IDS peak

Switch from IDS Software Suite to IDS peak

IDS Imaging Development Systems GmbH
Dimbacher Str. 6-8
D-74182 Obersulm, Germany
T: +49 7134 96196-0
E: info@ids-imaging.com
W: https://en.ids-imaging.com

1 Preface

Introduction

IDS Imaging Development Systems GmbH has taken every possible care in preparing this manual. We however assume no liability for the content, completeness or quality of the information contained therein. The content of this manual is regularly updated and adapted to reflect the current status of the software. We furthermore do not guarantee that this product will function without errors, even if the stated specifications are adhered to.

Under no circumstances can we guarantee that a particular objective can be achieved with the purchase of this product.

Insofar as permitted under statutory regulations, we assume no liability for direct damage, indirect damage or damages suffered by third parties resulting from the purchase of this product. In no event shall any liability exceed the purchase price of the product.

Please note that the content of this manual is neither part of any previous or existing agreement, promise, representation or legal relationship, nor an alteration or amendment thereof. All obligations of IDS Imaging Development Systems GmbH result from the respective contract of sale, which also includes the complete and exclusively applicable warranty regulations. These contractual warranty regulations are neither extended nor limited by the information contained in this manual. Should you require further information on this product, or encounter specific problems that are not discussed in sufficient detail in the manual, please contact your local dealer or system installer.

Trademarks

The IDS logo is a registered trademark of IDS Imaging Development Systems GmbH, registered for U.S. (Reg.No. 4,513,138) and other countries.

IDS Imaging Development Systems and **uEye** are registered trademarks of IDS Imaging Development Systems GmbH. Microsoft and Windows are trademarks or registered trademarks of Microsoft Corporation. All other products or company names mentioned in this manual are used solely for purposes of identification or description and may be trademarks or registered trademarks of the respective owners.

Copyright

Status: July 2019

Contact

Visit our web site https://en.ids-imaging.com where you will find all the latest information about our software and hardware products. Please contact your local IDS distributors for first level support in your language. For a list of IDS distributors worldwide please go to our website https://en.ids-imaging.com.

Address	IDS Imaging Development Systems GmbH Dimbacher Str. 6-8 D-74182 Obersulm, Germany
T	+49 7134 96196-0
E	info@ids-imaging.com
W	https://en.ids-imaging.com

2 IDS peak

IDS peak is a comprehensive software package from IDS Imaging Development Systems GmbH that can be used with GenICam-compliant industrial cameras. IDS peak provides all necessary tools to open cameras in an application with graphical user interface, to parametrize them, to capture images, etc. or to program your own application.

This documentation describes how to switch a IDS Software Suite based application into a new GenICam based IDS peak application.

3 Switch IDS Software Suite to IDS peak

In the following example, we identify a correctly configured Vision camera by its serial number, open it, capture images and adjust the exposure time.

Steps	IDS Software Suite	Steps	IDS peak
Requirement	The application knows the "uEye_api.dll" and the required header ("uEye.h") is included.	Requirement	The application knows the IDS peak libraries (DLLs) and also the DLLs for the IDS peak IPL (render and post-processing library). The header files for IDS peak and IDS peak IPL are included.

		Open the IDS peak library to access the IDS peak functionalities. At the end of the program, the library has to be closed. Objects still in use are deleted and the library can be closed completely.	`try`
			`{`
			`peak::Library::Initialize();`
			`}`
			`catch (const std::exception& e)`
			`{`
			`…`
			`}`

• Get camera list
• Search for the camera with the corresponding serial number
• Save matching device ID as unique identifier
• Open the camera with the device ID. Therefore, hCam is initialized with the device ID logically linked by "OR" to the `IS_USE_DEVICE_ID` flag
• Pass the handle of the window in which the rendering will be performed later.
	`UEYE_CAMERA_LIST* list = new UEYE_CAMERA_LIST;`
	`if (NULL == list)`
	`{`
	`// Error`
	`}`
	`// Initialize dwCount with zero. After the function call`
	`// dwCount will hold the number of cameras in the list`
	`pList->dwCount = 0;`
	`if (IS_SUCCESS != is_GetCameraList(list))`
	`{`
	`// Error`
	`}`
	
	`// Save number and delete list`
	`DWORD numCameras = pList->dwCount;`
	`delete list;`
	`// Allocate new list with correct size`
	`list = (UEYE_CAMERA_LIST*)new`
		• Get an instance of the DeviceManager
		• Update the DeviceManager
		• Search for the camera with the corresponding serial number. A device descriptor is returned which contains specific information about the (not yet opened) device.
		• Open the camera
			`std::shared_ptr<peak::core::Device> device;`
			`try`
			`{`
			`auto& deviceManager = peak::DeviceManager::Instance();`
			`deviceManager.Update();`
			`// Iterate through all connected devices`
			`for (const auto& deviceDescriptor : deviceManager.Devices())`
			`{`
			`// Compare specified serial number with serial number of the device`
			`if (deviceDescriptor->SerialNumber() == strSerialNumber)`
			`{`
			`// The function returns an object for all further accesses to the camera`
			`device = deviceDescriptor->OpenDevice(peak::core::DeviceAccessType::Control);`
			`break;`

Steps	IDS Software Suite	Steps	IDS peak

	char[sizeof(DWORD) + numCameras * sizeof(UEYE_CAMERA_INFO)];
	list->dwCount = numCameras;
	// Get list
	if (IS_SUCCESS != is_GetCameraList(list))
	{
	// Error
	}
	// Parse the list until we find the string serialNumber
	int deviceID = -1;
	for(int i = 0; i < static_cast<int>(list->dwCount); i++)
	{
	if (strcmp(list->uci[i].SerNo, serialNumber) == 0)
	{
	// Save corresponding device ID
	deviceID = pList->uci[i].dwDeviceID;
	}
	}
	// Delete list because we do not need it anymore
	delete [] list;
	if (-1 == deviceID)
	{
	// Error
	}
	// Open camera with the correct device ID.
	// hWnd is the handle of the window in which we will
	// render the image.
	HIDS hCam = ((HIDS) (deviceID | IS_USE_DEVICE_ID));
	if (IS_SUCCESS != is_InitCamera(&hCam, hWnd))
	{
	// Error
	}
			}
			}
			}
			catch (const std::exception& e)
			{
			…
			}

