virtual CIntNode height() = 0;

    /// \~chinese
    /// \brief 获取Width属性操作对象
    /// \return 返回属性操作对象
    /// \~english
    /// \brief get Width property operation object
    /// \return property's object
    virtual CIntNode width() = 0;
```

(3) 代码范例

// 获取 CSystem 对象

```
CSystem &systemObj = CSystem::getInstance();
```

// 构造 ImageFormatControl 对象

```
IImageFormatControlPtr sptrFormatControl;
sptrFormatControl = systemObj .createImageFormatControl(cameraPtr);
```

// 获取 width 属性节点

```
CIntNode nodeWidth = sptrFormatControl->width();
```

// 获取属性值

```
int64_t nCurVal = 0;
nodeWidth.getValue(nCurVal);
```

// 设置属性值

```
nodeWidth.setValue(nCurVal + 8);
```

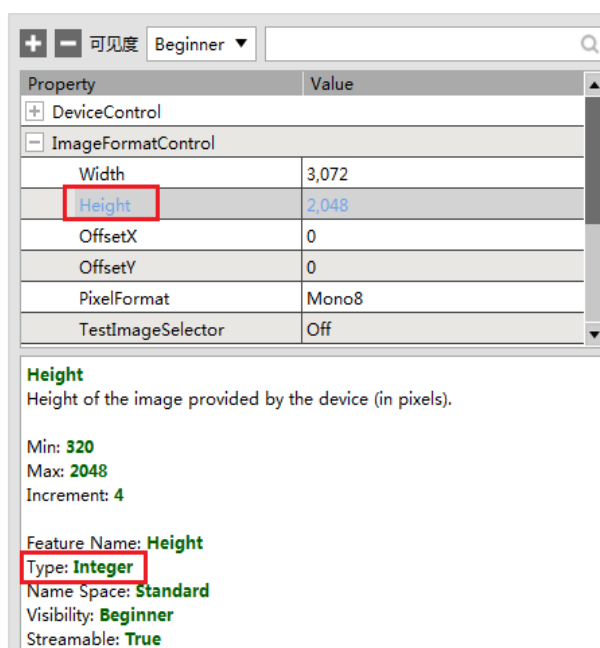


### 3.4.2. 使用通用属性方法

通用属性方法可以使用属性名 (字符串, 需要注意大小写) 构建相应类型的属性节点对象。然后在使用该属性节点对象, 读写属性值。具体使用方法如下:

- (1) 在相机客户端软件的属性列表里, 获取属性的名称、类型和范围等信息

以 Height 属性为例, Type 为 Integer, 说明 Height 是整数类型。



- (2) 按照属性的类型, 选择相应的接口。

以下接口依赖头文件: GenICam\ParameterNode.h

此处假设连接的相机对象为 pCamera

#### ① 整型属性

接口: CIntNode(const ICameraPtr& ptrCamera, const char\* pParamName);

功能说明: 获取整型属性

[in] ptrCamera 已连接相机对象

[in] pParamName 属性名称，注意大小写

接口: `bool getValue(int64_t& val) const;`

功能说明：获取属性值。成功返回 true，失败返回 false。

[out] val 获取到的属性值

接口: `bool setValue(int64_t val);`

功能说明：设置属性值。成功返回 true，失败返回 false。

[in] val 待设置的属性值

代码范例:

```
// 生成属性对象
CIntNode nodeHeight(ptrCamera, "Height");
int64_t nValue = 0;

// 获取属性值
nodeHeight.getValue(nValue);

// 设置属性值
nodeHeight.setValue(nValue-8);
```

## ② 浮点型属性

接口: `CDoubleNode(const ICameraPtr& ptrCamera, const char* pParamName);`

功能说明：获取浮点数属性

[in] ptrCamera 已连接相机对象

[in] pParamName 属性名称，注意大小写

接口： bool getValue(double& val) const;

功能说明：获取属性值。成功返回 true，失败返回 false。

[out] val 获取到的属性值

接口： bool setValue(double val);

功能说明：设置属性值。成功返回 true，失败返回 false。

[in] val 待设置的属性值

代码范例:

```
// 生成属性对象
CDoubleNode nodeExposureTime(cameraSptr, "ExposureTime");
double dbValue = 0.0;

// 获取属性值
nodeExposureTime.getValue(dbValue);

