

# Basler Cameras

```
// Create an instant camera object with the first
Camera_t camera( CTlFactory::GetInstance().Create

// Register an image event handler that accesses
camera.RegisterImageEventHandler( new CSampleImage
Ownership_TakeOwnership);

// Open the camera.
camera.Open();
```

## **PYLON SDK SAMPLES MANUAL**

Document Number: AW0014888

Version: 06 Language: 000 (English)

Release Date: 19 July 2022

Software Version: 7.x

# Contacting Basler Support Worldwide

## **Europe, Middle East, Africa**

Basler AG  
An der Strusbek 60–62  
22926 Ahrensburg  
Germany

Tel. +49 4102 463 515  
Fax +49 4102 463 599  
[support.europe@baslerweb.com](mailto:support.europe@baslerweb.com)

## **The Americas**

Basler, Inc.  
855 Springdale Drive, Suite 203  
Exton, PA 19341  
USA

Tel. +1 610 280 0171  
Fax +1 610 280 7608  
[support.usa@baslerweb.com](mailto:support.usa@baslerweb.com)

## **Asia-Pacific**

Basler Asia Pte. Ltd.  
35 Marsiling Industrial Estate Road 3  
#05–06  
Singapore 739257

Tel. +65 6367 1355  
Fax +65 6367 1255  
[support.asia@baslerweb.com](mailto:support.asia@baslerweb.com)  
[www.baslerweb.com](http://www.baslerweb.com)

**All material in this publication is subject to change without notice and is copyright  
Basler AG.**

## Table of Contents

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Data Processing API for C++ (Windows, Linux)</b>	<b>2</b>
2.1	Barcode	2
2.2	Camera	4
2.3	Composite Data Types	7
<b>3</b>	<b>C++ Samples</b>	<b>10</b>
3.1	DeviceRemovalHandling	10
3.2	Grab	11
3.3	Grab_CameraEvents	13
3.4	Grab_ChunkImage	15
3.5	Grab_MultiCast	16
3.6	Grab_MultipleCameras	19
3.7	Grab_Strategies	20
3.8	Grab_UsingActionCommand	22
3.9	Grab_UsingBufferFactory	24
3.10	Grab_UsingExposureEndEvent	25
3.11	Grab_UsingGrabLoopThread	27
3.12	Grab_UsingSequencer	29
3.13	GUI_ImageWindow	30
3.14	GUI_Sample	32
3.15	GUI_SampleMultiCam	34
3.16	GUI_QtMultiCam	36
3.17	ParametrizeCamera_AutoFunctions	38
3.18	ParametrizeCamera_Configurations	39
3.19	ParametrizeCamera_GenericParameterAccess	41
3.20	ParametrizeCamera_LoadAndSave	43
3.21	ParametrizeCamera_LookupTable	44
3.22	ParametrizeCamera_NativeParameterAccess	45
3.23	ParametrizeCamera_SerialCommunication	47
3.24	ParametrizeCamera_Shading	48
3.25	ParametrizeCamera_UserSets	49
3.26	Utility_GrabAvi	50
3.27	Utility_GrabVideo	51
3.28	Utility_Image	52
3.29	Utility_ImageDecompressor	54
3.30	Utility_ImageFormatConverter	55
3.31	Utility_ImageLoadAndSave	57
3.32	Utility_InstantInterface	59
3.33	Utility_IpConfig	60

<b>4</b>	<b>C Samples .....</b>	<b>61</b>
4.1	ActionCommands .....	61
4.2	Chunks .....	64
4.3	Events .....	66
4.4	GenApiParam.....	69
4.5	GrabTwoCameras .....	71
4.6	ImageDecompressor .....	73
4.7	OverlappedGrab.....	75
4.8	ParametrizeCamera .....	77
4.9	SimpleGrab .....	79
4.10	SurpriseRemoval.....	81
<b>5</b>	<b>.NET Samples .....</b>	<b>83</b>
5.1	DeviceRemovalHandling .....	83
5.2	Grab .....	85
5.3	Grab_CameraEvents.....	87
5.4	Grab_ChunkImage .....	89
5.5	Grab_MultiCast .....	91
5.6	Grab_Strategies .....	93
5.7	Grab_UsingActionCommand.....	94
5.8	Grab_UsingBufferFactory.....	95
5.9	Grab_UsingExposureEndEvent.....	96
5.10	Grab_UsingGrabLoopThread .....	97
5.11	Grab_UsingSequencer .....	99
5.12	GUISampleMultiCam.....	101
5.13	ParametrizeCamera .....	103
5.14	ParametrizeCamera_AutoFunctions .....	104
5.15	ParametrizeCamera_AutomaticImageAdjustment.....	106
5.16	ParametrizeCamera_Configurations .....	108
5.17	ParametrizeCamera_LoadAndSave .....	109
5.18	ParametrizeCamera_LookupTable.....	110
5.19	ParametrizeCamera_UserSets.....	111
5.20	PylonLiveView .....	112
5.21	Utility_AnnounceRemoteDevice .....	114
5.22	Utility_GrabAvi.....	115
5.23	Utility_GrabVideo .....	116
5.24	Utility_ImageDecompressor .....	117
5.25	Utility_IpConfig .....	118
5.26	VBGrab .....	119
5.27	VBParametrizeCamera .....	121

# 1 Overview

The pylon Camera Software Suite includes an SDK with the following APIs:

- pylon Data Processing API for C++ (Windows, Linux)
- pylon API for C++ (Windows, Linux, and macOS)
- pylon API for C (Windows and Linux)
- pylon API for .NET languages, e.g., C# and VB.NET (Windows only)

Along with the APIs, the pylon Camera Software Suite also includes a set of sample programs and documentation.

This manual describes the SDK sample programs.

- On **Windows** operating systems, the source code for the samples can be found here:  
`<pylon installation directory>\Basler\pylon 7\Development\Samples`  
**Example:** `C:\Program Files\Basler\pylon 7\Development\Samples`
- On **Linux** or **macOS** operating systems, the source code for the samples can be copied from the archive to any location on the target computer.

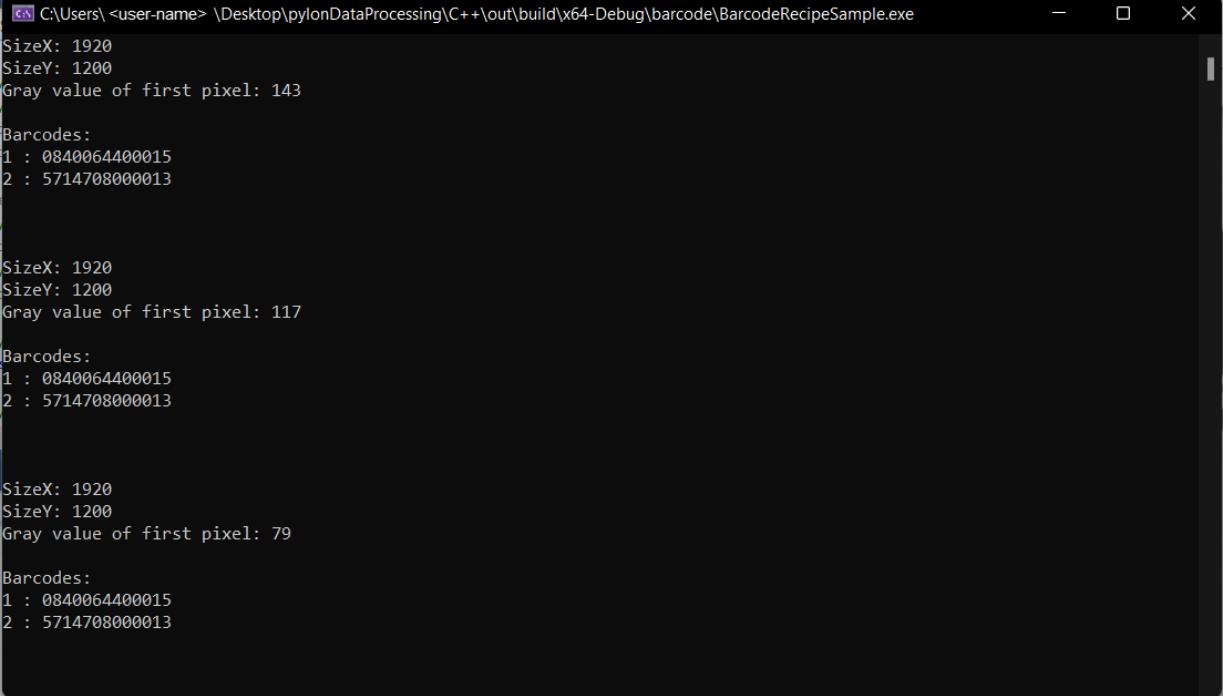
For more information about programming using the pylon API, refer to the *Programmer's Guide and Reference Documentation* documents delivered with the pylon Camera Software Suite.

## 2 Data Processing API for C++ (Windows, Linux)

### 2.1 Barcode

This sample demonstrates how to use the Barcode vTool. The Barcode vTool requires a valid evaluation license or runtime license.

This sample uses a predefined **barcode.precipe** file, the pylon camera emulation and sample images to demonstrate reading of up to two barcodes of an EAN type.



```
C:\Users\<user-name> \Desktop\pylonDataProcessing\C+\out\build\x64-Debug\barcode\BarcodeRecipeSample.exe
SizeX: 1920
SizeY: 1200
Gray value of first pixel: 143

Barcodes:
1 : 0840064400015
2 : 5714708000013

SizeX: 1920
SizeY: 1200
Gray value of first pixel: 117

Barcodes:
1 : 0840064400015
2 : 5714708000013

SizeX: 1920
SizeY: 1200
Gray value of first pixel: 79

Barcodes:
1 : 0840064400015
2 : 5714708000013
```



## Code

The `MyOutputObserver` class is used to create a helper object that shows how to handle output data provided via the `IOutputObserver::OutputDataPush` interface method. Also, `MyOutputObserver` shows how a thread-safe queue can be implemented for later processing while pulling the output data.

The `CRecipe` class is used to create a recipe object representing a recipe file that is created using the pylon Viewer Workbench.

The `recipe.Load()` method is used to load a recipe file.

The `recipe.PreAllocateResources()` method allocates all needed resources, e.g., it opens the camera device and allocates buffers for grabbing.

The `recipe.RegisterAllOutputsObserver()` method is used to register the `MyOutputObserver` object, which is used for collecting the output data, e.g., barcodes.

The `recipe.Stop()` method is eventually called to stop the processing.

## Applicable Interfaces

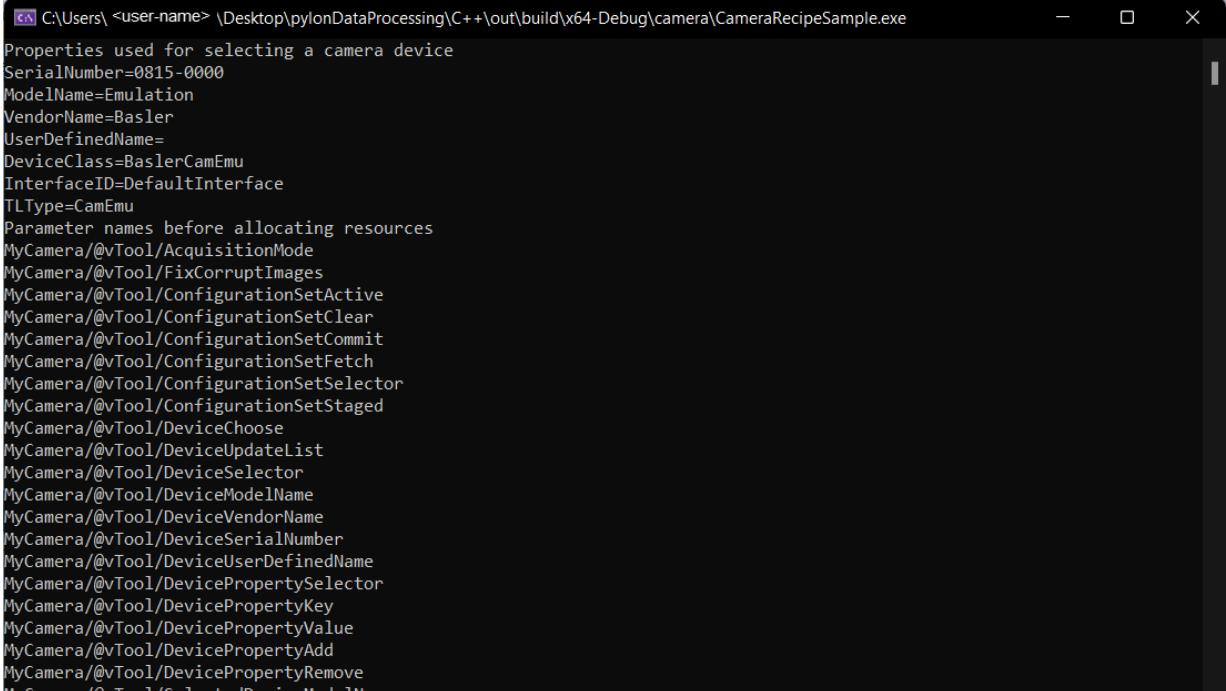
GigE Vision, USB3 Vision, CXP

 [Back to top](#)

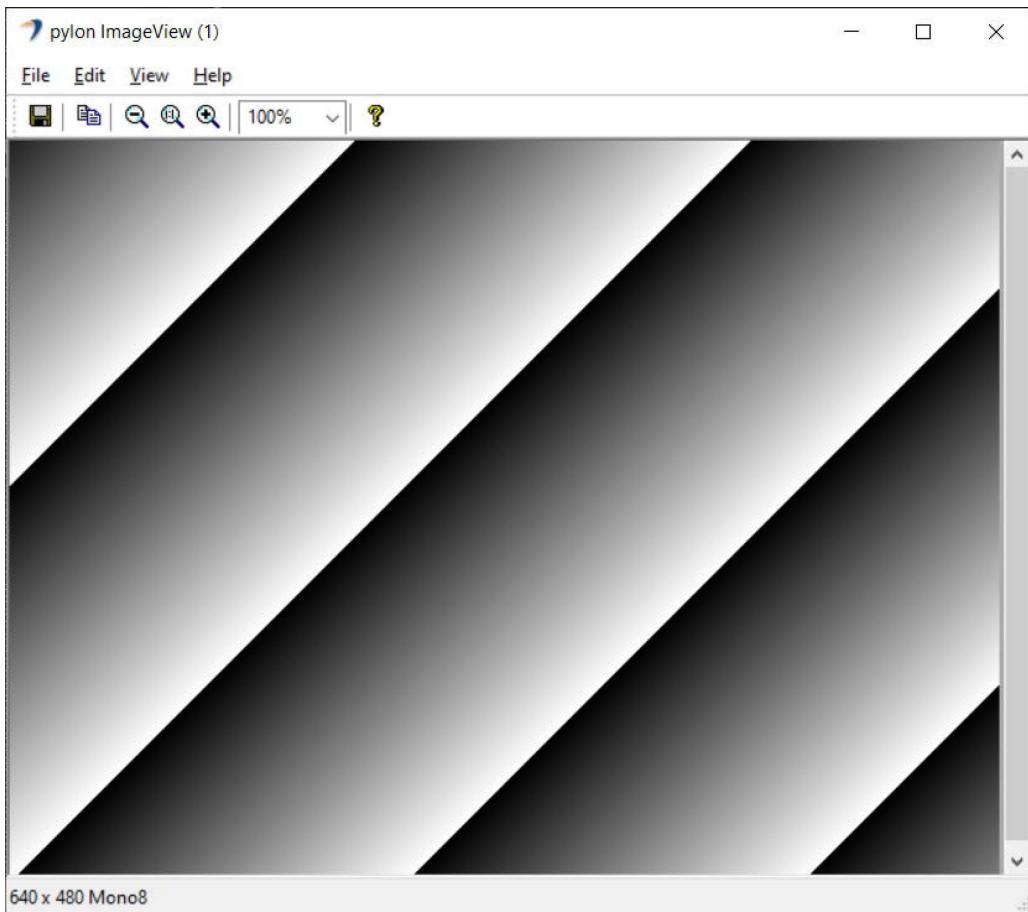
## 2.2 Camera

This sample demonstrates how to use and parametrize the Camera vTool. The Camera vTool doesn't require a license.

This sample uses a predefined **camera.precipe** file and the pylon Camera Emulation for demonstration purposes.



```
C:\Users\<user-name> \Desktop\pylonDataProcessing\C++\out\build\x64-Debug\camera\CameraRecipeSample.exe
Properties used for selecting a camera device
SerialNumber=0815-0000
ModelName=Emulation
VendorName=Basler
UserDefinedName=
DeviceClass=BaslerCamEmu
InterfaceID=DefaultInterface
TLType=CamEmu
Parameter names before allocating resources
MyCamera/@vTool/AcquisitionMode
MyCamera/@vTool/FixCorruptImages
MyCamera/@vTool/ConfigurationSetActive
MyCamera/@vTool/ConfigurationSetClear
MyCamera/@vTool/ConfigurationSetCommit
MyCamera/@vTool/ConfigurationSetFetch
MyCamera/@vTool/ConfigurationSetSelector
MyCamera/@vTool/ConfigurationSetStaged
MyCamera/@vTool/DeviceChoose
MyCamera/@vTool/DeviceUpdateList
MyCamera/@vTool/DeviceSelector
MyCamera/@vTool/DevicemodelName
MyCamera/@vTool/DeviceVendorName
MyCamera/@vTool/DeviceSerialNumber
MyCamera/@vTool/DeviceUserDefinedName
MyCamera/@vTool/DevicePropertySelector
MyCamera/@vTool/DevicePropertyKey
MyCamera/@vTool/DevicePropertyValue
MyCamera/@vTool/DevicePropertyAdd
MyCamera/@vTool/DevicePropertyRemove
MyCamera/@vTool/SelectedDevicemodelName
```



## Code

The [MyOutputObserver](#) class is used to create a helper object that shows how to handle output data provided via the [IOutputObserver::OutputDataPush](#) interface method. Also, [MyOutputObserver](#) shows how a thread-safe queue can be implemented for later processing while pulling the output data.

The [CRecipe](#) class is used to create a recipe object that represents a recipe file created using the pylon Viewer Workbench.

The [recipe.Load\(\)](#) method is used to load a recipe file. After loading the recipe, some [Pylon::CDeviceInfo](#) properties of the camera are accessed and displayed on the console for demonstration purposes.

The [recipe.PreAllocateResources\(\)](#) method allocates all needed resources, e.g., it opens the camera device and allocates buffers for grabbing. After opening the camera device, some camera parameters are read out and printed for demonstration purposes.

The [recipe.RegisterAllOutputsObserver\(\)](#) method is used to register the [MyOutputObserver](#) object, which is used for collecting the output data, e.g., images.

The [recipe.Stop\(\)](#) method is eventually called to stop the processing.

The [recipe.DeallocateResources\(\)](#) method is called to free all used resources.

## Applicable Interfaces

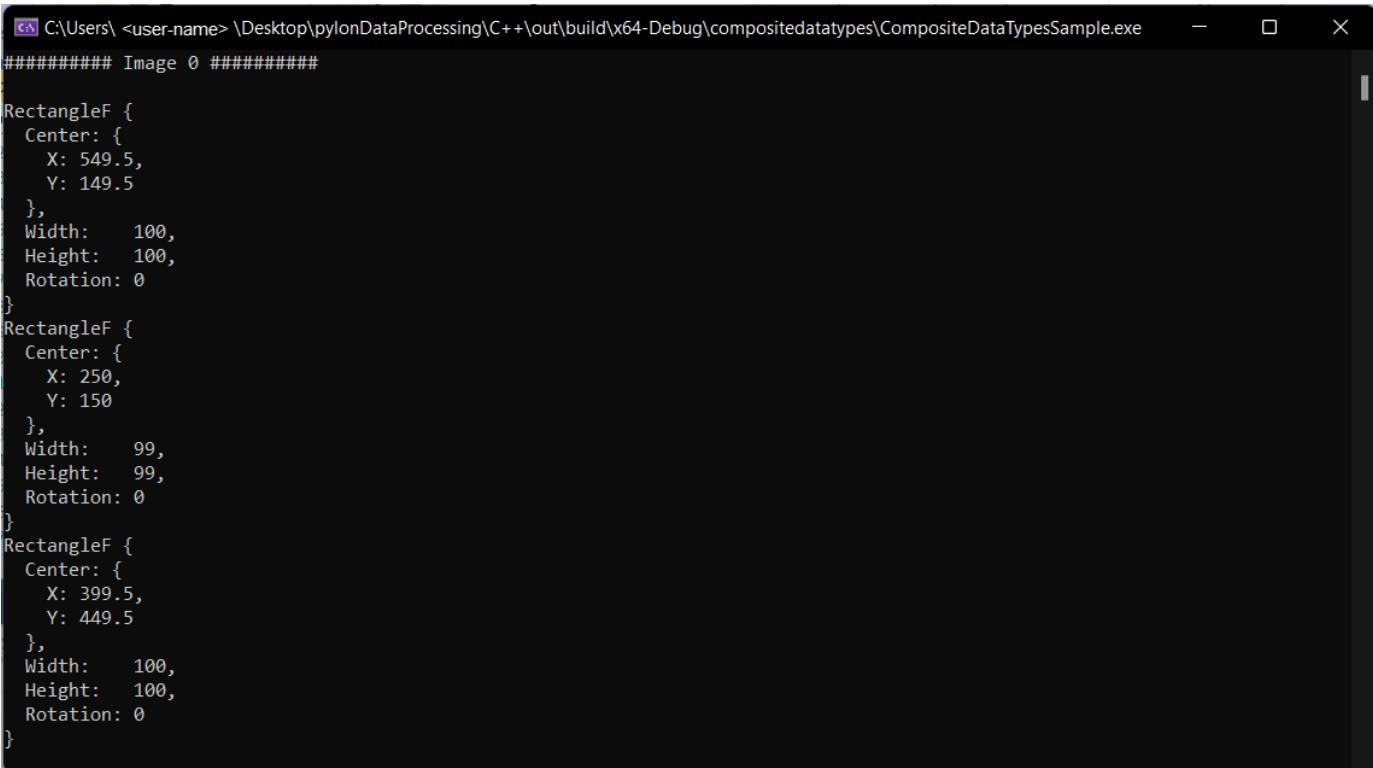
GigE Vision, USB3 Vision, CXP

 [Back to top](#)

## 2.3 Composite Data Types

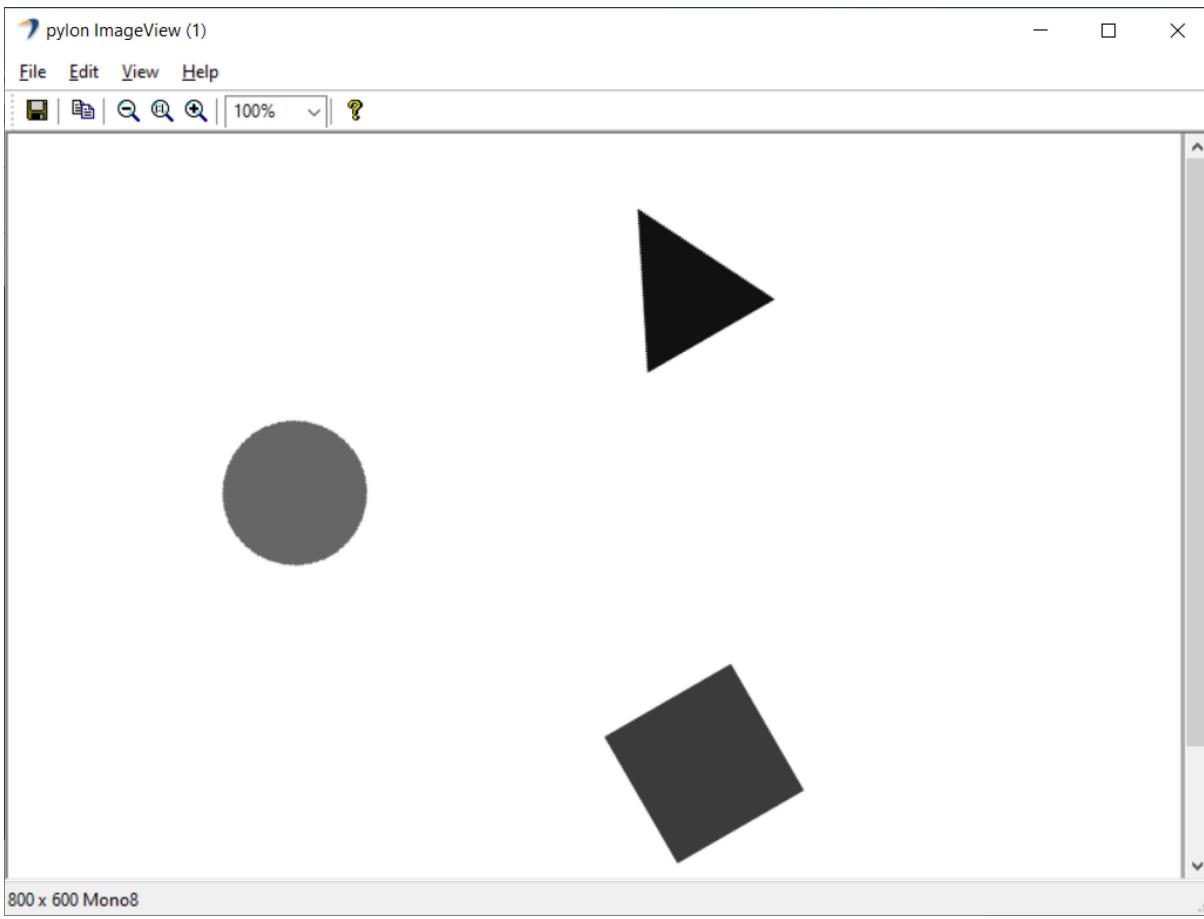
This sample demonstrates how to use vTools that output composite data types, e.g., [AbsoluteThresholding](#), [RegionMorphology](#), [RegionFeatureExtraction](#), etc. These vTools require a valid evaluation license or runtime license. Data types like [PointF](#) or [RectangleF](#) are composite data types.

This sample obtains information about the composition, i.e., [RectangleF](#), and accesses the data, e.g., [Center X](#), [Center Y](#), [Width](#), [Height](#) and [Rotation](#) by using a predefined **composite\_data\_types.precipe** file, the pylon camera emulation, and sample images. It depends on the input or output terminal pin(s), whether image coordinates in pixels and world coordinates in meter are used.



```
C:\Users\ <user-name> \Desktop\pylonDataProcessing\C++\out\build\x64-Debug\compositedatatypes\CompositeDataTypesSample.exe
#####
# Image 0 #####
#####

RectangleF {
    Center: {
        X: 549.5,
        Y: 149.5
    },
    Width:    100,
    Height:   100,
    Rotation: 0
}
RectangleF {
    Center: {
        X: 250,
        Y: 150
    },
    Width:    99,
    Height:   99,
    Rotation: 0
}
RectangleF {
    Center: {
        X: 399.5,
        Y: 449.5
    },
    Width:    100,
    Height:   100,
    Rotation: 0
}
```



## Code

The struct `RectangleF` in the class `ResultData` is used to store the data of the composite data type.

The `MyOutputObserver` class is used to create a helper object that shows how to handle output data provided via the `IOutputObserver::OutputDataPush` interface method. Also, `MyOutputObserver` shows how a thread-safe queue can be implemented for later processing while pulling the output data.

The `CVariant` and the `GetSubValue()` show how to access the data of composite data types of `RectangleF`.

The `CRecipe` class is used to create a recipe object that represents a recipe file created using the pylon Viewer Workbench.

The `recipe.Load()` method is used to load a recipe file.

The `recipe.PreAllocateResources()` method allocates all needed resources, e.g., it opens the camera device and allocates buffers for grabbing.

The `recipe.RegisterAllOutputsObserver()` method is used to register the `MyOutputObserver` object, which is used for collecting the output data, e.g., images and composite data types.

The `recipe.Stop()` method is eventually called to stop the processing.

With the `recipe.DeallocateResources()` method you can free all used resources.

## Applicable Interfaces

GigE Vision, USB3 Vision, CXP

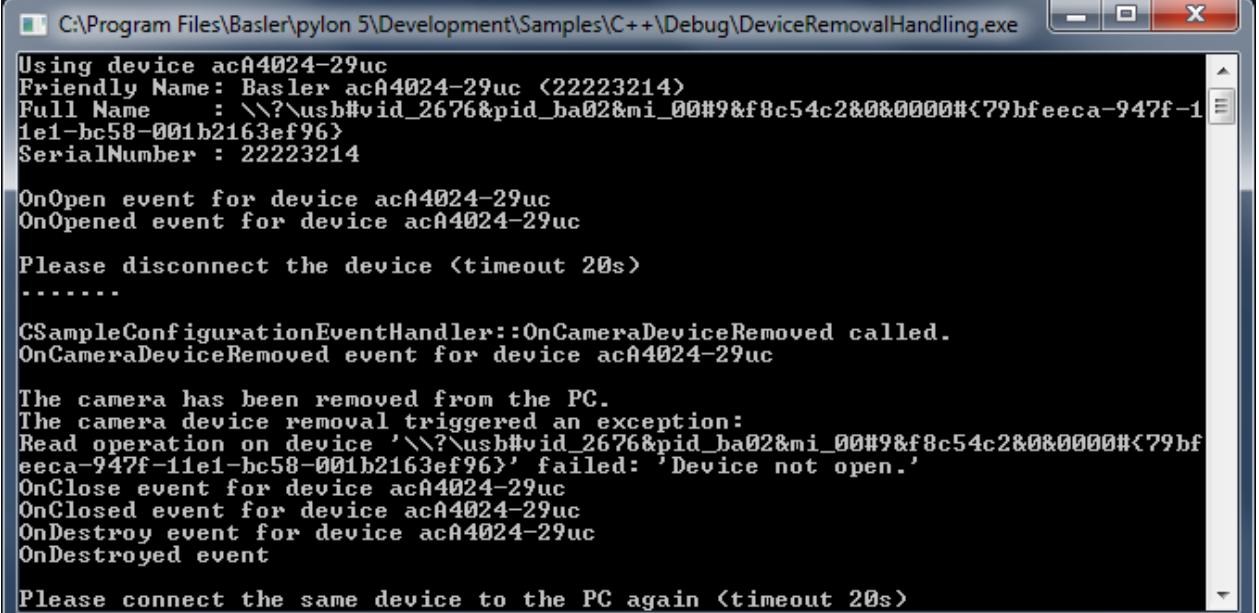
 [Back to top](#)

## 3 C++ Samples

### 3.1 DeviceRemovalHandling

This sample demonstrates how to detect the removal of a camera device. It also shows you how to reconnect to a removed device.

**Note:** If you build this sample in debug mode and run it using a GigE camera device, pylon will set the heartbeat timeout to 5 minutes. This is done to allow debugging and single-stepping without losing the camera connection due to missing heartbeats. However, with this setting, it would take 5 minutes for the application to notice that a GigE device has been disconnected. As a workaround, the heartbeat timeout is set to 1000 ms.



The screenshot shows a Windows command-line window with the title bar "C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug\DeviceRemovalHandling.exe". The window contains the following text output:

```
Using device acA4024-29uc
Friendly Name: Basler acA4024-29uc (22223214)
Full Name : \\?\usb#vid_2676&pid_ba02&mi_00#9&f8c54c2&0&0000#<79bfeeca-947f-1
1e1-bc58-001b2163ef96>
SerialNumber : 22223214

OnOpen event for device acA4024-29uc
OnOpened event for device acA4024-29uc

Please disconnect the device <timeout 20s>
.....
CSampleConfigurationEventHandler::OnCameraDeviceRemoved called.
OnCameraDeviceRemoved event for device acA4024-29uc

The camera has been removed from the PC.
The camera device removal triggered an exception:
Read operation on device '\\?\usb#vid_2676&pid_ba02&mi_00#9&f8c54c2&0&0000#<79bf
eeea-947f-11e1-bc58-001b2163ef96>' failed: 'Device not open.'
OnClose event for device acA4024-29uc
OnClosed event for device acA4024-29uc
OnDestroy event for device acA4024-29uc
OnDestroyed event

Please connect the same device to the PC again <timeout 20s>
```

#### Code

The [CTIFactory](#) class is used to create a generic transport layer.

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

The [CHearbeatHelper](#) class is used to set the HeartbeatTimeout to an appropriate value.

The [CSampleConfigurationEventHandler](#) is used to handle device removal events.

#### Applicable Interfaces

GigE Vision, USB3 Vision, Camera Link

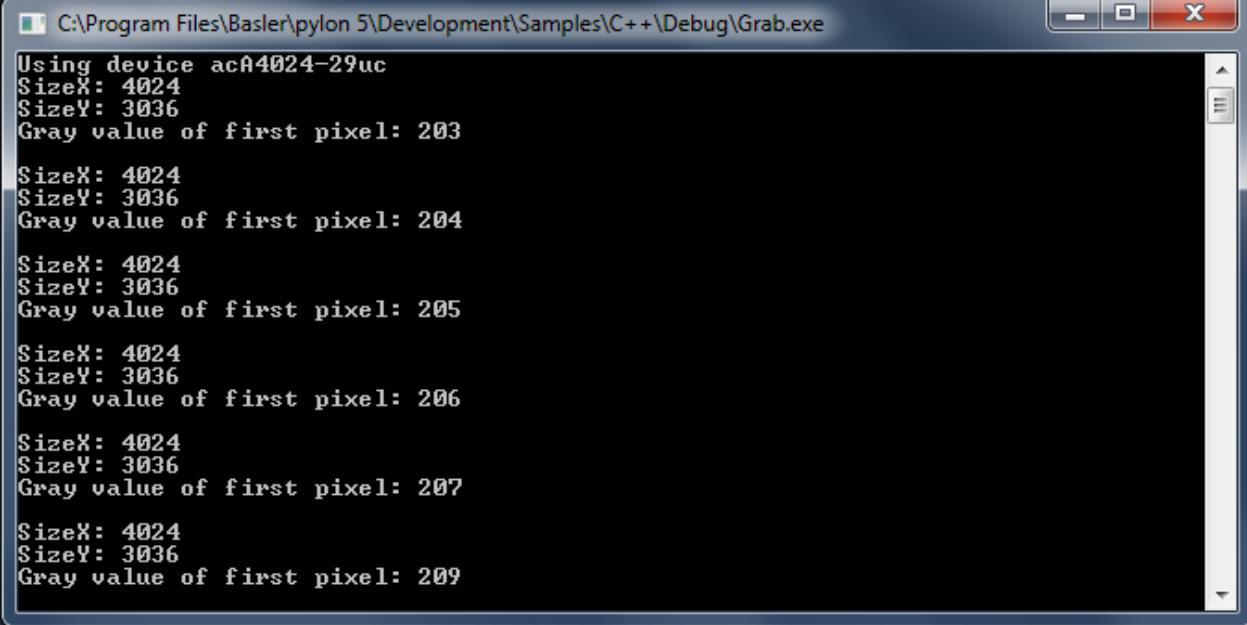
[^ Back to top](#)

## 3.2 Grab

This sample demonstrates how to grab and process images using the [CInstantCamera](#) class.

The images are grabbed and processed asynchronously, i.e., at the same time that the application is processing a buffer, the acquisition of the next buffer takes place.

The [CInstantCamera](#) class uses a pool of buffers to retrieve image data from the camera device. Once a buffer is filled and ready, the buffer can be retrieved from the camera object for processing. The buffer and additional image data are collected in a grab result. The grab result is held by a smart pointer after retrieval. The buffer is automatically reused when explicitly released or when the smart pointer object is destroyed.



```
C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug\Grab.exe
Using device acA4024-29uc
SizeX: 4024
SizeY: 3036
Gray value of first pixel: 203

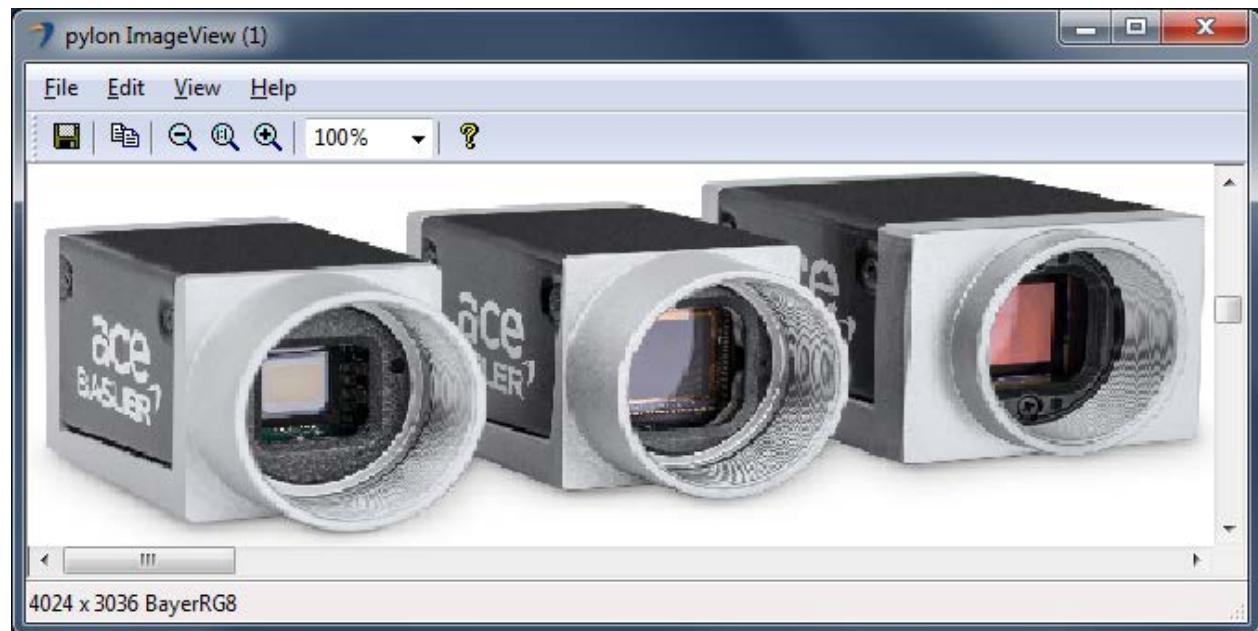
SizeX: 4024
SizeY: 3036
Gray value of first pixel: 204

SizeX: 4024
SizeY: 3036
Gray value of first pixel: 205

SizeX: 4024
SizeY: 3036
Gray value of first pixel: 206

SizeX: 4024
SizeY: 3036
Gray value of first pixel: 207

SizeX: 4024
SizeY: 3036
Gray value of first pixel: 209
```



## Code

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

The [DisplayImage](#) class is used to display the grabbed images.

## Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[↖ Back to top](#)

### 3.3 Grab\_CameraEvents

Basler USB3 Vision and GigE Vision cameras can send event messages. For example, when a sensor exposure has finished, the camera can send an Exposure End event to the computer. The event can be received by the computer before the image data of the finished exposure has been transferred completely. This sample demonstrates how to be notified when camera event message data is received.

The event messages are automatically retrieved and processed by the InstantCamera classes. The information carried by event messages is exposed as parameter nodes in the camera node map and can be accessed like standard camera parameters. These nodes are updated when a camera event is received. You can register camera event handler objects that are triggered when event data has been received.

These mechanisms are demonstrated for the Exposure End and the Event Overrun events.

The Exposure End event carries the following information:

- `ExposureEndEventFrameID`: Number of the image that has been exposed.
- `ExposureEndEventTimestamp`: Time when the event was generated.
- `ExposureEndEventStreamChannelIndex`: Number of the image data stream used to transfer the image. On Basler cameras, this parameter is always set to 0.

The Event Overrun event is sent by the camera as a warning that events are being dropped. The notification contains no specific information about how many or which events have been dropped.

Events may be dropped if events are generated at a high frequency and if there isn't enough bandwidth available to send the events.

This sample also shows you how to register event handlers that indicate the arrival of events sent by the camera. For demonstration purposes, different handlers are registered for the same event.

**Note:** Different camera families implement different versions of the Standard Feature Naming Convention (SFNC). That's why the name and the type of the parameters used can be different.

```
C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug_GigE\Grab_CameraEvents_Gi...
OnOpen event for device acA640-300gc
OnOpened event for device acA640-300gc
OnGrabStart event for device acA640-300gc
OnGrabStarted event for device acA640-300gc
OnCameraEvent event for device acA640-300gc
User provided ID: 100
Event data node name: ExposureEndEventTimestamp
Event node data: 620364212904

OnCameraEvent event for device acA640-300gc
User provided ID: 100
Event data node name: ExposureEndEventFrameID
Event node data: 1

Exposure End event. FrameID: 1 Timestamp: 620364212904
CSampleImageEventHandler::OnImageGrabbed called.

OnCameraEvent event for device acA640-300gc
User provided ID: 100
Event data node name: ExposureEndEventTimestamp
Event node data: 620367307046
```

## Code

The [CBaslerUniversalInstantCamera](#) class is used to create a camera object with the first found camera device independent of its interface.