Steps	IDS Software Suite	Steps	IDS peak

• Read sensor information
• Save maximum image size of the sensor
• Allocate 10 image buffers
• Insert image buffer into a sequence
• Set pixel format
• Set AOI
	`SENSORINFO info;`
	`if (IS_SUCCESS != is_GetSensorInfo(hCam, &info))`
	`{`
	`  // Error`
	`}`
	`int width = info.nMaxWidth;`
	`int height = info.nMaxHeight;`
	`// Allocate 3 BGRA8 (4 Byte/Pixel) buffers for a sequence`
	`char* imageMemory[10];`
	`int memoryId[10];`
	`int colorMode = IS_CM_BGRA8_PACKED;`
	`int bitsPerPixel = 32;`
	`for (int i = 0; i < 10; i++)`
	`{`
	`  if (IS_SUCCESS != is_AllocImageMem(hCam,`
	`  width,`
	`  height,`
	`  bitsPerPixel,`
	`  &(imageMemory[i]),`
	`  &(memoryId[i]))`
	`  {`
	`  // Error`
	`  }`
	`  if (IS_SUCCESS != is_AddToSequence(hCam,`
	`  imageMemory[i],`
	`  memoryId[i]);`
	`  {`
	`  // Error`
	`  }`
	`}`
	`// Set correct color mode`
	`if (IS_SUCCESS != is_SetColorMode(hCam, colorMode))`
		• Open camera data stream
		• Get a pointer to the remote device nodemap
		• Read payload size
		• Allocate 10 image buffers
		• Add image buffer to the buffer queue
		• Read maximum image size of the sensor
			`std::shared_ptr<peak::core::DataStream> dataStream;`
			`std::shared_ptr<peak::core::NodeMap> nodemapRemoteDevice;`
			`try`
			`{`
			`  // Get standard data stream`
			`  dataStream = m_device->DataStreams().at(0)->OpenDataStream();`
			`  // Get remote device nodemap for all accesses to the genicam nodemap tree`
			`  nodemapRemoteDevice = m_device->RemoteDevice()->NodeMaps().at(0);`
			`  // Get the payload size for correct buffer allocation`
			`  int64_t payloadSize = nodemapRemoteDevice->FindNode<peak::core::nodes::IntegerNode>("PayloadSize")->Value();`
			`  // Allocate 10 buffers (RAW image) and queue them`
			`  for (size_t count = 0; count < 10; ++count)`
			`  {`
			`  auto buffer = dataStream->AllocAndAnnounceBuffer(static_cast<size_t>(payloadSize), nullptr);`
			`  dataStream->QueueBuffer(buffer);`
			`  }`
			`  // Get the AOI setting`
			`  int64_t imageWidth  = nodemapRemoteDevice->FindNode<peak::core::nodes::IntegerNode>("Width")->Value();`
			`  int64_t imageHeight = nodemapRemoteDevice->FindNode<peak::core::nodes::IntegerNode>("Height")->Value();`
			`}`
			`catch (const std::exception& e)`
			`{`

Steps	IDS Software Suite	Steps	IDS peak

	{
	// Error
	}
	// Set AOI
	IS_SIZE_2D imageSize;
	imageSize.s32Width = width;
	imageSize.s32Height = height;
	if (IS_SUCCESS != is_AOI(hCam,
	IS_AOI_IMAGE_SET_SIZE,
	(void*)&imageSize,
	sizeof(imageSize)))
	{
	// Error
	}
			…
			}

Create and enable frame event	// Create event
	HANDLE hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
	if (hEvent == NULL)
	{
	// Error
	}
	// Init event
	if (IS_SUCCESS != is_InitEvent(hCam, hEvent, IS_SET_EVENT_FRAME))
	{
	// Error
	}
	// Enable event
	if (IS_SUCCESS != is_EnableEvent(hCam, IS_SET_EVENT_FRAME))
	{
	// Error
	}
		Is done automatically	

Start image acquisition	if (IS_SUCCESS != is_CaptureVideo(hCam, IS_WAIT))
	{
		• Start image acquisition
		• Set TLParamsLocked to 1. This avoids various nodes being written
			try
			{
			// Start continuous acquisition with an

Steps	IDS Software Suite	Steps	IDS peak

	`// Error`
	`}`
		during image acquisition.	`infinite number of images`
			`dataStream->StartAcquisition(peak::core::AcquisitionStartMode::Default, peak::core::INFINITE_NUMBER);`
			`nodemapRemoteDevice->FindNode<peak::core::nodes::CommandNode>("AcquisitionStart")->Execute();`
			`// Set this parameter to 1 to lock some nodes while acquisition is running. Set the value to 0 when acquisition is stopped.`
			`nodemapRemoteDevice->FindNode<peak::core::nodes::IntegerNode>("TLParamsLocked")->SetValue(1);`
			`}`
			`catch (const std::exception& e)`
			`{`
			`…`
			`}`