// 设置属性值
nodeExposureTime.setValue(dbValue - 2);
```

### ③ 布尔型属性

接口： CBoolNode(const ICameraPtr& ptrCamera, const char\* pParamName);

功能说明：获取浮点数属性

[in] ptrCamera 已连接相机对象

[in] pParamName 属性名称，注意大小写

接口： `bool getValue(bool& val) const;`

功能说明：获取属性值。成功返回 true，失败返回 false。

[out] val 获取到的属性值

接口： `bool setValue(bool val);`

功能说明：设置属性值。成功返回 true，失败返回 false。

[in] val 待设置的属性值

代码范例：

```
// 生成属性对象
CBoolNode nodeBool(cameraSptr, "ReverseX");
bool bReverse = false;

// 获取属性值
nodeBool.getValue(bReverse);

// 设置属性值
nodeBool.setValue(!bReverse);
```

#### ④ 枚举型属性

接口： `CEnumNode(const ICameraPtr& ptrCamera, const char* pParamName);`

功能说明：获取浮点数属性

[in] ptrCamera 已连接相机对象

[in] pParamName 属性名称，注意大小写

接口： `bool getValueSymbol(Infra::CString& val) const;`

功能说明：获取枚举属性 symbol 值。成功返回 true，失败返回 false。

[out] val 获取到的属性 symbol 值

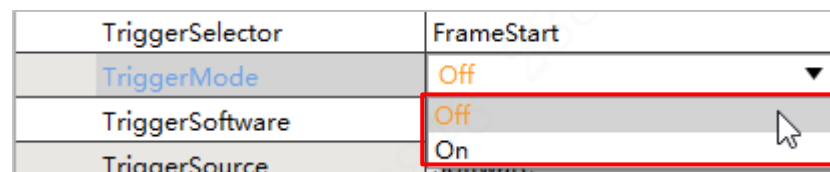
接口： bool setValueBySymbol(const Infra::CString& strSymbolName);

功能说明：设置属性值。成功返回 true，失败返回 false。

[in] strSymbolName 待设置的属性值

补充说明：枚举属性的 symbol 值范围，可以通过相机客户端软件的属性列表来确认。

代码中，需要注意 symbol 值的大小写，保持和客户端软件上看到的一致。



代码范例：

```
// 生成属性对象
CEnumNode nodeEnum(cameraSptr, "TriggerSelector");
CString strValue;

// 获取属性值
nodeEnum.getValueSymbol(strValue);

// 设置属性值
nodeEnum.setValueBySymbol("FrameStart");
```

## ⑤ 字符型属性

接口： CStringNode(const ICameraPtr& ptrCamera, const char\* pParamName);

功能说明：获取字符型属性

[in] ptrCamera 已连接相机对象

[in] pParamName 属性名称，注意大小写

接口： `bool getValue(Infra::CString &outStr);`

功能说明：获取属性值。成功返回 true，失败返回 false。

[out] outStr 获取到的属性值

接口： `bool setValue(const Infra::CString &inStr);`

功能说明：设置属性值。成功返回 true，失败返回 false。

[in] inStr 待输入的属性值

代码范例：

```
// 生成属性对象
CStringNode nodeString(cameraSptr, "DeviceUserID");
CString strID;

// 获取属性值
nodeString.getValue(strID);

// 设置属性值
nodeString.setValue("camera");
```

## ⑥ 命令型属性

接口： `CCmdNode(const ICameraPtr& ptrCamera, const char* pParamName);`

功能说明：获取命令型属性

[in] ptrCamera 已连接相机对象

[in] pParamName 属性名称，注意大小写

接口： bool execute();

功能说明：执行命令类型属性。成功返回 true，失败返回 false。

代码范例：

```
// 生成属性对象
CCmdNode nodeCmd(cameraSptr, "TriggerSoftware");

// 执行命令操作
nodeCmd.execute();
```

## 3.5. 采集图像

注意，进行图像采集相关操作之前，必须先连接相机。

### 3.5.1. 创建/销毁流对象

#### (1) 创建流对象

使用时先获取 System 单例对象，然后创建流对象。

头文件：GenICam\System.h

接口：IStreamSourcePtr createStreamSource(const ICameraPtr &cameraPtr,  
uint16\_t channelId = 0);

功能说明：创建流对象，返回流智能指针对象。

[in] cameraPtr 已连接相机智能指针对象

[in] channelId 流通道号，默认为 0 通道

代码范例：