The [CSoftwareTriggerConfiguration](#) class is used to register the standard configuration event handler for enabling software triggering. The software trigger configuration handler replaces the default configuration handler.

The [CSampleCameraEventHandler](#) class demonstrates the usage of example handlers for camera events.

The [CSampleImageEventHandler](#) class demonstrates the usage of an image event handler.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

## Applicable Interfaces

GigE Vision, USB3 Vision

[↖ Back to top](#)

### 3.4 Grab\_ChunkImage

Basler cameras supporting the Data Chunk feature can generate supplementary image data, e.g., frame count, time stamp, or CRC checksums, and append it to each acquired image.

This sample demonstrates how to enable the Data Chunks feature, how to grab images, and how to process the appended data. When the camera is in chunk mode, it transfers data blocks that are partitioned into chunks. The first chunk is always the image data. The data chunks that you have chosen follow the image data chunk.

```
Using device acA640-300gc
OnImageGrabbed: TimeStamp <Result> accessed via node map: 626039598326
OnImageGrabbed: TimeStamp <Result> accessed via result member: 626039598326
GrabSucceeded: 1
SizeX: 640
SizeY: 480
Gray value of first pixel: 0
TimeStamp <Result>: 626039598326
FrameCounter <Result>: 23327

OnImageGrabbed: TimeStamp <Result> accessed via node map: 626040231948
OnImageGrabbed: TimeStamp <Result> accessed via result member: 626040231948
GrabSucceeded: 1
SizeX: 640
SizeY: 480
Gray value of first pixel: 0
TimeStamp <Result>: 626040231948
FrameCounter <Result>: 23328

OnImageGrabbed: TimeStamp <Result> accessed via node map: 626040865572
OnImageGrabbed: TimeStamp <Result> accessed via result member: 626040865572
GrabSucceeded: 1
SizeX: 640
SizeY: 480
Gray value of first pixel: 0
```

#### Code

The [CBaslerUniversalInstantCamera](#) class is used to create a camera object with the first found camera device independent of its interface.

The [CBaslerUniversalGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result and chunk data independent of the camera interface.

The [CSampleImageEventHandler](#) class demonstrates the usage of an image event handler.

The [DisplayImage](#) class is used to display the grabbed images.

#### Applicable Interfaces

GigE Vision, USB3 Vision

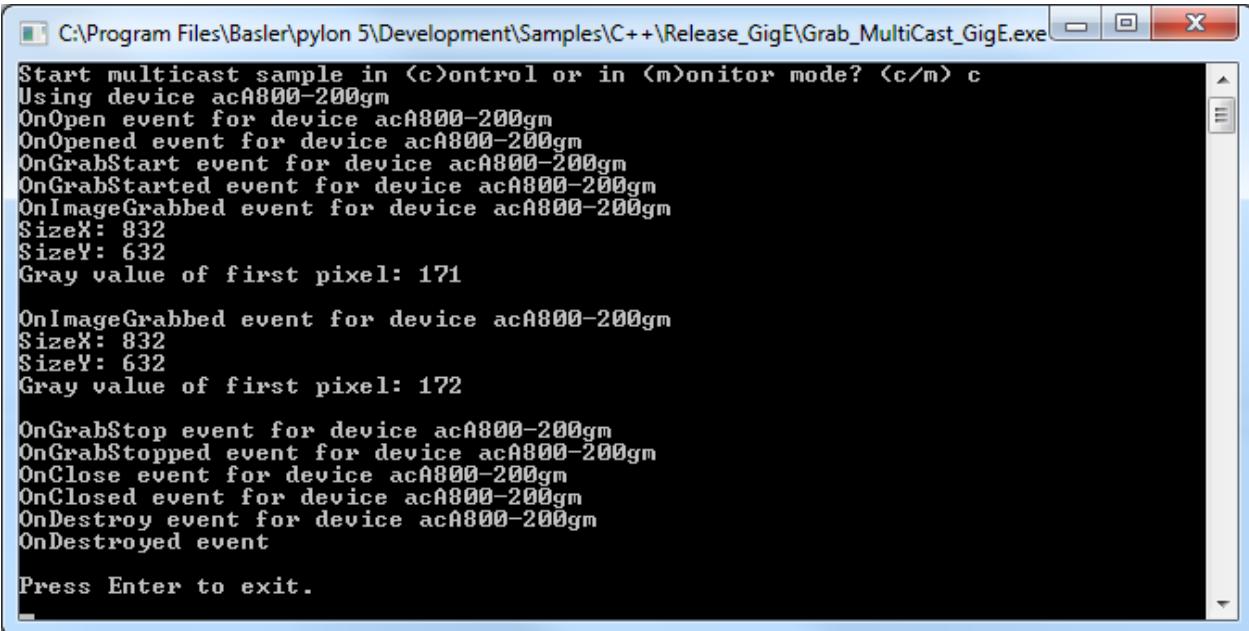
[Back to top](#)

### 3.5 Grab\_MultiCast

This sample applies to Basler GigE Vision cameras only and demonstrates how to open a camera in multicast mode and how to receive a multicast stream.

Two instances of an application must be run simultaneously on different computers. The first application started on computer A acts as the controlling application and has full access to the GigE camera. The second instance started on computer B opens the camera in monitor mode. This instance is not able to control the camera but can receive multicast streams.

To run the sample, start the application on computer A in control mode. After computer A has begun to receive frames, start the second instance of this application on computer B in monitor mode.



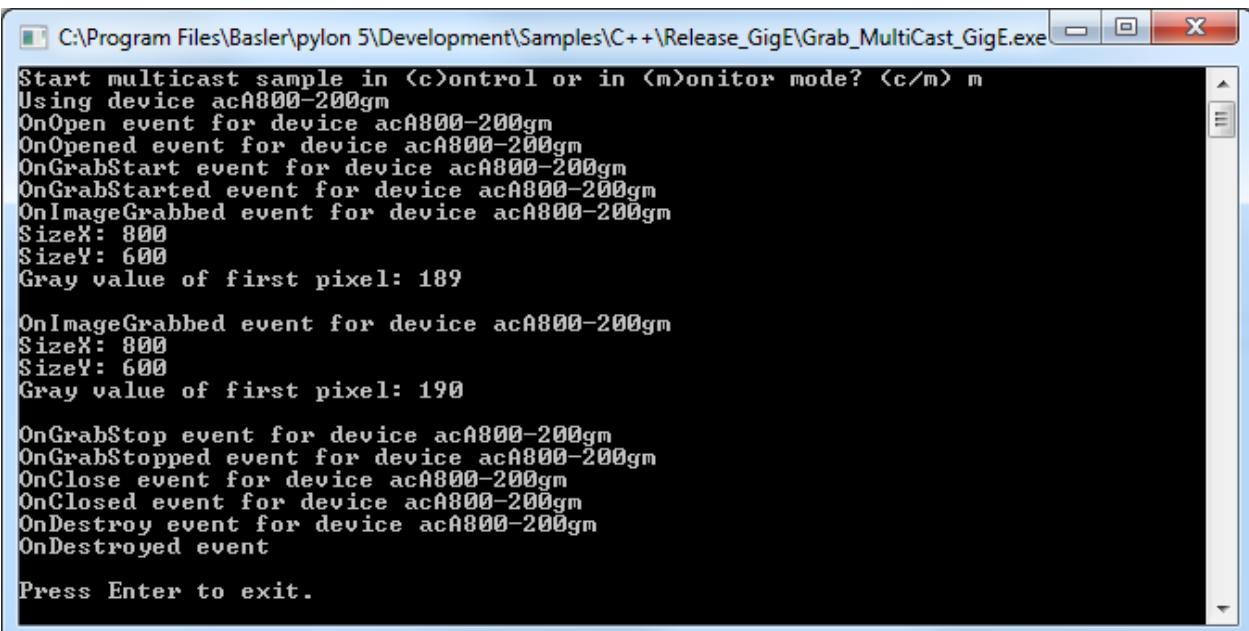
```
C:\Program Files\Basler\pylon 5\Development\Samples\C++\Release_GigE\Grab_MultiCast_GigE.exe
Start multicast sample in <c>ontrol or in <m>onitor mode? <c/m> c
Using device acA800-200gm
OnOpen event for device acA800-200gm
OnOpened event for device acA800-200gm
OnGrabStart event for device acA800-200gm
OnGrabStarted event for device acA800-200gm
OnImageGrabbed event for device acA800-200gm
SizeX: 832
SizeY: 632
Gray value of first pixel: 171

OnImageGrabbed event for device acA800-200gm
SizeX: 832
SizeY: 632
Gray value of first pixel: 172

OnGrabStop event for device acA800-200gm
OnGrabStopped event for device acA800-200gm
OnClose event for device acA800-200gm
OnClosed event for device acA800-200gm
OnDestroy event for device acA800-200gm
OnDestroyed event

Press Enter to exit.
```

Control application started on computer A.



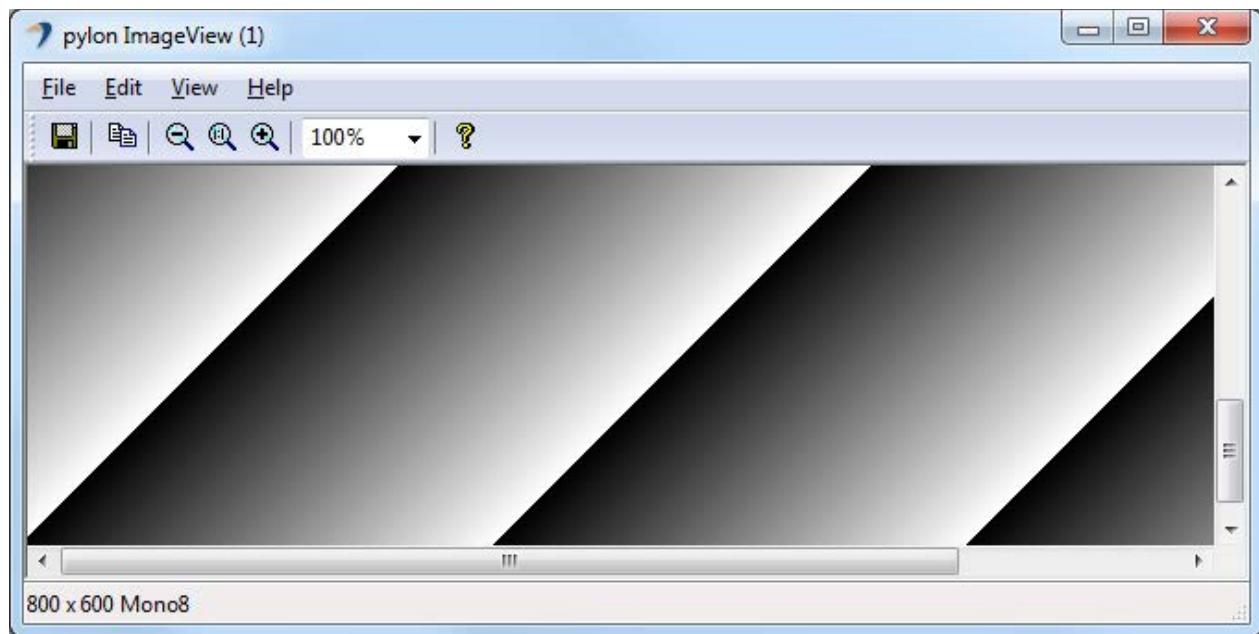
```
C:\Program Files\Basler\pylon 5\Development\Samples\C++\Release_GigE\Grab_MultiCast_GigE.exe
Start multicast sample in <c>ontrol or in <m>onitor mode? <c/m> m
Using device acA800-200gm
OnOpen event for device acA800-200gm
OnOpened event for device acA800-200gm
OnGrabStart event for device acA800-200gm
OnGrabStarted event for device acA800-200gm
OnImageGrabbed event for device acA800-200gm
SizeX: 800
SizeY: 600
Gray value of first pixel: 189

OnImageGrabbed event for device acA800-200gm
SizeX: 800
SizeY: 600
Gray value of first pixel: 190

OnGrabStop event for device acA800-200gm
OnGrabStopped event for device acA800-200gm
OnClose event for device acA800-200gm
OnClosed event for device acA800-200gm
OnDestroy event for device acA800-200gm
OnDestroyed event

Press Enter to exit.
```

Monitor application started on computer B.



## Code

The [CDeviceInfo](#) class is used to look for cameras with a specific interface, i.e., GigE Vision only ([BaslerGigEDeviceClass](#)).

The [CBaslerUniversalInstantCamera](#) class is used to find and create a camera object for the first GigE camera found.

When the camera is opened in control mode, the transmission type must be set to "multicast". In this case, the IP address and the IP port must also be set. This is done by the following command:

```
camera.GetStreamGrabberParams().TransmissionType = TransmissionType_Multicast;
```

When the camera is opened in monitor mode, i.e., the camera is already controlled by another application and configured for multicast, the active camera configuration can be used. In this case, the IP address and IP port will be set automatically:

```
camera.GetStreamGrabberParams().TransmissionType = TransmissionType_UseCameraConfig;
```

[RegisterConfiguration\(\)](#) is used to remove the default camera configuration. This is necessary when a monitor mode is selected because the monitoring application is not allowed to modify any camera parameter settings.

The [CConfigurationEventPrinter](#) and [CImageEventPrinter](#) classes are used for information purposes to print details about events being called and image grabbing.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

## Applicable Interfaces

GigE Vision

[↖ Back to top](#)

## 3.6 Grab\_MultipleCameras

This sample demonstrates how to grab and process images from multiple cameras using the [CInstantCameraArray](#) class. The [CInstantCameraArray](#) class represents an array of Instant Camera objects. It provides almost the same interface as the Instant Camera for grabbing.

The main purpose of the [CInstantCameraArray](#) is to simplify waiting for images and camera events of multiple cameras in one thread. This is done by providing a single `RetrieveResult` method for all cameras in the array.

Alternatively, the grabbing can be started using the internal grab loop threads of all cameras in the [CInstantCameraArray](#). The grabbed images can then be processed by one or more image event handlers. Note that this is not shown in this sample.

```
C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug\Grab_MultipleCameras.exe
Using device acA4024-29uc
Using device acA640-300gc
Camera 1: acA640-300gc
GrabSucceeded: 1
SizeX: 640
SizeY: 480
Gray value of first pixel: 0

Camera 0: acA4024-29uc
GrabSucceeded: 1
SizeX: 4024
SizeY: 3036
Gray value of first pixel: 188

Camera 1: acA640-300gc
GrabSucceeded: 1
SizeX: 640
SizeY: 480
Gray value of first pixel: 0

Camera 0: acA4024-29uc
GrabSucceeded: 1
SizeX: 4024
SizeY: 3036
Gray value of first pixel: 188
```

### Code

The [CInstantCameraArray](#) class demonstrates how to create an array of Instant Cameras for the devices found.

[StartGrabbing\(\)](#) starts grabbing sequentially for all cameras, starting with index 0, 1, etc.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

The [DisplayImage](#) class is used to show the image acquired by each camera in a separate window for each camera.

### Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[Back to top](#)

### 3.7 Grab\_Strategies

This sample demonstrates the use of the [CInstantCamera](#) grab strategies `GrabStrategy_OneByOne`, `GrabStrategy_LatestImageOnly`, `GrabStrategy_LatestImages`, and `GrabStrategy_UpcomingImage`.

When the "OneByOne" grab strategy is used, images are processed in the order of their acquisition. This strategy can be useful when all grabbed images need to be processed, e.g., in production and quality inspection applications.

The "LatestImageOnly" and "LatestImages" strategies can be useful when the acquired images are only displayed on screen. If the processor has been busy for a while and images could not be displayed automatically, the latest image is displayed when processing time is available again.

The "UpcomingImage" grab strategy can be used to make sure to get an image that has been grabbed after [RetrieveResult\(\)](#) has been called. This strategy cannot be used with USB3 Vision cameras.

```
Using device acA4024-29uc
OnOpen event for device acA4024-29uc
OnOpened event for device acA4024-29uc
Grab using the GrabStrategy_OneByOne default strategy:

OnGrabStart event for device acA4024-29uc
OnGrabStarted event for device acA4024-29uc

Grab results wait in the output queue.

OnImageGrabbed event for device acA4024-29uc
SizeX: 4024
SizeY: 3036
Gray value of first pixel: 197

OnImageGrabbed event for device acA4024-29uc
SizeX: 4024
SizeY: 3036
Gray value of first pixel: 198

OnImageGrabbed event for device acA4024-29uc
SizeX: 4024
SizeY: 3036
Gray value of first pixel: 199
```

#### Code

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

The [CSoftwareTriggerConfiguration](#) class is used to register the standard configuration event handler for enabling software triggering. The software trigger configuration handler replaces the default configuration.

[StartGrabbing\(\)](#) is used to demonstrate the usage of the different grab strategies.

## Applicable Interfaces

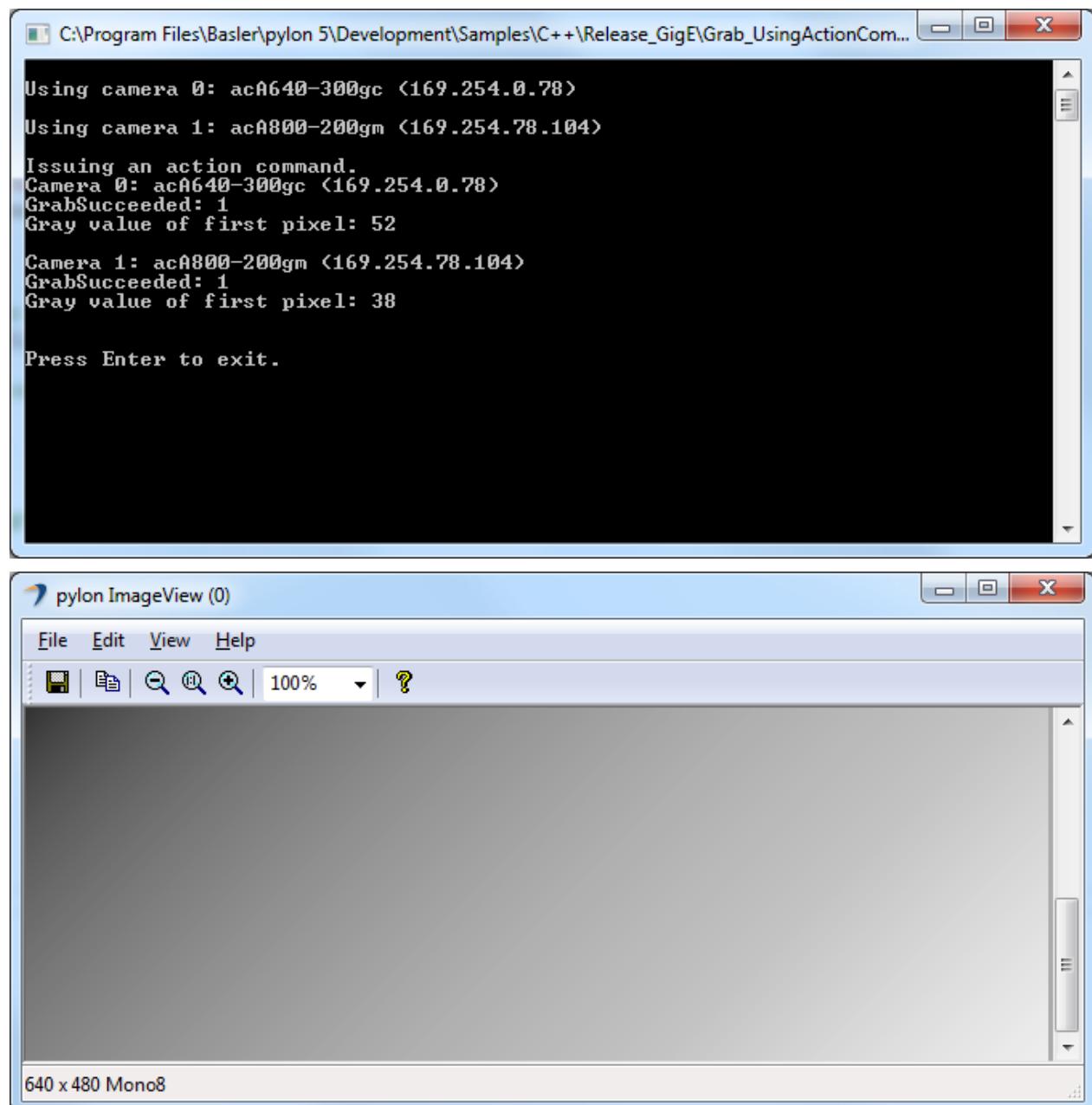
GigE Vision, USB3 Vision, CXP

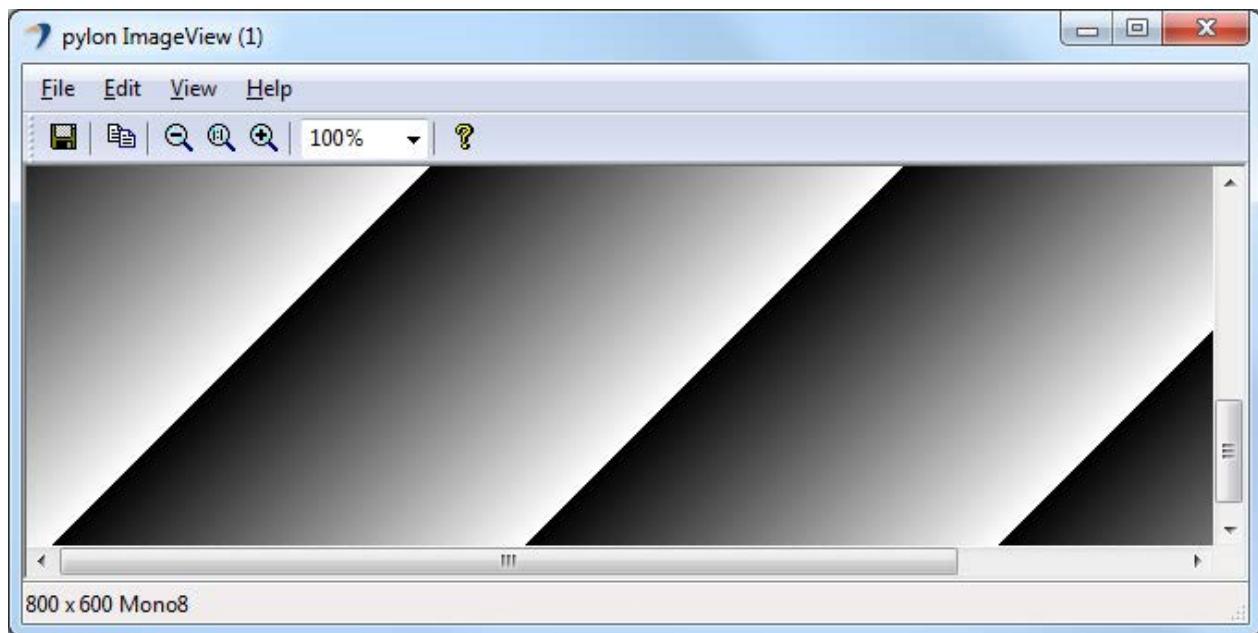
[↖ Back to top](#)

### 3.8 Grab\_UsingActionCommand

This sample applies to Basler GigE Vision cameras only and demonstrates how to issue a GigE Vision ACTION\_CMD to multiple cameras.

By using an action command, multiple cameras can be triggered at the same time as opposed to software triggering where each camera must be triggered individually.





## Code

To make the configuration of multiple cameras easier, this sample uses the [CBaslerUniversalInstantCameraArray](#) class.

The [IGigETransportLayer](#) interface is used to issue action commands.

The [CActionTriggerConfiguration](#) class is used to set up the basic action command features.

The [CBaslerUniversalGrabResultPtr](#) class is used to declare and initialize a smart pointer to receive the grab result data. When the cameras in the array are created, a camera context value is assigned to the index number of the camera in the array. The camera context is a user-settable value, which is attached to each grab result and can be used to determine the camera that produced the grab result, i.e., `ptrGrabResult->GetCameraContext()`.

The [DisplayImage](#) class is used to display the grabbed images.

## Applicable Interfaces

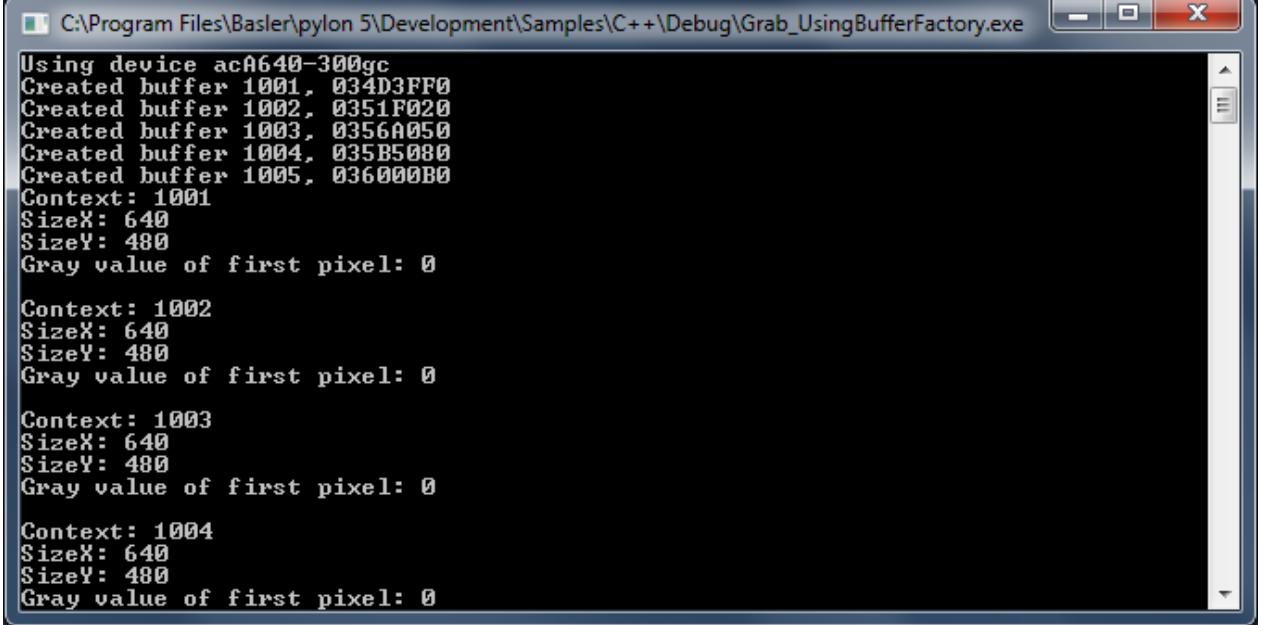
GigE Vision

[↖ Back to top](#)

### 3.9 Grab\_UsingBufferFactory

This sample demonstrates the use of a user-provided buffer factory.

The use of a buffer factory is optional and intended for advanced use cases only. A buffer factory is only required if you plan to grab into externally supplied buffers.



```
Using device acA640-300gc
Created buffer 1001, 034D3FF0
Created buffer 1002, 0351F020
Created buffer 1003, 0356A050
Created buffer 1004, 035B5080
Created buffer 1005, 036000B0
Context: 1001
SizeX: 640
SizeY: 480
Gray value of first pixel: 0

Context: 1002
SizeX: 640
SizeY: 480
Gray value of first pixel: 0

Context: 1003
SizeX: 640
SizeY: 480
Gray value of first pixel: 0

Context: 1004
SizeX: 640
SizeY: 480
Gray value of first pixel: 0
```

#### Code

The [MyBufferFactory](#) class demonstrates the usage of a user-provided buffer factory.

The buffer factory must be created first because objects on the stack are destroyed in reverse order of creation. The buffer factory must exist longer than the Instant Camera object in this sample.

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

[SetBufferFactory\(\)](#) provides its own implementation of a buffer factory. Since we control the lifetime of the factory object, we pass the [Cleanup\\_None](#) argument.

#### Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[Back to top](#)

### 3.10 Grab\_UsingExposureEndEvent

This sample demonstrates how to use the Exposure End event to speed up image acquisition.

For example, when a sensor exposure is finished, the camera can send an Exposure End event to the computer.

The computer can receive the event before the image data of the finished exposure has been transferred completely.

This can be used in order to avoid an unnecessary delay, e.g., when an imaged object is moved before the related image data transfer is complete.

Time [ms]	Event	FrameNumber
0.0000	ExposureEndEvent	1
0.0061	Move	1
3.1555	ImageReceived	1
1.6749	ExposureEndEvent	2
0.0016	Move	2
3.2464	ImageReceived	2
1.8275	ExposureEndEvent	3
0.0019	Move	3
3.2444	ImageReceived	3
1.8029	ExposureEndEvent	4
0.0022	Move	4
3.3795	ImageReceived	4
1.6758	ExposureEndEvent	5
0.0016	Move	5

#### Code

The [MyEvents](#) enumeration is used for distinguishing between different events, e.g., `ExposureEndEvent`, `FrameStartOvertrigger`, `EventOverrunEvent`, `ImageReceivedEvent`, `MoveEvent`, `NoEvent`.

The [CEventHandler](#) class is used to register image and camera event handlers.

**Note:** additional handling is required for GigE camera events because the event network packets can be lost, doubled or delayed on the network.

The [CBaslerUniversalInstantCamera](#) class is used to create a camera object with the first found camera device independent of its interface.

The [CConfigurationEventPrinter](#) class is used for information purposes to print details about camera use.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

## Applicable Interfaces

GigE Vision, USB3 Vision

[↖ Back to top](#)

### 3.11 Grab\_UsingGrabLoopThread

This sample demonstrates how to grab and process images using the grab loop thread provided by the [CInstantCamera](#) class.

```
C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug\Grab_UsingGrabLoopThread....  
OnOpen event for device acA4024-29uc  
OnOpened event for device acA4024-29uc  
OnGrabStart event for device acA4024-29uc  
OnGrabStarted event for device acA4024-29uc  
Enter "t" to trigger the camera or "e" to exit and press enter? <t/e>  
t  
OnImageGrabbed event for device acA4024-29uc  
SizeX: 4024  
SizeY: 3036  
Gray value of first pixel: 0  
CSampleImageEventHandler::OnImageGrabbed called.  
  
t  
OnImageGrabbed event for device acA4024-29uc  
SizeX: 4024  
SizeY: 3036  
Gray value of first pixel: 0  
CSampleImageEventHandler::OnImageGrabbed called.
```

#### Code

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

The [CSoftwareTriggerConfiguration](#) class is used to register the standard configuration event handler for enabling software triggering. The software trigger configuration handler replaces the default configuration.

The [CConfigurationEventPrinter](#) class is used for information purposes to print details about camera use.

The [CImageEventPrinter](#) class serves as a placeholder for an image processing task. When using the grab loop thread provided by the Instant Camera object, an image event handler processing the grab results must be created and registered.

[CanWaitForFrameTriggerReady\(\)](#) is used to query the camera device whether it is ready to accept the next frame trigger.

[StartGrabbing\(\)](#) demonstrates how to start grabbing using the grab loop thread by setting the grabLoopType parameter to GrabLoop\_ProvidedByInstantCamera. The grab results are delivered to the image event handlers. The GrabStrategy\_OneByOne default grab strategy is used in this case.

[WaitForFrameTriggerReady\(\)](#) is used to wait up to 500 ms for the camera to be ready for triggering.

[ExecuteSoftwareTrigger\(\)](#) is used to execute the software trigger.

The [DisplayImage](#) class is used to display the grabbed images.

### Applicable Interfaces

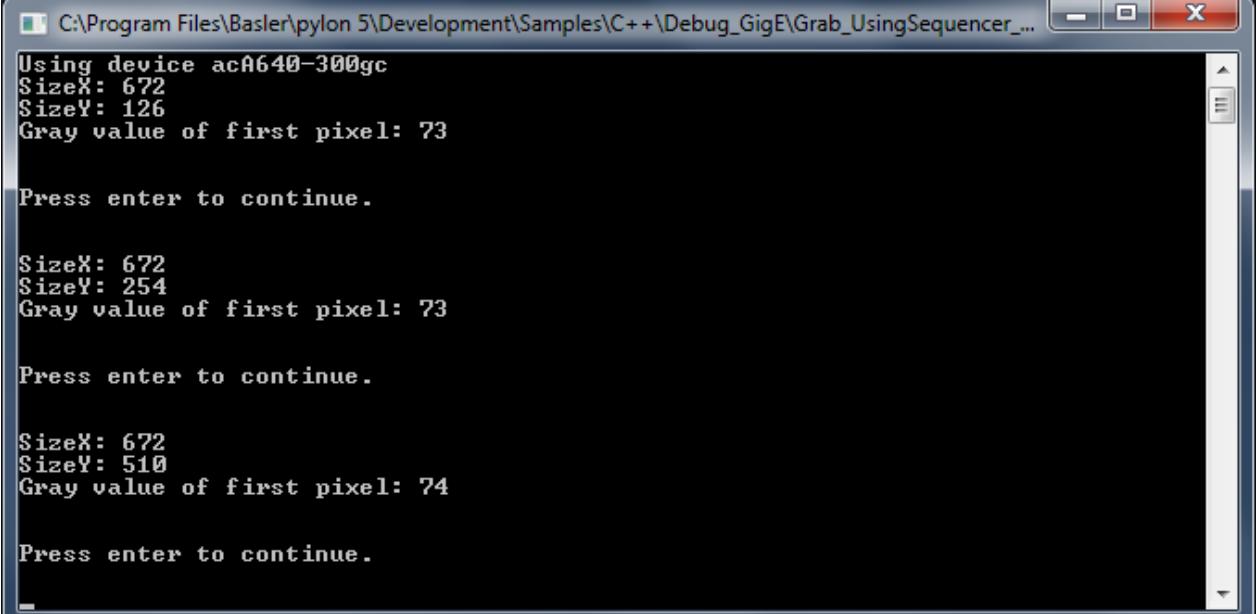
GigE Vision, USB3 Vision, CXP

[⊜ Back to top](#)

### 3.12 Grab\_UsingSequencer

This sample demonstrates how to grab images using the Sequencer feature of a Basler camera.

Three sequence sets are used for image acquisition. Each sequence set uses a different image height.



The screenshot shows a terminal window with the following text output:

```
C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug_GigE\Grab_UsingSequencer...
Using device acA640-300gc
SizeX: 672
SizeY: 126
Gray value of first pixel: 73

Press enter to continue.

SizeX: 672
SizeY: 254
Gray value of first pixel: 73

Press enter to continue.

SizeX: 672
SizeY: 510
Gray value of first pixel: 74

Press enter to continue.
```

#### Code

The [CBaslerUniversalInstantCamera](#) class is used to create a camera object with the first found camera device independent of its interface.

The [CSoftwareTriggerConfiguration](#) class is used to register the standard configuration event handler for enabling software triggering. The software trigger configuration handler replaces the default configuration.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

The [DisplayImage](#) class is used to display the grabbed images.

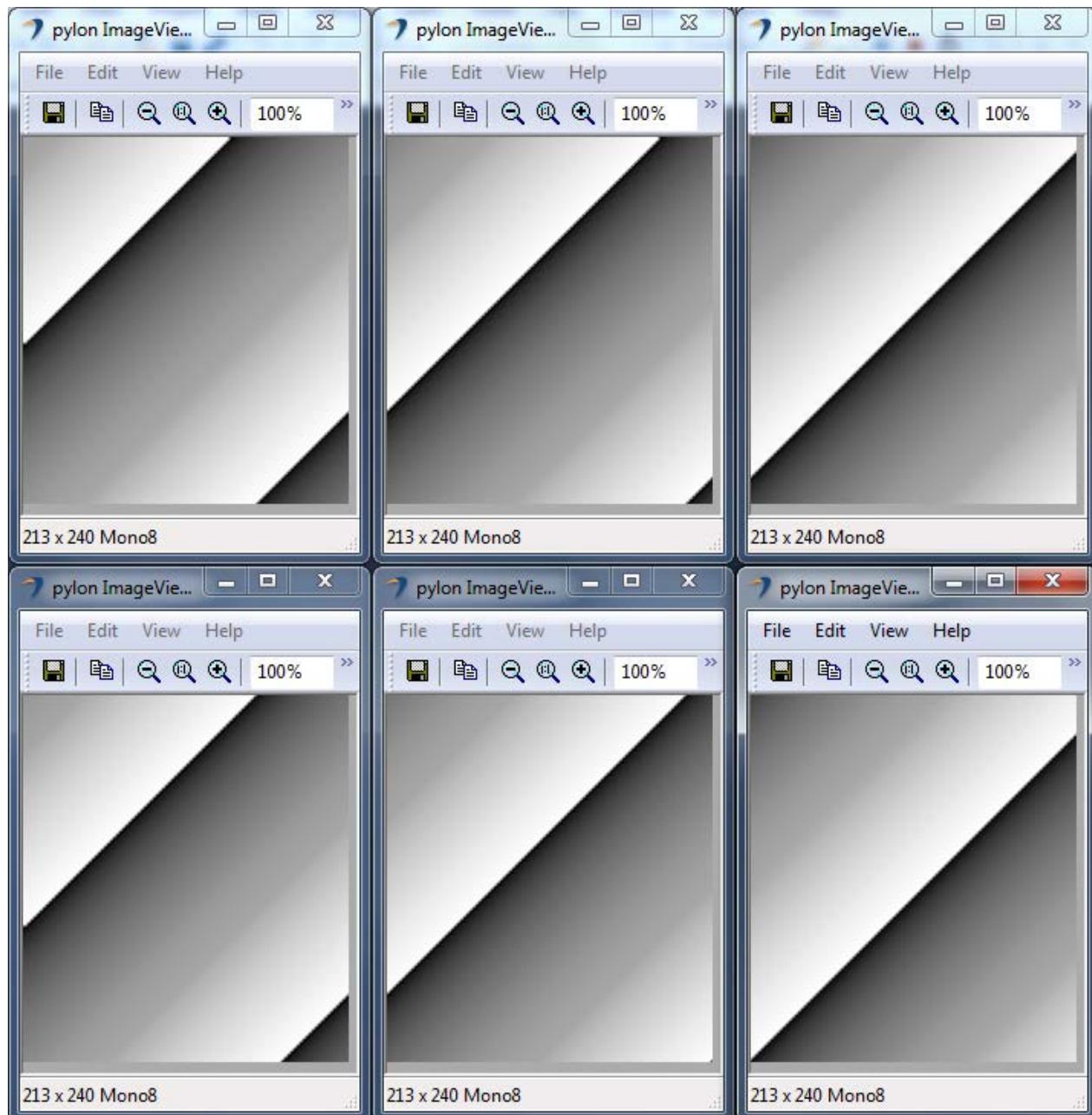
#### Applicable Interfaces

GigE Vision, USB3 Vision

[Back to top](#)

### 3.13 GUI\_ImageWindow

This sample demonstrates how to display images using the [CPylonImageWindow](#) class. Here, an image is grabbed and split into multiple tiles. Each tile is displayed in a separate image window.



#### Code

The [CPylonImageWindow](#) class is used to create an array of image windows for displaying camera image data.

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

[StartGrabbing\(\)](#) demonstrates how to start the grabbing by applying the [GrabStrategy\\_LatestImageOnly](#) grab strategy. Using this strategy is recommended when images have to be displayed.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

The [CPylonImage](#) class is used to split the grabbed image into tiles, which in turn will be displayed in different image windows.