• Wait for frame event (should be done in an extra worker thread)
• Get last finished buffer in the sequence
• Lock buffer
• Render buffer into the display
• Unlock buffer
	`// Wait 5000 ms for frame event`
	`DWORD ret = WaitForSingleObject(hEvent, 5000);`
	`if (WAIT_OBJECT_0 == ret)`
	`{`
	`int num;`
	`char* mem;`
	`char* memLast;`
	`// Get current buffer and last finished buffer`
	`if (IS_SUCCESS != is_GetActSeqBuf(hCam, &num, &mem, &memLast);`
	`{`
	`// Error`
	`}`
	`// Lock last buffer`
	`if (IS_SUCCESS != is_LockSeqBuf(hCam,`
		• Wait for frame event (should be done in an extra worker thread)
		• Create converted image from RAW buffer
		• Re-queue RAW buffer
		• Copy the converted image to the display
			`try`
			`{`
			`// Wait 5 seconds for an image from the camera`
			`uint64_t timeout_ms = 5000;`
			`auto imageBuffer = dataStream->WaitForFinishedBuffer(timeout_ms);`
			`// Convert image to BGRA8`
			`auto imageProcessed = peak::BufferTo<peak::ipl::Image>(imageBuffer).ConvertTo(peak::ipl::PixelFormatName::BGRa8);`
			`// Requeue buffer`
			`dataStream->QueueBuffer(pImageBuffer);`
			`// Copy converted image data (imageProcessed.Data()) to the image display`
			`// See sample SimpleLiveQtWidgets for further information`
			`…`

<table>
<tr><th>Steps</th><th>IDS Software Suite</th><th>Steps</th><th>IDS peak</th></tr>
<tr><td></td><td>

```
        num,
        memLast))
 {
  // Error
 }
 // Render buffer in display window
  if (IS_SUCCESS != is_RenderBitmap(hCam,
           num,
           hWnd,
           IS_RENDER_FIT_TO_WINDOW))
  // Error
 // Unlock last buffer
 if (IS_SUCCESS != is_UnlockSeqBuf(hCam,
           num,
           memLast))
 {
 }
 {
  // Error
 }
}
```

</td><td></td><td>

```
}
catch (const std::exception& e)
{
 …
}
```

If you want to use the higher quality 5x5 debayering instead of the standard 3x3 debayering the function call can be extended:

```
auto imageProcessed =
peak::BufferTo<peak::ipl::Image>(imageBuffer)
.ConvertTo(peak::ipl::PixelFormatName::BGRa8,
 peak::ipl::ConversionMode::HighQuality);
```

</td></tr>
<tr><td>Set exposure time to 1 s (1 000 ms)</td><td>

```
double exposureTime_ms = 1000;
if (IS_SUCCESS != is_Exposure(hCam,
        IS_EXPOSURE_CMD_SET_EXPOSURE,
        (void*)&exposureTime_ms,
           sizeof(exposureTime_ms)
       );
{
 // Error
}
```

</td><td>Set exposure time to 1 s (1 000 000 µs)</td><td>

```
double exposureTime_us = 1000000;
try
{
 nodemapRemoteDevice-
>FindNode<peak::core::nodes::FloatNode>("Expo
sureTime")->SetValue(exposureTime_us);
}
catch (const std::exception& e)
{
 …
}
```

</td></tr>
<tr><td></td><td></td><td>Close IDS peak library</td><td>

```
peak::Library::Close();
```

</td></tr>
</table>

4 Include IDS sherpa files into an IDS Software Suite MFC application to use IDS peak SDK

Specify include paths to the IDS peak API and the IDS peak IPL. Do this for all configurations (32/64-bit, Debug, Release).