```
// 假设 System 单例为 systemObj，连接上的相机对象为 cameraSptr  
  
// 创建流对象  
  
IStreamSourcePtr streamPtr = systemObj.createStreamSource(cameraSptr);
```

## (2) 销毁流对象

一般不需要销毁流对象。

在断线重连时，需要调用 streamPtr.reset();

### 3.5.2. 设置图像接收缓存个数

头文件：GenICam\StreamSource.h

接口：virtual bool setBufferCount(uint32\_t nSize) = 0;

功能说明：设置缓存个数，要在执行开始采集图像前，调用该接口。在图像采集状态

下调用该接口会失败。

默认缓存个数为 8。在内存较少的环境（尤其是使用大分辨率相机），相机帧率低时，可以通过设置缓存个数减少 buffer 的数量，防止内存分配失败。

[in] nSize 缓存数量，最小为 1，最大为 200

代码范例:

```
// 设置缓存个数为 4。假设创建的流对象为 streamPtr  
  
streamPtr->setBufferCount(4);
```



### 3.5.3. 开始/停止采集图像

#### (1) 开始采集图像

头文件：GenICam\StreamSource.h

接口：virtual bool startGrabbing(uint64\_t maxImagesGrabbed = 0,  
EGrabStrategy strategy = grabStrategySequential) = 0;

功能说明：开始采集图像。成功返回 true，失败返回 false。

[in] maxImagesGrabbed 允许最多的抓图数。达到指定抓图数后停止抓图。

如果为 0，表示忽略此参数连续抓图

[in] strategy 抓图策略。

grabStrategySequential 按顺序从缓存中取图（默认值）

grabStrategyLatestImage 取缓存中的最新图

代码范例:

```
// 开始采集图像。假设创建的流对象为 streamPtr  
streamPtr->startGrabbing();
```

#### (2) 停止采集图像

头文件：GenICam\StreamSource.h

接口：virtual bool stopGrabbing() = 0;

功能说明：停止采集图像

代码范例:

```
// 开始采集图像。假设创建的流对象为 streamPtr  
streamPtr->stopGrabbing();
```

### 3.5.4. 获取图像

#### (1) 主动采图

头文件：GenICam\StreamSource.h

接口：virtual bool getFrame(CFrame &frame, uint32\_t timeoutMS = INFINITE) const = 0;

功能说明：获取一帧图像，该接口不支持多线程调用。成功返回 true，失败返回 false。

[out] frame 一帧图像

[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。

代码范例：

```
// 主动采图

// 超时时间 100ms。即调用该接口 100ms 仍未采集到图像，返回失败

CFrame frame;
m_streamSptr->getFrame(frame, 100);
```

#### (2) 注册回调

##### ① 注册数据帧回调函数

头文件：GenICam\StreamSource.h

接口：virtual bool attachGrabbing(Proc proc) = 0;

功能说明：注册数据帧回调函数。

注册回调取图与主动采图互斥，只能选其一进行编写代码。

成功返回 true，失败返回 false。

必须在开始采集图像前调用。

[in] 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

代码范例：

```
// 回调函数的实现例

void onGetFrame(const CFrame &pFrame)
{
    uint64_t blockId = pFrame.getBlockId();
}

// 注册回调函数

streamPtr->attachGrabbing(onGetFrame);
```

补充说明：注册的回调函数为类的成员函数时，需要做强制类型转换。代码范例：

```
// 回调函数的实现例

void ClassA::onGetFrame(const CFrame &pFrame)
{
    uint64_t blockId = pFrame.getBlockId();
}

void ClassA::init()
{
    .....