## Applicable Interfaces

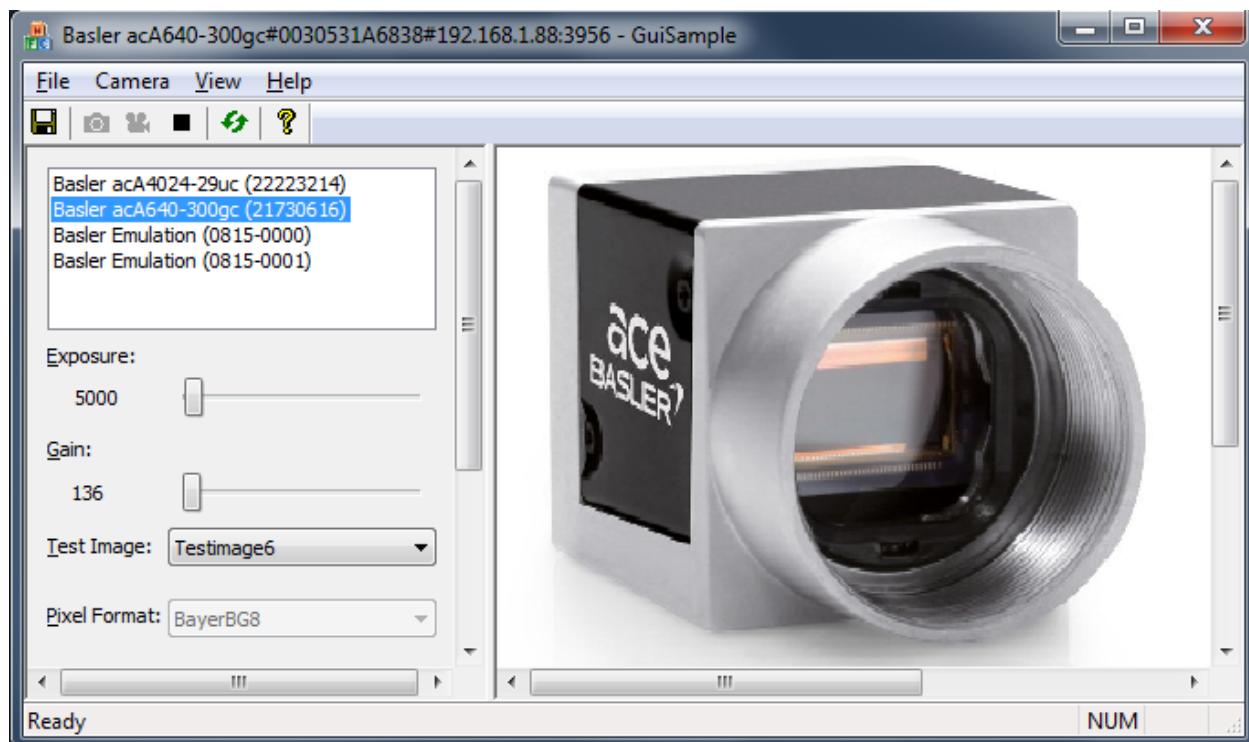
GigE Vision, USB3 Vision, CXP

[↖ Back to top](#)

### 3.14 GUI\_Sample

This sample demonstrates the use of a MFC GUI together with the pylon C++ API to enumerate attached cameras, to configure a camera, to start and stop the grab and to display and store grabbed images.

It also shows you how to use GUI controls to display and modify camera parameters.



#### Code

When the **Refresh** button is clicked, [CGuiSampleDoc::OnViewRefresh\(\)](#) is called, which in turn calls [CGuiSampleApp::EnumerateDevices\(\)](#) to enumerate all attached devices.

By selecting a camera in the device list, [CGuiSampleApp::OnOpenCamera\(\)](#) is called to open the selected camera. The **Single Shot** (Grab One) and **Start** (Grab Continuous) buttons as well as the **Exposure**, **Gain**, **Test Image** and **Pixel Format** parameters are initialized and enabled now.

By clicking on the **Single Shot** button, [CGuiSampleDoc::OnGrabOne\(\)](#) is called. To grab a single image, [StartGrabbing\(\)](#) is called with the following arguments:

```
m_camera.StartGrabbing(1, Pylon::GrabStrategy_OneByOne,
Pylon::GrabLoop_ProvidedByInstantCamera);
```

When the image is received, pylon will call the [CGuiSampleDoc::OnImageGrabbed\(\)](#) handler. To display the image, [CGuiSampleDoc::OnNewGrabresult\(\)](#) is called.

By clicking on the **Start** button, [CGuiSampleDoc::OnStartGrabbing\(\)](#) is called.

To grab images continuously, [StartGrabbing\(\)](#) is called with the following arguments:

```
m_camera.StartGrabbing(Pylon::GrabStrategy_OneByOne,
Pylon::GrabLoop_ProvidedByInstantCamera);
```

In this case, the camera will grab images until [StopGrabbing\(\)](#) is called.

When a new image is received, pylon will call the `CGuiSampleDoc::OnImageGrabbed()` handler. To display the image, `CGuiSampleDoc::OnNewGrabresult()` is called.

The **Stop** button gets enabled only after the **Start** button has been clicked. To stop continuous image acquisition, the **Stop** button has to be clicked. Upon clicking the **Stop** button, `CGuiSampleDoc::OnStopGrab()` is called.

When the **Save** button is clicked, `CGuiSampleDoc::OnFileImageSaveAs()` is called and a Bitmap (BMP) image will be saved (BMP is the default file format). Alternatively, the image can be saved in TIFF, PNG, JPEG, or Raw file formats.

## Applicable Interfaces

GigE Vision, USB3 Vision, CXP

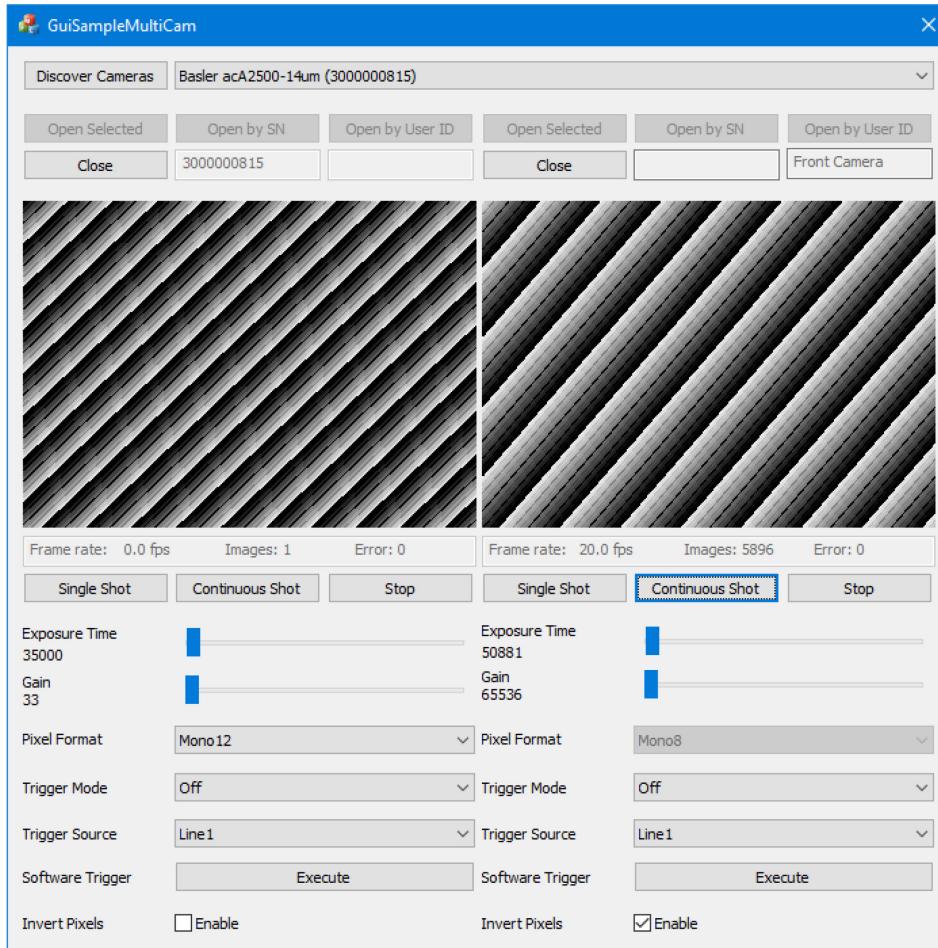
[⌘ Back to top](#)

### 3.15 GUI\_SampleMultiCam

This sample demonstrates how to operate multiple cameras using an MFC GUI together with the pylon C++ API.

The sample demonstrates different techniques for opening a camera, e.g., by using its serial number or user device ID. It also contains an image processing example and shows how to handle device disconnections.

The sample covers single and continuous image acquisition using software as well as hardware triggering.



#### Code

When the **Discover Cameras** button is clicked, the [CGuiSampleMultiCamDlg::OnBnClickedButtonScan\(\)](#) function is called, which in turn calls the [CGuiSampleMultiCamDlg::EnumerateDevices\(\)](#) function to enumerate all attached devices.

By clicking the **Open Selected** button, the [CGuiSampleMultiCamDlg::InternalOnBnClickedOpenSelected\(\)](#) function is called, which in turn calls the [CGuiSampleMultiCamDlg::InternalOpenCamera\(\)](#) function to create a new device info object.

Then, the [CGuiCamera::CGuiCamera\(\)](#) function is called to create a camera object and open the selected camera. In addition, callback functions for parameter changes are registered, e.g., for **Exposure Time**, **Gain**, **Pixel Format**, etc.

Cameras can be opened by clicking the **Open by SN** (SN = serial number) or **Open by User ID** button. The latter assumes that you have already assigned a user ID to the camera, e.g., in the pylon Viewer or via the pylon API.

After a camera has been opened, the following GUI elements become available:

- **Single Shot**, **Continuous Shot**, **Stop**, and **Execute** (for executing a software trigger) buttons
- **Exposure Time** and **Gain** sliders
- **PixelFormat**, **Trigger Mode**, and **Trigger Source** drop-down lists
- **Invert Pixels** check box

By clicking the **Single Shot** button, the `CGuiCamera::SingleGrab()` function is called. To grab a single image, the `StartGrabbing()` function is called with the following arguments:

```
m_camera.StartGrabbing(1, Pylon::GrabStrategy_OneByOne,  
Pylon::GrabLoop_ProvidedByInstantCamera);
```

When the image is received, pylon will call the `CGuiCamera::OnImageGrabbed()` handler. To display the image, the `CGuiSampleMultiCamDlg::OnNewGrabresult()` function is called.

By clicking the Continuous Shot button, the `CGuiCamera::ContinuousGrab()` function is called. To grab images continuously, the `StartGrabbing()` function is called with the following arguments:

```
m_camera.StartGrabbing(Pylon::GrabStrategy_OneByOne,  
Pylon::GrabLoop_ProvidedByInstantCamera);
```

In this case, the camera will grab images until `StopGrabbing()` is called.

When a new image is received, pylon will call the `CGuiCamera::OnImageGrabbed()` handler. To display the image, the `CGuiSampleMultiCamDlg::OnNewGrabresult()` function is called.

This sample also demonstrates the triggering of cameras by using a software trigger. For this purpose, the **Trigger Mode** parameter has to be set to **On**, and the **Trigger Source** parameter has to be set to **Software**. When starting a single or a continuous image acquisition, the camera will then be waiting for a software trigger.

By clicking the **Execute** button, the `CGuiCamera::ExecuteSoftwareTrigger()` function will be called, which will execute a software trigger.

For triggering the camera by hardware trigger, set **Trigger Mode** to **On** and **Trigger Source** to, e.g., **Line1**. When starting a single or a continuous image acquisition, the camera will then be waiting for a hardware trigger.

By selecting the **Invert Pixels** check box, an example of image processing will be shown. In the example, the pixel data will be inverted. This is done in the `CGuiCamera::OnNewGrabResult()` function.

Finally, this sample also shows the use of Device Removal callbacks. If an already opened camera is disconnected, the `CGuiCamera::OnCameraDeviceRemoved()` function is called. In turn, the `CGuiSampleMultiCamDlg::OnDeviceRemoved()` function will be called to inform the user about the disconnected camera.

## Applicable interfaces

GigE Vision, USB3 Vision, CoaXPress

[Back to top](#)

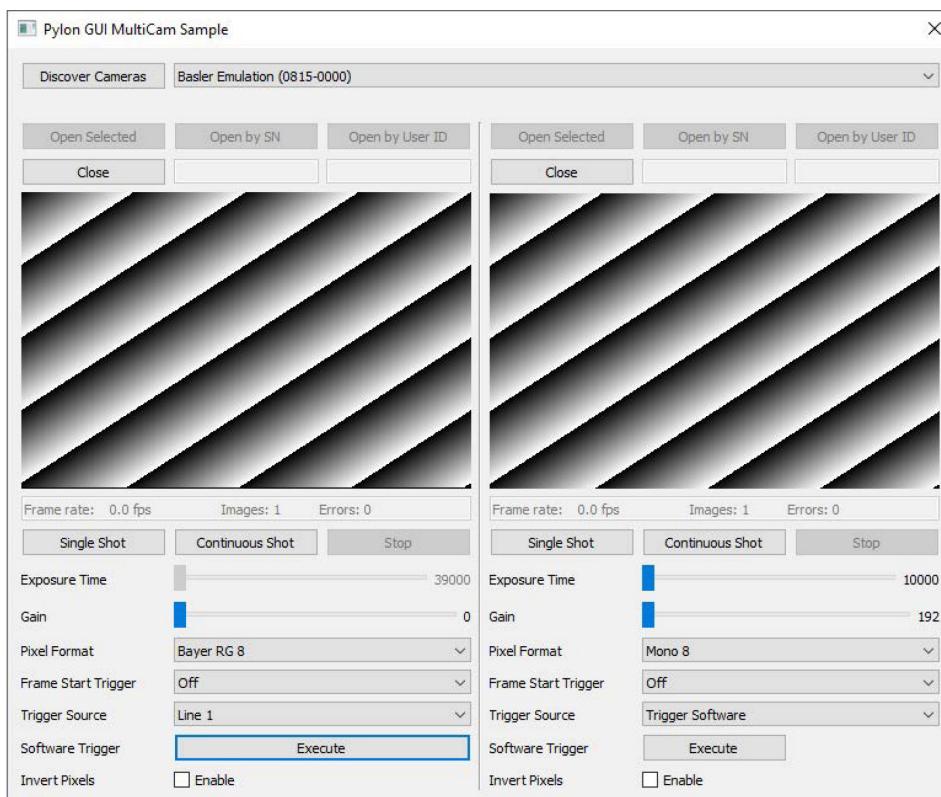
### 3.16 GUI\_QtMultiCam

This sample demonstrates how to operate multiple cameras using a Qt GUI together with the pylon C++ API.

Note: An installation of Qt Creator 5.12 or newer and a Microsoft Visual C++ Compiler is required.

The sample demonstrates different techniques for opening a camera, e.g., by using its serial number or user device ID. It also contains an image-processing example and shows how to handle device disconnections.

The sample covers single and continuous image acquisition using software as well as hardware triggering.



#### Code

When you click the **Discover Cameras** button, the `MainDialog::on_scanButton_clicked()` function is called, which in turn calls the `MainDialog::EnumerateDevices()` function to enumerate all attached devices.

By clicking the **Open Selected** button, the `MainDialog::on_openSelected_1_clicked()` or the `MainDialog::on_openSelected_2_clicked()` function is called, which in turn calls the `CGuiCamera::Open()` function to create a camera object and to open the selected camera. In addition, callback functions for parameter changes are registered, e.g., for **Exposure Time**, **Gain**, **Pixel Format**, etc.

Cameras can be opened by clicking the **Open by SN** (SN = serial number) or **Open by User ID** button. The latter assumes that you have already assigned a user ID to the camera, e.g., in the pylon Viewer or via the pylon API.

After a camera has been opened, the following GUI elements become available:

- **Single Shot**, **Continuous Shot**, **Stop**, and **Execute** (for executing a software trigger) buttons
- **Exposure Time** and **Gain** sliders
- **PixelFormat**, **Trigger Mode**, and **Trigger Source** drop-down lists
- **Invert Pixels** check box

By clicking the **Single Shot** button, the [CGuiCamera::SingleGrab\(\)](#) function is called. To grab a single image, the [StartGrabbing\(\)](#) function is called with the following arguments:

```
m_camera.StartGrabbing(1, Pylon::GrabStrategy_OneByOne,  
Pylon::GrabLoop_ProvidedByInstantCamera);
```

When the image is received, pylon calls the [CGuiCamera::OnImageGrabbed\(\)](#) handler.

By clicking the **Continuous Shot** button, the [CGuiCamera::ContinuousGrab\(\)](#) function is called. To grab images continuously, the [StartGrabbing\(\)](#) function is called with the following arguments:

```
m_camera.StartGrabbing(Pylon::GrabStrategy_OneByOne,  
Pylon::GrabLoop_ProvidedByInstantCamera);
```

In this case, the camera grabs images until [StopGrabbing\(\)](#) is called.

When a new image is received, pylon calls the [CGuiCamera::OnImageGrabbed\(\)](#) handler.

This sample also demonstrates the triggering of cameras by using a software trigger. For this purpose, the **Trigger Mode** parameter has to be set to **On**, and the **Trigger Source** parameter has to be set to **Software**. When starting a single or a continuous image acquisition, the camera then waits for a software trigger.

By clicking the **Execute** button, the [CGuiCamera::ExecuteSoftwareTrigger\(\)](#) function is called, which executes a software trigger.

For triggering the camera by hardware trigger, set **Trigger Mode** to **On** and **Trigger Source** to, e.g., **Line1**. When starting a single or a continuous image acquisition, the camera then waits for a hardware trigger.

By selecting the **Invert Pixels** check box, an example of image processing is shown. In the example, the pixel data is inverted. This is done in the [CGuiCamera::OnImageGrabbed\(\)](#) function.

Finally, this sample also shows the use of device removal callbacks. If an already opened camera is disconnected, the [CGuiCamera::OnCameraDeviceRemoved\(\)](#) function is called. In turn, the [MainDialog::OnDeviceRemoved\(\)](#) function is called to inform the user about the disconnected camera.

## Applicable interfaces

GigE Vision, USB3 Vision, CoaXPress

[Back to top](#)

### 3.17 ParametrizeCamera\_AutoFunctions

This sample demonstrates how to use the auto functions of Basler cameras, e.g., Gain Auto, Exposure Auto and Balance White Auto (color cameras only).

**Note:** Different camera families implement different versions of the Standard Feature Naming Convention (SFNC). That's why the name and the type of the parameters used can be different.

```
C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug_GigE\ParametrizeCamera_Aut...
Using device acA640-300gc
Trying 'GainAuto = Once'.
Initial Gain = 542
GainAuto went back to 'Off' after 3 frames.
Final Gain = 542

Press Enter to continue.

Trying 'GainAuto = Continuous'.
Initial Gain = 542
Final Gain = 542

Press Enter to continue.

Trying 'ExposureAuto = Once'.
Initial exposure time = 80 us
ExposureAuto went back to 'Off' after 4 frames.
Final exposure time = 500000 us

Press Enter to continue.
```

#### Code

The [CBaslerUniversalInstantCamera](#) class is used to create a camera object with the first found camera device independent of its interface.

The [CAcquireSingleFrameConfiguration](#) class is used to register the standard event handler for configuring single frame acquisition. This overrides the default configuration as all event handlers are removed by setting the registration mode to `RegistrationMode_ReplaceAll`. Note that the camera device auto functions do not require grabbing by single frame acquisition. All available acquisition modes can be used.

The [AutoGainOnce\(\)](#) and [AutoGainContinuous\(\)](#) functions control brightness by using the Once and the Continuous modes of the Gain Auto auto function.

The [AutoExposureOnce\(\)](#) and [AutoExposureContinuous\(\)](#) functions control brightness by using the Once and the Continuous modes of the Exposure Auto auto function.

The [CBaslerUniversalGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. The [DisplayImage](#) class is used to display the grabbed images.

#### Applicable Interfaces

GigE Vision, USB3 Vision

[↖ Back to top](#)

### 3.18 ParametrizeCamera\_Configurations

The Instant Camera class provides configuration event handlers to configure the camera and handle grab results. This is very useful for standard camera setups and image processing tasks.

This sample demonstrates how to use the existing configuration event handlers and how to register your own configuration event handlers.

Configuration event handlers are derived from the [CConfigurationEventHandler](#) base class. This class provides virtual methods that can be overridden. If the configuration event handler is registered, these methods are called when the state of the Instant Camera object changes, e.g., when the camera object is opened or closed.

The standard configuration event handler provides an implementation for the [OnOpened\(\)](#) method that parametrizes the camera.

To override Basler's implementation, create your own handler and attach it to [CConfigurationEventHandler](#).

Device-specific camera classes, e.g., for GigE cameras, provide specialized event handler base classes, e.g., [CBaslerGigEConfigurationEventHandler](#).

```
C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug\ParametrizeCamera_Configur...
Using device acA4024-29uc
Grab using continuous acquisition:

OnImageGrabbed event for device acA4024-29uc
SizeX: 4024
SizeY: 3036
Gray value of first pixel: 139

OnImageGrabbed event for device acA4024-29uc
SizeX: 4024
SizeY: 3036
Gray value of first pixel: 137

OnImageGrabbed event for device acA4024-29uc
SizeX: 4024
SizeY: 3036
Gray value of first pixel: 137

Grab using software trigger mode:

OnImageGrabbed event for device acA4024-29uc
SizeX: 4024
SizeY: 3036
Gray value of first pixel: 139
```

#### Code

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

The [CImageEventPrinter](#) class is used to output details about the grabbed images.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that receives the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

The [CAcquireContinuousConfiguration](#) class is the default configuration of the Instant Camera class. It is automatically registered when an Instant Camera object is created. This Instant Camera configuration is provided as header-only file. The code can be copied and modified to create your own configuration classes.

In this sample, the standard configuration event handler is registered for configuring the camera for continuous acquisition. By setting the registration mode to `RegistrationMode_ReplaceAll`, the new configuration handler replaces the default configuration handler that has been automatically registered when creating the Instant Camera object. The handler is automatically deleted when deregistered or when the registry is cleared if `Cleanup_Delete` is specified.

The [CSoftwareTriggerConfiguration](#) class is used to register the standard configuration event handler for enabling software triggering. This Instant Camera configuration is provided as header-only file. The code can be copied and modified to create your own configuration classes, e.g., to enable hardware triggering. The software trigger configuration handler replaces the default configuration.

The [CAcquireSingleFrameConfiguration](#) class is used to register the standard event handler for configuring single frame acquisition. This overrides the default configuration as all event handlers are removed by setting the registration mode to `RegistrationMode_ReplaceAll`.

The [CPixelFormatAndAoiConfiguration](#) class is used to register an additional configuration handler to set the image format and adjust the image ROI. This Instant Camera configuration is provided as header-only file. The code can be copied and modified to create your own configuration classes.

By setting the registration mode to `RegistrationMode_Append`, the configuration handler is added instead of replacing the configuration handler already registered.

## Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[↖ Back to top](#)

### 3.19 ParametrizeCamera\_GenericParameterAccess

This sample illustrates how to read and write different camera parameter types.

For camera configuration and for accessing other parameters, the pylon API uses the technologies defined by the GenICam standard (<http://www.genicam.org>). The standard also defines a format for camera description files.

These files describe the configuration interface of GenICam compliant cameras. The description files are written in XML and describe camera registers, their interdependencies, and all other information needed to access high-level features. This includes features such as Gain, Exposure Time, or Pixel Format. The features are accessed by means of low level register read and write operations.

The elements of a camera description file are represented as parameter objects. For example, a parameter object can represent a single camera register, a camera parameter such as Gain, or a set of parameter values. Each node implements the GenApi::INode interface.

The nodes are linked together by different relationships as explained in the GenICam standard document. The complete set of nodes is stored in a data structure called a node map. At runtime, the node map is instantiated from an XML description file.

This sample shows the generic approach for configuring a camera using the GenApi node maps represented by the GenApi::INodeMap interface. The names and types of the parameter nodes can be found in the Basler pylon *Programmer's Guide and API Reference Documentation*, in the Basler Product Documentation, in the camera's Register Structure and Access Methods documentation (if applicable), and by using the pylon Viewer tool.

See also the ParametrizeCamera\_NativeParameterAccess sample for the native approach for configuring a camera.

```
C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug\ParametrizeCamera_GenericP...
=====
Camera Device Information
=====
Vendor      : Basler
Model       : acA4024-29uc
Firmware version : 107405-01;U;acA4024_29u;V1.2-0;1

Camera Device Settings
=====
OffsetX    : 0
OffsetY    : 0
Width      : 200
Height     : 100
Old PixelFormat : Mono8
New PixelFormat : Mono8
Gain <50%>   : 13.5169 <Min: -0; Max: 27.0458>

Press Enter to exit.
```

#### Code

The `CInstantCamera` class is used to create an Instant Camera object with the first camera device found.

The [INodeMap](#) interface is used to access the feature node map of the camera device. It provides access to all features supported by the camera.

[CIntegerPtr](#) is a smart pointer for the [IInteger](#) interface pointer. It is used to access camera features of the [int64\\_t](#) type, e.g., image ROI (region of interest).

[CEnumerationPtr](#) is a smart pointer for the [IEnumerator](#) interface pointer. It is used to access camera features of the enumeration type, e.g., Pixel Format.

[CFloatPtr](#) is a smart pointer for the [IFloat](#) interface pointer. It is used to access camera features of the float type, e.g., Gain (only on camera devices compliant with SFNC version 2.0).

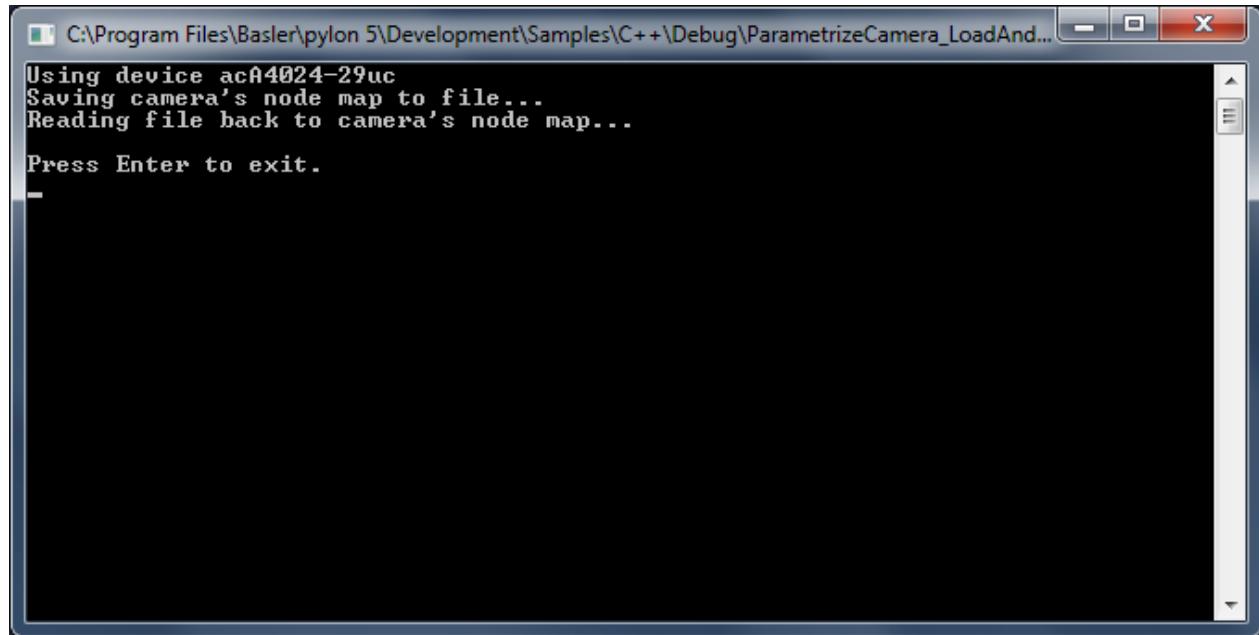
## Applicable Interfaces

GigE Vision, USB3 Vision, Camera Link, CXP

[↖ Back to top](#)

## 3.20 ParametrizeCamera\_LoadAndSave

This sample application demonstrates how to save or load the features of a camera to or from a file.



### Code

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

The [CFeaturePersistence](#) class is a pylon utility class for saving and restoring camera features to and from a file or string.

**Note:** When saving features, the behavior of cameras supporting sequencers depends on the current setting of the "SequenceEnable" (some GigE models) or "SequencerConfigurationMode" (USB only) features respectively. The sequence sets are only exported, if the sequencer is in configuration mode. Otherwise, the camera features are exported without sequence sets.

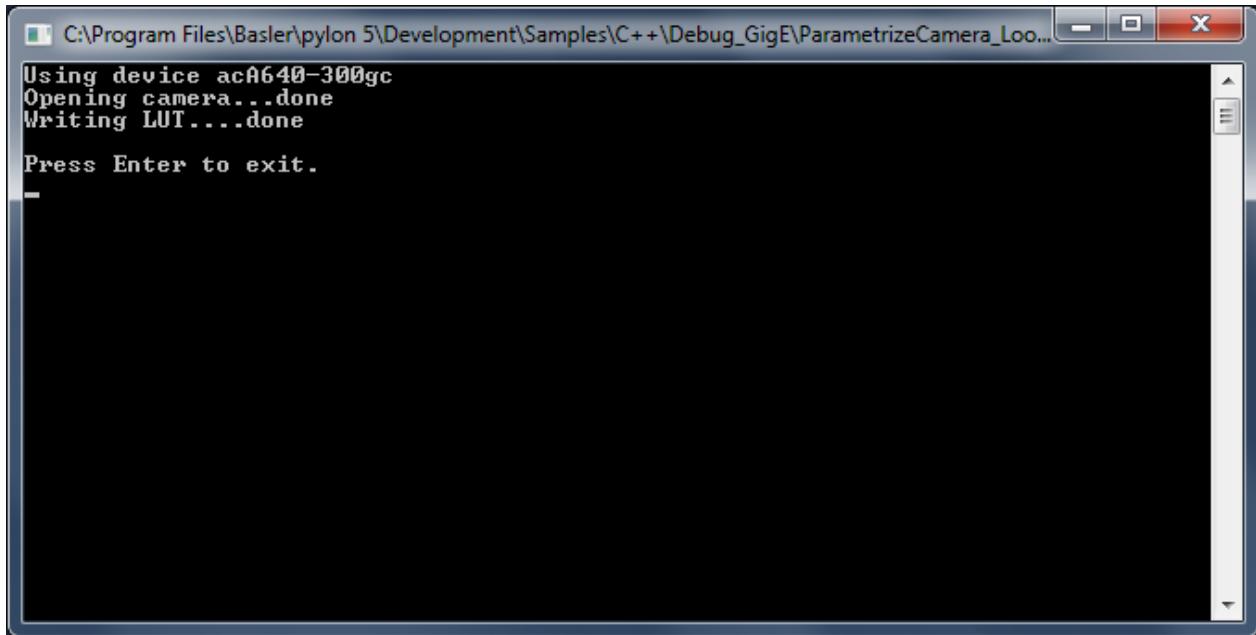
### Applicable Interfaces

GigE Vision, USB3 Vision, Camera Link

[Back to top](#)

### 3.21 ParametrizeCamera\_LookupTable

This sample demonstrates the use of the Luminance Lookup Table feature independent of the camera interface.



#### Code

The [CBaslerUniversalInstantCamera](#) class is used to create a camera object with the first found camera device independent of its interface.

The camera feature [LUTSelector](#) is used to select the lookup table. As some cameras have 10-bit and others have 12-bit lookup tables, the type of the lookup table for the current device must be determined first. The [LUTIndex](#) and [LUTValue](#) parameters are used to access the lookup table values. This sample demonstrates how the lookup table can be used to cause an inversion of the sensor values.

#### Applicable Interfaces

GigE Vision, USB3 Vision

[↖ Back to top](#)

## 3.22 ParametrizeCamera\_NativeParameterAccess

This sample shows the native approach for configuring a camera using device-specific Instant Camera classes.

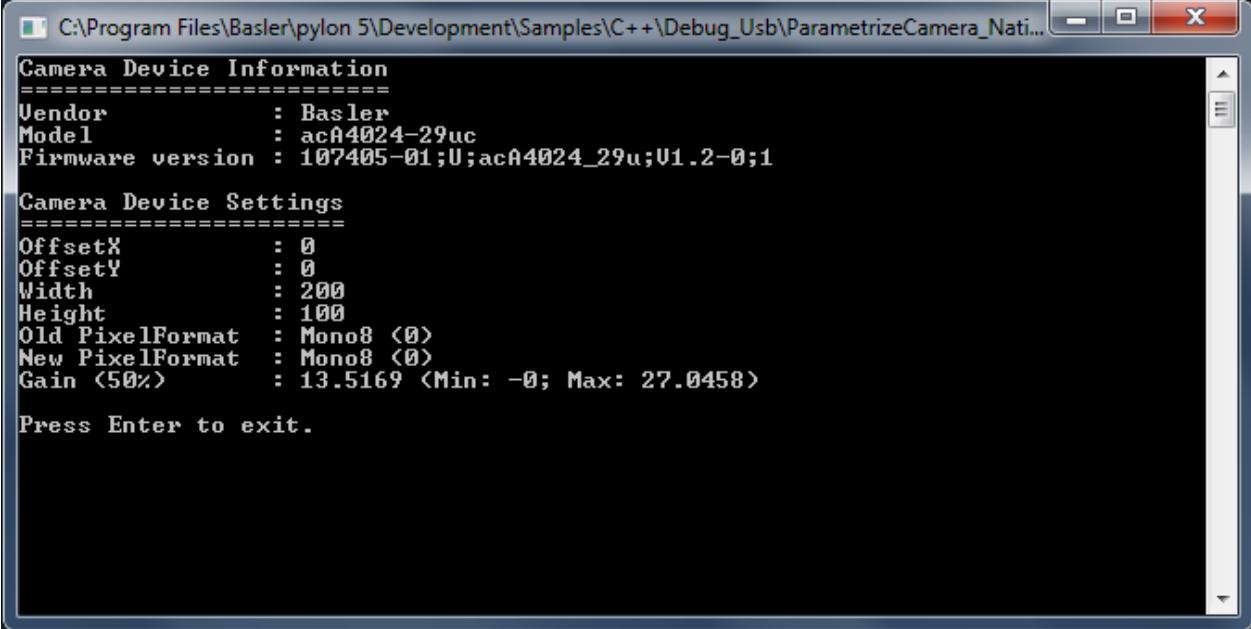
See also the `ParametrizeCamera_GenericParameterAccess` sample for the generic approach for configuring a camera.

For camera configuration and for accessing other parameters, the pylon API uses the technologies defined by the GenICam standard (<http://www.genicam.org>). The standard also defines a format for camera description files.

These files describe the configuration interface of GenICam compliant cameras. The description files are written in XML and describe camera registers, their interdependencies, and all other information needed to access high-level features. This includes features such as Gain, Exposure Time, or Pixel Format. The features are accessed by means of low level register read and write operations.

The elements of a camera description file are represented as parameter objects. For example, a parameter object can represent a single camera register, a camera parameter such as Gain, or a set of parameter values. Each node implements the `GenApi::INode` interface.

Using the code generators provided by GenICam's GenAPI module, a programming interface is created from a camera description file. This provides a function for each parameter that is available for the camera device. The programming interface is exported by the device-specific Instant Camera classes. **This is the easiest way to access parameters.**



A screenshot of a Windows command-line interface window titled "C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug\_Usb\ParametrizeCamera\_Nati...". The window displays the following text:

```
Camera Device Information
=====
Vendor      : Basler
Model       : acA4024-29uc
Firmware version : 107405-01;U;acA4024_29u;V1.2-0;1

Camera Device Settings
=====
OffsetX     : 0
OffsetY     : 0
Width       : 200
Height      : 100
Old PixelFormat : Mono8 <0>
New PixelFormat  : Mono8 <0>
Gain <50%>   : 13.5169 <Min: -0; Max: 27.0458>

Press Enter to exit.
```

### Code

The `CBaslerUniversalInstantCamera` class is used to create a camera object with the first found camera device independent of its interface.

This sample demonstrates the use of camera features of the `IInteger` type, e.g., `Width`, `Height`, `GainRaw` (available on camera devices compliant with SFNC versions before 2.0), of the `IEnumeration` type, e.g., `PixelFormat`, or of the `IFloat` type, e.g., `Gain` (available on camera devices compliant with SFNC version 2.0).

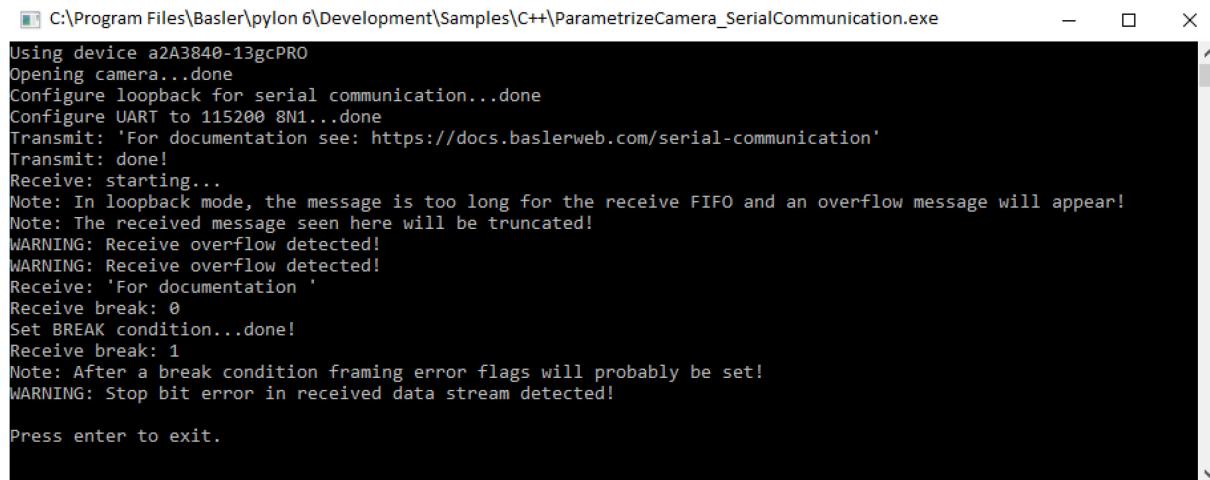
## Applicable Interfaces

GigE Vision, USB3 Vision, Camera Link, CXP

[↖ Back to top](#)

### 3.23 ParametrizeCamera\_SerialCommunication

This sample demonstrates the use of the Serial Communication feature (UART) supported by ace 2 Pro cameras. This feature allows you to establish serial communication between a host and an external device through the camera's I/O lines. For more information on the Serial Communication feature, refer to the Basler Product Documentation under [docs.baslerweb.com](https://docs.baslerweb.com).



```
C:\Program Files\Basler\pylon 6\Development\Samples\C++\ParametrizeCamera_SerialCommunication.exe
Using device a2A3840-13gcPRO
Opening camera...done
Configure loopback for serial communication...done
Configure UART to 115200 8N1...done
Transmit: 'For documentation see: https://docs.baslerweb.com/serial-communication'
Transmit: done!
Receive: starting...
Note: In loopback mode, the message is too long for the receive FIFO and an overflow message will appear!
Note: The received message seen here will be truncated!
WARNING: Receive overflow detected!
WARNING: Receive overflow detected!
Receive: 'For documentation '
Receive break: 0
Set BREAK condition...done!
Receive break: 1
Note: After a break condition framing error flags will probably be set!
WARNING: Stop bit error in received data stream detected!

Press enter to exit.
```

#### Code

The [CBaslerUniversalInstantCamera](#) class is used to create a camera object with the first camera found. Make sure to use an ace 2 Pro camera that supports the serial communication feature. Otherwise, an exception will be returned when trying to access and configure the camera's I/O lines.

To test the serial communication without having an external device connected to the camera, or to rule out errors caused by the external device, you can configure a loopback mode on the camera. This is done by setting the `BslSerialRxSource` parameter to `SerialTx`.

In this case, the serial input is connected to the serial output internally, so the camera receives exactly what it transmits.

To configure the serial communication between the camera and an external device, the GPIO Line 2 (SerialTx) and GPIO Line 3 (BslSerialRxSource) must be configured accordingly. Make sure not to use the opto-coupled I/O lines for UART communications.

In addition, depending on the configuration of the external device, the camera's baud rate (`BslSerialBaudRate`), the number of data bits (`BslSerialNumberOfDataBits`), the number of stop bits (`BslSerialNumberOfStopBits`), and the kind of parity check (`BslSerialParity`) must be configured.

After the serial communication has been configured, you can send data to the external device, via the `SerialTransmit()` function, and receive data from it, via the `SerialReceive()` function.