- `$(IDS_PEAK_SDK_PATH)\api\include`
- `$(IDS_PEAK_SDK_PATH)\ipl\include`

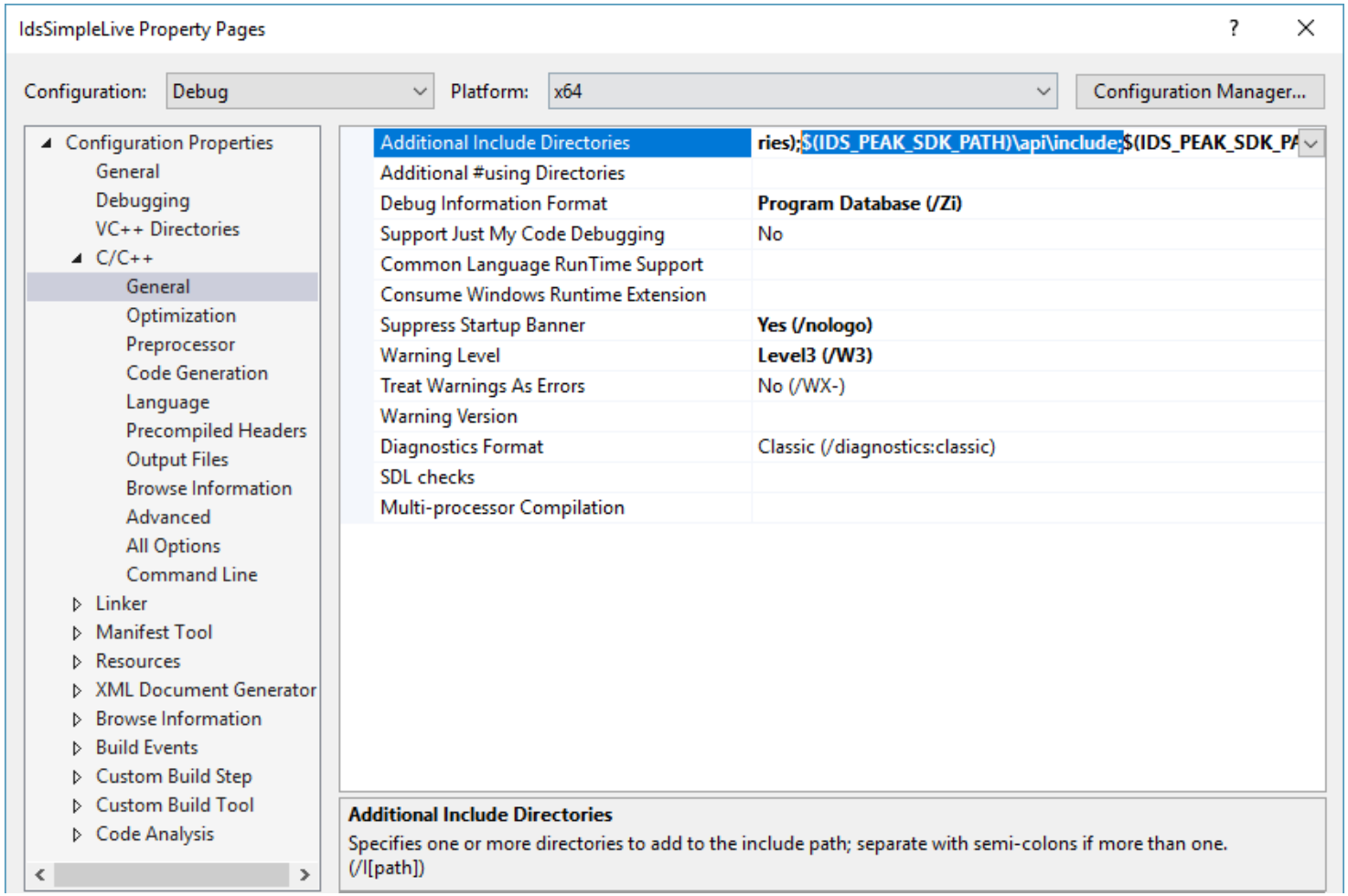

Fig. 1: Include directories

Add IdsSherpa header file ("IdsSherpa.h") and cpp file ("IdsSherpa.cpp") to the project.

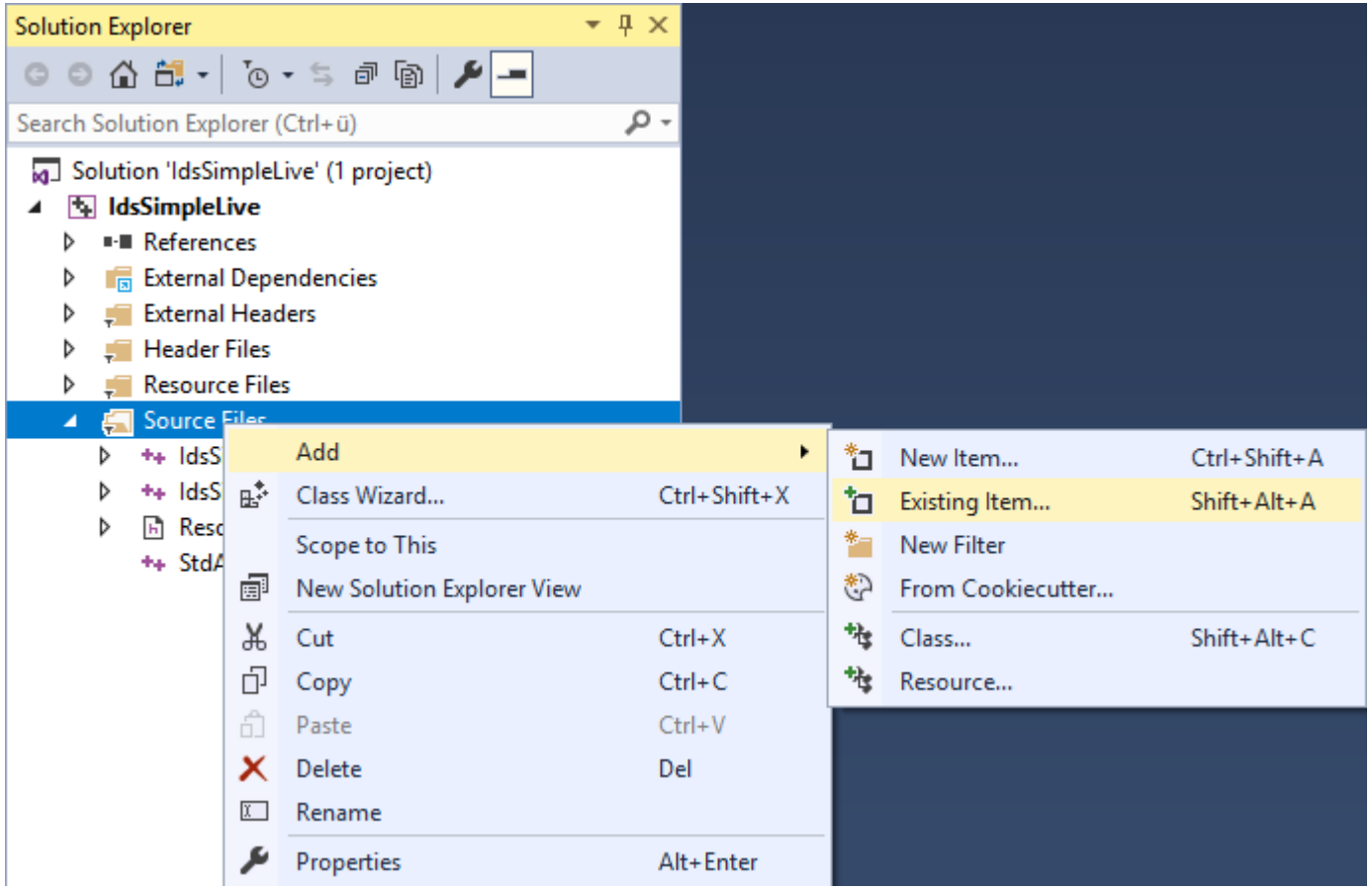

Fig. 2: Add files to project

Add "IdsSherpa.h" to "IdsSimpleLiveDlg.h":

```
#include "IdsSherpa.h"
```

Add "IdsSherpa.h" to "IdsSimpleLiveDlg.cpp":

```
#include "IdsSherpa.h"
```

Specify include paths to the IDS peak API library and the IDS peak IPL library. Do this for all configurations (32/64-bit, Debug, Release).