// 注册回调函数

    streamPtr->attachGrabbing(GenICam::IStreamSource::Proc(&ClassA::onGetFrame, this));

    .....
}
```

接口：virtual bool attachGrabbingEx(ProcEx proc, void\* pUser) = 0;

功能说明：注册数据帧回调函数（包含用户自定义数据）。

[in] 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 用户自定义数据。使用多台相机时，可以用来区分相机。

补充说明：注册的回调函数为类的成员函数时，需要做强制类型转换。强制类型转换的方法，请参考 3.5.4 - (2) -①的补充说明中的代码范例。

代码范例：

```
// 回调函数的实现例

void onGetFrame(const CFrame &pFrame, const void* pUser)
{
    uint64_t blockId = pFrame.getBlockId();
}

// 注册回调函数

streamPtr->attachGrabbingEx(onGetFrame, sptrCamera);
```

## ② 反注册数据帧回调函数

头文件：GenlCam\StreamSource.h

接口：virtual bool detachGrabbing(Proc proc) = 0;

功能说明：反注册数据帧回调函数。成功返回 true，失败返回 false。

反注册数据帧回调函数必须在 stopGrabbing 之前调用。

[in] proc 去注册数据帧回调函数

补充说明：注册的回调函数为类的成员函数时，需要做强制类型转换。强制类型转换的方法，请参考 3.5.4 - (2) -①的补充说明中的代码范例。

代码范例：

```
// 注销回调函数

bRet = streamPtr->detachGrabbing(onGetFrame);
```

接口： virtual bool detachGrabbingEx(ProcEx proc, void\* pUser) = 0;

功能说明：注册数据帧回调函数（包含用户自定义数据）。

反注册数据帧回调函数必须在 stopGrabbing 之前调用。

[in] proc 去注册数据帧回调函数

[in] pUser 用户自定义数据(与 attachGrabbingEx 的 pUser 相同)

补充说明：注册的回调函数为类的成员函数时，需要做强制类型转换。强制类型转换的方法，请参考 3.5.4 - (2) -①的补充说明中的代码范例。

代码范例：

```
// 注销回调函数

bRet = streamPtr->detachGrabbingEx(onGetFrame, sptrCamera);
```

### (3) 获取图像裸数据(RAW Data)

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(CFrame 对象)。

CFrame 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。

头文件：Frame.h

接口：const void \* getImage() const;

功能说明：获取该帧图像裸数据的内存首地址。

补充说明：可以通过 Frame.h 中的 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。

代码范例:

```
// 假设 CFrame 对象为 frame  
const void* pImage = frame.getImage();
```

#### (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 1.3 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。

代码范例:

```
// 假设获取图像帧 CFrame 对象为 frame

// 分配转换所需 buffer。注意，目标格式为 RGB、BGR 时，需要乘以 3
int nBGRBufferSize = frame.getImageWidth() * frame.getImageHeight() * 3;
uint8_t *pBGRbuffer = new uint8_t[nBGRBufferSize];

// 设置转换配置参数
IMGCNV_SOpenParam openParam;
openParam.width      = frame.getImageWidth();
openParam.height     = frame.getImageHeight();
openParam.paddingX   = frame.getImagePaddingX();
openParam.paddingY   = frame.getImagePaddingY();
openParam.dataSize   = frame.getImageSize();
openParam.pixelFormat = frame.getImagePixelFormat();

// 转换为 BGR24。 转换为其他格式时，调用相应的接口
IMGCNV_EErr status = IMGCNV_ConvertToBGR24(frame.getImage(),
                                           &openParam,
                                           pBGRbuffer,
                                           &nBGRBufferSize);

if (IMGCNV_SUCCESS != status)
{
    delete[] pBGRbuffer;    // 转码失败时，释放内存
}
```

## 3.6. 事件通知

### 3.6.1. 设备连接状态事件

头文件：GenICam\ EventSubscribe.h

接口：virtual bool subscribeConnectArgs(ConnectArgProc const& proc) = 0;

功能说明：设备连接状态事件回调注册。成功返回 true，失败返回 false。

[in] proc 设备连接状态事件回调注册函数

补充说明：注册的回调函数为类的成员函数时，需要做强制类型转换。强制类型转换的方法，请参考 3.5.4 - (2) -①的补充说明中的代码范例。

接口：virtual bool unsubscribeConnectArgs(ConnectArgProc const& proc) = 0;

功能说明：设备连接状态事件回调去注册。成功返回 true，失败返回 false。

[in] proc 设备连接状态事件回调去注册函数

补充说明：注册的回调函数为类的成员函数时，需要做强制类型转换。强制类型转换的方法，请参考 3.5.4 - (2) -①的补充说明中的代码范例。

接口：virtual bool subscribeConnectArgsEx(ConnectArgProcEx const& proc,  
void\* pUser) = 0;

功能说明：设备连接状态事件回调去注册。成功返回 true，失败返回 false。

[in] proc 设备连接状态事件回调注册函数

[in] pUser 用户自定义数据。使用多台相机时，可以用来区分相机。

补充说明：注册的回调函数为类的成员函数时，需要做强制类型转换。强制类型转换的方法，请参考 3.5.4 - (2) -①的补充说明中的代码范例。



接口：virtual bool unsubscribeConnectArgsEx(ConnectArgProcEx const& proc,  
void\* pUser) = 0;

功能说明：设备连接状态事件回调去注册。成功返回 true，失败返回 false。

[in] proc 设备连接状态事件回调去注册函数

[in] pUser 用户自定义数据。使用多台相机时，可以用来区分相机。

补充说明：注册的回调函数为类的成员函数时，需要做强制类型转换。强制类型转换的方法，请参考 3.5.4 - (2) -①的补充说明中的代码范例。

代码范例：( 以 subscribeConnectArgsEx 和 unsubscribeConnectArgsEx 为例 )