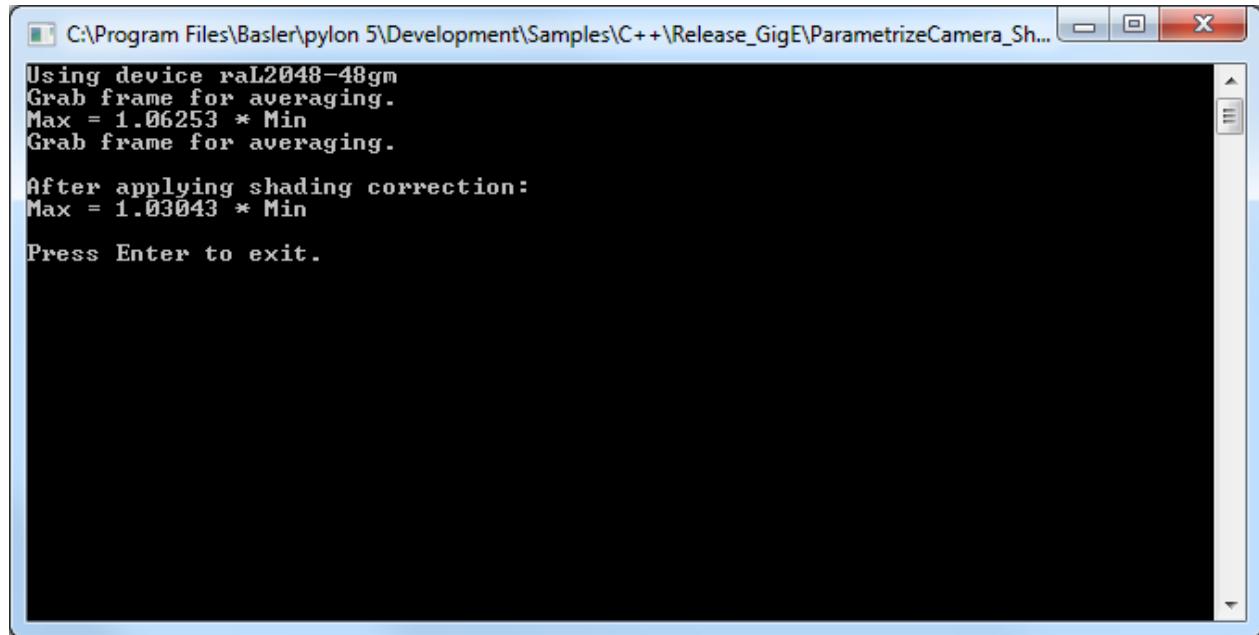
#### Applicable Interfaces

GigE Vision, USB3 Vision

[Back to top](#)

## 3.24 ParametrizeCamera\_Shading

This sample demonstrates how to calculate and upload gain shading sets to Basler racer and Basler runner line scan GigE Vision cameras.



### Code

The [CDeviceInfo](#) class is used to look for cameras with a specific interface, e.g., GigE Vision only ([BaslerGigEDeviceClass](#)).

The [CBaslerUniversalInstantCamera](#) class is used to create a camera object with the first found GigE camera.

The [CAcquireSingleFrameConfiguration](#) class is used to register the standard event handler for configuring single frame acquisition. This overrides the default configuration as all event handlers are removed by setting the registration mode to `RegistrationMode_ReplaceAll`.

[CreateShadingData\(\)](#) assumes that the conditions for exposure (illumination, exposure time, etc.) have been set up to deliver images of uniform intensity (gray value), but that the acquired images are not uniform. The gain shading data is calculated so that the observed non-uniformity will be compensated when the data is applied. The data is saved in a local file.

[UploadFile\(\)](#) transfers the calculated gain shading data from the local file to the camera.

[CheckShadingData\(\)](#) tests to what extent the non-uniformity has been compensated.

### Applicable Interfaces

GigE Vision

[Back to top](#)

### 3.25 ParametrizeCamera\_UserSets

This sample demonstrates how to use user configuration sets (user sets) and how to configure the camera to start up with the user-defined settings of user set 1.

You can also use the pylon Viewer to configure your camera and store custom settings in a user set of your choice.

**Note:** Different camera families implement different versions of the Standard Feature Naming Convention (SFNC). That's why the name and the type of the parameters used can be different.

ATTENTION: Executing this sample will overwrite all current settings in user set 1.

```
C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug_GigE\ParametrizeCamera_Us...
Using device acA640-300gc
Loading default settings
Turning off Gain Auto and Exposure Auto.
Saving currently active settings to user set 1.

Loading default settings.
Default settings
=====
Gain      : 136
Exposure time : 5000

Loading user set 1 settings.
User set 1 settings
=====
Gain      : 136
Exposure time : 80

Press Enter to exit.
```

#### Code

The [CBaslerUniversalInstantCamera](#) class is used to create a camera object with the first found camera device independent of its interface.

The camera parameters [UserSetSelector](#), [UserSetLoad](#), [UserSetSave](#), and [UserSetDefaultSelector](#) are used to demonstrate the use of user configuration sets (user sets) and how to configure the camera to start up with user-defined settings.

#### Applicable Interfaces

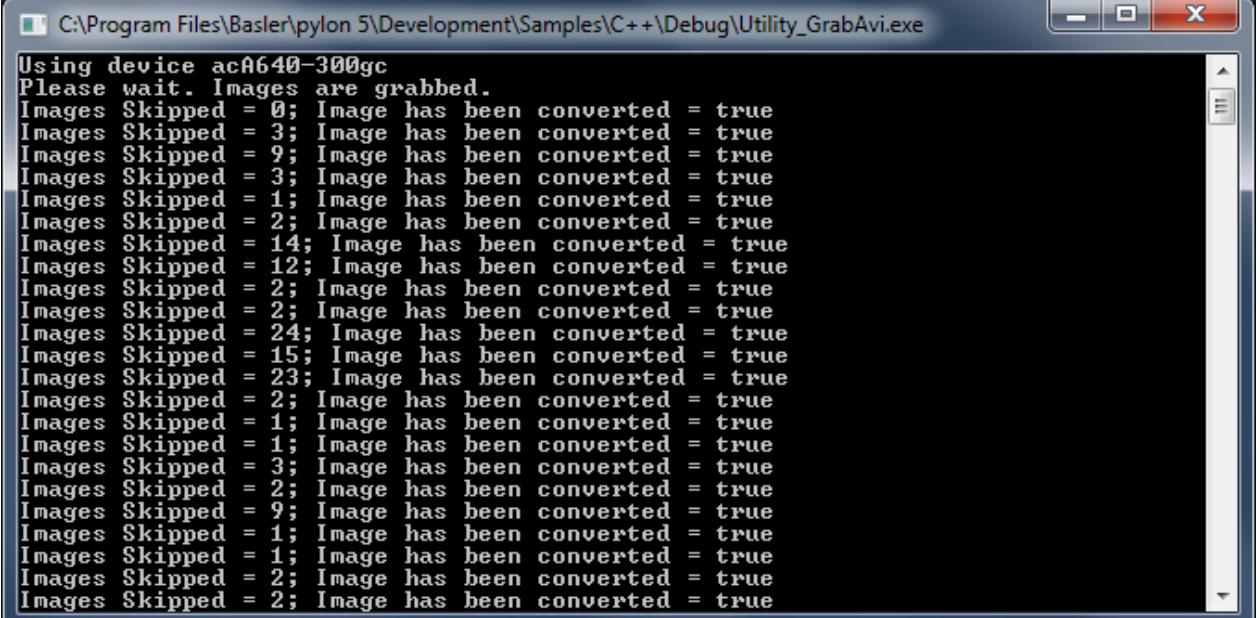
GigE Vision, USB3 Vision, Camera Link, CXP

[^ Back to top](#)

## 3.26 Utility\_GrabAvi

This sample demonstrates how to create a video file in Audio Video Interleave (AVI) format on Windows operating systems only.

**Note:** AVI is best for recording high-quality lossless videos because it allows you to record without compression. The disadvantage is that the file size is limited to 2 GB. Once that threshold is reached, the recording stops and an error message is displayed.



```
Using device acA640-300gc
Please wait. Images are grabbed.
Images Skipped = 0; Image has been converted = true
Images Skipped = 3; Image has been converted = true
Images Skipped = 9; Image has been converted = true
Images Skipped = 3; Image has been converted = true
Images Skipped = 1; Image has been converted = true
Images Skipped = 2; Image has been converted = true
Images Skipped = 14; Image has been converted = true
Images Skipped = 12; Image has been converted = true
Images Skipped = 2; Image has been converted = true
Images Skipped = 2; Image has been converted = true
Images Skipped = 24; Image has been converted = true
Images Skipped = 15; Image has been converted = true
Images Skipped = 23; Image has been converted = true
Images Skipped = 2; Image has been converted = true
Images Skipped = 1; Image has been converted = true
Images Skipped = 1; Image has been converted = true
Images Skipped = 3; Image has been converted = true
Images Skipped = 2; Image has been converted = true
Images Skipped = 9; Image has been converted = true
Images Skipped = 1; Image has been converted = true
Images Skipped = 1; Image has been converted = true
Images Skipped = 2; Image has been converted = true
Images Skipped = 2; Image has been converted = true
```

### Code

The [CAviWriter](#) class is used to create an AVI writer object. The writer object takes the following arguments: file name, playback frame rate, pixel output format, width and height of the image, vertical orientation of the image data, and compression options (optional).

[StartGrabbing\(\)](#) demonstrates how to start the grabbing by applying the [GrabStrategy\\_LatestImages](#) grab strategy. Using this strategy is recommended when images have to be recorded.

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

The [DisplayImage](#) class is used to display the grabbed images.

[Add\(\)](#) converts the grabbed image to the correct format, if required, and adds it to the AVI file.

### Applicable Interfaces

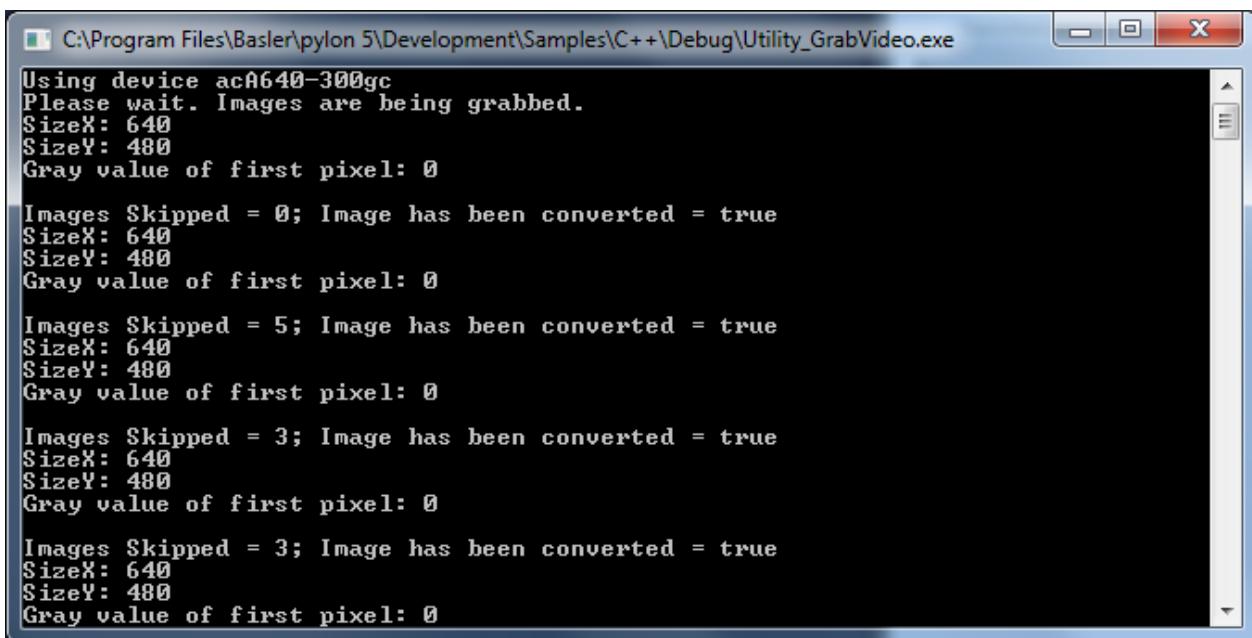
GigE Vision, USB3 Vision, CXP

[Back to top](#)

### 3.27 Utility\_GrabVideo

This sample demonstrates how to create a video file in MP4 format. It is presumed that the pylon Supplementary Package for MPEG-4 is already installed.

**Note:** There are no file size restrictions when recording MP4 videos. However, the MP4 format always compresses data to a certain extent, which results in loss of detail.



```
C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug\Utility_GrabVideo.exe
Using device acA640-300gc
Please wait. Images are being grabbed.
SizeX: 640
SizeY: 480
Gray value of first pixel: 0

Images Skipped = 0; Image has been converted = true
SizeX: 640
SizeY: 480
Gray value of first pixel: 0

Images Skipped = 5; Image has been converted = true
SizeX: 640
SizeY: 480
Gray value of first pixel: 0

Images Skipped = 3; Image has been converted = true
SizeX: 640
SizeY: 480
Gray value of first pixel: 0

Images Skipped = 3; Image has been converted = true
SizeX: 640
SizeY: 480
Gray value of first pixel: 0
```

#### Code

The [CVideoWriter](#) class is used to create a video writer object. Before opening the video writer object, it is initialized with the current parameter values of the ROI width and height, the pixel output format, the playback frame rate, and the quality of compression.

[StartGrabbing\(\)](#) demonstrates how to start the grabbing by applying the [GrabStrategy\\_LatestImages](#) grab strategy. Using this strategy is recommended when images have to be recorded.

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

The [DisplayImage](#) class is used to display the grabbed images.

[Add\(\)](#) converts the grabbed image to the correct format, if required, and adds it to the video file.

#### Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[Back to top](#)

### 3.28 Utility\_Image

This sample demonstrates how to use the pylon image classes [CPylonImage](#) and [CPylonBitmapImage](#).

[CPylonImage](#) supports handling image buffers of the various existing pixel types.

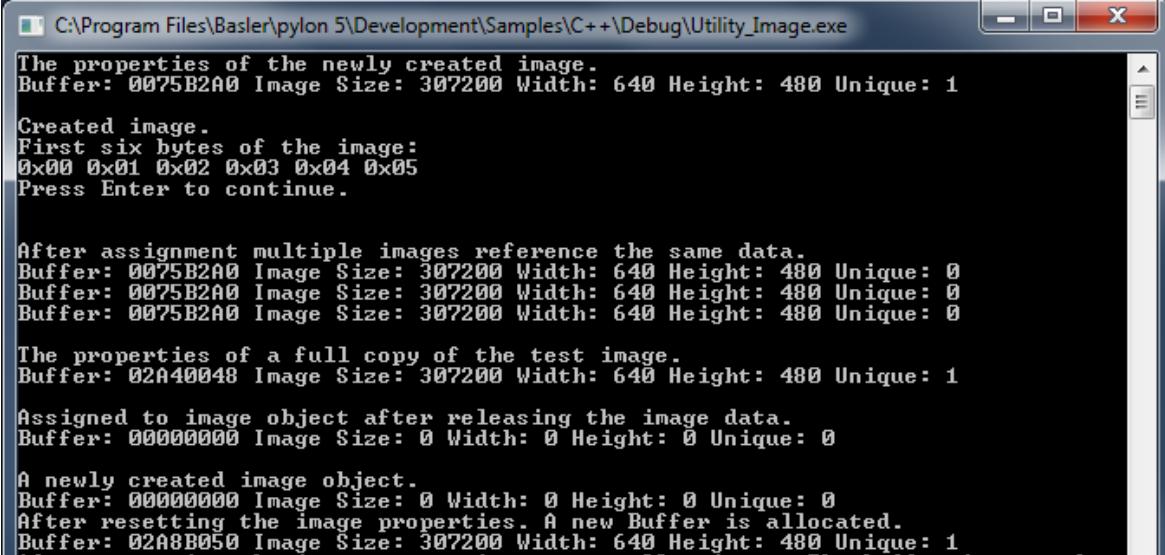
[CPylonBitmapImage](#) can be used to easily create Windows bitmaps for displaying images. In addition, there are two image class-related interfaces in pylon ([IImage](#) and [IReusableImage](#)).

[IImage](#) can be used to access image properties and the image buffer.

The [IReusableImage](#) interface extends the [IImage](#) interface to be able to reuse the resources of the image to represent a different image.

Both [CPylonImage](#) and [CPylonBitmapImage](#) implement the [IReusableImage](#) interface.

The [CGrabResultPtr](#) grab result class provides a cast operator to the [IImage](#) interface. This makes using the grab result together with the image classes easier.



```
C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug\Utility_Image.exe
The properties of the newly created image.
Buffer: 0075B2A0 Image Size: 307200 Width: 640 Height: 480 Unique: 1

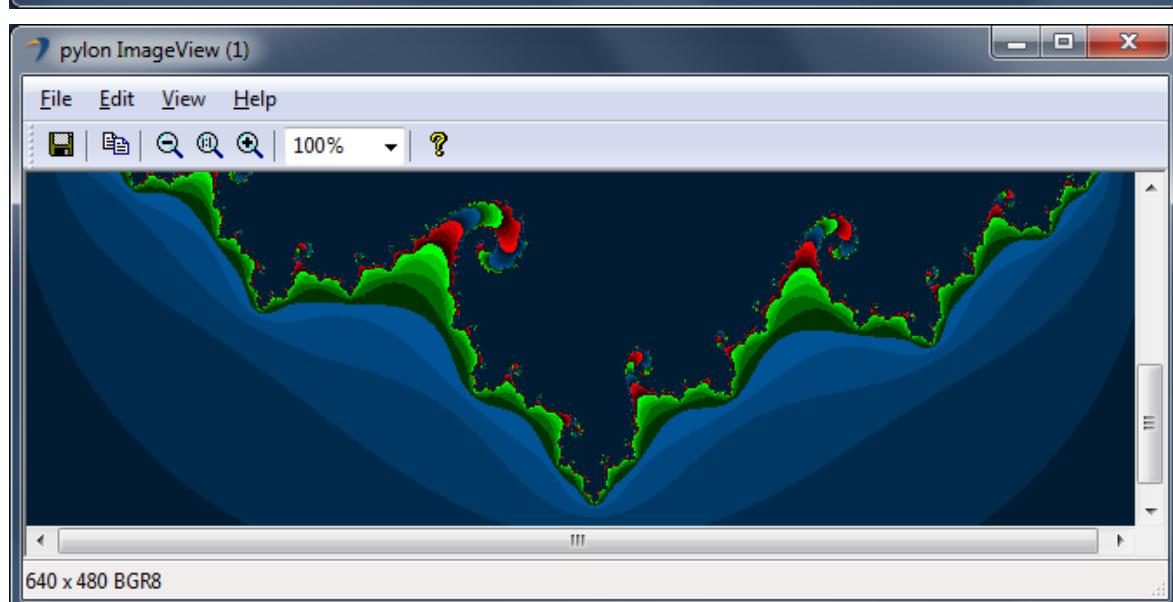
Created image.
First six bytes of the image:
0x00 0x01 0x02 0x03 0x04 0x05
Press Enter to continue.

After assignment multiple images reference the same data.
Buffer: 0075B2A0 Image Size: 307200 Width: 640 Height: 480 Unique: 0
Buffer: 0075B2A0 Image Size: 307200 Width: 640 Height: 480 Unique: 0
Buffer: 0075B2A0 Image Size: 307200 Width: 640 Height: 480 Unique: 0

The properties of a full copy of the test image.
Buffer: 02A40048 Image Size: 307200 Width: 640 Height: 480 Unique: 1

Assigned to image object after releasing the image data.
Buffer: 00000000 Image Size: 0 Width: 0 Height: 0 Unique: 0

A newly created image object.
Buffer: 00000000 Image Size: 0 Width: 0 Height: 0 Unique: 0
After resetting the image properties. A new Buffer is allocated.
Buffer: 02A8B050 Image Size: 307200 Width: 640 Height: 480 Unique: 1
After resetting the image properties to a smaller image. The buffer is reused.
```



## Code

The [CPylonImage](#) class describes an image. It takes care of the following:

- Automatically manages size and lifetime of the image.
- Allows taking over a grab result to prevent its reuse as long as required.
- Allows connecting user buffers or buffers provided by third-party software packages.
- Provides methods for loading and saving an image in different file formats.
- Serves as the main target format for the [CImageFormatConverter](#) class.
- Makes working with planar images easier.
- Makes extracting AOIs easier, e.g., for thumbnail images of defects.

The [CPylonBitmapImage](#) class can be used to easily create Windows bitmaps for displaying images. It takes care of the following:

- Automatically handles the bitmap creation and lifetime.
- Provides methods for loading and saving an image in different file formats.
- Serves as target format for the [CImageFormatConverter](#) class.

The bitmap image class provides a cast operator for [HBitmap](#). The cast operator can be used for instance to provide the handle to Windows API functions.

The [CImageFormatConverter](#) class creates new images by converting a source image to another format.

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

The [DisplayImage](#) class is used to display the grabbed images.

## Applicable Interfaces

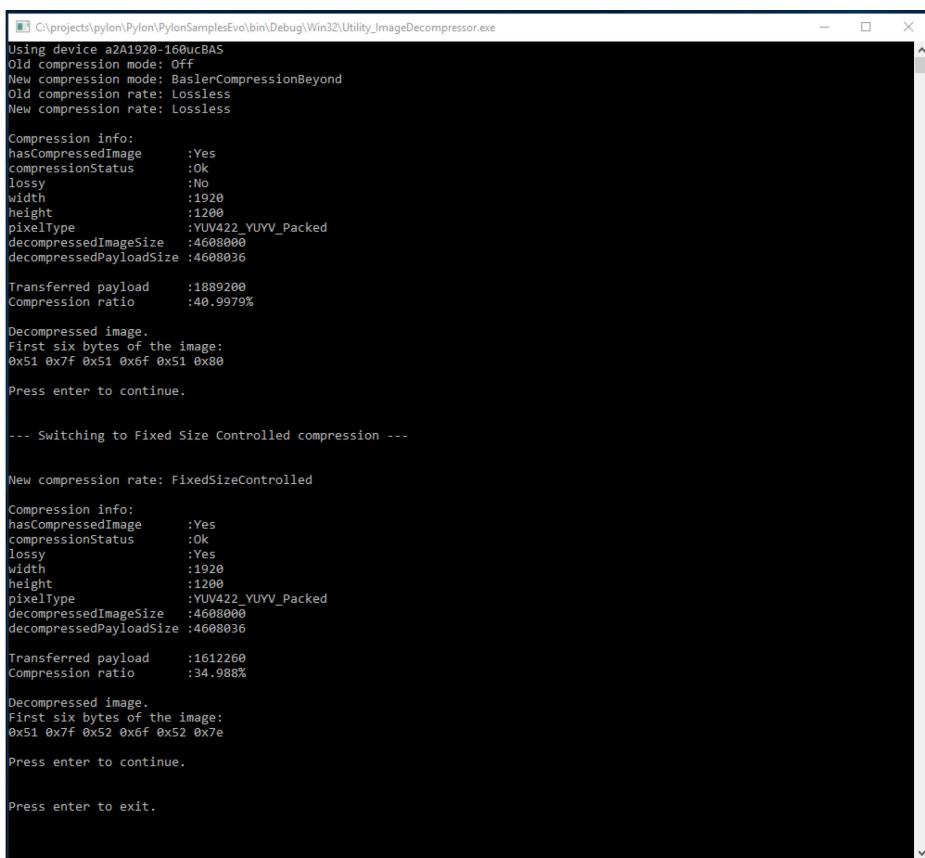
GigE Vision, USB3 Vision, CXP

[↖ Back to top](#)

### 3.29 Utility\_ImageDecompressor

This sample illustrates how to enable and use the Basler Compression Beyond feature in Basler ace 2 GigE and Basler ace 2 USB 3.0 cameras.

This sample also demonstrates how to decompress the images using the [CImageDecompressor](#) class.



The screenshot shows a Windows command-line interface window titled 'C:\projects\pylon\Pylon\PylonSamplesEvo\bin\Debug\Win32\Utility\_ImageDecompressor.exe'. The window displays the following text:

```
Using device a2A1920-160ucBAS
Old compression mode: Off
New compression mode: BaslerCompressionBeyond
Old compression rate: Lossless
New compression rate: Lossless

Compression info:
hasCompressedImage      :Yes
compressionStatus        :Ok
lossy                   :No
width                   :1920
height                  :1200
pixelType               :YUV422_YUYV_Packed
decompressedImageSize   :4668000
decompressedPayloadSize :4668036

Transferred payload     :1889200
Compression ratio       :40.9979%

Decompressed image.
First six bytes of the image:
0x51 0x7f 0x51 0x6f 0x51 0x80

Press enter to continue.

--- Switching to Fixed Size Controlled compression ---

New compression rate: FixedSizeControlled

Compression info:
hasCompressedImage      :Yes
compressionStatus        :Ok
lossy                   :Yes
width                   :1920
height                  :1200
pixelType               :YUV422_YUYV_Packed
decompressedImageSize   :4668000
decompressedPayloadSize :4668036

Transferred payload     :1612260
Compression ratio       :34.988%

Decompressed image.
First six bytes of the image:
0x51 0x7f 0x52 0x6f 0x52 0x7e

Press enter to continue.

Press enter to exit.
```

#### Code

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

The [CImageDecompressor](#) class is used to decompress grabbed images. In this sample, compression and decompression are demonstrated, using lossless and lossy algorithms.

The [CPylonImage](#) class is used to create a decompressed target image. The target image is displayed in an image window.

#### Applicable Interfaces

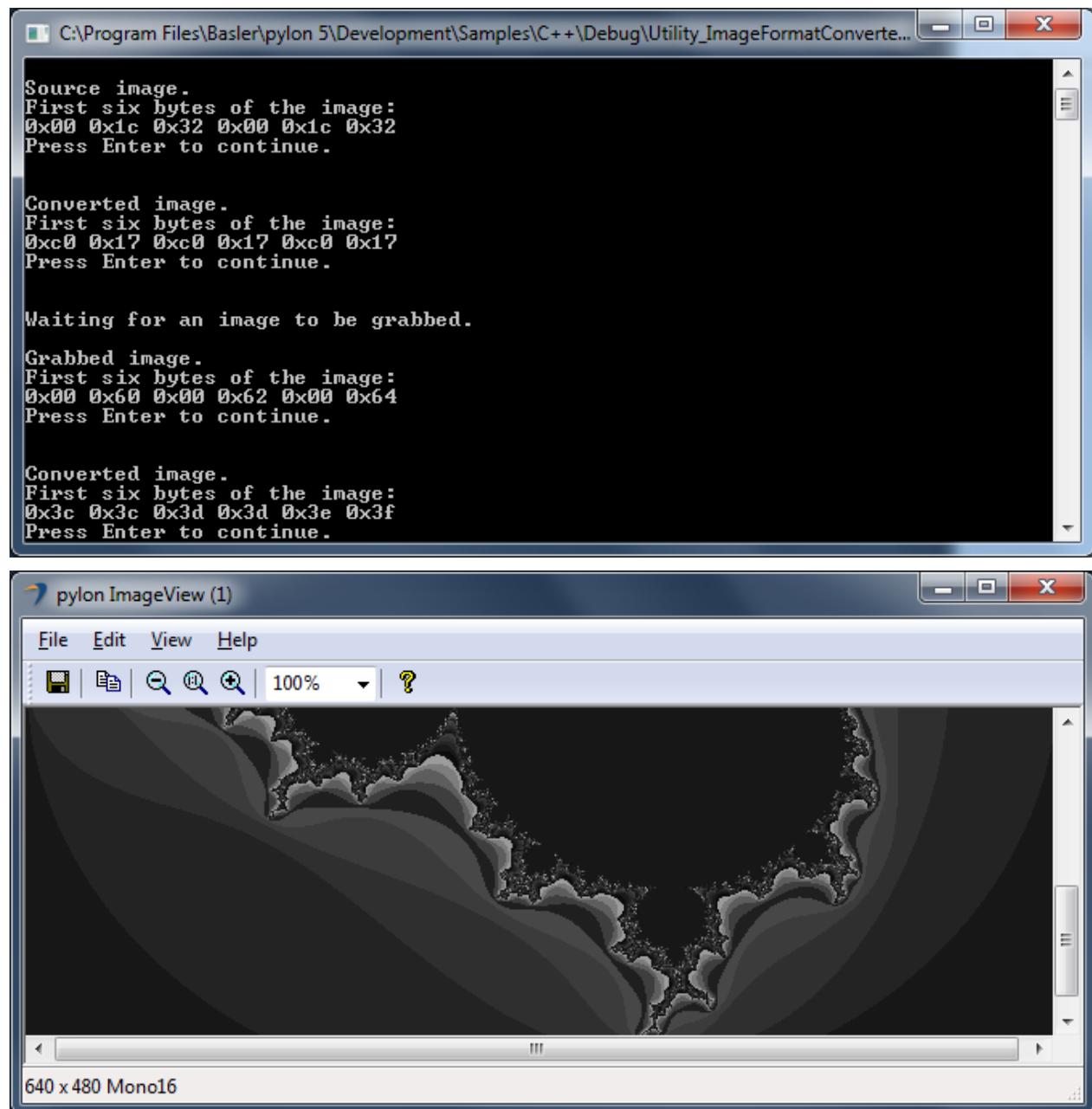
GigE Vision, USB3 Vision

[Back to top](#)

### 3.30 Utility\_ImageFormatConverter

This sample demonstrates how to use the [CImageFormatConverter](#) class. The image format converter accepts all image formats produced by Basler camera devices. It can convert these to a number of output formats.

The conversion can be controlled by several parameters. For more information, see the converter class documentation.



#### Code

The [CImageFormatConverter](#) class creates new images by converting a source image to another format.

The [CPylonImage](#) class describes an image. It takes care of the following:

- Automatically manages size and lifetime of the image.
- Allows taking over a grab result to prevent its reuse as long as required.
- Allows connecting user buffers or buffers provided by third-party software packages.
- Provides methods for loading and saving an image in different file formats.
- Serves as the main target format for the [CImageFormatConverter](#) class.
- Makes working with planar images easier.
- Makes extracting image ROIs easier, e.g., for thumbnail images of defects.

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

The [DisplayImage](#) class is used to display the grabbed images.

## Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[↖ Back to top](#)

### 3.31 Utility\_ImageLoadAndSave

This sample demonstrates how to load and save images.

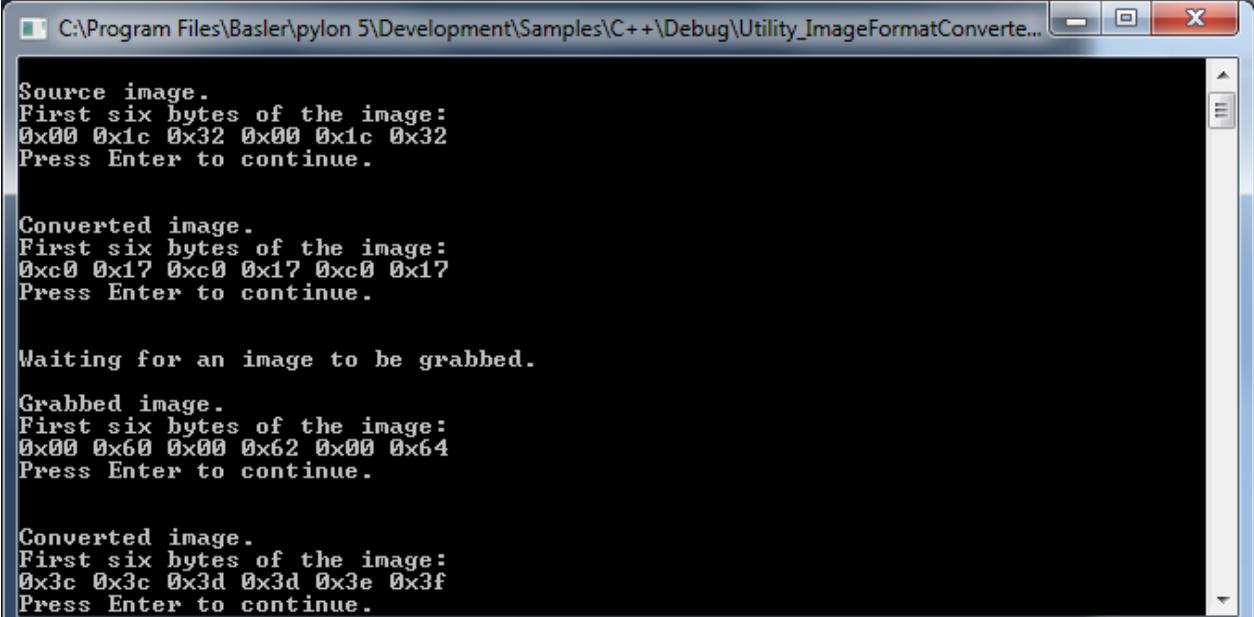
The [CImagePersistence](#) class provides functions for loading and saving images. It uses the image class-related pylon interfaces [IImage](#) and [IReusableImage](#).

[IImage](#) can be used to access image properties and the image buffer. Therefore, it is used when saving images. In addition to that, images can also be saved by passing an image buffer and the corresponding properties.

The [IReusableImage](#) interface extends the [IImage](#) interface to be able to reuse the resources of the image to represent a different image. The [IReusableImage](#) interface is used when loading images.

The [CPylonImage](#) and [CPylonBitmapImage](#) image classes implement the [IReusableImage](#) interface. These classes can therefore be used as targets for loading images.

The grab result smart pointer classes provide a cast operator to the [IImage](#) interface. This makes it possible to pass a grab result directly to the function that saves images to disk.



The screenshot shows a terminal window with the title bar "C:\Program Files\Basler\pylon 5\Development\Samples\C++\Debug\Utility\_ImageFormatConverte...". The window contains the following text:

```
Source image.
First six bytes of the image:
0x00 0x1c 0x32 0x00 0x1c 0x32
Press Enter to continue.

Converted image.
First six bytes of the image:
0xc0 0x17 0xc0 0x17 0xc0 0x17
Press Enter to continue.

Waiting for an image to be grabbed.

Grabbed image.
First six bytes of the image:
0x00 0x60 0x00 0x62 0x00 0x64
Press Enter to continue.

Converted image.
First six bytes of the image:
0x3c 0x3c 0x3d 0x3d 0x3e 0x3f
Press Enter to continue.
```

#### Code

The [CImagePersistence](#) class demonstrates how images can be loaded or saved. It can be used to check whether the image can be saved without prior conversion. Supported image file formats are TIFF, BMP, JPEG, and PNG.

The [CInstantCamera](#) class is used to create an Instant Camera object with the first camera device found.

The [CGrabResultPtr](#) class is used to initialize a smart pointer that will receive the grab result data. It controls the reuse and lifetime of the referenced grab result. When all smart pointers referencing a grab result go out of scope, the referenced grab result is reused or destroyed. The grab result is still valid after the camera object it originated from has been destroyed.

The [CPylonImage](#) class describes an image. It takes care of the following:

- Automatically manages size and lifetime of the image.
- Allows taking over a grab result to prevent its reuse as long as required.
- Allows connecting user buffers or buffers provided by third-party software packages.
- Provides methods for loading and saving an image in different file formats.
- Serves as the main target format for the [CImageFormatConverter](#) class.
- Makes working with planar images easier.
- Makes extracting AOIs easier, e.g., for thumbnail images of defects.

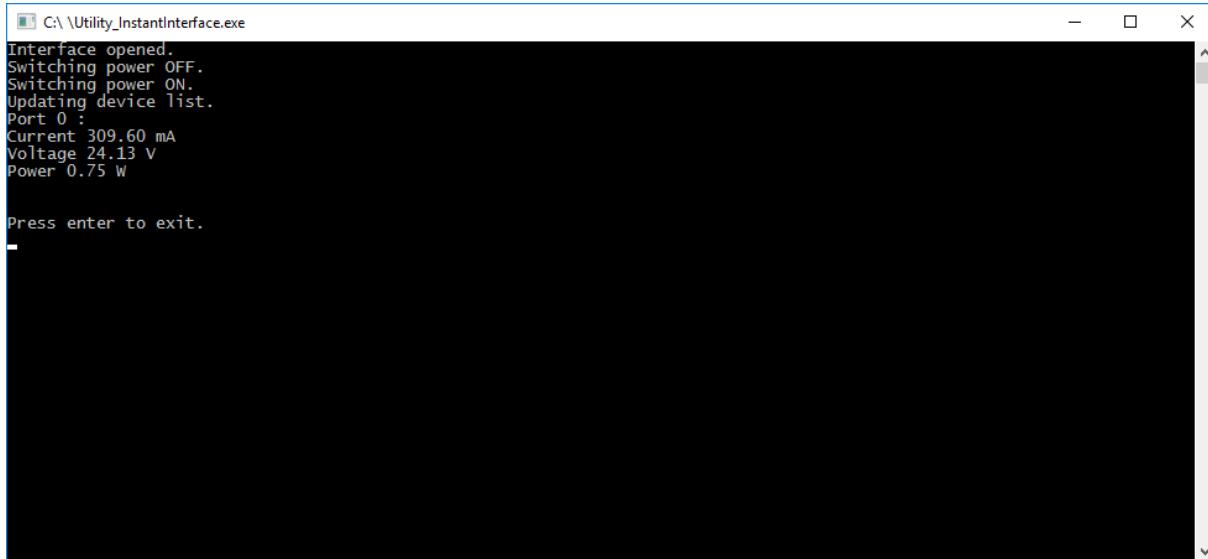
### Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[↖ Back to top](#)

### 3.32 Utility\_InstantInterface

This sample illustrates how to use the [CInstantInterface](#) class to access parameters of the interface using the Basler CXP-12 interface card. The sample shows how to access the Power-over-CoaXPress settings and monitor the power usage.



#### Code

The [CInterfaceInfo](#) class is used for storing information about an interface object provided by a specific transport layer, e.g., BaslerGenTICxpDeviceClass.

The [CUniversalInstantInterface](#) class is used to open the first interface on the CoaXPress interface card and access its parameters. In this sample, the Power-over-CoaXPress parameter CxpPoCxpStatus is enabled/disabled. In addition, the current, voltage, and power consumption information is displayed.

#### Applicable Interfaces

CXP

[Back to top](#)

### 3.33 Utility\_IpConfig

This sample demonstrates how to configure the IP address of a GigE Vision camera. The functionalities described in this sample are similar to those used in the pylon IP Configurator.

In addition, this sample can be used to automatically and programmatically configure multiple GigE Vision cameras. As the sample accepts command line arguments, it can be directly executed, e.g., from a batch script file.

The screenshot shows a Windows command-line application window titled 'C:\Utility\_IpConfig.exe'. The window contains the following text:

```
Usage: Utility_IpConfig <MAC> <IP> [<MASK>] [<GATEWAY>]
<MAC> is the MAC address without separators, e.g., 0030531596CF
<IP> is one of the following:
  - AUTO to use Auto-IP (LLA).
  - DHCP to use DHCP.
  - Everything else is interpreted as a new IP address in dotted notation, e.g., 192.168.1.1.
[<MASK>] is the network mask in dotted notation. This is optional. 255.255.255.0 is used as default.
[<GATEWAY>] is the gateway address in dotted notation. This is optional. 0.0.0.0 is used as default.
Please note that this is a sample and no sanity checks are made.

Available Devices
Friendly Name          MAC      IP Address     Subnet Mask    Gateway      Mode     IP?   DHCP?  supports
Basler acA1600-60gm (21414863) 0030531596CF 169.254.200.200 255.255.255.0 0.0.0.0  StaticIP True  True   LLA?
Basler acA640-120gc (21767329) 0030531AF7A1 169.254.162.247 255.255.0.0 0.0.0.0  DHCP   True  True   True
Basler acA1920-50gc (21588208) 003053183BF0 169.254.241.59  255.255.0.0 0.0.0.0  AutoIP True  True   True

Press enter to exit.
```

#### Code

The [CTIFactory](#) class is used to create a GigE transport layer. The GigE transport layer is required to discover all GigE Vision cameras independent of their current IP address configuration. For that purpose, the [EnumerateAllDevices\(\)](#) function is used.

To set a new IP address of a GigE Vision camera, the [BroadcastIpConfiguration\(\)](#) function is used.