32-bit	64-bit
• `$(IDS_PEAK_SDK_PATH)\api\lib\x86_32` • `$(IDS_PEAK_SDK_PATH)\ipl\lib\x86_32`	• `$(IDS_PEAK_SDK_PATH)\api\lib\x86_64` • `$(IDS_PEAK_SDK_PATH)\ipl\lib\x86_64`

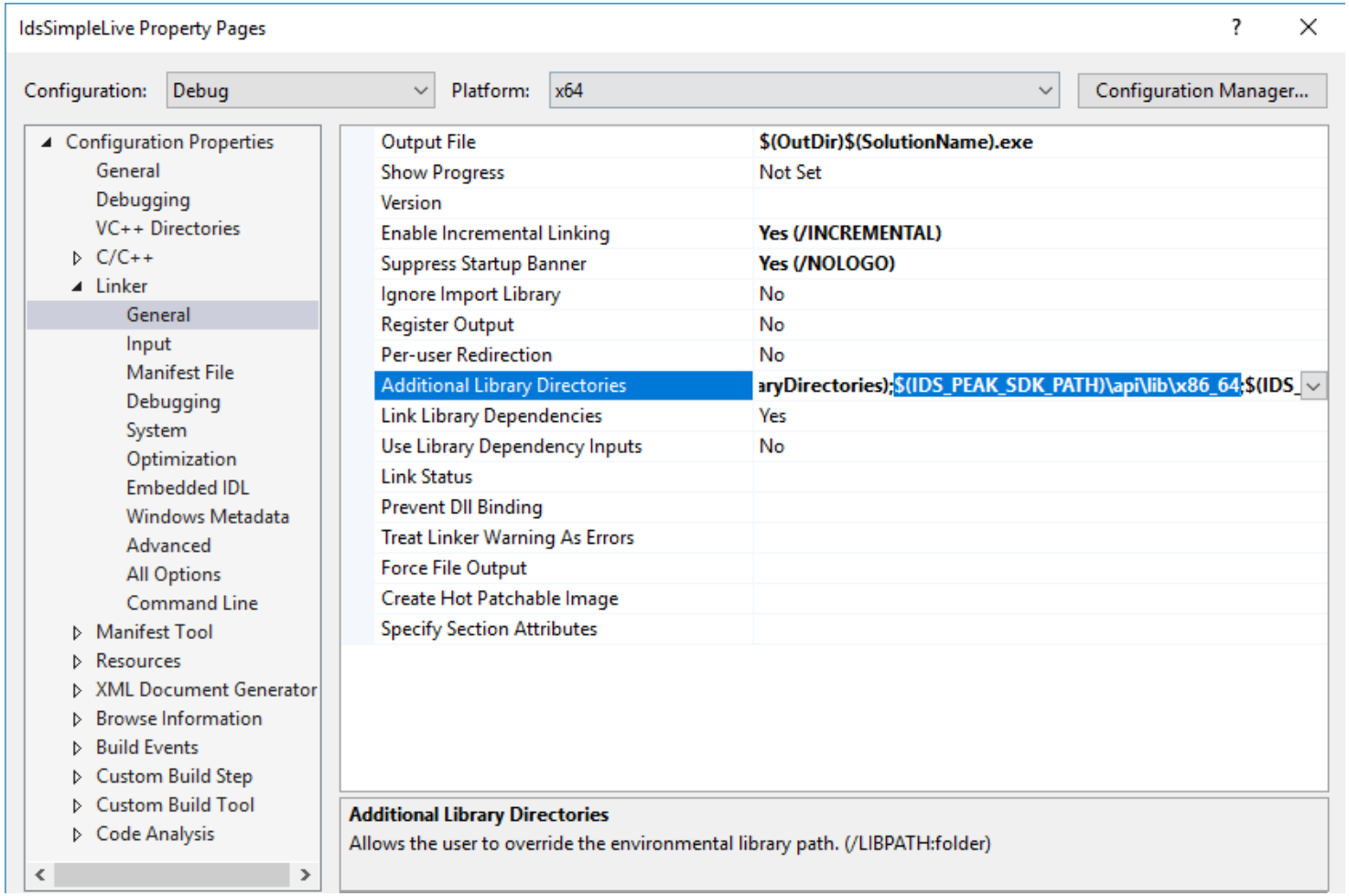

Fig. 3: Additional libraries

Specify IDS peak and IDS peak IPL libraries.