```
// 相机断线通知回调函数（用户自定义处理内容）
void onDeviceLinkNotify(const SConnectArg& conArg, void* pUser)
{
    if (SConnectArg::offLine == conArg.m_event)
    {
        printf("***** 相机断线 *****\n");

        // 用户自定义处理：一般是做停止采图。

        // 如果需要相机上线通知，这里不可以断开相机连接
    }
    else if (SConnectArg::onLine == conArg.m_event)
    {
        printf("相机上线，重新连接.\n");

        // 用户自定义处理：一般是重新连接相机，初始化，开始采图
    }
}

// 创建事件对象
IEventSubscribePtr eventPtr = systemObj.createEventSubscribe(cameraSptr);
```

```
char camera1[10] = "Camera1";

// 注册事件回调的方法
eventPtr->subscribeConnectArgsEx(onDeviceLinkNotify, camera1);

// 反注册事件回调的方法
eventPtr->unsubscribeConnectArgsEx(onDeviceLinkNotify, camera1);
```

### 3.6.2. 流事件

头文件：GenICam\ EventSubscribe.h

接口：virtual bool subscribeStreamArg(StreamArgProc const& proc) = 0;

功能说明：流通道事件回调注册。成功返回 true，失败返回 false。

[in] proc 流通道事件回调注册函数

接口：virtual bool unsubscribeStreamArg(StreamArgProc const& proc) = 0;

功能说明：流通道事件回调去注册。成功返回 true，失败返回 false。

[in] proc 流通道事件回调去注册函数

接口：virtual bool subscribeStreamArgEx(StreamArgProcEx const& proc,  
void\* pUser) = 0;

功能说明：流通道事件回调注册。成功返回 true，失败返回 false。

[in] proc 流通道事件回调注册函数

[in] pUser 用户自定义数据。使用多台相机时，可以用来区分相机。

补充说明：注册的回调函数为类的成员函数时，需要做强制类型转换。强制类型转换

的方法，请参考 3.5.4 - (2) -①的补充说明中的代码范例。

接口：virtual bool unsubscribeStreamArgEx(StreamArgProcEx const& proc,  
void\* pUser) = 0;

功能说明：流通道事件回调去注册。成功返回 true，失败返回 false。

[in] proc 流通道事件回调去注册函数

[in] pUser 用户自定义数据。使用多台相机时，可以用来区分相机。

补充说明：注册的回调函数为类的成员函数时，需要做强制类型转换。强制类型转换的方法，请参考 3.5.4 - (2) -①的补充说明中的代码范例。

代码范例：(以 subscribeStreamArgEx 和 unsubscribeStreamArgEx 为例)

```
// 相机流事件通知回调函数 (用户自定义处理内容)
void onStreamEvent(const SStreamArg& arg, void* pUser)
{
    if (streamEventStreamChannelError == arg.eStreamEventStatus) // 流通道错误
    {
        //停止拉流
        streamPtr->stopGrabbing();

        //重新开始拉流
        streamPtr->startGrabbing();
    }

    else if (streamEventLostPacket == arg.eStreamEventStatus) // 丢包引起丢帧
    {
        printf("pUser:%s  EventLostPacket!\n", pUser);
    }
    else if (streamEventNormal == arg.eStreamEventStatus)
    {
        printf("pUser:%s  EventNormal!\n", pUser);
    }
}
```