#### Applicable Interfaces

GigE Vision

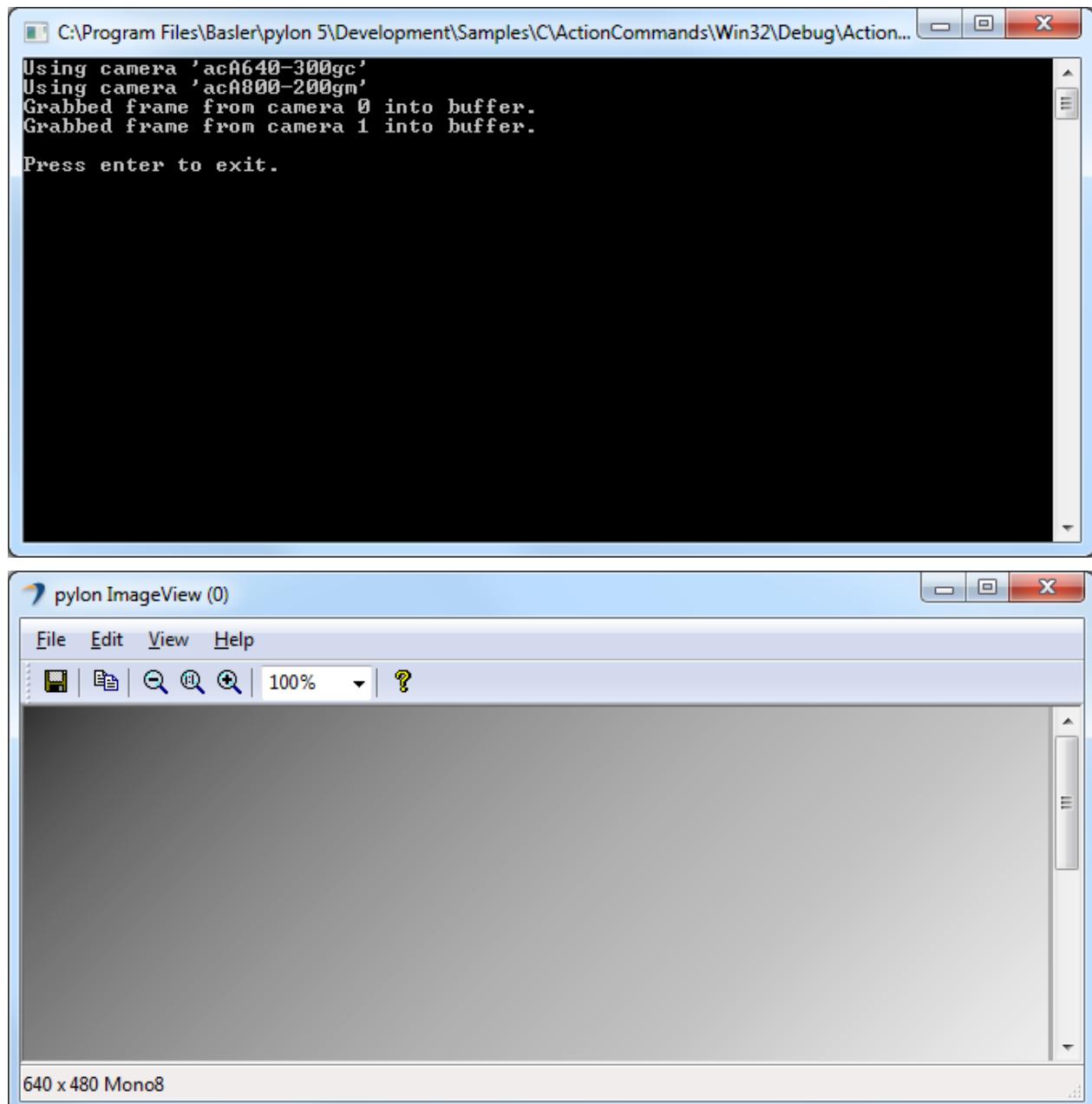
[Back to top](#)

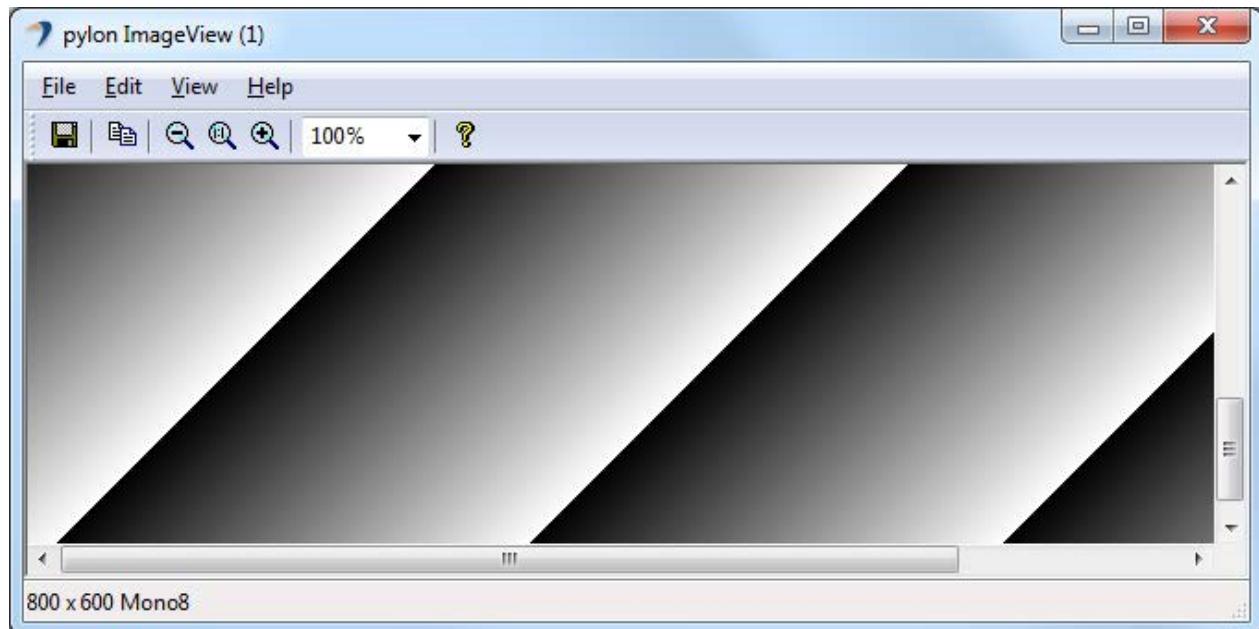
## 4 C Samples

### 4.1 ActionCommands

This sample illustrates how to grab images and trigger multiple cameras using a GigE Vision action command.

At least two connected GigE cameras are required for this sample.





## Code

Before using any pylon methods, the pylon runtime is initialized by calling [PylonInitialize\(\)](#).

Then, [PylonEnumerateDevices\(\)](#) is called to enumerate all attached camera devices.

Before using a camera device, it must be opened by calling [PylonDeviceOpen\(\)](#). This allows us to set parameters and grab images.

This sample works only for cameras supporting GigE Vision action commands. This is checked by calling [PylonDeviceFeatureIsAvailable\(\)](#) and passing the device handle and the camera parameter "ActionControl" as arguments. Cameras with action command support are then configured accordingly, i.e., the parameters ActionSelector, ActionDeviceKey, ActionGroupKey, ActionGroupMask, TriggerSelector, TriggerMode, and TriggerSource are set.

If the cameras are connected to a switch, Basler recommends setting the Inter-Packet Delay (GevSCPD) and the Frame Transmission Delay (GevSCFTD) so that the switch can properly line up packets.

Images are grabbed using a stream grabber. For each camera device, a stream grabber is created by calling [PylonDeviceGetStreamGrabber\(\)](#) and passing the device handle and the stream grabber handle as arguments. A handle for the stream grabber's wait object is retrieved within [PylonStreamGrabberGetWaitObject\(\)](#). The wait object allows waiting for buffers to be filled with grabbed data.

We must also tell the stream grabber the number and size of the buffers we are using. This is done with [PylonStreamGrabberSetMaxNumBuffer\(\)](#) and [PylonStreamGrabberSetMaxBufferSize\(\)](#). By calling [PylonStreamGrabberPrepareGrab\(\)](#), we allocate the resources required for grabbing. After this, critical parameters that impact the payload size must not be changed until [PylonStreamGrabberFinishGrab\(\)](#) is called.

Before using the buffers for grabbing, they must be registered and queued into the stream grabber's input queue. This is done with [PylonStreamGrabberRegisterBuffer\(\)](#) and [PylonStreamGrabberQueueBuffer\(\)](#).

To enable image acquisition, [PylonDeviceExecuteCommandFeature\(\)](#) is called with the device handle and the AcquisitionStart camera parameter as arguments. After that, the cameras are triggered using [PylonGigEIssueActionCommand\(\)](#).

In [PylonWaitObjectsWaitForAny\(\)](#), we wait for the next buffer to be filled with a timeout of 5000 ms. The grabbed image is retrieved by calling [PylonStreamGrabberRetrieveResult\(\)](#).

With [PylonImageWindowDisplayImageGrabResult\(\)](#), images are displayed in an image window.

When image acquisition is stopped, we must perform a cleanup for all cameras, i.e., all wait objects must be removed, all allocated buffer memory must be released, and the stream grabber as well as the camera device handles must be closed and destroyed.

Finally, we shut down the pylon runtime system by calling [PylonTerminate\(\)](#). No pylon functions should be called after calling [PylonTerminate\(\)](#).

## Applicable Interfaces

GigE Vision

[↖ Back to top](#)

## 4.2 Chunks

Basler cameras supporting the Data Chunk feature can generate supplementary image data, e.g., frame count, time stamp, or CRC checksums, and append it to each acquired image.

This sample illustrates how to enable the Data Chunk feature, how to grab images, and how to process the appended data. When the camera is in chunk mode, it transfers data blocks partitioned into chunks. The first chunk is always the image data. If one or more data chunks are enabled, these chunks are transmitted as chunk 2, 3, and so on.

This sample also demonstrates how to use software triggers. Two buffers are used. Once a buffer is filled, the acquisition of the next frame is triggered before processing the received buffer. This approach allows acquiring images while the previous image is still being processed.

```

C:\Program Files\Basler\pylon 5\Development\Samples\C\Chunks\Win32\Debug\Chunks.exe
Enumerating devices ...
Opening first device ...
Grabbed frame # 1 into buffer 0.
Frame 1 contains a CRC checksum. The checksum is ok.
Frame 1 doesn't contain a frame counter chunk.
Frame 1 doesn't contain a width chunk.
Frame 1 doesn't contain a height chunk.
Min. gray value = 255, Max. gray value = 0
Grabbed frame # 2 into buffer 1.
Frame 2 contains a CRC checksum. The checksum is ok.
Frame 2 doesn't contain a frame counter chunk.
Frame 2 doesn't contain a width chunk.
Frame 2 doesn't contain a height chunk.
Min. gray value = 255, Max. gray value = 0
Grabbed frame # 3 into buffer 0.
Frame 3 contains a CRC checksum. The checksum is ok.
Frame 3 doesn't contain a frame counter chunk.
Frame 3 doesn't contain a width chunk.
Frame 3 doesn't contain a height chunk.
Min. gray value = 255, Max. gray value = 0
Grabbed frame # 4 into buffer 1.
Frame 4 contains a CRC checksum. The checksum is ok.
Frame 4 doesn't contain a frame counter chunk.
Frame 4 doesn't contain a width chunk.
Frame 4 doesn't contain a height chunk.

```

### Code

Before using any pylon methods, the pylon runtime is initialized by calling [PylonInitialize\(\)](#).

Then, [PylonEnumerateDevices\(\)](#) is called to enumerate all attached camera devices.

Before using a camera device, it must be opened by calling [PylonDeviceOpen\(\)](#). This allows us to set parameters and grab images.

As the camera will be triggered by software trigger, the TriggerMode and TriggerSource camera parameters are configured accordingly.

When using software triggering, the Continuous frame mode should be used. This is done by passing the device handle and the camera parameters "AcquisitionMode" and "Continuous" as arguments to [PylonDeviceFeatureFromString\(\)](#).

Before enabling individual chunks, the chunk mode must be activated. In this sample, the frame counter and the CRC checksum data chunks are enabled as well.

The data block containing the image chunk and the other chunks has a self-descriptive layout. A chunk parser is used to extract the appended chunk data from the grabbed image frame. A chunk parser is created with [PylonDeviceCreateChunkParser\(\)](#) by passing the device and the chunk parser handles as arguments.

Images are grabbed using a stream grabber. For each camera device, a stream grabber is created by calling [PylonDeviceGetStreamGrabber\(\)](#) and passing the device handle and the stream grabber handle as arguments. A handle for the stream grabber's wait object is retrieved within [PylonStreamGrabberGetWaitObject\(\)](#). The wait object allows waiting for buffers to be filled with grabbed data.

We must also tell the stream grabber the number and size of the buffers we are using. This is done with [PylonStreamGrabberSetMaxNumBuffer\(\)](#) and [PylonStreamGrabberSetMaxBufferSize\(\)](#). By calling [PylonStreamGrabberPrepareGrab\(\)](#) we allocate the resources required for grabbing. After this, critical parameters that impact the payload size must not be changed until [PylonStreamGrabberFinishGrab\(\)](#) is called.

Before using the buffers for grabbing, they must be registered and queued into the stream grabber's input queue. This is done with [PylonStreamGrabberRegisterBuffer\(\)](#) and [PylonStreamGrabberQueueBuffer\(\)](#).

To enable image acquisition, [PylonDeviceExecuteCommandFeature\(\)](#) is called with the device handle and the `AcquisitionStart` camera parameter as arguments.

Because the trigger mode is enabled, issuing the acquisition start command itself will not trigger any image acquisitions. Issuing the start command simply prepares the camera to acquire images. Once the camera is prepared it will acquire one image for every trigger it receives.

Software triggers are issued by calling [PylonDeviceExecuteCommandFeature\(\)](#) while passing the device handle and the "TriggerSoftware" camera parameter as arguments.

In [PylonWaitObjectsWait\(\)](#), we wait for the next buffer to be filled with a timeout of 1000 ms. The grabbed image is retrieved by calling [PylonStreamGrabberRetrieveResult\(\)](#).

If the image was grabbed successfully, we let the chunk parser extract the chunk data by calling [PylonChunkParserAttachBuffer\(\)](#).

After image processing is completed and before re-queueing the buffer, we detach it from the chunk parser by calling [PylonChunkParserDetachBuffer\(\)](#). Then, we re-queue the buffer to be filled with image data by calling [PylonStreamGrabberQueueBuffer\(\)](#).

When image acquisition is stopped, we must perform a cleanup for all cameras, i.e., all wait objects must be removed, all allocated buffer memory must be released, and the stream grabber as well as the camera device handles must be closed and destroyed.

Finally, we shut down the pylon runtime system by calling [PylonTerminate\(\)](#). No pylon functions should be called after calling [PylonTerminate\(\)](#).

## Applicable Interfaces

GigE Vision, USB3 Vision

[↖ Back to top](#)

## 4.3 Events

Basler GigE Vision and USB3 Vision cameras can send event messages. For example, when a sensor exposure has finished, the camera can send an Exposure End event to the computer. The event can be received by the computer before the image data for the finished exposure has been completely transferred. This sample illustrates how to retrieve and process event messages.

Receiving events is similar to grabbing images. An event grabber provides a wait object that is notified when an event message is available. When an event message is available, it can be retrieved from the event grabber. In contrast to grabbing images, you don't need to provide memory buffers to receive events. The memory buffers are organized by the event grabber itself.

The specific layout of event messages depends on the event type and the camera type. The event message layout is described in the camera's GenICam XML description file. From the file, a GenApi node map is created. This means that the information carried by the event messages is exposed as nodes in the node map and can be accessed like standard camera parameters.

You can register callback functions that are fired when a parameter has been changed. To be informed that a received event message contains a specific event, you must register a callback for the parameters associated with the event.

These mechanisms are demonstrated with the Exposure End event. The event carries the following information:

- `ExposureEndEventFrameID`: Number of the image that has been exposed.
- `ExposureEndEventTimestamp`: Time when the event was generated.
- `ExposureEndEventStreamChannelIndex`: Number of the image data stream used to transfer the image. On Basler cameras, this parameter is always set to 0.

A callback for the `ExposureEndEventFrameID` will be registered as an indicator for the arrival of an end-of-exposure event.

```
Using camera acA4024-29uc
Got end-of-exposure event. Frame number: 0
Grabbed frame # 1 into buffer 0. Min. gray value = 6, Max. gray value = 255
Got end-of-exposure event. Frame number: 1
Got end-of-exposure event. Frame number: 2
Grabbed frame # 2 into buffer 1. Min. gray value = 7, Max. gray value = 255
Got end-of-exposure event. Frame number: 3
Got end-of-exposure event. Frame number: 4
Grabbed frame # 3 into buffer 2. Min. gray value = 5, Max. gray value = 255
Got end-of-exposure event. Frame number: 5
Grabbed frame # 4 into buffer 3. Min. gray value = 7, Max. gray value = 255
Got end-of-exposure event. Frame number: 6
Got end-of-exposure event. Frame number: 7
Grabbed frame # 5 into buffer 4. Min. gray value = 7, Max. gray value = 255
Got end-of-exposure event. Frame number: 8
Got end-of-exposure event. Frame number: 9
Grabbed frame # 6 into buffer 0. Min. gray value = 7, Max. gray value = 255
Got end-of-exposure event. Frame number: 10
Grabbed frame # 7 into buffer 1. Min. gray value = 5, Max. gray value = 255
Got end-of-exposure event. Frame number: 11
Got end-of-exposure event. Frame number: 12
Grabbed frame # 8 into buffer 2. Min. gray value = 6, Max. gray value = 255
Got end-of-exposure event. Frame number: 13
Got end-of-exposure event. Frame number: 14
Grabbed frame # 9 into buffer 3. Min. gray value = 7, Max. gray value = 255
```

### Code

Before using any pylon methods, the pylon runtime is initialized by calling [PylonInitialize\(\)](#).

Then, [PylonEnumerateDevices\(\)](#) is called to enumerate all attached camera devices.

Before using a camera device, it must be opened by calling [PylonDeviceOpen\(\)](#). This allows us to set parameters and grab images.

In this sample, we will use the Continuous acquisition mode, i.e., the camera delivers images continuously. We do this by calling [PylonDeviceFeatureFromString\(\)](#) while passing the device handle and the camera parameters "AcquisitionMode" and "Continuous" as arguments.

To make use of camera events, we enable camera event reporting and select the Exposure End event.

To handle events, we create and prepare an event grabber by calling [PylonDeviceGetEventGrabber\(\)](#) while passing the device and event grabber handles as arguments. We tell the grabber how many buffers to use by calling [PylonEventGrabberSetNumBuffers\(\)](#).

In [PylonEventGrabberGetWaitObject\(\)](#), we retrieve the wait object that is associated with the event grabber. The event will be notified when an event message has been received.

To extract the event data from an event message, an event adapter is used. We create it by calling [PylonDeviceCreateEventAdapter\(\)](#).

We then register a callback function for the ExposureEndEventFrameID parameter by getting it from the device node map and calling [GenApiNodeRegisterCallback\(\)](#).

We create a container ([PylonWaitObjectsCreate](#)) and put the wait objects for image and event data into it ([PylonWaitObjectsAddMany](#)).

Images are grabbed using a stream grabber. For each camera device, a stream grabber is created by calling [PylonDeviceGetStreamGrabber\(\)](#) and passing the device handle and the stream grabber handle as arguments. A handle for the stream grabber's wait object is retrieved with [PylonStreamGrabberGetWaitObject\(\)](#). The wait object allows waiting for buffers to be filled with grabbed data.

We must also tell the stream grabber the number and size of the buffers we are using. This is done with [PylonStreamGrabberSetMaxNumBuffer\(\)](#) and [PylonStreamGrabberSetMaxBufferSize\(\)](#). By calling [PylonStreamGrabberPrepareGrab\(\)](#), we allocate the resources required for grabbing. After this, critical parameters that impact the payload size must not be changed until [PylonStreamGrabberFinishGrab\(\)](#) is called.

Before using the buffers for grabbing, they must be registered and queued into the stream grabber's input queue. This is done with [PylonStreamGrabberRegisterBuffer\(\)](#) and [PylonStreamGrabberQueueBuffer\(\)](#).

To enable image acquisition, [PylonDeviceExecuteCommandFeature\(\)](#) is called with the device handle and the AcquisitionStart camera parameter as arguments.

In [PylonWaitObjectsWaitForAny\(\)](#), we wait for either an image buffer grabbed or an event received with a timeout of 1000 ms.

Grabbed images are retrieved by calling [PylonStreamGrabberRetrieveResult\(\)](#).

Grabbed events are retrieved by calling [PylonEventGrabberRetrieveEvent\(\)](#).

Once finished with the processing, we re-queue the buffer to be filled again by calling [PylonStreamGrabberQueueBuffer\(\)](#).

When image acquisition is stopped, we must perform a cleanup for all cameras, i.e., all wait objects must be removed, all allocated buffer memory must be released, and the stream grabber as well as the camera device handles must be closed and destroyed.

Finally, we shut down the pylon runtime system by calling `PylonTerminate()`. No pylon functions should be called after calling `PylonTerminate()`.

### Applicable Interfaces

GigE Vision, USB3 Vision

[↖ Back to top](#)

## 4.4 GenApiParam

This sample illustrates how to access the different camera parameter types. It uses the low-level functions provided by GenApiC instead of those provided by pylonC.

```

Using camera acA640-300gc
The 'Width' feature is implemented
The 'Weirdness' feature is not implemented
The 'BinningVertical' feature is not available
The 'Width' feature is readable
The 'Width' feature is writable

Width: min= 16 max= 656 incr=16 Value=640
Cannot read feature 'Gamma' - node not readable
The GammaEnable features is on
Switching the GammaEnable feature off
Width: 336
Buffer is too small for the value of 'Width'. The required buffer size is 4
PylonC error #c2000002 'Failed to set node value from string.'

PixelFormat: Mono8
Mono8 is a supported value for the PixelFormat feature
YUV422Packed is a supported value for the PixelFormat feature
Mono16 is not a supported value for the PixelFormat feature
Setting PixelFormat to YUV422Packed

```

### Code

Before using any pylon methods, the pylon runtime is initialized by calling [PylonInitialize\(\)](#).

Then, [PylonEnumerateDevices\(\)](#) is called to enumerate all attached camera devices.

Before using a camera device, it must be opened. Open it by calling [PylonDeviceOpen\(\)](#) for setting parameters afterwards.

The following helper functions are used:

- [demonstrateAccessibilityCheck\(\)](#): Demonstrates how to check the accessibility of a camera feature, e.g., whether the camera feature "BinningVertical" is implemented and available for the current camera.
- [demonstrateIntFeature\(\)](#): Demonstrates how to handle integer camera parameters, e.g., the camera feature "Width".
- [demonstrateFloatFeature\(\)](#): Demonstrates how to handle floating point camera parameters, e.g., the camera feature "Gamma".
- [demonstrateBooleanFeature\(\)](#): Demonstrates how to handle boolean camera parameters, e.g., the camera feature "GammaEnable".
- [demonstrateFromStringToString\(\)](#): Demonstrates how to read or set camera features as a string. Regardless of the parameter's type, any parameter value can be retrieved as a string. In addition, each parameter can be set by passing in a string. This function illustrates how to set and get the integer parameter "Width" as string.
- [demonstrateEnumFeature\(\)](#): Demonstrates how to handle enumeration camera parameters, e.g., the camera feature "PixelFormat".
- [demonstrateEnumIteration\(\)](#): Demonstrates how to iterate enumeration entries, e.g., the enumeration entries of the camera feature "PixelFormat".

- [demonstrateCommandFeature\(\)](#): Demonstrates how to execute commands, e.g., load the camera factory settings by executing the "UserSetLoad" command.
- [demonstrateCategory\(\)](#): Demonstrates category node. The function traverses the feature tree, displaying all categories and all features.

Finally, a cleanup is done, e.g., the pylon device is closed and released. The pylon runtime system is shut down by calling [PylonTerminate\(\)](#). No pylon functions should be called after calling [PylonTerminate\(\)](#).

## Applicable Interfaces

GigE Vision, USB3 Vision, Camera Link, CXP

[↖ Back to top](#)

## 4.5 GrabTwoCameras

This sample illustrates how to grab images and process images using multiple cameras simultaneously.

The sample uses a pool of buffers that are passed to a stream grabber to be filled with image data. Once a buffer is filled and ready for processing, the buffer is retrieved from the stream grabber, processed, and passed back to the stream grabber to be filled again. Buffers retrieved from the stream grabber are not overwritten as long as they are not passed back to the stream grabber.

```

Using camera 'acA4024-29uc'
Using camera 'acA640-300gc'
Grabbed frame # 0 from camera 1 into buffer 0. Min. val= 0, Max. val= 23
Grabbed frame # 1 from camera 1 into buffer 1. Min. val= 0, Max. val= 12
Grabbed frame # 2 from camera 1 into buffer 2. Min. val= 0, Max. val= 12
Grabbed frame # 3 from camera 0 into buffer 0. Min. val= 18, Max. val=247
Grabbed frame # 4 from camera 0 into buffer 1. Min. val= 18, Max. val=247
Grabbed frame # 5 from camera 0 into buffer 2. Min. val= 18, Max. val=247
Grabbed frame # 6 from camera 0 into buffer 3. Min. val= 18, Max. val=247
Grabbed frame # 7 from camera 0 into buffer 4. Min. val= 18, Max. val=247
Grabbed frame # 8 from camera 0 into buffer 0. Min. val= 18, Max. val=247
Grabbed frame # 9 from camera 0 into buffer 1. Min. val= 18, Max. val=247
Grabbed frame #10 from camera 0 into buffer 2. Min. val= 18, Max. val=247
Grabbed frame #11 from camera 0 into buffer 3. Min. val= 18, Max. val=247
Grabbed frame #12 from camera 0 into buffer 4. Min. val= 18, Max. val=247
Grabbed frame #13 from camera 0 into buffer 0. Min. val= 18, Max. val=247
Grabbed frame #14 from camera 0 into buffer 1. Min. val= 18, Max. val=247
Grabbed frame #15 from camera 0 into buffer 2. Min. val= 18, Max. val=248
Grabbed frame #16 from camera 0 into buffer 3. Min. val= 18, Max. val=247
Grabbed frame #17 from camera 0 into buffer 4. Min. val= 18, Max. val=247
Grabbed frame #18 from camera 0 into buffer 0. Min. val= 18, Max. val=247
Grabbed frame #19 from camera 0 into buffer 1. Min. val= 18, Max. val=247
Grabbed frame #20 from camera 0 into buffer 2. Min. val= 18, Max. val=247
Grabbed frame #21 from camera 0 into buffer 3. Min. val= 18, Max. val=247
Grabbed frame #22 from camera 0 into buffer 4. Min. val= 18, Max. val=247

```

### Code

Before using any pylon methods, the pylon runtime is initialized by calling [PylonInitialize\(\)](#).

Then, [PylonEnumerateDevices\(\)](#) is called to enumerate all attached camera devices.

Before using a camera device, it must be opened by calling [PylonDeviceOpen\(\)](#). This allows us to set parameters and grab images.

Images are grabbed using a stream grabber. For each camera device, a stream grabber is created by calling [PylonDeviceGetStreamGrabber\(\)](#) and passing the device handle and the stream grabber handle as arguments. A handle for the stream grabber's wait object is retrieved within [PylonStreamGrabberGetWaitObject\(\)](#). The wait object allows waiting for buffers to be filled with grabbed data.

We must also tell the stream grabber the number and size of the buffers we are using. This is done with [PylonStreamGrabberSetMaxNumBuffer\(\)](#) and [PylonStreamGrabberSetMaxBufferSize\(\)](#). By calling [PylonStreamGrabberPrepareGrab\(\)](#) we allocate the resources required for grabbing. After this, critical parameters that impact the payload size must not be changed until [PylonStreamGrabberFinishGrab\(\)](#) is called.

Before using the buffers for grabbing, they must be registered and queued into the stream grabber's input queue. This is done with [PylonStreamGrabberRegisterBuffer\(\)](#) and [PylonStreamGrabberQueueBuffer\(\)](#).

We call [PylonDeviceExecuteCommandFeature\(\)](#) with the device handle and the `AcquisitionStart` camera parameter as arguments on each camera to start the image acquisition.

In [PylonWaitObjectsWaitForAny\(\)](#), we wait for the next buffer to be filled with a timeout of 1000 ms. The grabbed image is retrieved by calling [PylonStreamGrabberRetrieveResult\(\)](#).

With [PylonImageWindowDisplayImageGrabResult\(\)](#), images are displayed in different image windows.

Once finished with the processing, we re-queue the current grabbed buffer to be filled again by calling [PylonStreamGrabberQueueBuffer\(\)](#).

When image acquisition is stopped, we must perform a cleanup for all cameras, i.e., all wait objects must be removed, all allocated buffer memory must be released, and the stream grabber as well as the camera device handles must be closed and destroyed.

Finally, we shut down the pylon runtime system by calling [PylonTerminate\(\)](#). No pylon functions should be called after calling [PylonTerminate\(\)](#).

## Applicable Interfaces

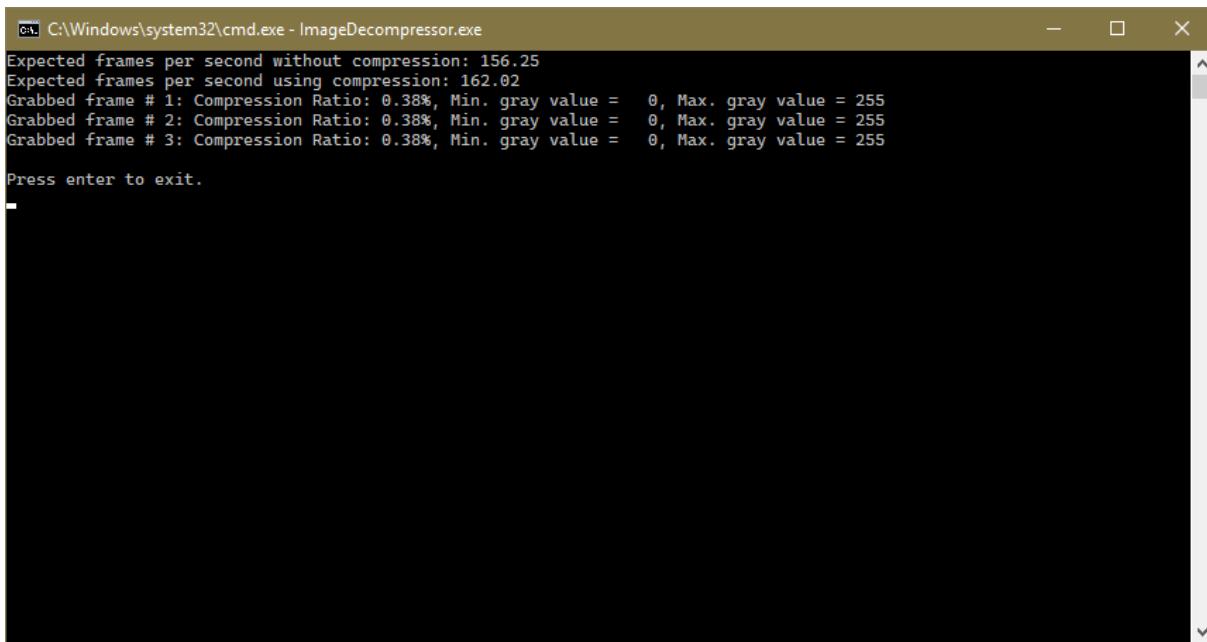
GigE Vision, USB3 Vision, CXP

[↖ Back to top](#)

## 4.6 ImageDecompressor

This sample illustrates how to enable and use the Basler Compression Beyond feature in Basler ace 2 Pro GigE and Basler ace 2 Pro USB 3.0 cameras.

This sample also demonstrates how to create and configure a pylon decompressor and use it to decompress the compressed camera images.



```
C:\Windows\system32\cmd.exe - ImageDecompressor.exe
Expected frames per second without compression: 156.25
Expected frames per second using compression: 162.02
Grabbed frame # 1: Compression Ratio: 0.38%, Min. gray value = 0, Max. gray value = 255
Grabbed frame # 2: Compression Ratio: 0.38%, Min. gray value = 0, Max. gray value = 255
Grabbed frame # 3: Compression Ratio: 0.38%, Min. gray value = 0, Max. gray value = 255
Press enter to exit.
```

### Code

Before using any pylon methods, the pylon runtime is initialized by calling [PylonInitialize\(\)](#).

Then, [PylonEnumerateDevices\(\)](#) is called to enumerate all attached camera devices.

Before using a camera device, it must be opened by calling [PylonDeviceOpen\(\)](#). This allows us to set parameters, e.g., to set the ImageCompressionMode parameter to On or Off.

The [configureCompression\(\)](#) function is used either to switch off the Compression Beyond feature or to configure the camera for lossless compression. In addition, you can also enable and use lossy compression.

The image decompressor is created by passing the decompressor handle to [PylonImageDecompressorCreate\(\)](#).

To be able to decompress image data, we have to set the compression descriptor first. This is done by calling [PylonImageDecompressorSetCompressionDescriptor\(\)](#) while passing the decompressor handle, the buffer used to store the compression descriptor, and the size of the compression descriptor as arguments.

Image grabbing is typically done by using a stream grabber. As we grab a single image in this sample, we allocate a single image buffer (malloc) without setting up a stream grabber.

The camera is set to Single Frame acquisition mode. We grab one single frame in a loop by calling [PylonDeviceGrabSingleFrame\(\)](#). We wait up to 2000 ms for the image to be grabbed.

As the information about the compressed image data is transmitted as chunk data, we retrieve this information by calling [PylonImageDecompressorGetCompressionInfo\(\)](#).

The decompression of the image data is done in [PylonImageDecompressorDecompressImage\(\)](#). When decompression is complete, information about the resulting frame rate, i.e., possible speed increases and the compression ratio applied, is printed in the terminal.

With [PylonImageWindowDisplayImageGrabResult\(\)](#), images are displayed in an image window.

At application exit, a cleanup for the camera device must be done, i.e., all allocated buffer memory must be released and the decompressor and the camera device handles must be freed and destroyed.

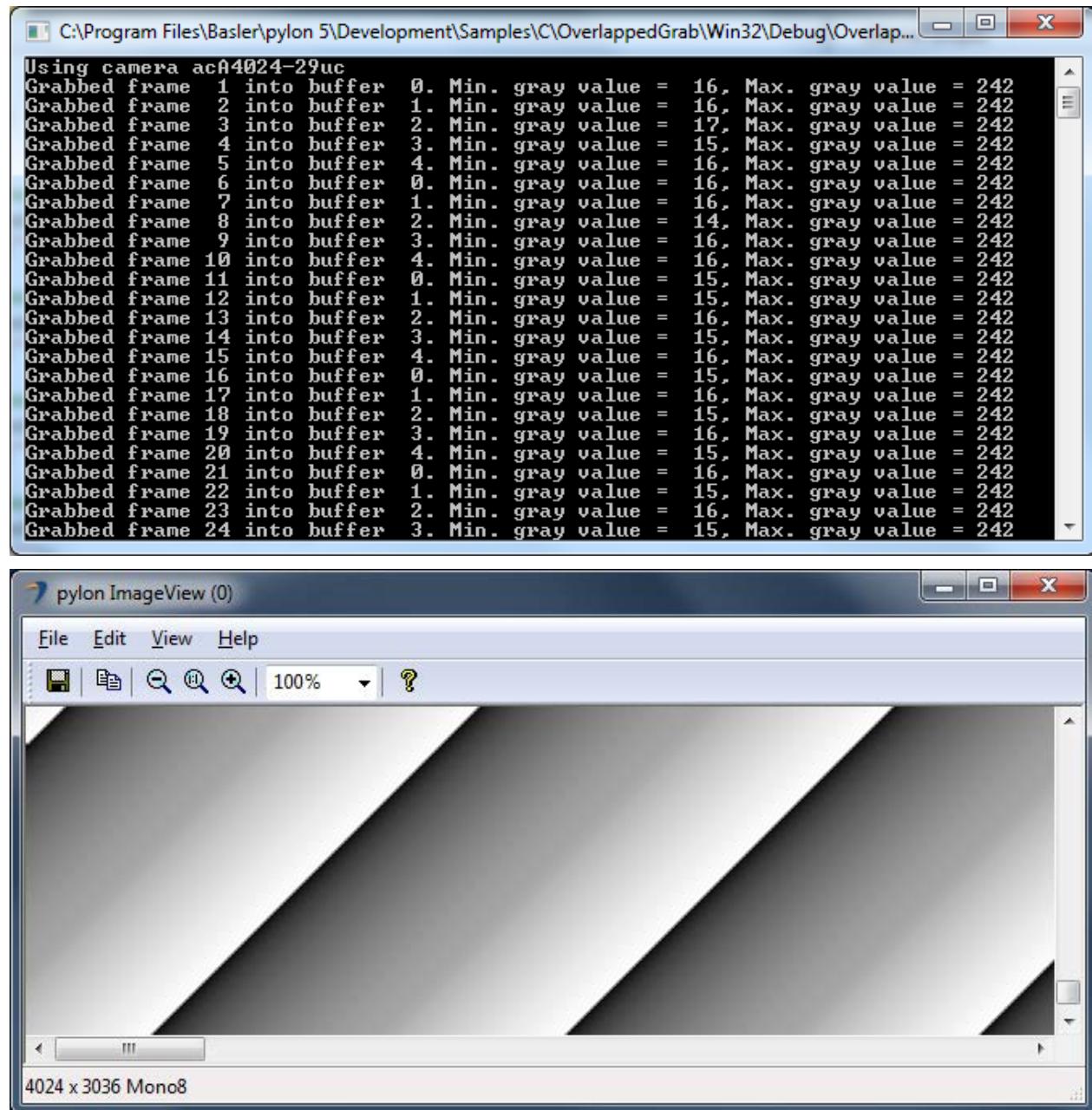
## Applicable Interfaces

GigE Vision, USB3 Vision

[↖ Back to top](#)

## 4.7 OverlappedGrab

This sample illustrates how to grab and process images asynchronously, i.e., while the application is processing a buffer, the acquisition of the next buffer is done in parallel. The sample uses a pool of buffers that are passed to a stream grabber to be filled with image data. Once a buffer is filled and ready for processing, the buffer is retrieved from the stream grabber, processed, and passed back to the stream grabber to be filled again. Buffers retrieved from the stream grabber are not overwritten as long as they are not passed back to the stream grabber.



### Code

Before using any pylon methods, the pylon runtime is initialized by calling [PylonInitialize\(\)](#).

Then, [PylonEnumerateDevices\(\)](#) is called to enumerate all attached camera devices.

Before using a camera device, it must be opened by calling [PylonDeviceOpen\(\)](#). This allows us to set parameters and grab images.

Images are grabbed using a stream grabber. For each camera device, a stream grabber is created by calling [PylonDeviceGetStreamGrabber\(\)](#) and passing the device handle and the stream grabber handle as arguments. A handle for the stream grabber's wait object is retrieved within [PylonStreamGrabberGetWaitObject\(\)](#). The wait object allows waiting for buffers to be filled with grabbed data.

We must also tell the stream grabber the number and size of the buffers we are using. This is done with [PylonStreamGrabberSetMaxNumBuffer\(\)](#) and [PylonStreamGrabberSetMaxBufferSize\(\)](#). By calling [PylonStreamGrabberPrepareGrab\(\)](#) we allocate the resources required for grabbing. After this, critical parameters that impact the payload size must not be changed until [PylonStreamGrabberFinishGrab\(\)](#) is called.

Before using the buffers for grabbing, they must be registered and queued into the stream grabber's input queue. This is done with [PylonStreamGrabberRegisterBuffer\(\)](#) and [PylonStreamGrabberQueueBuffer\(\)](#).

Call [PylonDeviceExecuteCommandFeature\(\)](#) with the device handle and the AcqusitionStart camera parameter as arguments on each camera to start the image acquisition.

In [PylonWaitObjectsWait\(\)](#) we wait for the next buffer to be filled with a timeout of 1000 ms. The grabbed image is retrieved by calling [PylonStreamGrabberRetrieveResult\(\)](#).

With [PylonImageWindowDisplayImageGrabResult\(\)](#) , images are displayed in an image window.

Once finished with the processing, we re-queue the current grabbed buffer to be filled again by calling [PylonStreamGrabberQueueBuffer\(\)](#).

When image acquisition is stopped, we must perform a cleanup for all cameras, i.e., all allocated buffer memory must be released and the stream grabber as well as the camera device handles must be closed and destroyed.

Finally, we shut down the pylon runtime system by calling [PylonTerminate\(\)](#). No pylon functions should be called after calling [PylonTerminate\(\)](#).

## Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[^ Back to top](#)

## 4.8 ParametrizeCamera

This sample illustrates how to read and write different camera parameter types.

```

Using camera acA4024-29uc
The 'Width' feature is implemented
The 'MyCustomFeature' feature isn't implemented
The 'BinningVertical' feature isn't available
The 'Width' feature is readable
The 'MyCustomFeature' feature isn't readable
The 'Width' feature is writable

Width: min= 8 max= 4024 incr=4 Value=4024
Height: min= 6 max= 3036 incr=2 Value=3036
Gamma: min = 0.00, max = 4.00, value = 1.00
Setting Gamma to 2.00

The GammaEnable feature isn't writable

Width: 2016
Buffer is too small for the value of 'Width'. The required buffer size is 5
PylonC error #c2000002 'Failed to set node value from string.'

PixelFormat: Mono8
Mono8 is a supported value for the PixelFormat feature
YUV422Packed isn't a supported value for the PixelFormat feature

```

### Code

Before using any pylon methods, the pylon runtime is initialized by calling [PylonInitialize\(\)](#).

Then, [PylonEnumerateDevices\(\)](#) is called to enumerate all attached camera devices.

Before using a camera device, it must be opened by calling [PylonDeviceOpen\(\)](#). This allows us to set parameters.