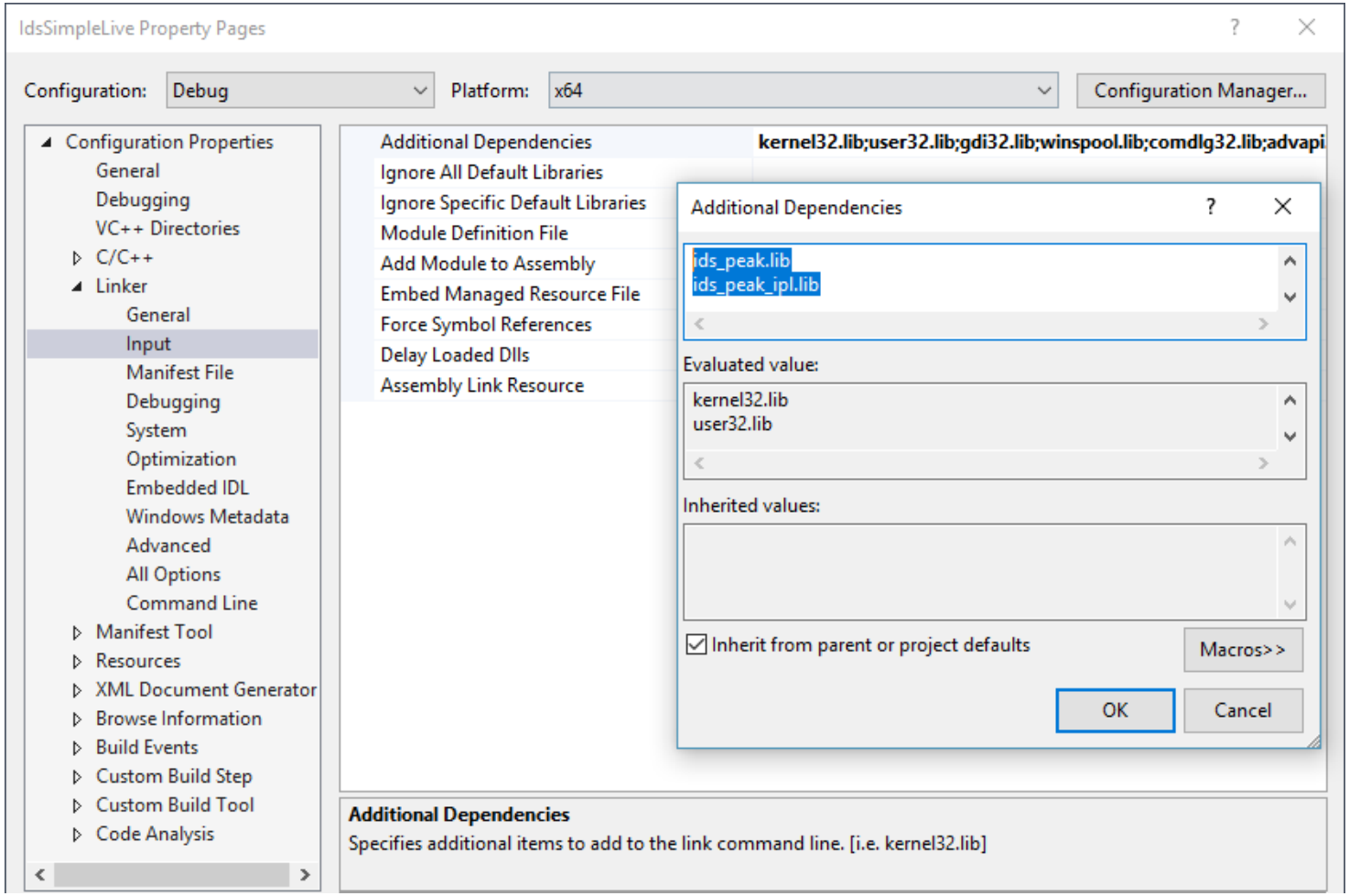

Fig. 4: Specify libraries

Enable additional compiler flags to compile C++ code.