```
// 创建事件对象
```

```
IEventSubscribePtr eventPtr = systemObj.createEventSubscribe(cameraSptr);
```

```
char camera1[10] = "Camera1";
```

```
// 注册事件回调的方法
```

```
eventPtr->subscribeStreamArgEx(onStreamEvent, camera1);
```

```
// 反注册事件回调的方法
```

```
eventPtr->unsubscribeStreamArgEx(onStreamEvent, camera1);
```

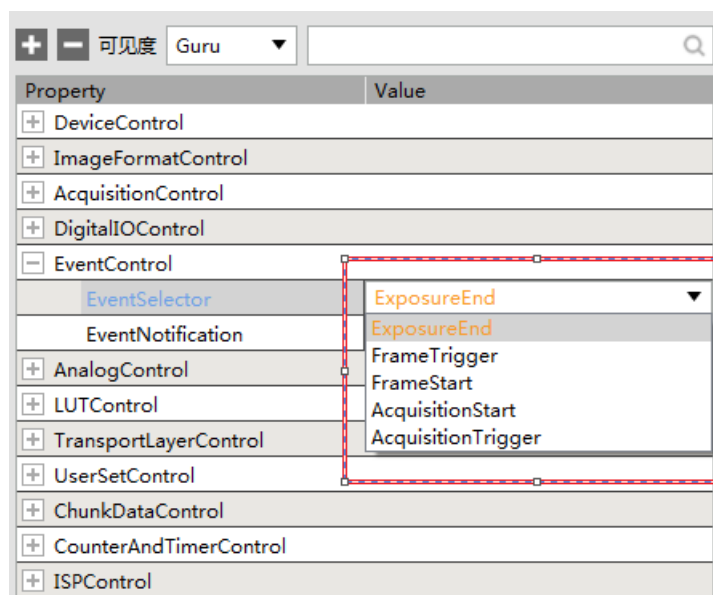
### 3.6.3. 相机消息事件

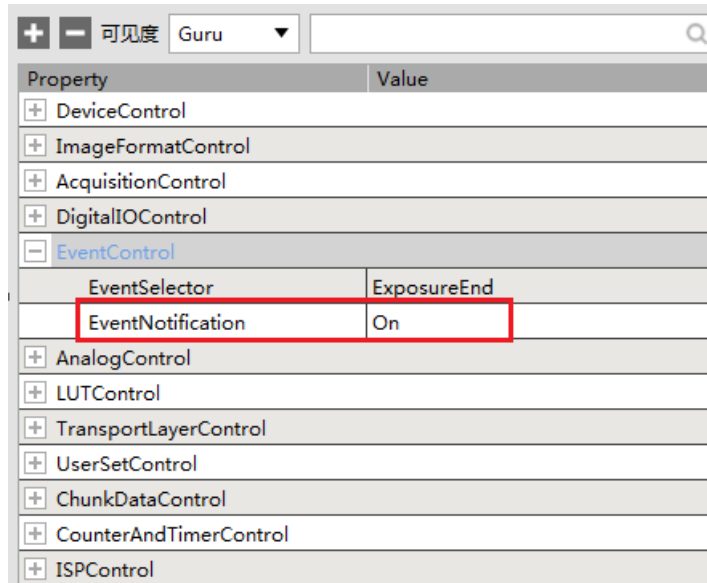
相机提供了“曝光结束”、“帧触发”、“帧开始”等代表相机内部处理状态的事件通知。

用户可以使用 SDK 提供的相机消息事件通知机制，来获得相应的事件的回调。

在使用这些事件之前，需要打开对应的事件通知开关（EventNotification 设置为 On）。

以下，以曝光结束（ExposureEnd）的事件为例。





头文件：GenICam\ EventSubscribe.h

接口：virtual bool subscribeMsgChannel(MsgChannelProc const& proc) = 0;

功能说明：消息通道事件回调注册。成功返回 true，失败返回 false。

[in] proc 消息通道事件回调注册函数

接口：virtual bool unsubscribeMsgChannel(MsgChannelProc const& proc) = 0;

功能说明：消息通道事件回调去注册。成功返回 true，失败返回 false。

[in] proc 消息通道事件回调去注册函数

接口：virtual bool subscribeMsgChannelEx(MsgChannelProcEx const& proc,

void\* pUser) = 0;

功能说明：消息通道事件回调注册。成功返回 true，失败返回 false。

[in] proc 消息通道事件回调注册函数

[in] pUser 用户自定义数据。使用多台相机时，可以用来区分相机。

补充说明：注册的回调函数为类的成员函数时，需要做强制类型转换。强制类型转换

的方法，请参考 3.5.4 - (2) -①的补充说明中的代码范例。

接口：virtual bool unsubscribeMsgChannelEx(MsgChannelProcEx const& proc,  
void\* pUser) = 0;

功能说明：消息通道事件回调去注册。成功返回 true，失败返回 false。

[in] proc 消息通道事件回调去注册函数

[in] pUser 用户自定义数据。使用多台相机时，可以用来区分相机。

补充说明：注册的回调函数为类的成员函数时，需要做强制类型转换。强制类型转换的方法，请参考 3.5.4 - (2) -①的补充说明中的代码范例。

代码范例：(以 subscribeMsgChannelEx 和 unsubscribeMsgChannelEx 为例)