The following helper functions are used:

- [demonstrateAccessibilityCheck\(\)](#): Demonstrates how to check the accessibility of a camera feature, e.g., whether the camera feature "BinningVertical" is implemented and available for the current camera.
- [demonstrateIntFeature\(\)](#): Demonstrates how to handle integer camera parameters, e.g., the camera feature "Width".
- [demonstrateInt32Feature\(\)](#): Demonstrates how to handle integer camera parameters, e.g., the camera feature "Height".
- [demonstrateFloatFeature\(\)](#): Demonstrates how to handle floating point camera parameters, e.g., the camera feature "Gamma".
- [demonstrateBooleanFeature\(\)](#): Demonstrates how to handle boolean camera parameters, e.g., the camera feature "GammaEnable".
- [demonstrateFromStringToString\(\)](#): Demonstrates how to read or set camera features as a string. Regardless of the parameter's type, any parameter value can be retrieved as a string. In addition, each parameter can be set by passing in a string. This function illustrates how to set and get the integer parameter "Width" as string.
- [demonstrateEnumFeature\(\)](#): Demonstrates how to handle enumeration camera parameters, e.g., the camera feature "PixelFormat".

- [demonstrateCommandFeature\(\)](#): Demonstrates how to execute commands, e.g., load the camera factory settings by executing the "UserSetLoad" command.

Finally, a cleanup is done, e.g., the pylon device is closed and released. The pylon runtime system is shut down by calling [PylonTerminate\(\)](#). No pylon functions should be called after calling [PylonTerminate\(\)](#).

## Applicable Interfaces

GigE Vision, USB3 Vision, Camera Link, CXP

[↖ Back to top](#)

## 4.9 SimpleGrab

This sample illustrates how to use the [PylonDeviceGrabSingleFrame\(\)](#) convenience method for grabbing images in a loop. [PylonDeviceGrabSingleFrame\(\)](#) grabs one single frame in single frame mode.

Grabbing in Single Frame acquisition mode is the easiest way to grab images.

**Note:** In Single Frame mode, the maximum frame rate of the camera can't be achieved. The maximum frame rate can be achieved by setting the camera to the Continuous frame acquisition mode and by grabbing in overlapped mode, i.e., image acquisition begins while the camera is still processing the previous image. This is illustrated in the OverlappedGrab sample program.

```
Using camera acA4024-29uc
Grabbed frame # 1. Min. gray value = 11, Max. gray value = 188
Grabbed frame # 2. Min. gray value = 11, Max. gray value = 187
Grabbed frame # 3. Min. gray value = 11, Max. gray value = 187
Grabbed frame # 4. Min. gray value = 11, Max. gray value = 187
Grabbed frame # 5. Min. gray value = 12, Max. gray value = 187
Grabbed frame # 6. Min. gray value = 11, Max. gray value = 187
Grabbed frame # 7. Min. gray value = 11, Max. gray value = 187
Grabbed frame # 8. Min. gray value = 11, Max. gray value = 186
Grabbed frame # 9. Min. gray value = 11, Max. gray value = 187
Grabbed frame #10. Min. gray value = 12, Max. gray value = 188

Press enter to exit.
```

### Code

Before using any pylon methods, the pylon runtime is initialized by calling [PylonInitialize\(\)](#).

Then, [PylonEnumerateDevices\(\)](#) is called to enumerate all attached camera devices.

Before using a camera device, it must be opened by calling [PylonDeviceOpen\(\)](#). This allows us to set parameters and grab images.

Image grabbing is typically done by using a stream grabber. As we grab a single image in this sample, we allocate a single image buffer (malloc) without setting up a stream grabber.

The camera is set to Single Frame acquisition mode. We grab one single frame in a loop by calling [PylonDeviceGrabSingleFrame\(\)](#). We wait up to 500 ms for the image to be grabbed.

With [PylonImageWindowDisplayImageGrabResult\(\)](#), images are displayed in an image window.

When the image acquisition is stopped, a cleanup for the camera device must be done, i.e., all allocated buffer memory must be released and the camera device handles must be closed and destroyed.

Finally, we shut down the pylon runtime system by calling `PylonTerminate()`. No pylon functions should be called after calling `PylonTerminate()`.

### Applicable Interfaces

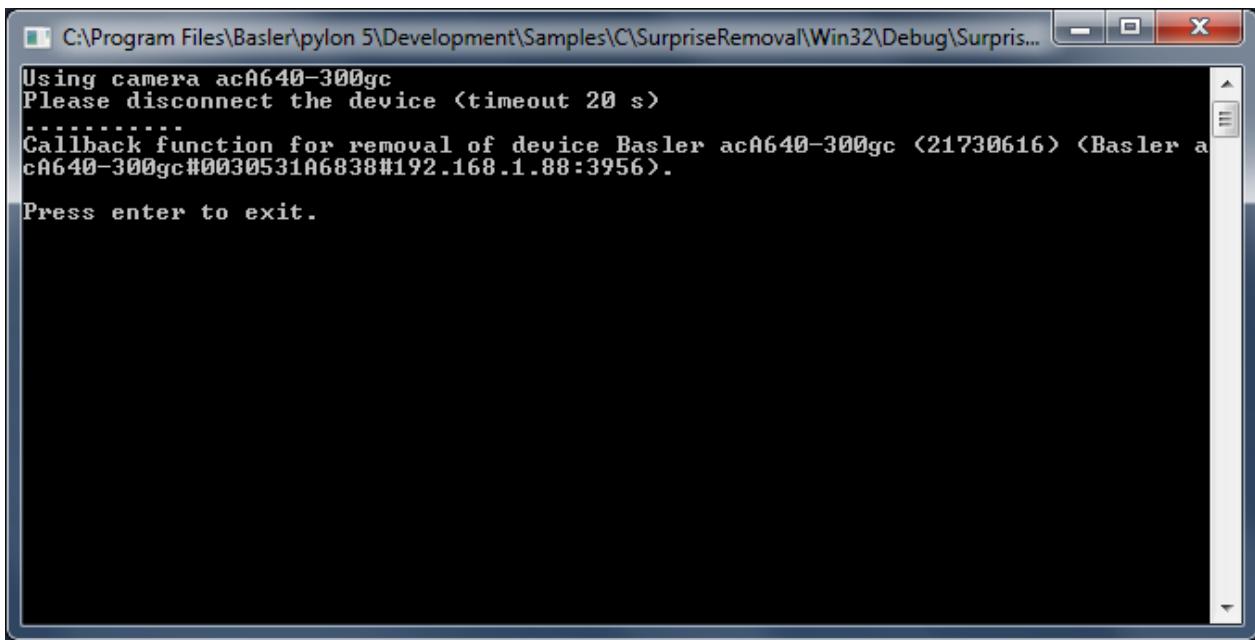
GigE Vision, USB3 Vision, CXP

[↖ Back to top](#)

## 4.10 SurpriseRemoval

This sample program demonstrates how to be informed about a sudden removal of a device.

**Note:** If you build this sample in debug mode and run it using a GigE camera device, pylon will set the heartbeat timeout to 5 minutes. This is done to allow debugging and single-stepping without losing the camera connection due to missing heartbeats. However, with this setting, it would take 5 minutes for the application to notice that a GigE device has been disconnected. As a workaround, the heartbeat timeout is set to 1000 ms.



The screenshot shows a Windows command-line interface window titled 'C:\Program Files\Basler\pylon 5\Development\Samples\C\SurpriseRemoval\Win32\Debug\SurpriseRemoval'. The window contains the following text:

```
Using camera acA640-300gc
Please disconnect the device <timeout 20 s>
.....
Callback function for removal of device Basler acA640-300gc <21730616> <Basler acA640-300gc#0030531A6838#192.168.1.88:3956>.

Press enter to exit.
```

### Code

Before using any pylon methods, the pylon runtime is initialized by calling [PylonInitialize\(\)](#).

Then, [PylonEnumerateDevices\(\)](#) is called to enumerate all attached camera devices.

Before using a camera device, it must be opened by calling [PylonDeviceOpen\(\)](#). This allows us to set parameters and grab images.

In [PylonDeviceRegisterRemovalCallback\(\)](#), we register the callback function [removalCallbackFunction\(\)](#). This function will be called when the opened device has been removed.

The [setHeartbeatTimeout\(\)](#) function is used to adjust the heartbeat timeout. For GigE cameras, the application periodically sends heartbeat signals to the camera to keep the connection to the camera alive. If the camera doesn't receive heartbeat signals within the time period specified by the heartbeat timeout counter, the camera resets the connection. When the application is stopped by the debugger, the application cannot create the heartbeat signals. For that reason, the pylon runtime extends the heartbeat timeout when debugging to 5 minutes. For GigE cameras, we will set the heartbeat timeout to a shorter period before testing the callbacks.

The heartbeat mechanism is also used for detection of device removal. When the pylon runtime doesn't receive acknowledges for the heartbeat signal, it is assumed that the device has been removed. A removal callback will be fired in that case. By decreasing the heartbeat timeout, the surprise removal will be noticed earlier.

When we exit the application, a cleanup for the camera device must be done, i.e., the removal callback must be deregistered and the camera device handle must be closed and destroyed.

Finally, we shut down the pylon runtime system by calling [PylonTerminate\(\)](#). No pylon functions should be called after calling [PylonTerminate\(\)](#).

## Applicable Interfaces

GigE Vision, USB3 Vision, Camera Link

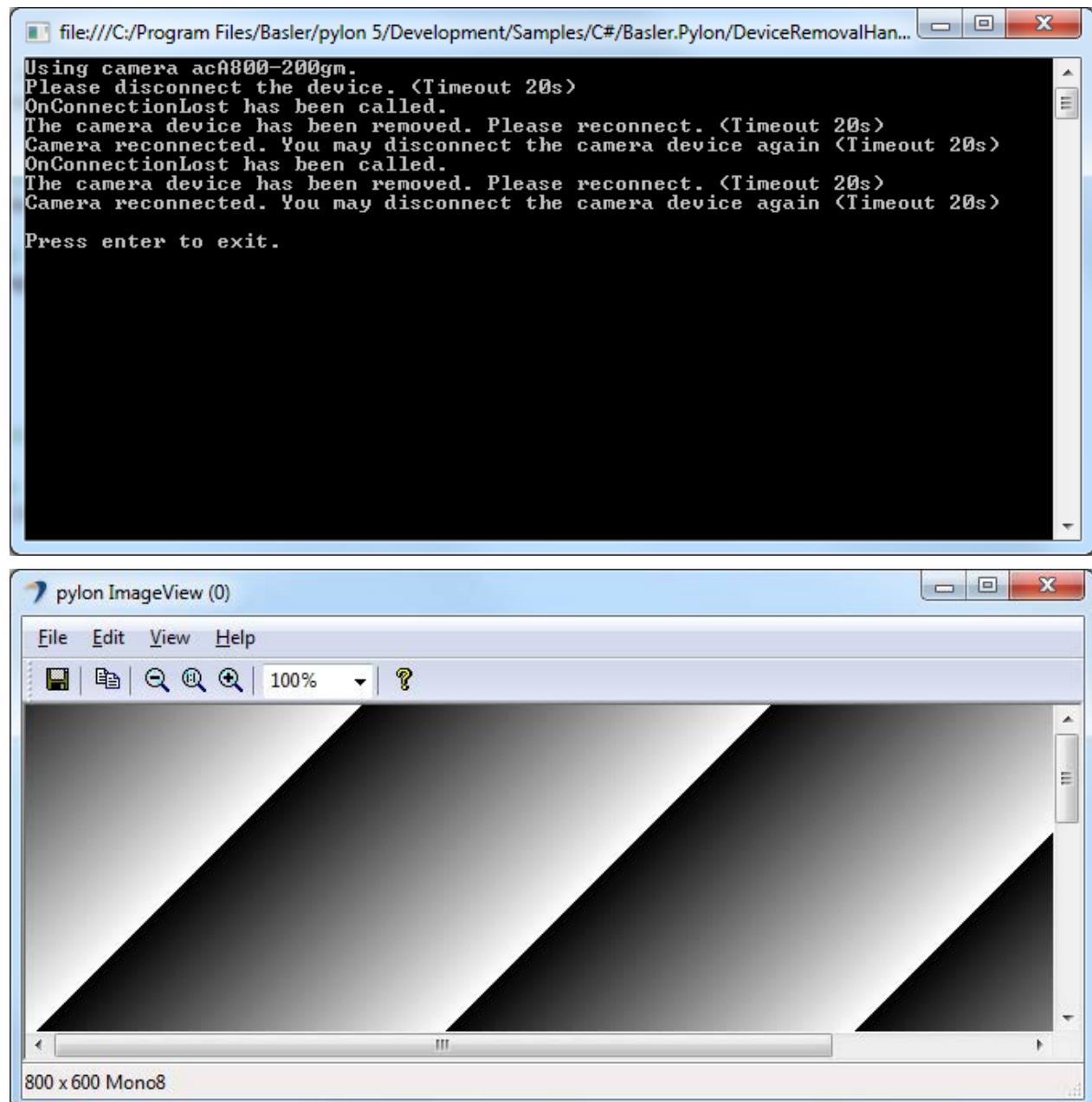
[ŷ Back to top](#)

## 5 .NET Samples

### 5.1 DeviceRemovalHandling

This sample program demonstrates how to be informed about the removal of a camera device. It also shows how to reconnect to a removed device.

**Note:** If you build this sample in debug mode and run it using a GigE camera device, pylon will set the heartbeat timeout to 5 minutes. This is done to allow debugging and single-stepping without losing the camera connection due to missing heartbeats. However, with this setting, it would take 5 minutes for the application to notice that a GigE device has been disconnected. As a workaround, the heartbeat timeout is set to 1000 ms.



## Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [Configuration](#) class is used to set the acquisition mode to free running continuous acquisition when the camera is opened.

For demonstration purposes, the event handler [OnConnectionLost\(\)](#) is added. This event is always called on a separate thread when the physical connection to the camera has been lost.

The [PLTransportLayer](#) class provides a list of all available transport layer parameters, e.g., GigE or USB 3.0 parameters. It can be used to manually set the heartbeat timeout to a shorter value when using GigE cameras.

The [ImageWindow](#) class is used to display the grabbed image on the screen.

## Applicable Interfaces

GigE Vision, USB3 Vision, Camera Link

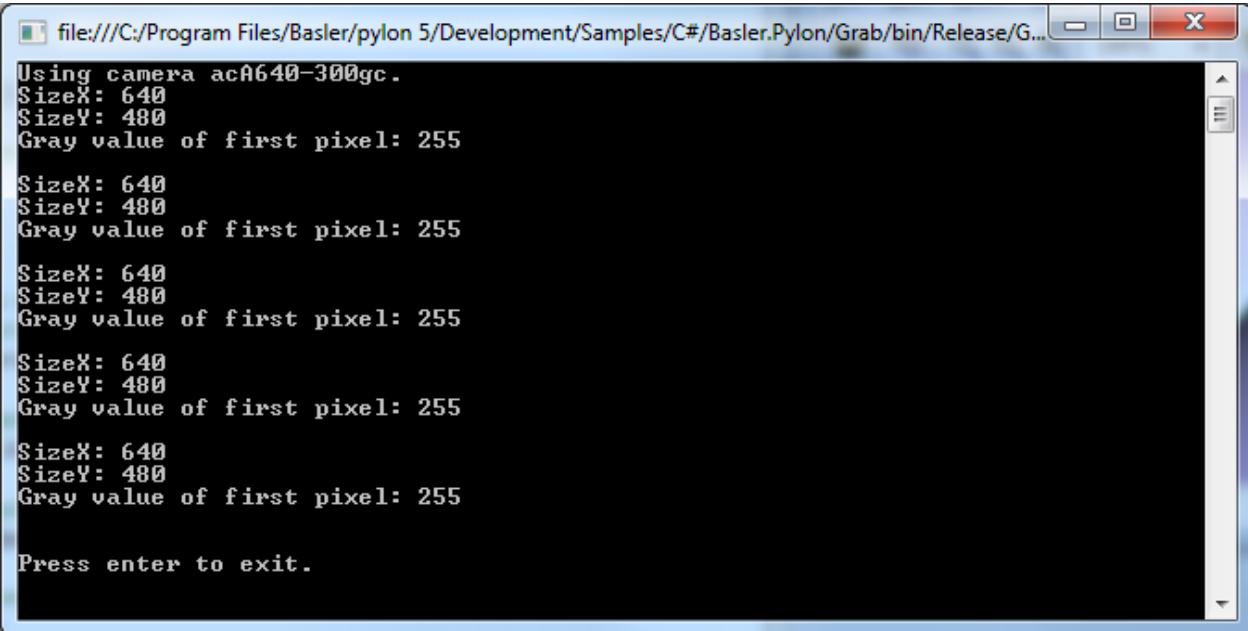
[^ Back to top](#)

## 5.2 Grab

This sample illustrates how to grab images and process images asynchronously. This means that while the application is processing a buffer, the acquisition of the next buffer is done in parallel.

The sample uses a pool of buffers. The buffers are allocated automatically. Once a buffer is filled and ready for processing, the buffer is retrieved from the stream grabber as part of a grab result. The grab result is processed and the buffer is passed back to the stream grabber by disposing the grab result. The buffer is reused and refilled.

A buffer retrieved from the stream grabber as a grab result is not overwritten in the background as long as the grab result is not disposed.



```
file:///C:/Program Files/Basler/pylon 5/Development/Samples/C#/Basler.Pylon/Grab/bin/Release/G...
Using camera acA640-300gc.
SizeX: 640
SizeY: 480
Gray value of first pixel: 255

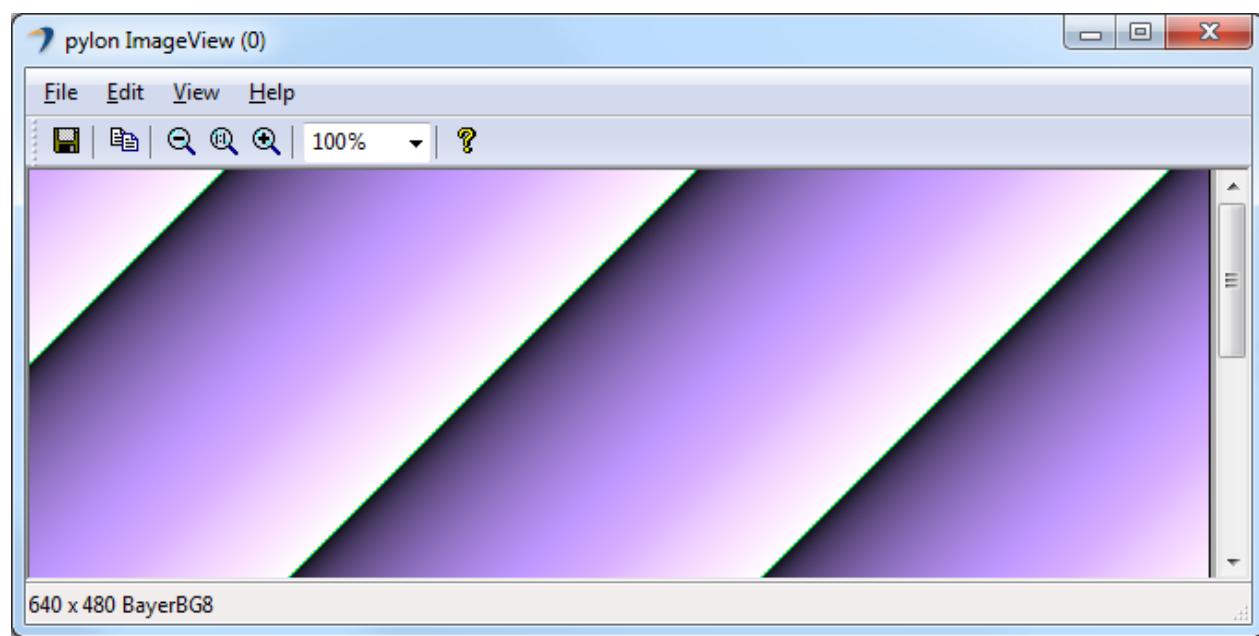
SizeX: 640
SizeY: 480
Gray value of first pixel: 255

SizeX: 640
SizeY: 480
Gray value of first pixel: 255

SizeX: 640
SizeY: 480
Gray value of first pixel: 255

SizeX: 640
SizeY: 480
Gray value of first pixel: 255

Press enter to exit.
```



## Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [Configuration](#) class is used to set the acquisition mode to free running continuous acquisition when the camera is opened.

The [PLCCameraInstance](#) class provides a list of all parameter names available for the Camera class instance. It is used to set the parameter MaxNumBuffer that controls the amount of buffers allocated for grabbing.

The [ImageWindow](#) class is used to display the grabbed image on the screen.

## Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[Back to top](#)

### 5.3 Grab\_CameraEvents

Basler USB3 Vision and GigE Vision cameras can send event messages. For example, when a sensor exposure has finished, the camera can send an Exposure End event to the computer. The event can be received by the computer before the image data for the finished exposure has been completely transferred. This sample illustrates how to be notified when camera event message data has been received.

The event messages are retrieved automatically and processed by the Camera classes.

The information contained in event messages is exposed as parameter nodes in the camera node map and can be accessed like standard camera parameters. These nodes are updated when a camera event is received. You can register camera event handler objects that are triggered when event data has been received.

The handler object provides access to the changed parameter, but not to its source (the camera).

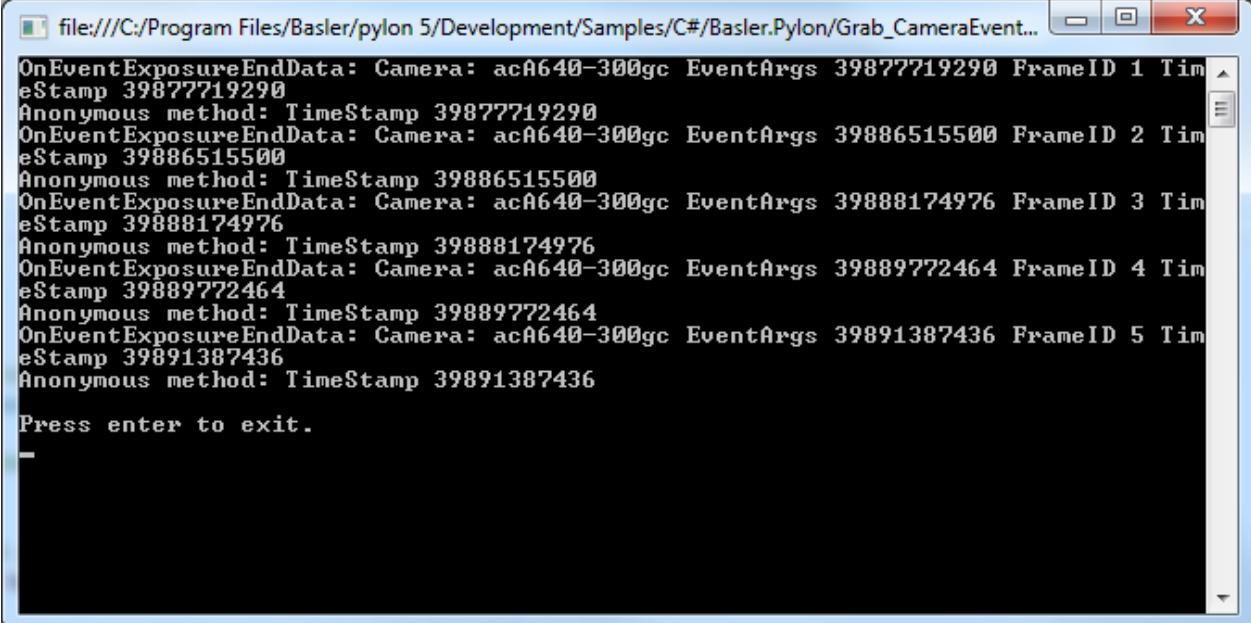
In this sample, we solve this problem with a derived camera class with a handler object as member.

These mechanisms are demonstrated for the Exposure End event.

The Exposure End event carries the following information:

- EventExposureEndFrameID (USB) / ExposureEndEventFrameID (GigE): Number of the image that has been exposed.
- EventExposureEndTimestamp (USB) / ExposureEndEventTimestamp (GigE): Time when the event was generated.

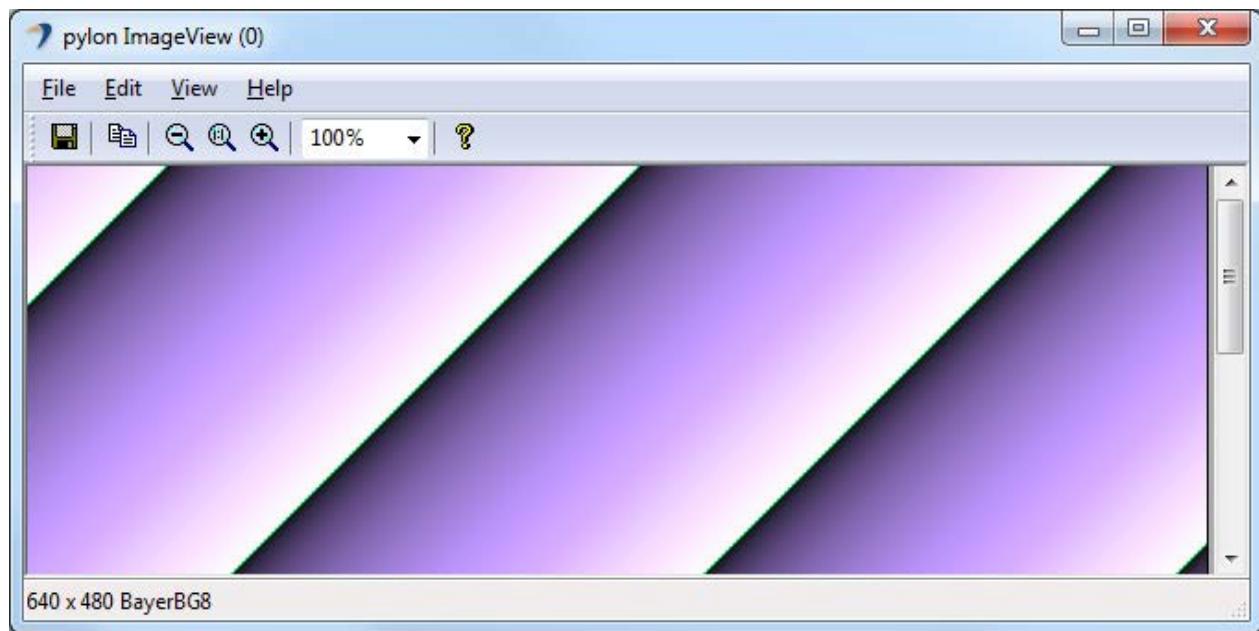
This sample shows how to register event handlers that indicate the arrival of events sent by the camera. For demonstration purposes, different handlers are registered for the same event.



A screenshot of a terminal window titled "file:///C:/Program Files/Basler/pylon 5/Development/Samples/C#/Basler.Pylon/Grab\_CameraEvent...". The window displays a series of log entries for "OnEventExposureEndData" events. Each entry includes the camera model ("acA640-300gc"), the event arguments (a timestamp), the frame ID (e.g., 1, 2, 3, 4, 5), and the timestamp at which the event was generated. The entries are as follows:

```
OnEventExposureEndData: Camera: acA640-300gc EventArgs 39877719290 FrameID 1 TimeStamp 39877719290
Anonymous method: TimeStamp 39877719290
OnEventExposureEndData: Camera: acA640-300gc EventArgs 39886515500 FrameID 2 TimeStamp 39886515500
Anonymous method: TimeStamp 39886515500
OnEventExposureEndData: Camera: acA640-300gc EventArgs 39888174976 FrameID 3 TimeStamp 39888174976
Anonymous method: TimeStamp 39888174976
OnEventExposureEndData: Camera: acA640-300gc EventArgs 39889772464 FrameID 4 TimeStamp 39889772464
Anonymous method: TimeStamp 39889772464
OnEventExposureEndData: Camera: acA640-300gc EventArgs 39891387436 FrameID 5 TimeStamp 39891387436
Anonymous method: TimeStamp 39891387436

Press enter to exit.
```



## Code

The [EventCamera](#) class is derived from the [Camera](#) class. It is used to create a camera object that opens the first camera device found. This class provides different methods for camera configuration and event handling. [Configure\(\)](#) is used to configure the camera for event trigger and register exposure end event handler.

The [Configuration](#) class is used to configure the camera for software trigger mode to demonstrate synchronous processing of the grab results.

The [PLCameraInstance](#) class provides a list of all parameter names available for the Camera class instance. Here, it is used to enable event notification.

The [PLGigE Camera](#) and [PLUsb Camera](#) camera classes are used to access GigE and USB3 Vision specific camera features related to the Exposure End event.

The [PLCamera](#) class is used to enable Exposure End event transmission.

[OnEventExposureEndData\(\)](#) is used to register an event handler to receive the changed FrameID value of the exposure end event.

**Note:** Only short processing tasks should be performed by this method. Otherwise, the event notification will block the processing of images.

The [ImageWindow](#) class is used to display the grabbed image on the screen.

## Applicable Interfaces

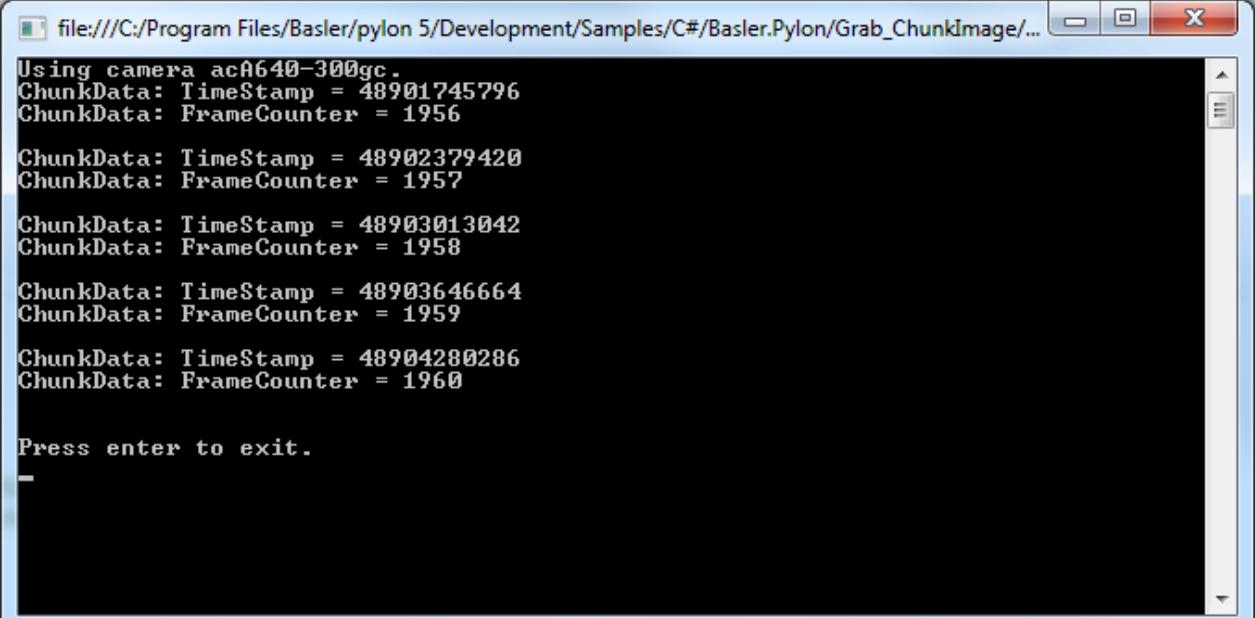
GigE Vision, USB3 Vision

[Back to top](#)

## 5.4 Grab\_ChunkImage

Basler cameras supporting the Data Chunk feature can generate supplementary image data, e.g., frame count, time stamp, or CRC checksums, and append it to each acquired image.

This sample illustrates how to enable the Data Chunk feature, how to grab images, and how to process the appended data. When the camera is in chunk mode, it transfers data blocks partitioned into chunks. The first chunk is always the image data. If one or more data chunks are enabled, these chunks are transmitted as chunk 2, 3, and so on.



```
Using camera acA640-300gc.
ChunkData: TimeStamp = 48901745796
ChunkData: FrameCounter = 1956

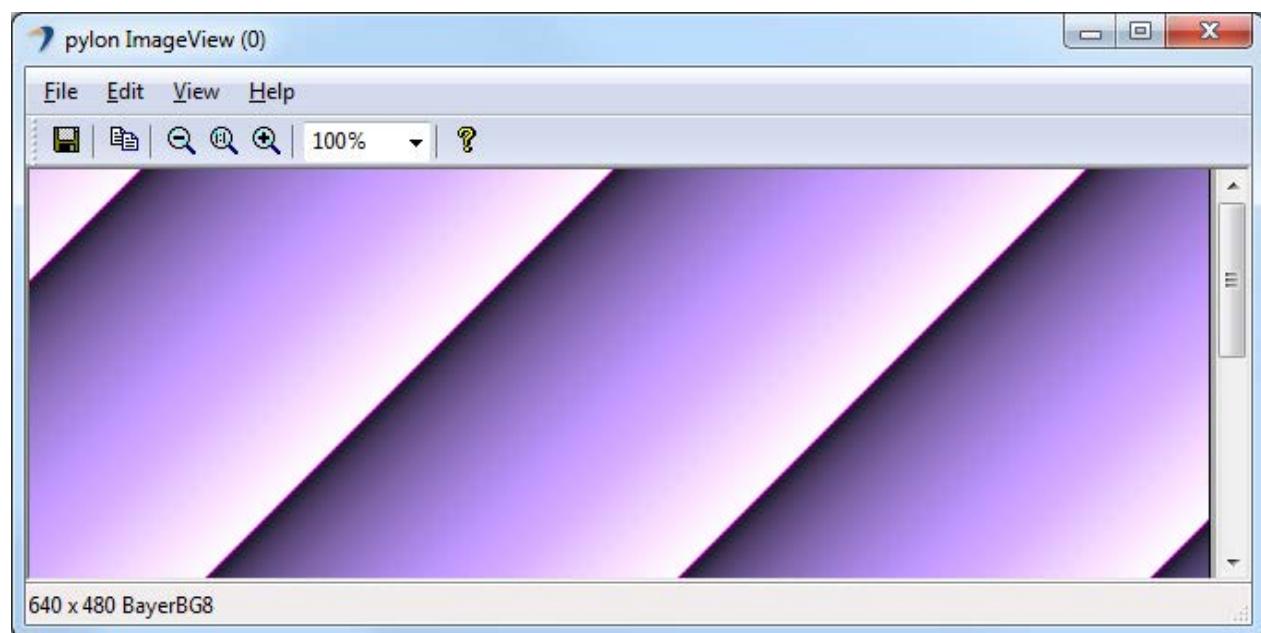
ChunkData: TimeStamp = 48902379420
ChunkData: FrameCounter = 1957

ChunkData: TimeStamp = 48903013042
ChunkData: FrameCounter = 1958

ChunkData: TimeStamp = 48903646664
ChunkData: FrameCounter = 1959

ChunkData: TimeStamp = 48904280286
ChunkData: FrameCounter = 1960

Press enter to exit.
```



### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [Configuration](#) class is used to set the acquisition mode to free running continuous acquisition when the camera is opened.

The [PLCamera](#) class is used to enable the chunk mode in general as well as specific camera chunks like timestamp, frame counter, CRC checksum, etc.

The [ImageWindow](#) class is used to display the grabbed image on the screen.

## Applicable Interfaces

GigE Vision, USB3 Vision

[↖ Back to top](#)

## 5.5 Grab\_MultiCast

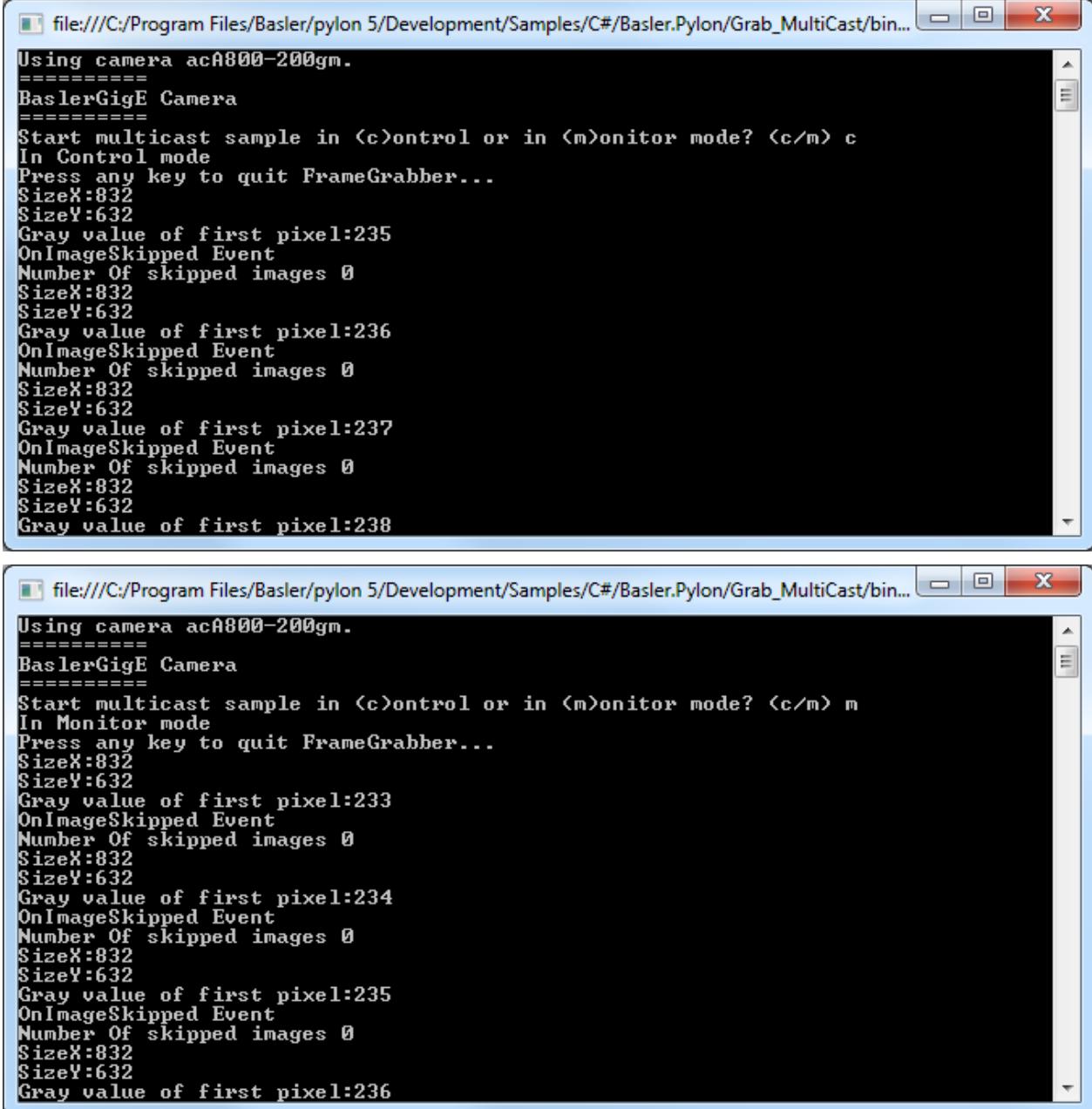
This sample demonstrates how to open a camera in multicast mode and how to receive a multicast stream.

Two instances of this application must be started simultaneously on different computers.

The first application started on computer A acts as the controlling application and has full access to the GigE camera.

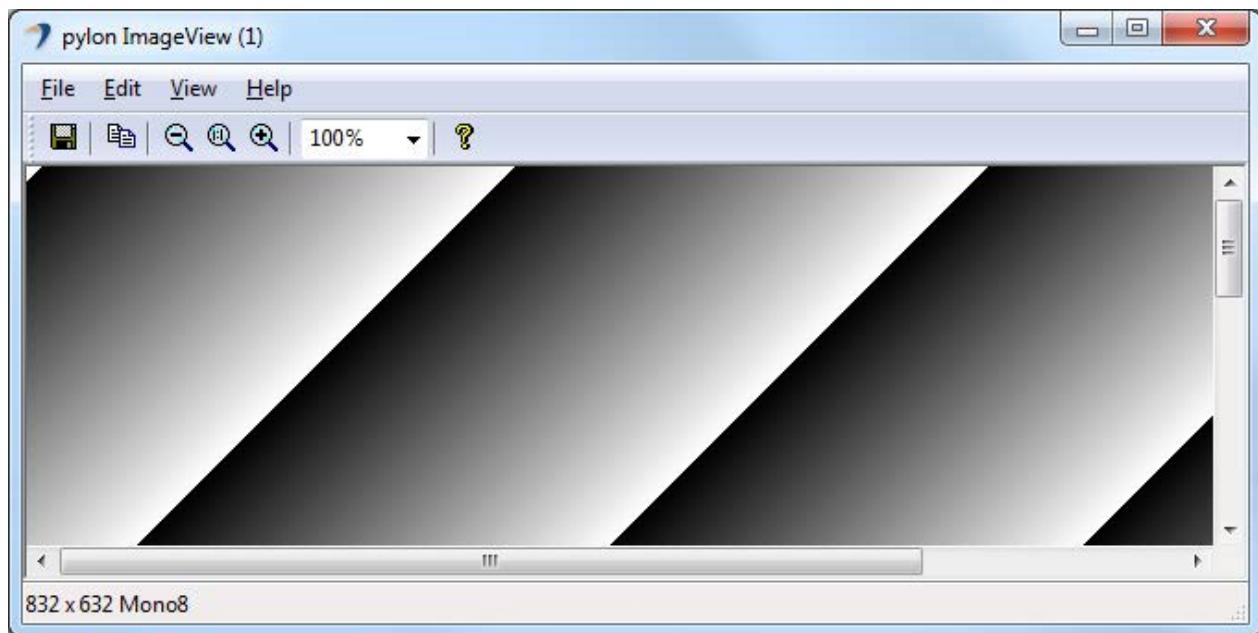
The second instance started on computer B opens the camera in monitor mode. This instance can't control the camera but can receive multicast streams.