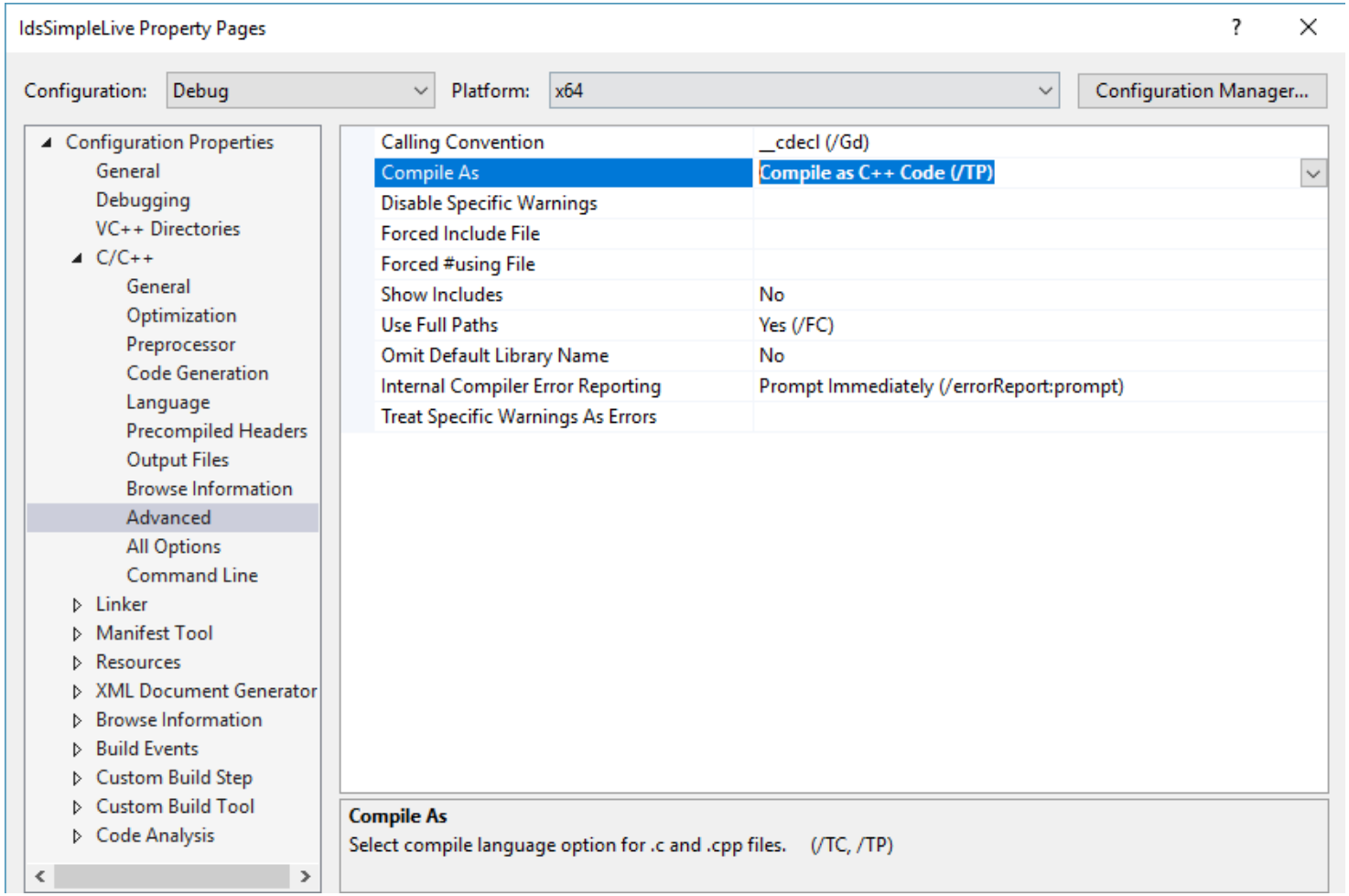

Fig. 5: Compiler flags

Enable additional compiler flags to compile C++ 14. Add the "bigobj" flag to compile large objects.

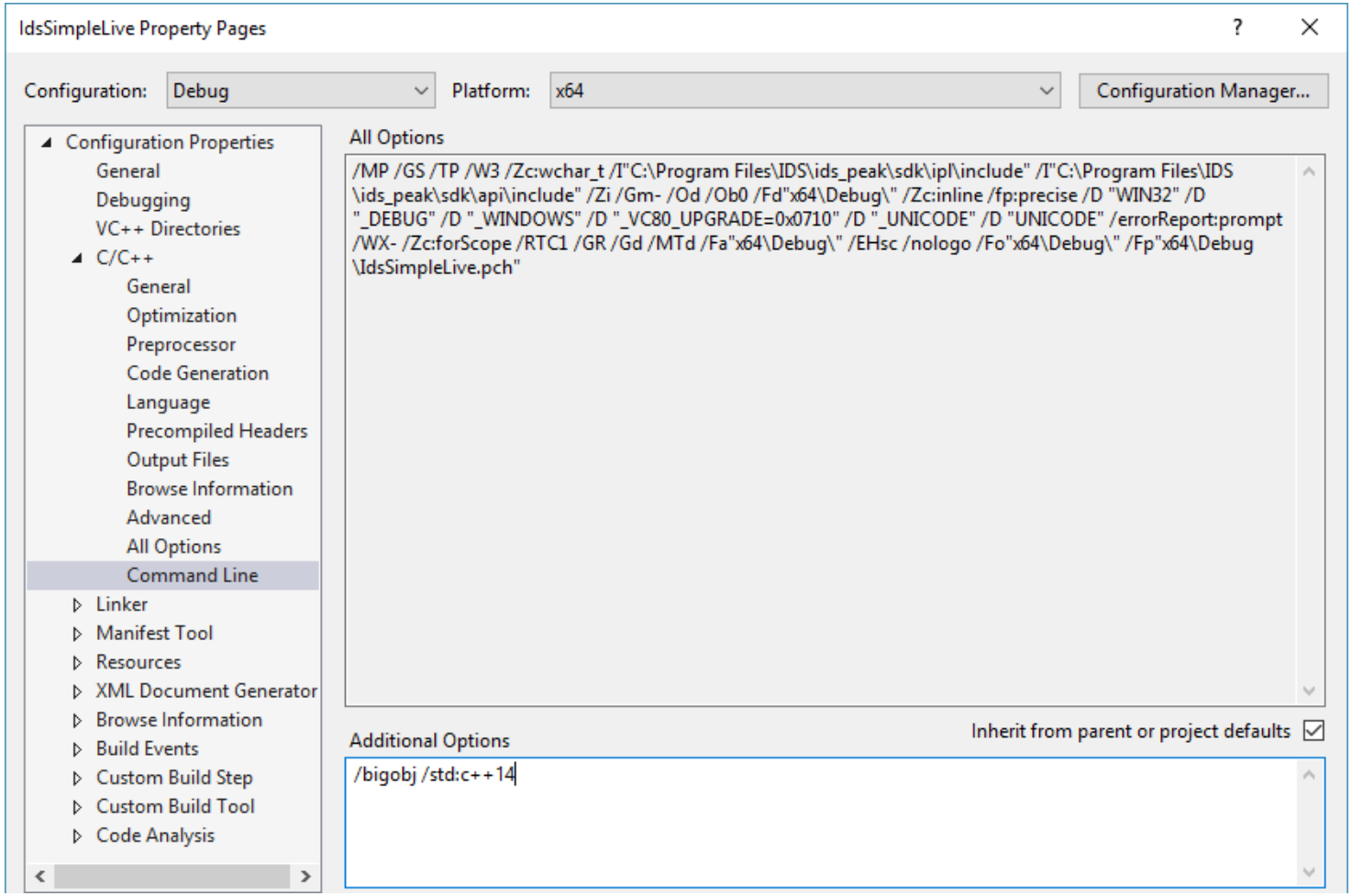

Fig. 6: "bigobj" flag

Copy the IDS peak DLLs and GenICam DLLs to the destination folder, that is, where the compiled *.exe is located. The files are located in the following folders of your IDS peak installation (`IDS_PEAK_SDK_PATH` might be "C:\Program Files\IDS\ids_peak\sdk").