```
// 相机消息事件通知回调函数（用户自定义处理内容）
void onChannelCallback(const SMsgChannelArg& channelArg, void* pUser)
{
    if (channelArg.eventID == MSG_EVENT_ID_EXPOSURE_END)
    {
        // 曝光结束。具体处理内容，用户自定义。
        printf("pUser:%s Exposure End!\n", pUser);
    }
    else if (channelArg.eventID == MSG_EVENT_ID_FRAME_TRIGGER)
    {
        // 帧触发。具体处理内容，用户自定义。
        printf("pUser:%s Frame Trigger!\n", pUser);
    }
    else if (channelArg.eventID == MSG_EVENT_ID_FRAME_START)
    {
        // 帧开始。具体处理内容，用户自定义
        printf("pUser:%s Frame Start!\n", pUser);
    }
}
```

```
// 创建事件对象

IEventSubscribePtr eventPtr = systemObj.createEventSubscribe(cameraSptr);

char camera1[10] = "Camera1";

// 注册事件回调的方法

eventPtr->subscribeMsgChannelEx(onChannelCallback, camera1);

// 反注册事件回调的方法

eventPtr->unsubscribeMsgChannelEx(onChannelCallback, camera1);
```

## 4. 第三方平台的图像对象的转换方法

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 (CFrame) 对象为 frame。

```
Hobject grabImg;  
  
gen_image1_extern(&(grabImg),  
                  "byte",  
                  (Hlong) frame.getImageWidth(),  
                  (Hlong) frame.getImageHeight(),  
                  (Hlong)( frame.getImage()),  
                  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, 假设转换后的 BGR24 格式图像数据保存在 char\* bgrImage 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)bgrImage,  
                      "bgr",  
                      frame.getImageWidth(),  
                      frame.getImageHeight(),  
                      0,  
                      "byte",  
                      frame.getImageWidth(),  
                      frame.getImageHeight(),  
                      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 ( CFrame ) 对象为 frame。

```
cv::Mat image = cv::Mat(frame.getImageHeight(),
                        frame.getImageWidth(),
                        CV_8U,
                        (uint8_t*)frame.getImage());
```

### 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，假设转换后的 BGR24 格式图像数据保存在 char\* bgrImage 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(frame.getImageHeight(),
                        frame.getImageWidth(),
                        CV_8UC3,
                        (uint8_t*)bgrImage);
```

## 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（CFrame）对象为 frame。

```
QImage image = QImage((uint8_t*)frame.getImage(),
                      frame.getImageWidth(),
                      frame.getImageHeight(),
                      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 RGB24 格式

转换方法参考 3.5.4 - (4)的图像格式转换的说明。

此处，假设转换后的 RGB24 格式图像数据保存在 char\* rgbImage 里。

#### (2) 使用转换后的 RGB24 格式图像数据，构建 QImage 对象

```
QImage image = QImage((uint8_t*)rgbImage,
                      frame.getImageWidth(),
                      frame.getImageHeight(),
                      QImage::Format_RGB888);
```