To get the sample running, start the application on computer A in control mode. After computer A has begun to receive frames, start a second instance of the application on computer B in monitor mode.



```
file:///C:/Program Files/Basler/pylon 5/Development/Samples/C#/Basler.Pylon/Grab_MultiCast/bin... - X
Using camera acA800-200gm.
=====
BaslerGigE Camera
=====
Start multicast sample in <c>ontrol or in <m>onitor mode? <c/m> c
In Control mode
Press any key to quit FrameGrabber...
SizeX:832
SizeY:632
Gray value of first pixel:235
OnImageSkipped Event
Number Of skipped images 0
SizeX:832
SizeY:632
Gray value of first pixel:236
OnImageSkipped Event
Number Of skipped images 0
SizeX:832
SizeY:632
Gray value of first pixel:237
OnImageSkipped Event
Number Of skipped images 0
SizeX:832
SizeY:632
Gray value of first pixel:238

file:///C:/Program Files/Basler/pylon 5/Development/Samples/C#/Basler.Pylon/Grab_MultiCast/bin... - X
Using camera acA800-200gm.
=====
BaslerGigE Camera
=====
Start multicast sample in <c>ontrol or in <m>onitor mode? <c/m> m
In Monitor mode
Press any key to quit FrameGrabber...
SizeX:832
SizeY:632
Gray value of first pixel:233
OnImageSkipped Event
Number Of skipped images 0
SizeX:832
SizeY:632
Gray value of first pixel:234
OnImageSkipped Event
Number Of skipped images 0
SizeX:832
SizeY:632
Gray value of first pixel:235
OnImageSkipped Event
Number Of skipped images 0
SizeX:832
SizeY:632
Gray value of first pixel:236
```



## Code

The [Camera](#) class is used to create a camera object that opens the first GigE camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [PLCameralnstance](#) class provides a list of all parameter names available for the Camera class instance. It is used to open the camera in control or monitor mode depending on the user's input. While being opened in control mode, the control user application can adjust camera parameters and control image acquisition. While being opened in monitor mode, the monitor customer application can only read camera features and receive image data.

The [PLGigEStream](#) class provides a list of all parameter names available for the GigE stream grabber. It is used to configure the camera transmission type, e.g., for multicasting.

The [PLGigECamera](#) class provides a list of all parameter names available for GigE cameras only. It is used to configure the image area of interest and set the pixel data format.

The [ImageWindow](#) class is used to display the grabbed image on the screen.

## Applicable Interfaces

GigE Vision

[Back to top](#)

## 5.6 Grab\_Strategies

This sample demonstrates the use of the Camera grab strategies `GrabStrategy.OneByOne` and `GrabStrategy.LatestImages`.

When the "OneByOne" grab strategy is used, images are processed in the order of their acquisition. This strategy can be useful when all grabbed images need to be processed, e.g., in production and quality inspection applications.

The "LatestImages" strategy can be useful when the acquired images are only displayed on screen. If the processor has been busy for a while and images could not be displayed automatically, the latest image is displayed when processing time is available again.

```
Using camera acA640-300gc.
Grab using the GrabStrategy.OneByOne default strategy:
Grab results wait in the output queue.
SizeX:640
SizeY:480
Gray value of first pixel:255
SizeX:640
SizeY:480
Gray value of first pixel:255
SizeX:640
SizeY:480
Gray value of first pixel:255
Retrieved 3 grab results from output queue.
Grab using strategy GrabStrategy.LatestImages
Grab results wait in the output queue.
SizeX:640
SizeY:480
Gray value of first pixel:255
Skipped 1 images.
SizeX:640
SizeY:480
Gray value of first pixel:255
Retrieved 2 grab result from output queue.

Press enter to exit.
```

### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [PLCameraInstance](#) class provides a list of all parameter names available for the Camera class instance. It is used to enable the grabbing of camera events in general and control the buffer size of the output queue.

The [Configuration](#) class is used to configure the camera for software trigger mode.

The [PLStream](#) class provides a list of all parameter names available for the stream grabber. It is used to set the `MaxNumBuffer` parameter that controls the count of buffers allocated for grabbing. The default value of this parameter is 10.

The grab strategies `GrabStrategy.OneByOne` and `GrabStrategy.LatestImages` are applied by passing them as an argument to [Start\(\)](#), which is called on the stream grabber.

### Applicable Interfaces

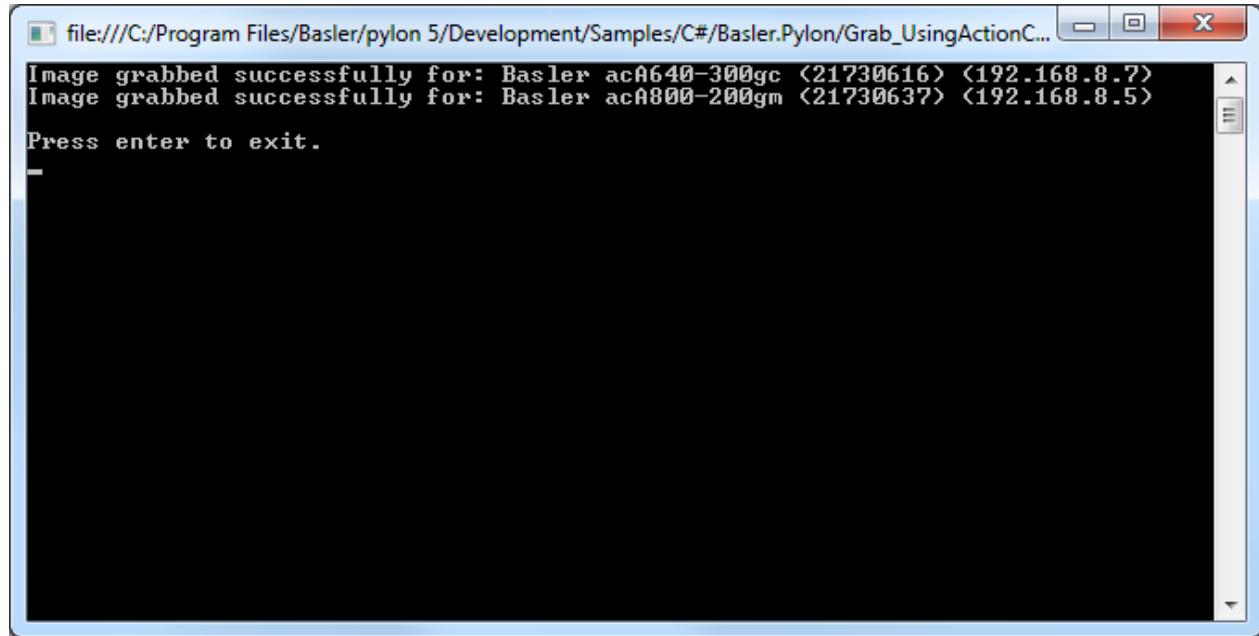
GigE Vision, USB3 Vision, CXP

[↖ Back to top](#)

## 5.7 Grab\_UsingActionCommand

This sample shows how to issue a GigE Vision action command to multiple cameras. By using an action command, multiple cameras can be triggered at the same time as opposed to software triggering where each camera has to be triggered individually.

To make the configuration of multiple cameras and the execution of the action commands easier, this sample uses the [ActionCommandTrigger](#) class.



### Code

The [CameraFinder](#) class provides a list of all found GigE camera devices.

The [ActionCommandTrigger](#) class provides simplified access to GigE action commands. It is used to configure the DeviceKey, GroupKey, and GroupMask parameters for cameras automatically. It also configures the camera's trigger and sets the trigger source to Action1. In addition, there are some static methods for issuing and scheduling an action command.

### Applicable Interfaces

GigE Vision

[Back to top](#)

## 5.8 Grab\_UsingBufferFactory

This sample demonstrates how to use a user-provided buffer factory.

Using a buffer factory is optional and intended for advanced use cases only. A buffer factory is only necessary if you want to grab into externally supplied buffers.



```
C:\WINDOWS\system32\cmd.exe

SizeX: 1920
SizeY: 1200
First value of pixel data: 16

SizeX: 1920
SizeY: 1200
First value of pixel data: 1553

SizeX: 1920
SizeY: 1200
First value of pixel data: 2578

SizeX: 1920
SizeY: 1200
First value of pixel data: 2313

Freed buffer 56907080.
Freed buffer 59211112.
Freed buffer 61515144.
Freed buffer 63819176.
Freed buffer 66123208.
Freed buffer 68427240.
Freed buffer 70731272.
Freed buffer 113119256.
Freed buffer 115423288.
Freed buffer 117727320.

Press enter to exit.
```

### Code

The [MyBufferFactory](#) class demonstrates how to use a user-provided buffer factory.

The buffer factory must be created before streaming is started in order to allocate the buffer memory.

Note that the .NET garbage collector automatically manages the release of allocated memory for your application.

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name or serial number.

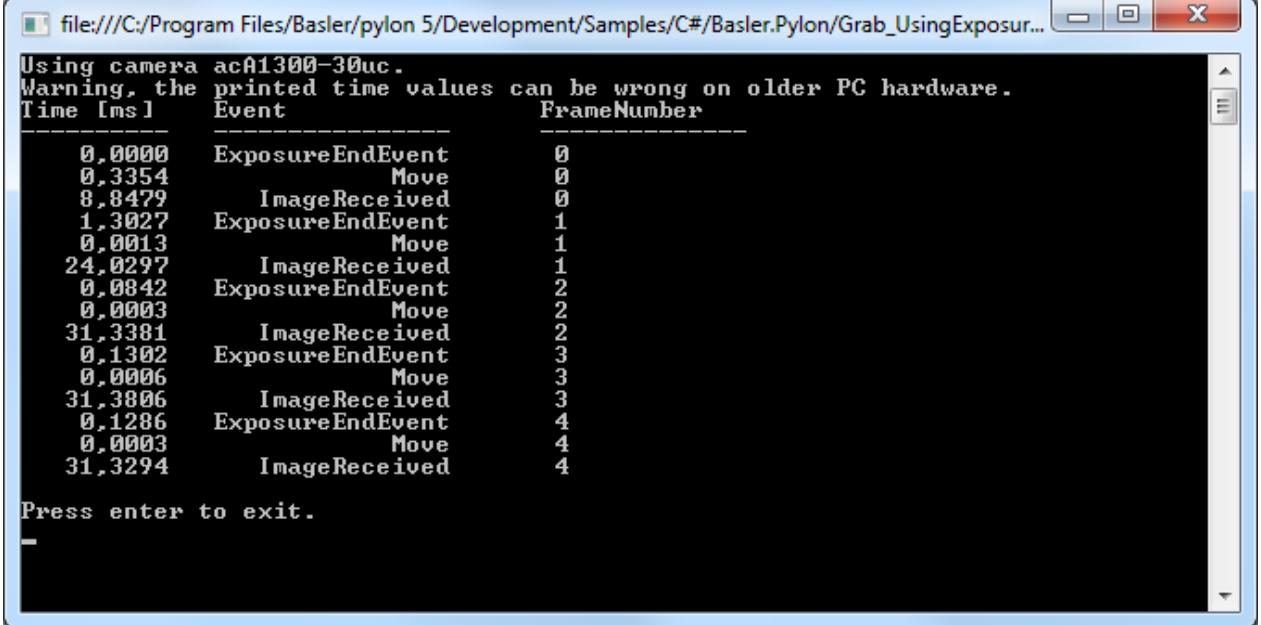
### Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[Back to top](#)

## 5.9 Grab\_UsingExposureEndEvent

This sample shows how to use the Exposure End event to speed up the image acquisition. For example, when a sensor exposure is finished, the camera can send an Exposure End event to the computer. The computer can receive the event before the image data has been completely transferred. This allows you to avoid unnecessary delays, e.g., when an imaged object is moved further before the related image data transfer is complete.



The screenshot shows a terminal window with the following text:

```
Using camera acA1300-30uc.
Warning, the printed time values can be wrong on older PC hardware.

Time [ms] Event FrameNumber
-----
```

Time [ms]	Event	FrameNumber
0,0000	ExposureEndEvent	0
0,3354	Move	0
8,8479	ImageReceived	0
1,3027	ExposureEndEvent	1
0,0013	Move	1
24,0297	ImageReceived	1
0,0842	ExposureEndEvent	2
0,0003	Move	2
31,3381	ImageReceived	2
0,1302	ExposureEndEvent	3
0,0006	Move	3
31,3806	ImageReceived	3
0,1286	ExposureEndEvent	4
0,0003	Move	4
31,3294	ImageReceived	4

Press enter to exit.

### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The helper function [Configure\(\)](#) is used to configure the camera for sending events.

The [PLCCameraInstance](#) class provides a list of all parameter names available for the Camera class instance. Here, it is used to enable event notification.

The [PLCCamera](#) class is used to configure and enable the sending of Exposure End, Event Overrun and Frame Start Overtrigger events.

In this sample, different event handlers are used to receive the grabbed image data and the camera events.

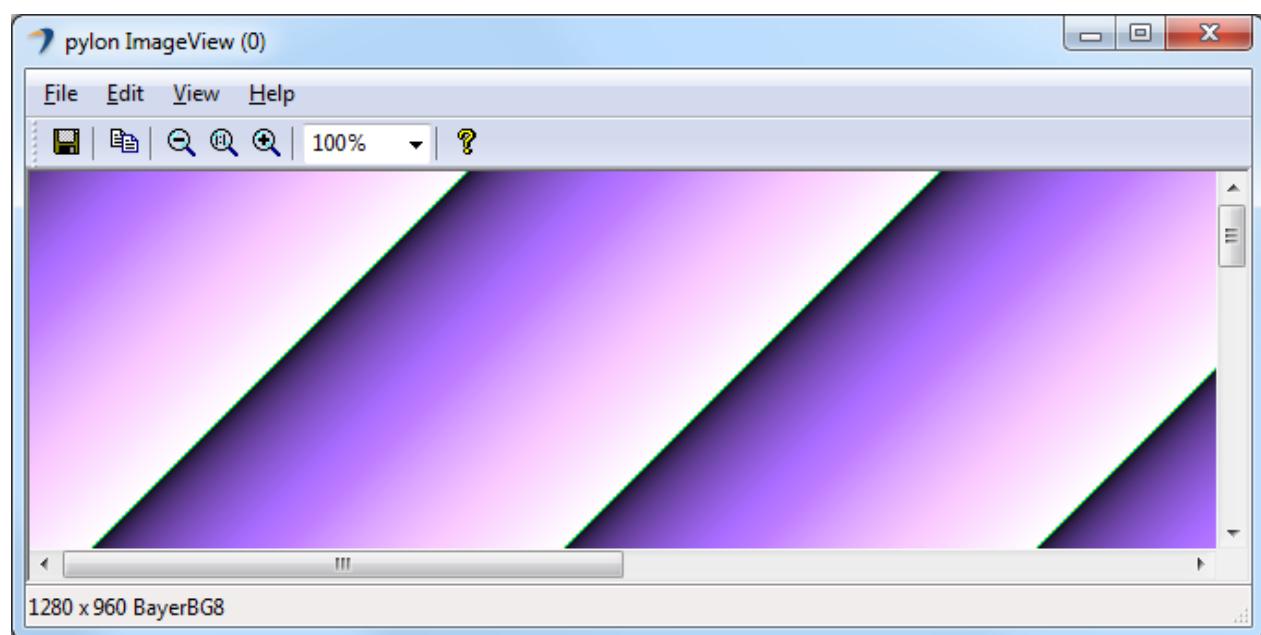
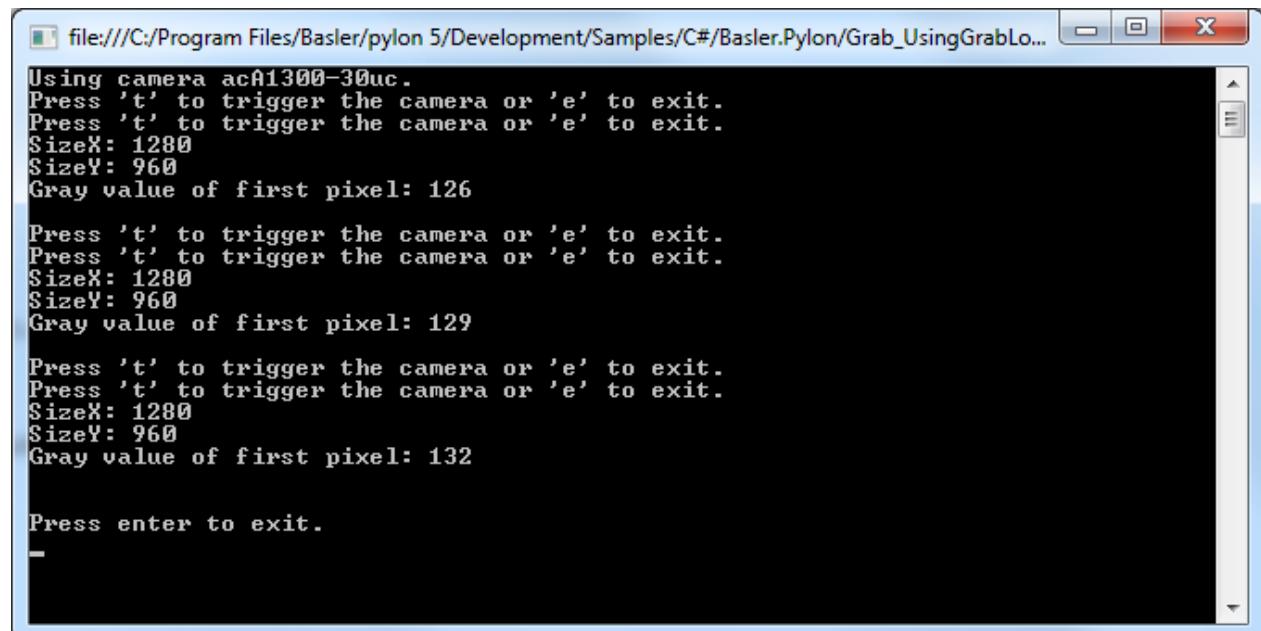
### Applicable Interfaces

GigE Vision, USB3 Vision

[Back to top](#)

## 5.10 Grab\_UsingGrabLoopThread

This sample illustrates how to grab and process images using the grab loop thread provided by the Camera class.



### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [PLCameraInstance](#) class provides a list of all parameter names available for the Camera class instance. It is used to enable the grabbing of camera events in general and control the buffer size of the output queue.

The [Configuration](#) class is used to configure the camera for software trigger mode.

Image grabbing is started by using an additional grab loop thread provided by the stream grabber. This is done by setting the grabLoopType parameter to GrabLoop.ProvidedByStreamGrabber. The grab results are delivered to the image event handler OnImageGrabbed. The default grab strategy GrabStrategy.OneByOne is used.

The [ImageWindow](#) class is used to display the grabbed image on the screen.

The [ImagePersistence](#) class is used to save the grabbed image to a Bitmap image file.

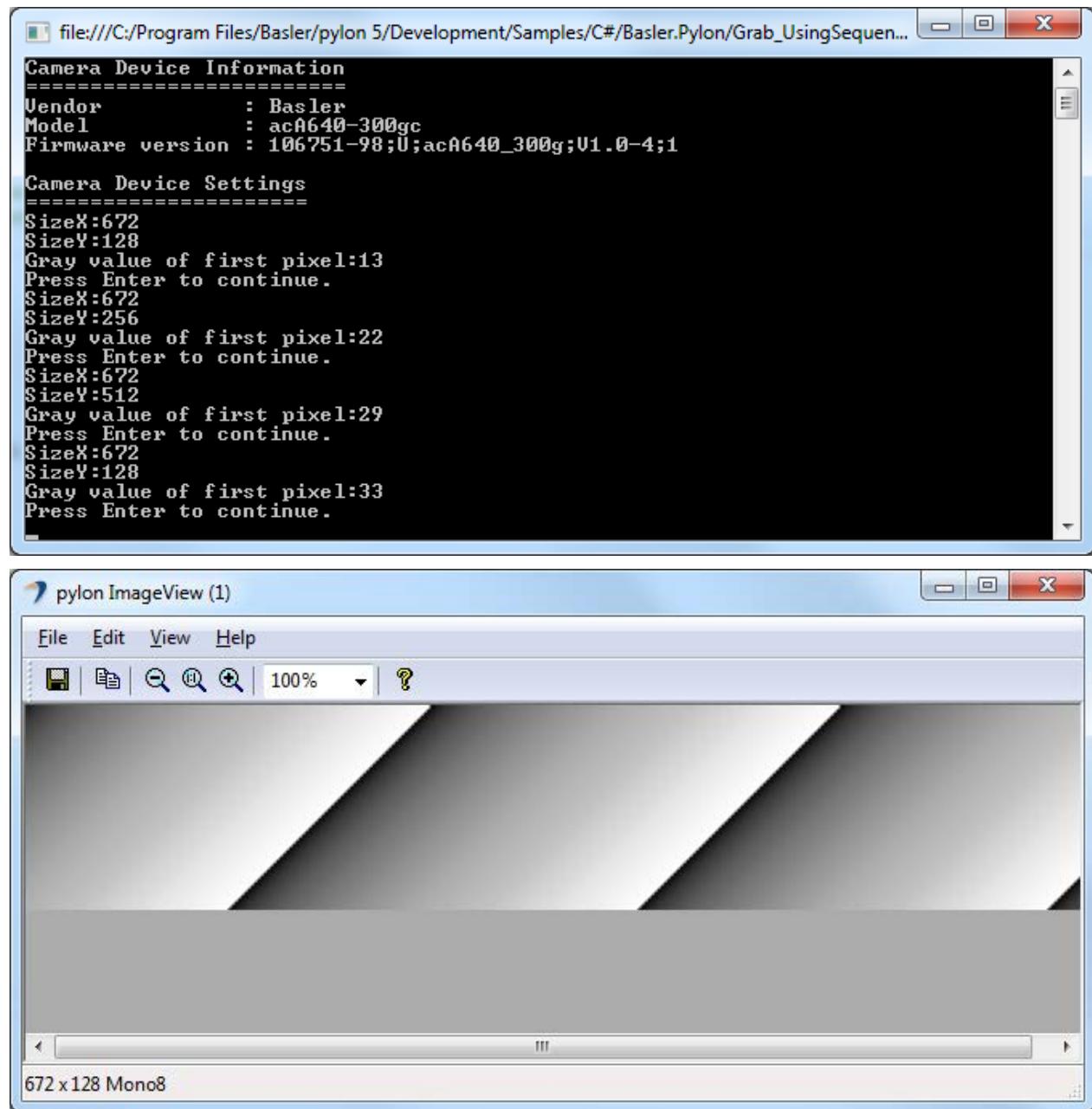
## Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[↖ Back to top](#)

## 5.11 Grab\_UsingSequencer

This sample shows how to grab images using the Sequencer feature of a camera. Three sequence sets are used for image acquisition. Each sequence set uses a different image height.



### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [Configuration](#) class is used to configure the camera for software trigger mode.

The [PLCamera](#) class is used to enable and configure the camera Sequencer feature.

The [ImageWindow](#) class is used to display the grabbed image on the screen.

## Applicable Interfaces

GigE Vision, USB3 Vision

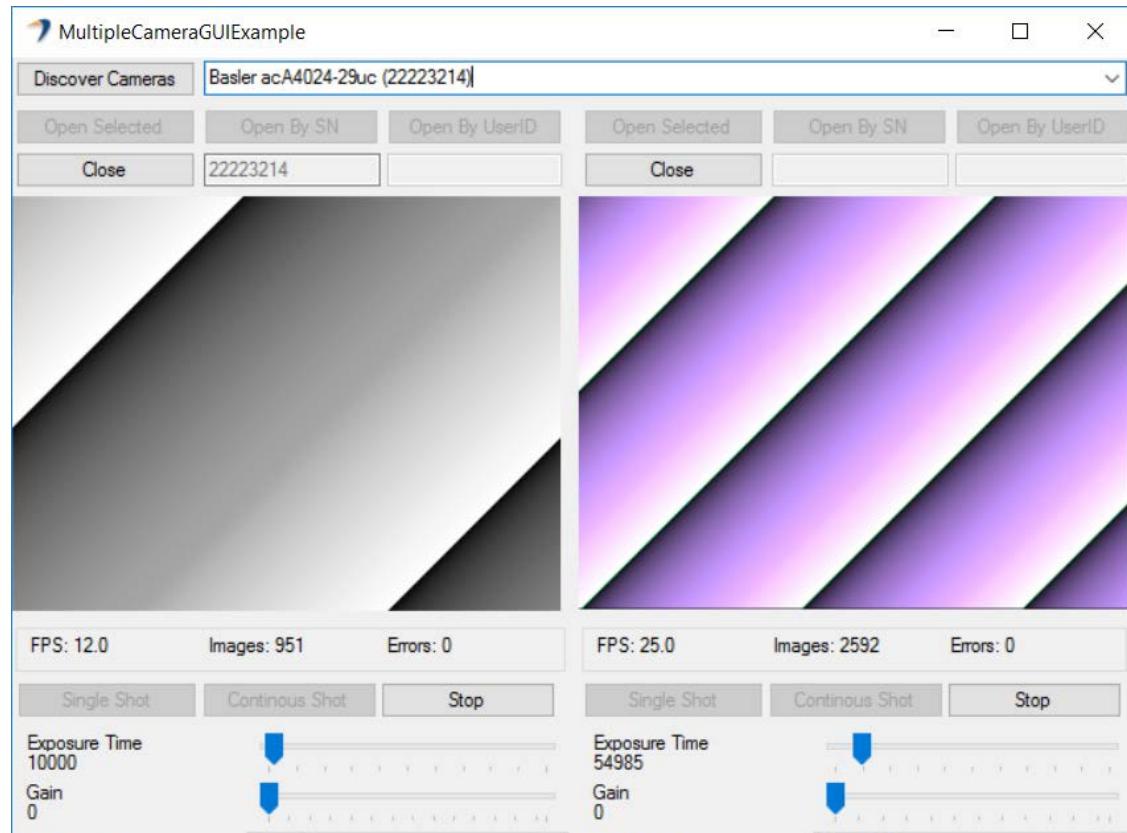
[⊜ Back to top](#)

## 5.12 GUI Sample MultiCam

This sample demonstrates how to operate multiple cameras using a Windows Forms GUI together with the pylon .NET API.

The sample demonstrates different techniques for opening a camera, e.g., by using its serial number or user device ID. It also contains an image-processing example and shows how to handle device disconnections.

The sample covers single and continuous image acquisition using software as well as hardware triggering.



### Code

When the **Discover Cameras** button is clicked, the [UpdateDeviceList\(\)](#) function in the [MainForm](#) class is called, which in turn calls the [CameraFinder.Enumerate\(\)](#) function to enumerate all attached devices.

By clicking the **Open Selected** button, the [SelectByCameraInfo\(\)](#) function is called to create a new device info object.

Then, the [OpenCamera\(\)](#) function in the [GUICamera](#) class is called to create a camera object and open the selected camera. In addition, event handlers for image grabbing and device removal are registered.

Cameras can be opened by clicking the **Open by SN** (SN = serial number) or **Open by User ID** button. The latter assumes that you have already assigned a user ID to the camera, e.g., in the pylon Viewer or via the pylon API.

After a camera has been opened, the following GUI elements become available:

- **Single Shot, Continuous Shot, Stop, and Execute** (for executing a software trigger) buttons
- **Exposure Time** and **Gain** sliders
- **PixelFormat, Trigger Mode, and Trigger Source** drop-down lists
- **Invert Pixels** check box

By clicking the **Single Shot** button, the [SingleShot\(\)](#) function is called. To grab a single image, the stream grabber [Start\(\)](#) function is called with the following arguments:

```
camera.StreamGrabber.Start(1, GrabStrategy.OneByOne, GrabLoop.ProvidedByStreamGrabber);
```

When the image is received, pylon will call the [OnImageGrabbed\(\)](#) handler and the image will be displayed.

By clicking the **Continuous Shot** button, the [ContinuousShot\(\)](#) function is called. To grab images continuously, the stream grabber [Start\(\)](#) function is called with the following arguments:

```
camera.StreamGrabber.Start(GrabStrategy.OneByOne, GrabLoop.ProvidedByStreamGrabber);
```

In this case, the camera will grab images until the stream grabber [Stop\(\)](#) function is called.

When a new image is received, pylon will call the [OnImageGrabbed\(\)](#) handler and the grabbed images will be displayed continuously.

This sample also demonstrates the triggering of cameras by using a software trigger. For this purpose, the **Trigger Mode** parameter has to be set to **On**, and the **Trigger Source** parameter has to be set to **Software**. When starting a single or a continuous image acquisition, the camera will then be waiting for a software trigger.

By clicking the **Execute** button, the [SoftwareTrigger\(\)](#) function will be called, which will execute a software trigger.

For triggering the camera by hardware trigger, set **Trigger Mode** to **On** and **Trigger Source** to, e.g., **Line1**. When starting a single or a continuous image acquisition, the camera will then be waiting for a hardware trigger.

By selecting the **Invert Pixels** check box, an example of image processing will be shown. In the example, the pixel data will be inverted. This is done in the [InvertColors\(\)](#) function, which is called from [OnImageGrabbed\(\)](#).

Finally, this sample also shows the use of Device Removal callbacks. If an already opened camera is disconnected, the [OnDeviceRemoved\(\)](#) function is called. In turn, the [OnCameraDisconnected\(\)](#) function will be called to inform the user about the disconnected camera.

## Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[Back to top](#)

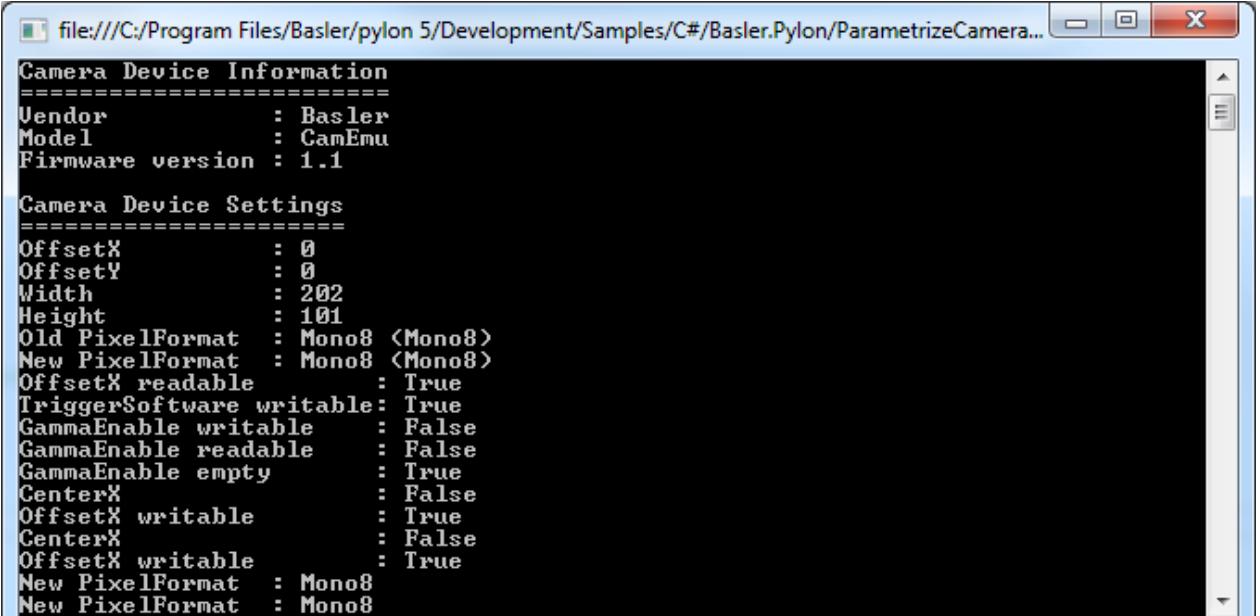
## 5.13 ParametrizeCamera

This sample illustrates how to read and write different camera parameter types.

For camera configuration and for accessing other parameters, the pylon API uses the technologies defined by the GenICam standard (<http://www.genicam.org>). The standard also defines a format for camera description files.

These files describe the configuration interface of GenICam compliant cameras. The description files are written in XML and describe camera registers, their interdependencies, and all other information needed to access high-level features. This includes features such as Gain, Exposure Time, or Pixel Format. The features are accessed by means of low level register read and write operations.

The elements of a camera description file are represented as parameter objects. For example, a parameter object can represent a single camera register, a camera parameter such as Gain, or a set of parameter values.



```
file:///C:/Program Files/Basler/pylon 5/Development/Samples/C#/Basler.Pylon/ParametrizeCamera...
=====
Camera Device Information
=====
Vendor : Basler
Model  : CamEmu
Firmware version : 1.1

Camera Device Settings
=====
OffsetX : 0
OffsetY : 0
Width   : 202
Height  : 101
Old PixelFormat : Mono8 <Mono8>
New PixelFormat : Mono8 <Mono8>
OffsetX readable : True
TriggerSoftware writable: True
GammaEnable writable : False
GammaEnable readable : False
GammaEnable empty    : True
CenterX : False
OffsetX writable   : True
CenterX : False
OffsetX writable   : True
New PixelFormat : Mono8
New PixelFormat : Mono8
```

### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [PLCamera](#) class is used to configure camera features such as Width, Height, OffsetX, OffsetY, PixelFormat, etc.

The [PLUsbCamera](#) class is used to configure features compatible with the SFNC version 2.0, e.g., the feature Gain available on USB3 Vision cameras.

### Applicable Interfaces

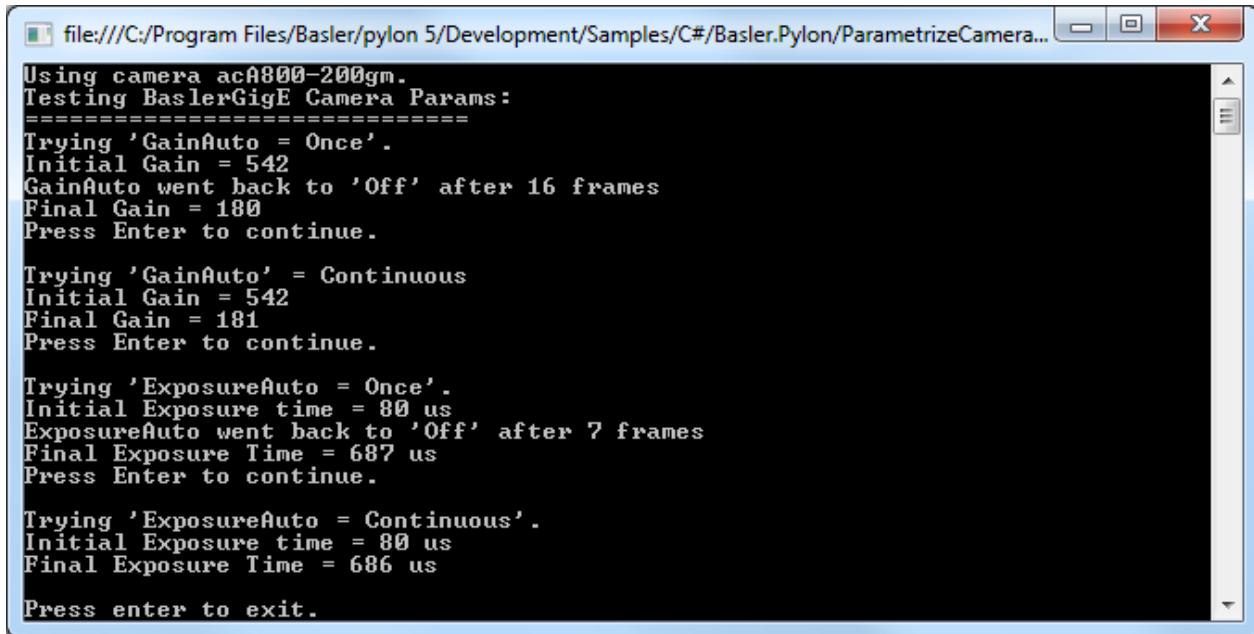
GigE Vision, USB3 Vision, Camera Link, CXP

[Back to top](#)

## 5.14 ParametrizeCamera\_AutoFunctions

This sample illustrates how to use the Auto Functions feature of Basler cameras.

**Note:** Different camera families implement different versions of the Standard Feature Naming Convention (SFNC). That's why the name and the type of the parameters used can be different.



```
file:///C:/Program Files/Basler/pylon 5/Development/Samples/C#/Basler.Pylon/ParametrizeCamera...
Using camera acA800-200gm.
Testing BaslerGigE Camera Params:
=====
Trying 'GainAuto = Once'.
Initial Gain = 542
GainAuto went back to 'Off' after 16 frames
Final Gain = 180
Press Enter to continue.

Trying 'GainAuto' = Continuous
Initial Gain = 542
Final Gain = 181
Press Enter to continue.

Trying 'ExposureAuto = Once'.
Initial Exposure time = 80 us
ExposureAuto went back to 'Off' after 7 frames
Final Exposure Time = 687 us
Press Enter to continue.

Trying 'ExposureAuto = Continuous'.
Initial Exposure time = 80 us
Final Exposure Time = 686 us

Press enter to exit.
```



### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [PLCamera](#) class is used to demonstrate the configuration of different camera features:

- [AutoGainOnce\(\)](#): Carries out luminance control by using the Gain Auto auto function in the Once operating mode.

- [AutoGainContinuous\(\)](#): Carries out luminance control by using the Gain Auto auto function in the Continuous operating mode.
- [AutoExposureOnce\(\)](#): Carries out luminance control by using the Exposure Auto auto function in the Once operating mode.
- [AutoExposureContinuous\(\)](#): Carries out luminance control by using the Exposure Auto auto function in the Continuous operating mode.
- [AutoWhiteBalance\(\)](#): Carries out white balance using the Balance White Auto auto function. **Note:** Only color cameras support this auto function.