32-bit	64-bit
`<IDS_PEAK_SDK_PATH>\api\lib\x86_32\` • ids_peak.dll • ids_peak_ipl.dll • FirmwareUpdate_MD_VC140_v3_1_IDS.dll • GCBase_MD_VC140_v3_1_IDS.dll • GenApi_MD_VC140_v3_1_IDS.dll • Log_MD_VC140_v3_1_IDS.dll • MathParser_MD_VC140_v3_1_IDS.dll • NodeMapData_MD_VC140_v3_1_IDS.dll • XmlParser_MD_VC140_v3_1_IDS.dll	`<IDS_PEAK_SDK_PATH>\api\lib\x86_64\` • ids_peak.dll • ids_peak_ipl.dll • FirmwareUpdate_MD_VC140_v3_1_IDS.dll • GCBase_MD_VC140_v3_1_IDS.dll • GenApi_MD_VC140_v3_1_IDS.dll • Log_MD_VC140_v3_1_IDS.dll • MathParser_MD_VC140_v3_1_IDS.dll • NodeMapData_MD_VC140_v3_1_IDS.dll • XmlParser_MD_VC140_v3_1_IDS.dll
`<IDS_PEAK_SDK_PATH>\ipl\lib\x86_32\` • ids_peak_ipl.dll	`<IDS_PEAK_SDK_PATH>\ipl\lib\x86_64\` • ids_peak_ipl.dll

Now the program should compile and link. You can also use the sample **IdsSherpaSimple** where all the necessary modifications have already been made. The DLLs are copied automatically in a post-build step to the output folder.

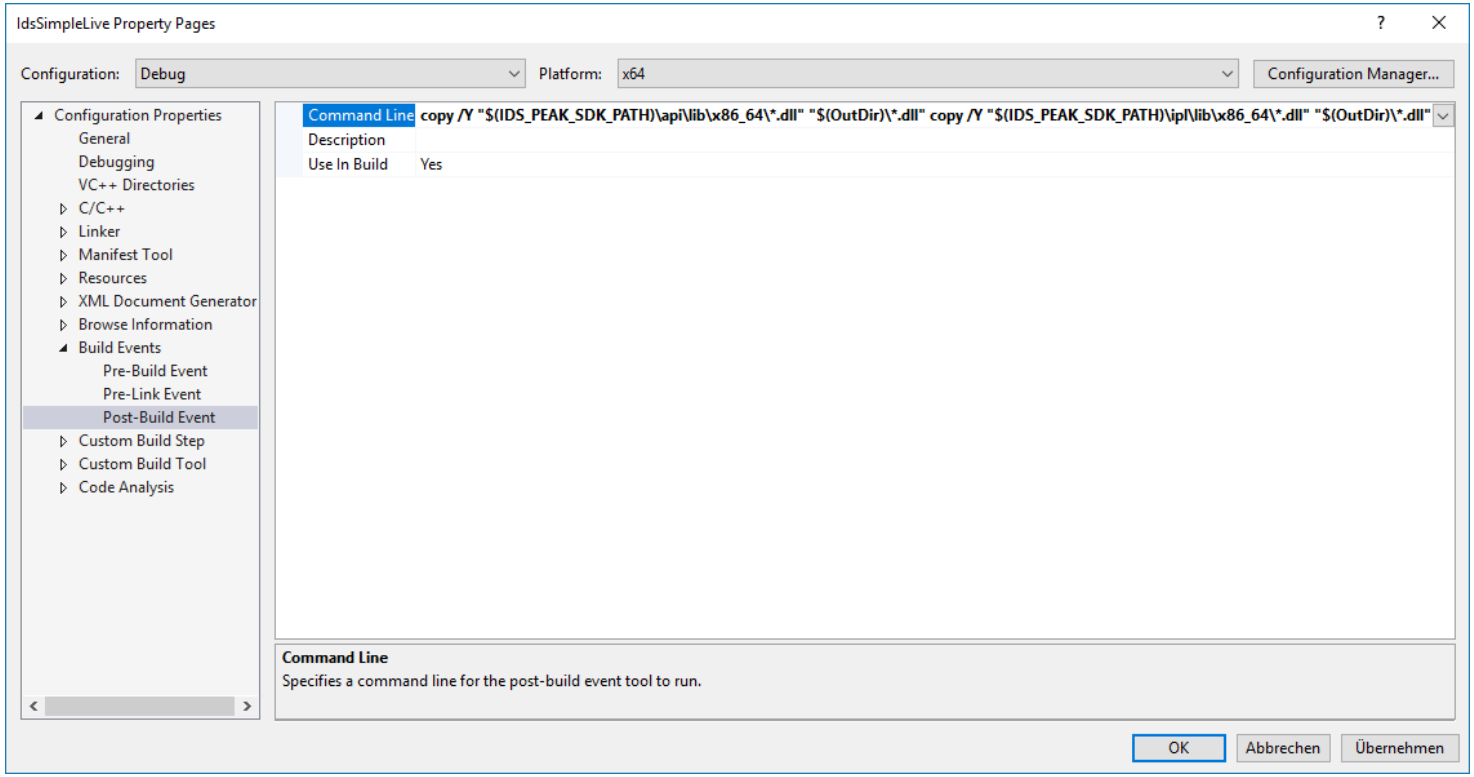

Fig. 7: Post-build step

First steps

Make changes in "IdsSimpleLiveDlg.h". Add a shared pointer on a **IdsSherpa** class for the camera you want to use.

```
class CIdsSimpleLiveDlg : public CDialog
{
    …
private:
    …
    std::shared_ptr<IdsSherpa> m_camera;
```

Changes in "IdsSimpleLiveDlg.cpp":

- Create sherpa object and bind it to a MFC window. You can also pass the window handle later using the `SetWindowHandle()` function.
- Open the first available camera. You can also open a specific camera by passing its serial number.

- Allocate memory and create a sequence. If you specify no number of sequence buffers or zero, the minimum number of required buffers is allocated. See **IdsSherpaSimple** for further details.
- Specify if you want to render the images scaled to the window or in original size.
- Start image acquisition

```
// initialize IdsSherpa object
m_camera = std::make_shared<IdsSherpa>(GetDlgItem(IDC_DISPLAY_PEAK)->m_hWnd);

// Open first available vision camera
bool ret = m_camera->InitCamera();

ret = m_camera->AllocImageBuffersAndCreateSequence();
ret = m_camera->SetRenderType(PEAK_DISPLAY_FIT_TO_WINDOW);
ret = m_camera->StartCapture();
```