## 5. 配套例程说明

SDK（C++接口）提供了 C++、MFC、QT 这 3 种配套的例程。具体如下：

C++例程路径：【SDK 安装目录】\Development\Samples\C++

MFC 例程路径：【SDK 安装目录】\Development\Samples\MFC

QT 例程路径：【SDK 安装目录】\Development\Samples\QT

类别	例程名	例程说明
C++	CommPropAccess	通用属性使用例程

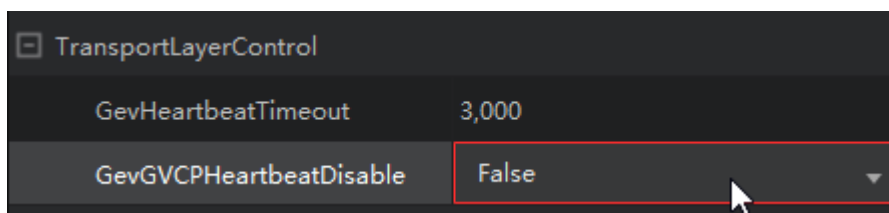
	LineTrigger	外部触发例程(回调取图)
	SimpleSample	主动取流例程
	SoftTrigger	软触发例程(回调取图)
MFC	SingleDisplay	MFC 单相机操作例程（包括拉流、格式转换、显示和保存 bmp）
	MultiDisplay	在 SingleDisplay 基础上，支持最多 4 个相机
	GenICamVc6Demo	VC6 例程（包括属性设置、取图、格式转换和显示）
QT	ShowImage	QT 例程。转码到 BGR24 进行显示。
	ShowQImage	QT 例程。转码到 QImage 进行显示。

## 6. 常见问题&注意点

### 6.1. 使用 GigE 相机时，进行断点调试开发时，发现操作相机失败。

原因：GigE 相机和 SDK 之间有心跳保活机制。当使用断点调试时，会阻塞心跳线程，导致心跳保活超时，相机断线。故之后对相机的操作都会失败。

解决方法：关闭相机的心跳检测，或者，加大相机的心跳超时时间设置。具体参照下图的相机属性项。（TransportLayerControl 类别下）。



需要注意，当关闭相机心跳后，如果调试程序异常结束，则需要断电重启相机，

否则无法再次连接相机。

## 6.2. 只能采集到与图像接收缓存个数一样的图像数

原因：获得图像数据对象后 ( CFrame )，没有及时将该对象释放，导致图像接收缓存一直被占用。当所有的图像接收缓存都被占用 ( 默认值：8 个 )，SDK 驱动没有空闲的缓存用于接收新的图像数据，则上层应用就取不到新的图像。

解决方法：不要使用全局变量，或者，类的成员变量存储 CFrame 对象。

可以使用局部变量存储 CFrame 变量，然后及时将其中的图像裸数据、宽高等信息及时拷贝到全局变量，或者，类的成员变量里，用于后续的处理。

## 6.3. 注册回调函数为类的成员函数时，编译报错。

原因：不支持直接将类的成员函数注册为回调函数。如果直接注册，则会提示以下编译错误

不 存 在 从 "void (const Dahua::GenICam::CFrame &frame)" 转 换 到 "Dahua::Infra::TFunction1<void, const Dahua::GenICam::CFrame &>" 的适当构造函数

解决方法：注册的回调函数为类的成员函数时，需要做强制类型转换。

强制类型转换的方法，请参考 3.5.4 - (2) -①的补充说明中的代码范例。

## 6.4. 【MFC】用户编译程序时，提示 "CString ： 不明确的符号 “、” 无法从 Dahua::Infra::CString “转换为” CString “

```
error C2872: "CString" : 不明确的符号
error C2664: "void IPCfg::fillAddress(CString,CIPAddressCtrl *)" : 无法将参数 1 从 "Dahua::Infra::CString" 转换为 "CString"
error C2664: "void IPCfg::fillAddress(CString,CIPAddressCtrl *)" : 无法将参数 1 从 "Dahua::Infra::CString" 转换为 "CString"
error C2664: "void IPCfg::fillAddress(CString,CIPAddressCtrl *)" : 无法将参数 1 从 "Dahua::Infra::CString" 转换为 "CString"
```

原因： Dahua Infra 库中也有 CString 类型变量，与 MFC 中的 CString 类型有重复。

解决方法： 出现问题时，代码中不要用 using namespace Dahua::Infra。如果需要使用

Infra 库的接口、类型，在定义变量、调用接口时，明确带上 Dahua::Infra。如

Dahua::Infra::CString 。

- END -