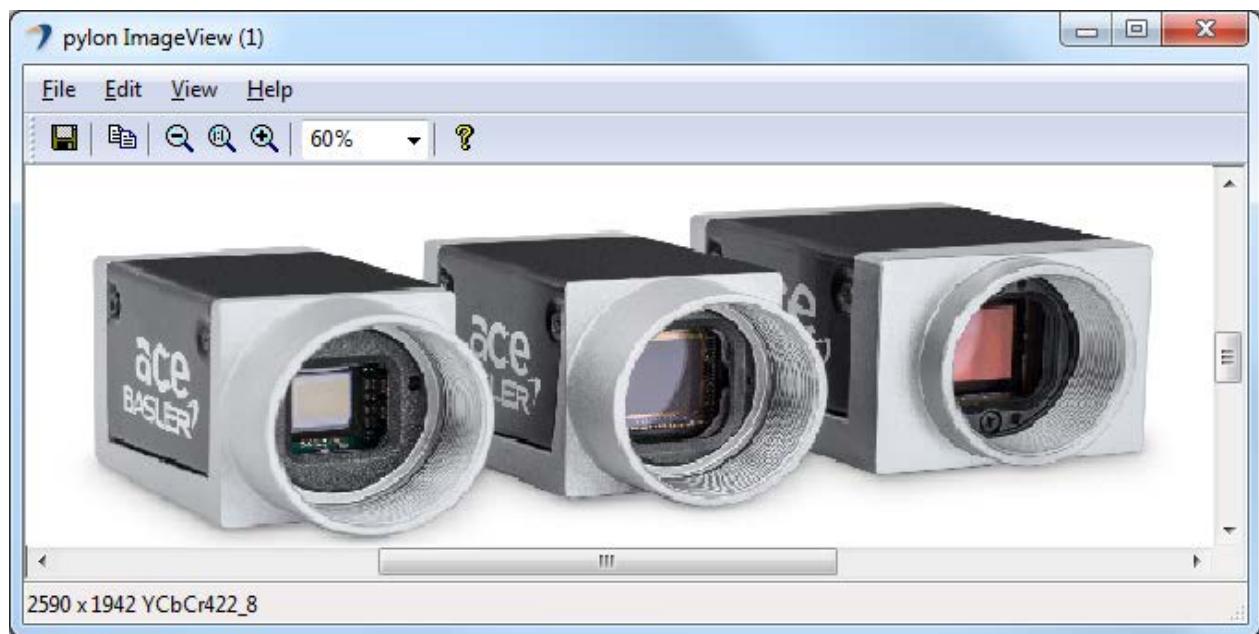
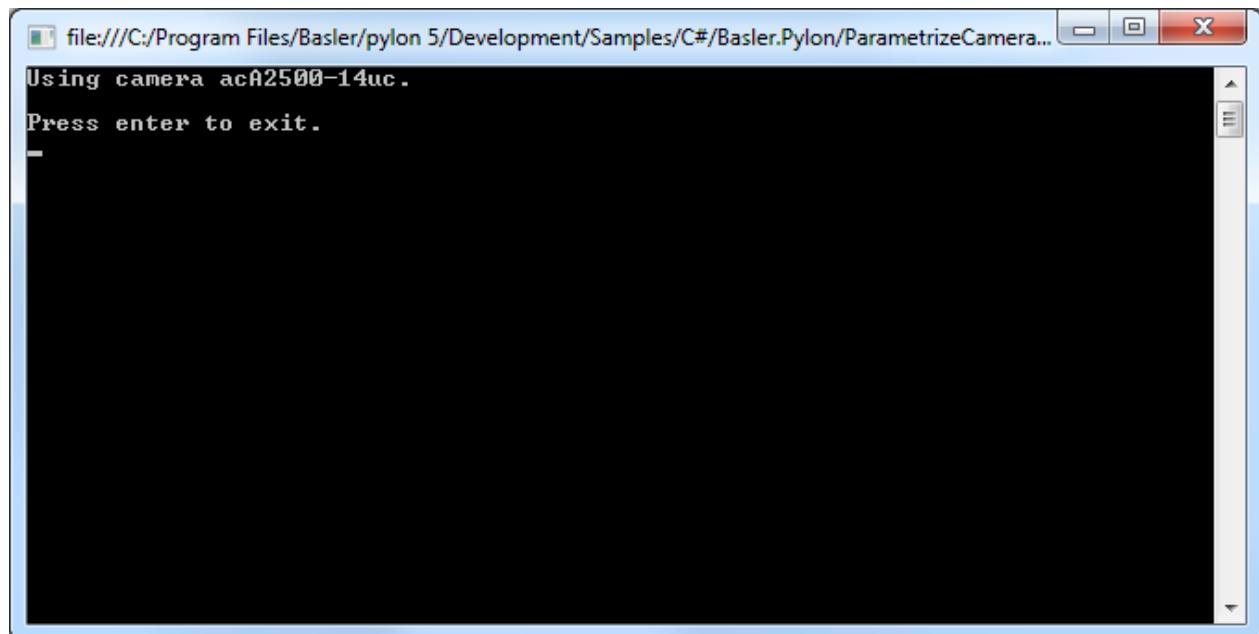
## Applicable Interfaces

GigE Vision, USB3 Vision

[↖ Back to top](#)

## 5.15 ParametrizeCamera\_AutomaticImageAdjustment

This sample illustrates how to mimic the "Automatic Image Adjustment" button of the Basler pylon Viewer.



### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [PLCamera](#) class is used to demonstrate the usage of automatic image adjustment features like GainAuto, ExposureAuto and BalanceWhiteAuto. In addition, features related to the color image quality like Gamma and LightSourcePreset are used.

The [ImageWindow](#) class is used to display the grabbed image on the screen.

## Applicable Interfaces

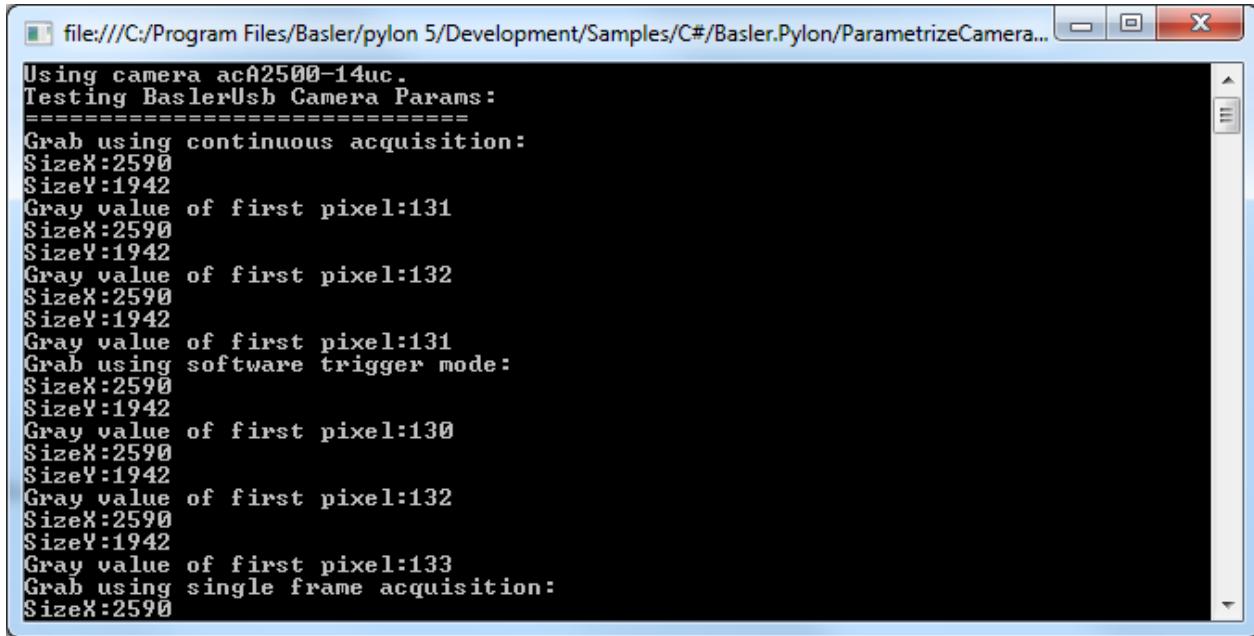
GigE Vision, USB3 Vision

[ŷ Back to top](#)

## 5.16 ParametrizeCamera\_Configurations

This sample shows how to use configuration event handlers by applying the standard configurations and registering sample configuration event handlers.

If the configuration event handler is registered, the registered methods are called when the state of the camera objects changes, e.g., when the camera object is opened or closed. In pylon.NET, a configuration event handler is a method that parametrizes the camera.



```
Using camera acA2500-14uc.
Testing BaslerUsb Camera Params:
=====
Grab using continuous acquisition:
SizeX:2590
SizeY:1942
Gray value of first pixel:131
SizeX:2590
SizeY:1942
Gray value of first pixel:132
SizeX:2590
SizeY:1942
Gray value of first pixel:131
Grab using software trigger mode:
SizeX:2590
SizeY:1942
Gray value of first pixel:130
SizeX:2590
SizeY:1942
Gray value of first pixel:132
SizeX:2590
SizeY:1942
Gray value of first pixel:133
Grab using single frame acquisition:
SizeX:2590
```

### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [Configuration](#) class is used to demonstrate the usage of different configuration event handlers.

The [Configuration.AcquireContinuous](#) handler is a standard configuration event handler to configure the camera for continuous acquisition.

The [Configuration.SoftwareTrigger](#) handler is a standard configuration event handler to configure the camera for software triggering.

The [Configuration.AcquireSingleFrame](#) handler is a standard configuration event handler to configure the camera for single frame acquisition.

The [PixelFormatAndAoiConfiguration](#) handler is a custom event handler for pixel format and area of interest configuration.

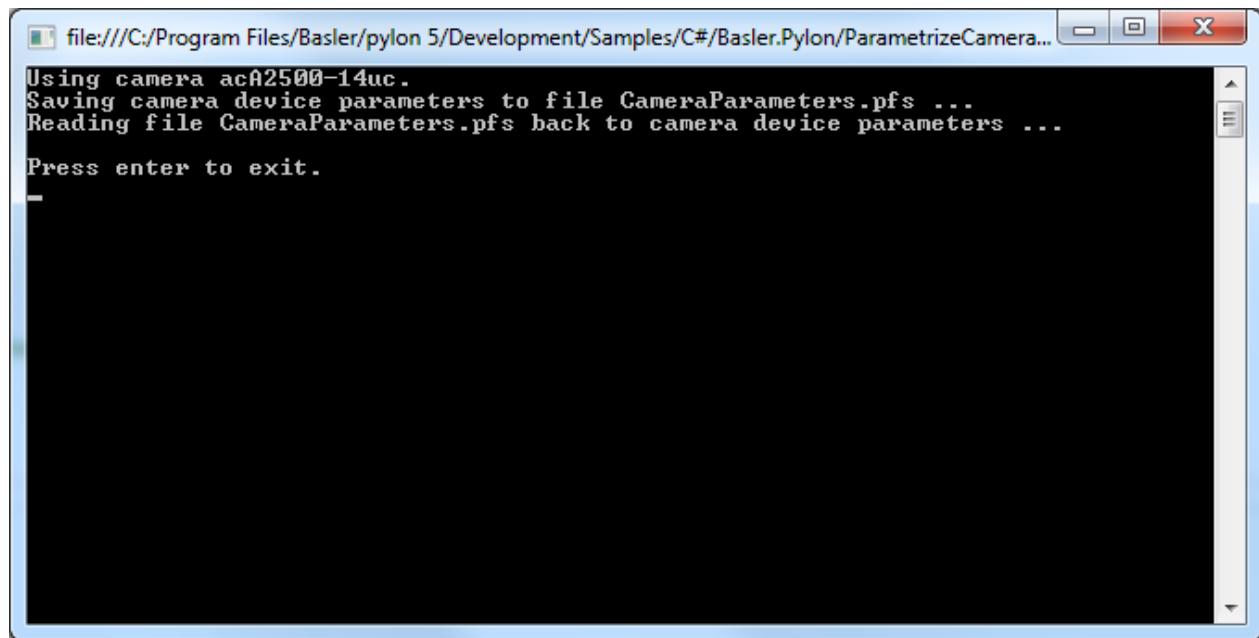
### Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[Back to top](#)

## 5.17 ParametrizeCamera\_LoadAndSave

This sample application demonstrates how to save or load the features of a camera to or from a file.



### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The interface [Parameters](#) returns a parameter collection of the camera for accessing all parameters. It is used to access the [Save\(\)](#) and the [Load\(\)](#) functions which allow saving or loading of camera parameters to or from a file. This feature can be used to transfer the configuration of a "reference" camera to other cameras.

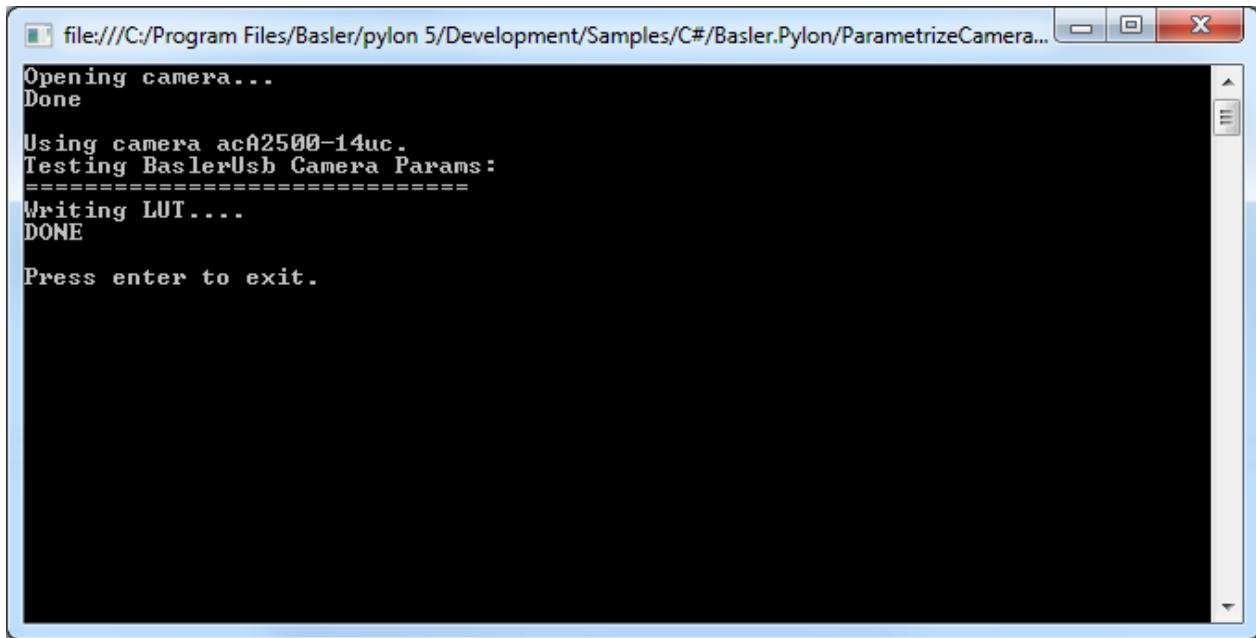
### Applicable Interfaces

GigE Vision, USB3 Vision, Camera Link

[↖ Back to top](#)

## 5.18 ParametrizeCamera\_LookupTable

This sample program demonstrates the use of the Luminance Lookup Table (LUT) feature.



### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [PLCamera](#) class is used to enable and configure all parameters related to the lookup table camera feature.

### Applicable Interfaces

GigE Vision, USB3 Vision, Camera Link

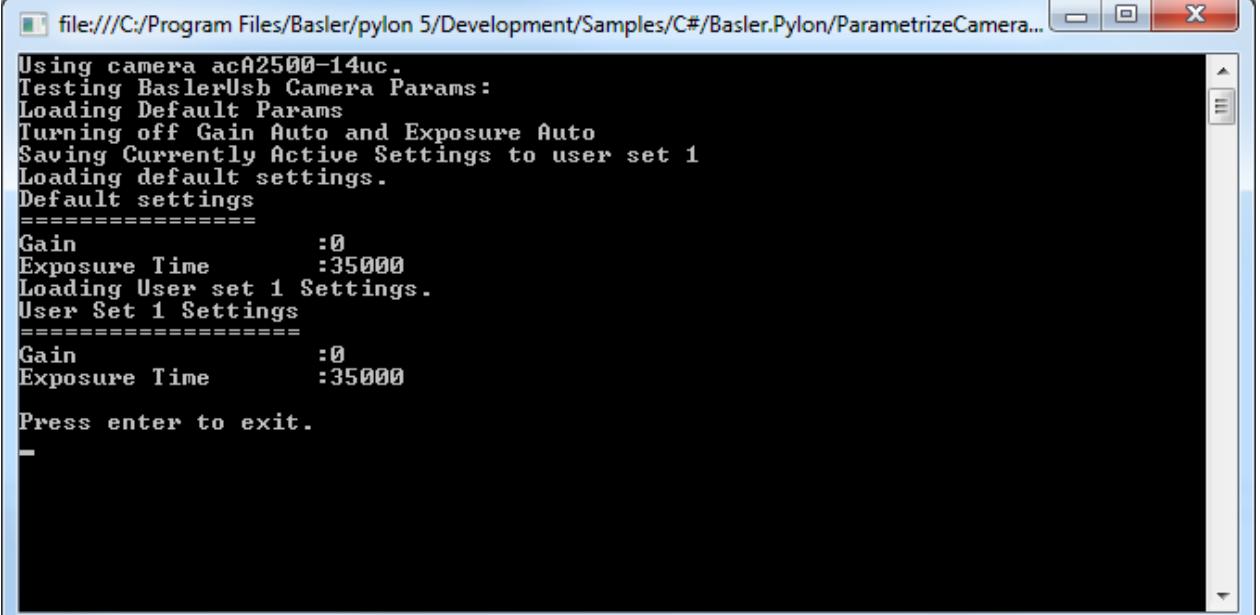
[↖ Back to top](#)

## 5.19 ParametrizeCamera\_UserSets

This sample application demonstrates how to use user sets (also called "configuration sets") and how to configure the camera to start up with the user-defined settings of user set 1.

You can also configure your camera using the pylon Viewer and store your custom settings in a user set of your choice.

**Note:** Executing this sample will overwrite all current settings in user set 1.



```
file:///C:/Program Files/Basler/pylon 5/Development/Samples/C#/Basler.Pylon/ParametrizeCamera...
Using camera acA2500-14uc.
Testing BaslerUsb Camera Params:
Loading Default Params
Turning off Gain Auto and Exposure Auto
Saving Currently Active Settings to user set 1
Loading default settings.
Default settings
=====
Gain          :0
Exposure Time :35000
Loading User set 1 Settings.
User Set 1 Settings
=====
Gain          :0
Exposure Time :35000

Press enter to exit.
-
```

### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [PLCamera](#) class is used to demonstrate the use of the camera user sets feature.

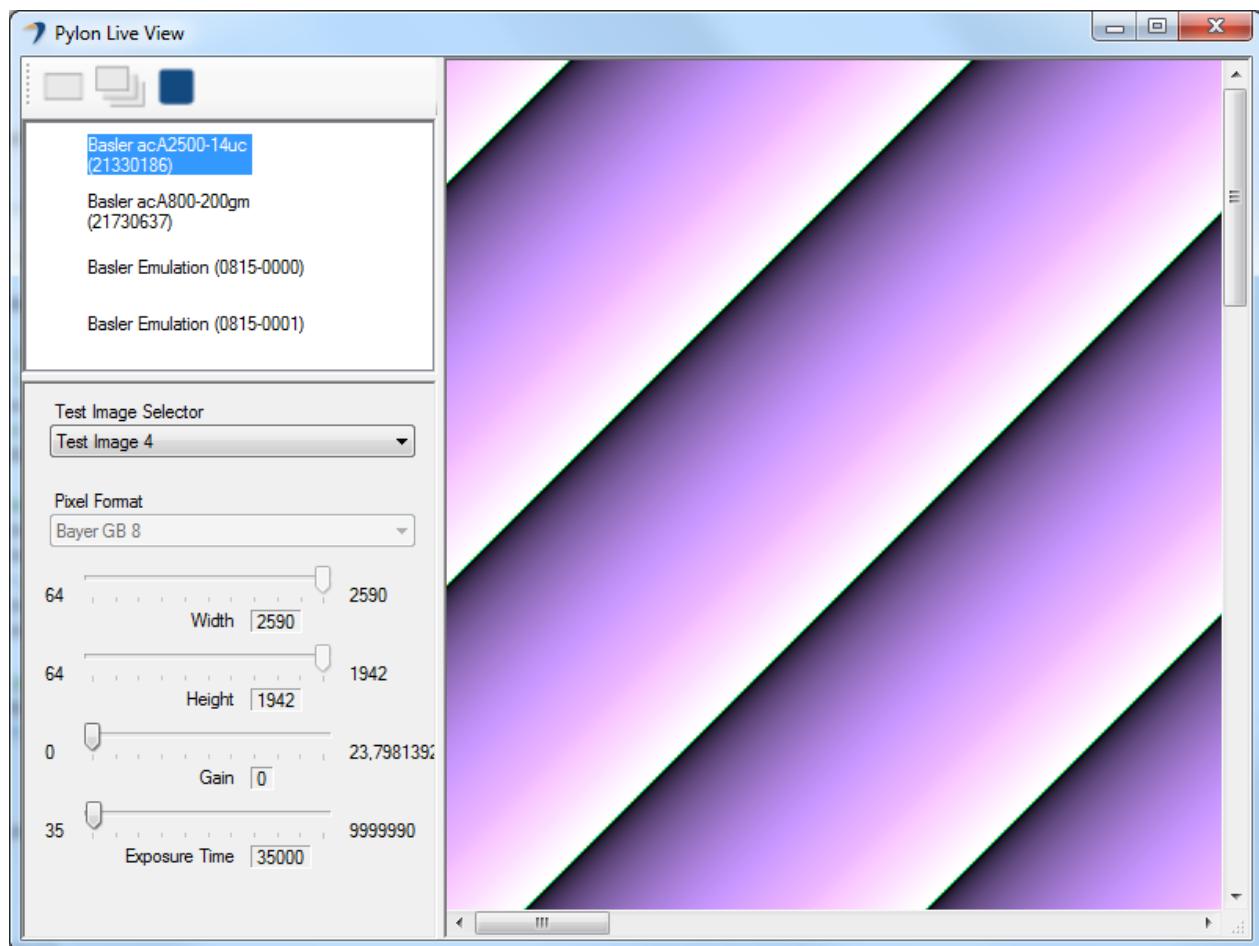
### Applicable Interfaces

GigE Vision, USB3 Vision, Camera Link, CXP

[Back to top](#)

## 5.20 PylonLiveView

This sample demonstrates the use of a GUI to enumerate attached cameras, to configure a camera, to start and stop grabbing and to display grabbed images.



### Code

The [MainForm](#) class contains the implementation of the main controls and events to be used.

When a camera device is selected in the device list, the [OnCameraOpened\(\)](#) callback is called and the camera device is opened.

When the One Shot button is clicked, the [toolStripButtonOneShot\\_Click\(\)](#) callback is called, which in turn calls [OneShot\(\)](#) to start the grabbing of one image. The [PLCamera](#) class is used to select the SingleFrame acquisition mode. The default grab strategy OneByOne is applied while an additional grab loop thread provided by the stream grabber is used.

The grab results are delivered to the image event handler [OnImageGrabbed\(\)](#).

When the Continuous Shot button is clicked, the [toolStripButtonContinuousShot\\_Click\(\)](#) callback is called, which in turn calls [ContinuousShot\(\)](#) to start the grabbing of images until grabbing is stopped. The [PLCamera](#) class is used to select the Continuous acquisition mode. The default grab strategy OneByOne is applied while an additional grab loop thread provided by the stream grabber is used.

The grab results are delivered to the image event handler [OnImageGrabbed\(\)](#).

When the Stop Grab button is clicked, the `toolStripButtonStop_Click()` callback is called, which in turn calls `Stop()` to stop the grabbing of images.

### Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[↖ Back to top](#)

## 5.21 Utility\_AnnounceRemoteDevice

This sample illustrates how to discover and work with GigE Vision cameras that are behind a router.

When a camera is behind a router, the router will prevent any broadcast device discovery messages to pass through and reach the camera. In turn, this will usually prevent the camera from being discovered by the pylon IP Configurator, the pylon Viewer, or a customer application.

The screenshot shows a terminal window with the title 'C:\Utility\_AnnounceRemoteDevice.exe'. The window displays three tables of device information:

Available Devices (at start)					
Friendly Name	MAC	IP Address	Subnet Mask	Gateway	Mode
Basler acA1920-48gc (21992935)	0030531E68E7	169.254.232.104	255.255.0.0	0.0.0.0	AutoIP

Available Devices (after AnnounceRemoteDevice)					
Friendly Name	MAC	IP Address	Subnet Mask	Gateway	Mode
Basler acA1920-48gc (21992935)	0030531E68E7	169.254.232.104	255.255.0.0	0.0.0.0	AutoIP

Available Devices (after RenounceRemoteDevice)					
Friendly Name	MAC	IP Address	Subnet Mask	Gateway	Mode
Basler acA1920-48gc (21992935)	0030531E68E7	169.254.232.104	255.255.0.0	0.0.0.0	AutoIP

Press enter to exit.

### Code

The [CameraFinder](#) class is used to discover all GigE Vision cameras that are not connected behind a router, i.e., cameras that can be accessed by broadcast device discovery messages.

The [IpConfigurator](#) class is used to access a GigE Vision camera behind a router. For that purpose, the [AnnounceRemoteDevice\(\)](#) function is used, which sends a unicast device discovery message to the specific IP address of the camera.

### Applicable Interfaces

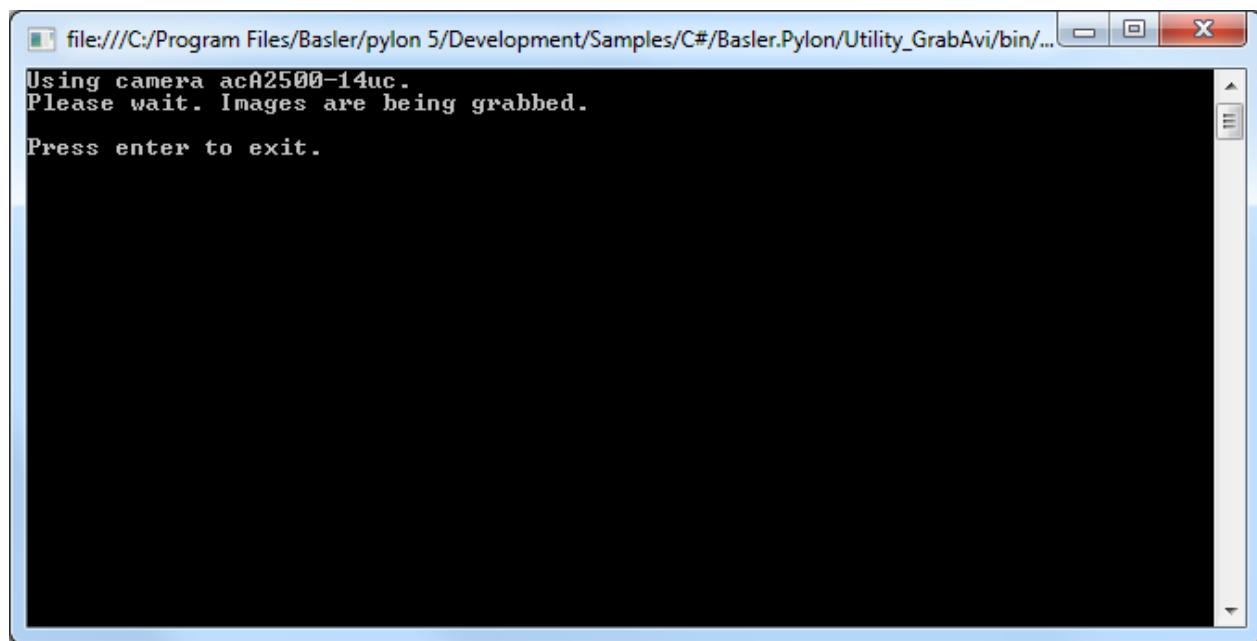
GigE Vision

[↖ Back to top](#)

## 5.22 Utility\_GrabAvi

This sample illustrates how to create a video file in Audio Video Interleave (AVI) format.

**Note:** AVI is best for recording high-quality lossless videos because it allows you to record without compression. The disadvantage is that the file size is limited to 2 GB. Once that threshold is reached, the recording stops and an error message is displayed.



### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [PLCCamera](#) class is used to set the region of interest and the pixel format of the camera.

The [PLCCameraInstance](#) class provides a list of all parameter names available for the Camera class instance. It is used to set the parameter MaxNumBuffer that controls the amount of buffers allocated for grabbing.

The [AviVideoWriter](#) class is used to create and save AVI video file to the computer's hard drive.

### Applicable Interfaces

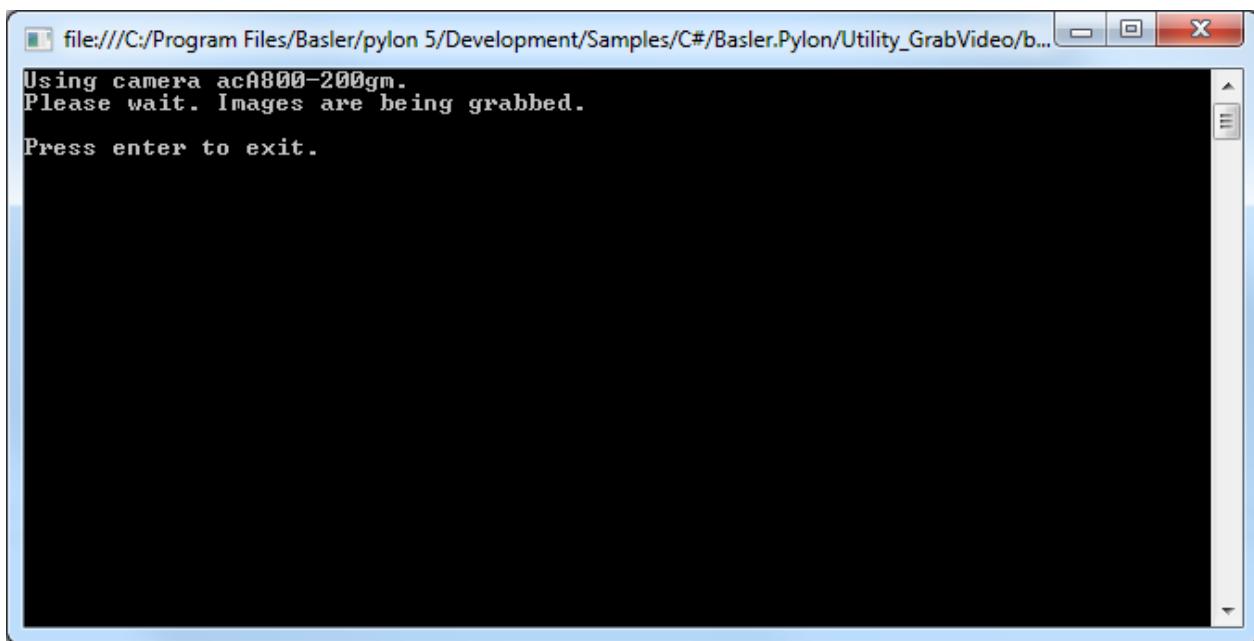
GigE Vision, USB3 Vision, CXP

[↖ Back to top](#)

## 5.23 Utility\_GrabVideo

This sample demonstrates how to create a video file in MP4 format. It is presumed that the pylon Supplementary Package for MPEG-4 is already installed.

**Note:** There are no file size restrictions when recording MP4 videos. However, the MP4 format always compresses data to a certain extent, which results in loss of detail.



### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [PLCamera](#) class is used to set the region of interest and the pixel format of the camera.

The [PLCameraInstance](#) class provides a list of all parameter names available for the Camera class instance. It is used to set the parameter MaxNumBuffer that controls the amount of buffers allocated for grabbing.

The [VideoWriter](#) class is used to create and save MP4 video file to the computer's hard drive.

The [PLVideoWriter](#) class provides a list of parameter names available for the video writer class. It is used to set the quality of the resulting compressed stream. The quality has a direct influence on the resulting bit rate. The optimal bit rate is calculated based on the input values height, width, and playback frame. This is then normalized to the quality value range 1–100, where 100 corresponds to the optimum bit rate and 1 to the lowest bit rate.

### Applicable Interfaces

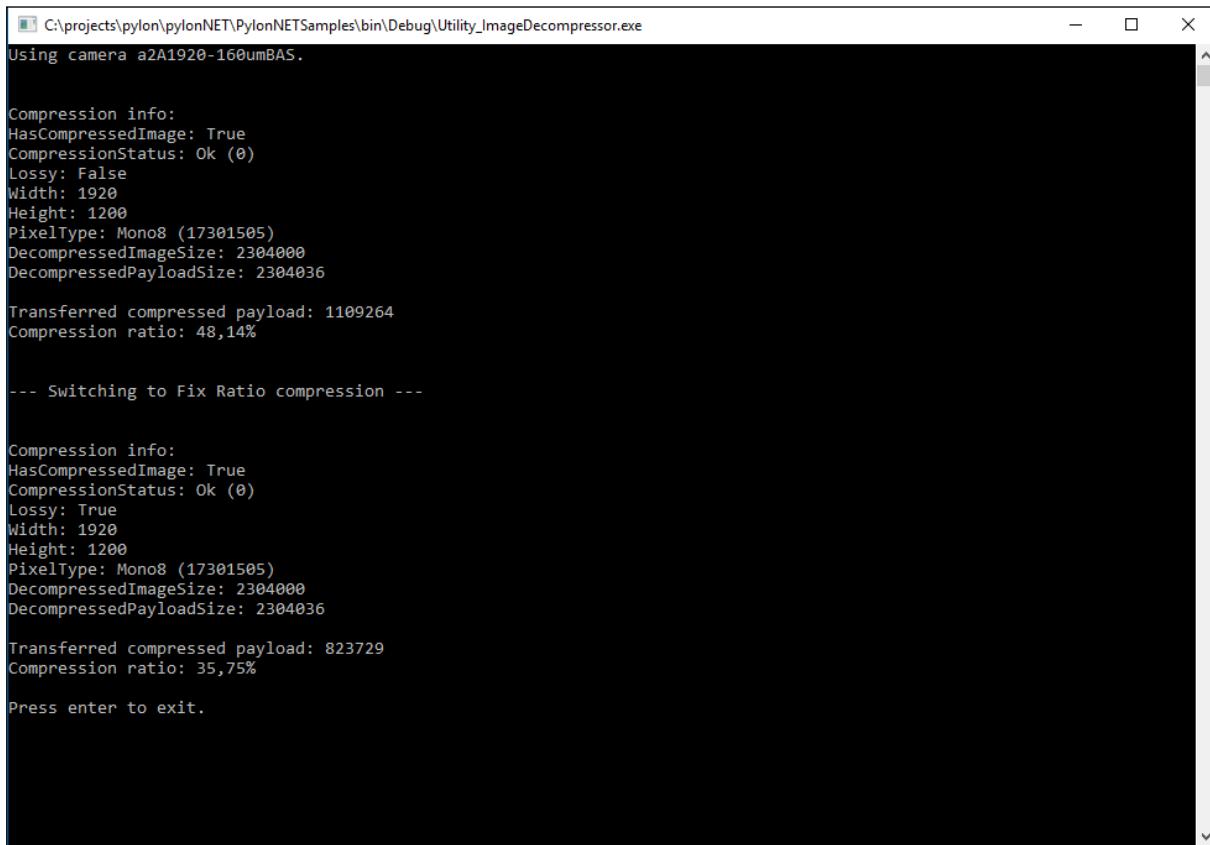
GigE Vision, USB3 Vision, CXP

[Back to top](#)

## 5.24 Utility\_ImageDecompressor

This sample illustrates how to enable and use the Basler Compression Beyond feature in Basler ace 2 GigE and Basler ace 2 USB 3.0 cameras.

This sample also demonstrates how to decompress the images using the [CImageDecompressor](#) class.



```
C:\projects\pylon\pylonNET\PylonNETSamples\bin\Debug\Utility_ImageDecompressor.exe
Using camera a2A1920-160umBAS.

Compression info:
HasCompressedImage: True
CompressionStatus: Ok (0)
Lossy: False
Width: 1920
Height: 1200
PixelFormat: Mono8 (17301505)
DecompressedImageSize: 2304000
DecompressedPayloadSize: 2304036

Transferred compressed payload: 1109264
Compression ratio: 48,14%

--- Switching to Fix Ratio compression ---

Compression info:
HasCompressedImage: True
CompressionStatus: Ok (0)
Lossy: True
Width: 1920
Height: 1200
PixelFormat: Mono8 (17301505)
DecompressedImageSize: 2304000
DecompressedPayloadSize: 2304036

Transferred compressed payload: 823729
Compression ratio: 35,75%

Press enter to exit.
```

### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [Configuration](#) class is used to set the acquisition mode to a single image acquisition when the camera is opened.

The [ImageDecompressor](#) class is used to decompress grabbed images. In this sample, compression and decompression are demonstrated, using lossless and lossy algorithms.

The [CompressionInfo](#) class is used to fetch information of a compressed image for display.

The [ImageWindow](#) class is used to display the grabbed image on the screen.

### Applicable Interfaces

GigE Vision, USB3 Vision

[Back to top](#)

## 5.25 Utility\_IpConfig

This sample demonstrates how to configure the IP address of a GigE Vision camera. The functionalities described in this sample are similar to those used in the pylon IP Configurator.

In addition, this sample can be used to automatically and programmatically configure multiple GigE Vision cameras. As the sample accepts command line arguments, it can be directly executed, e.g., from a batch script file.

The screenshot shows a Windows command-line application window titled 'C:\Utility\_IpConfig.exe'. It displays the usage information for the command-line tool:

```
Usage: Utility_IpConfig <MAC> <IP> [MASK] [GATEWAY]
<MAC> is the MAC address without separators, e.g., 0030531596CF
<IP> is one of the following:
  - AUTO to use Auto-IP (LLA).
  - DHCP to use DHCP.
  - Everything else is interpreted as a new IP address in dotted notation, e.g., 192.168.1.1.
  [MASK] is the network mask in dotted notation. This is optional. 255.255.255.0 is used as default.
  [GATEWAY] is the gateway address in dotted notation. This is optional. 0.0.0.0 is used as default.
Please note that this is a sample and no sanity checks are made.
```

Below the usage information, there is a table titled 'Available Devices' showing the configuration for three cameras:

Friendly Name	MAC	IP Address	Subnet Mask	Gateway	Mode	IP?	DHCP?	LLA?
Basler acA1600-60gm (21414863)	0030531596CF	169.254.200.200	255.255.255.0	0.0.0.0	StaticIP	True	True	True
Basler acA640-120gc (21767329)	0030531AF7A1	169.254.162.247	255.255.0.0	0.0.0.0	DHCP	True	True	True
Basler acA1920-50gc (21588208)	003053183BF0	169.254.241.59	255.255.0.0	0.0.0.0	AutoIP	True	True	True

At the bottom of the window, it says 'Press enter to exit.'

### Code

The [IpConfigurator](#) class is used to discover all GigE Vision cameras independent of their current IP address configuration. For that purpose, the [EnumerateAllDevices\(\)](#) function is used.

To set a new IP address of a GigE Vision camera, the [ChangeIpConfiguration\(\)](#) function is used.

### Applicable Interfaces

GigE Vision

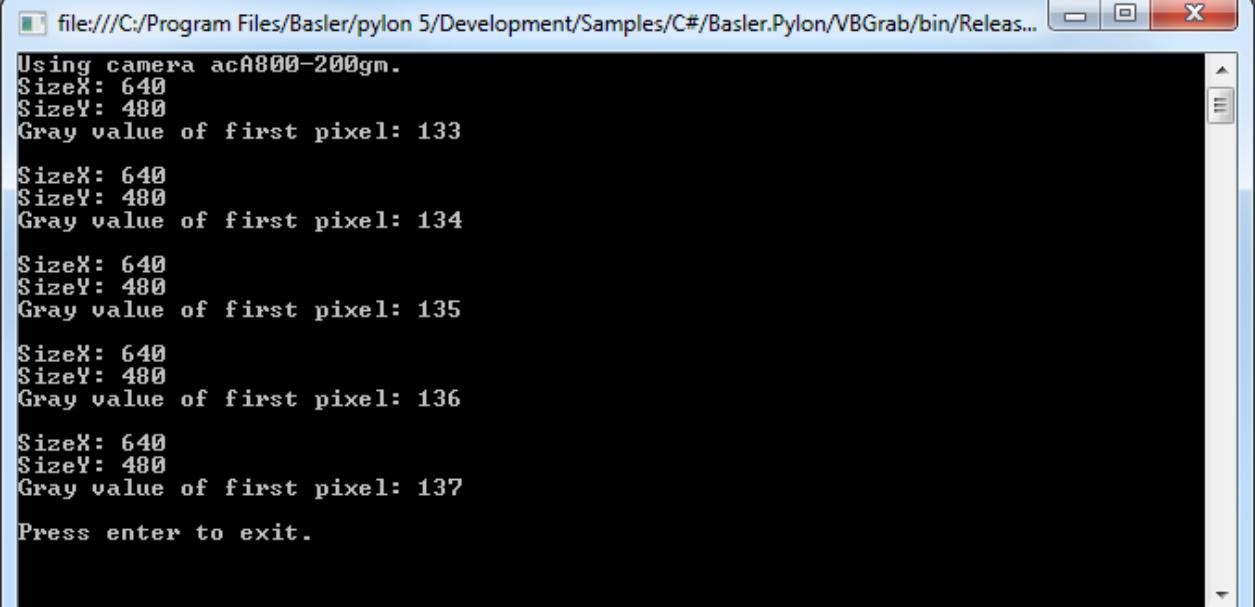
[Back to top](#)

## 5.26 VBGrab

This sample illustrates how to grab images and process images asynchronously.

This means that while the application is processing a buffer, the acquisition of the next buffer is done in parallel. The sample uses a pool of buffers. The buffers are allocated automatically. Once a buffer is filled and ready for processing, the buffer is retrieved from the stream grabber as part of a grab result.

The grab result is processed and the buffer is passed back to the stream grabber by disposing the grab result. The buffer is reused and refilled. A buffer retrieved from the stream grabber as a grab result is not overwritten in the background as long as the grab result is not disposed.



```
Using camera acA800-200gm.
SizeX: 640
SizeY: 480
Gray value of first pixel: 133

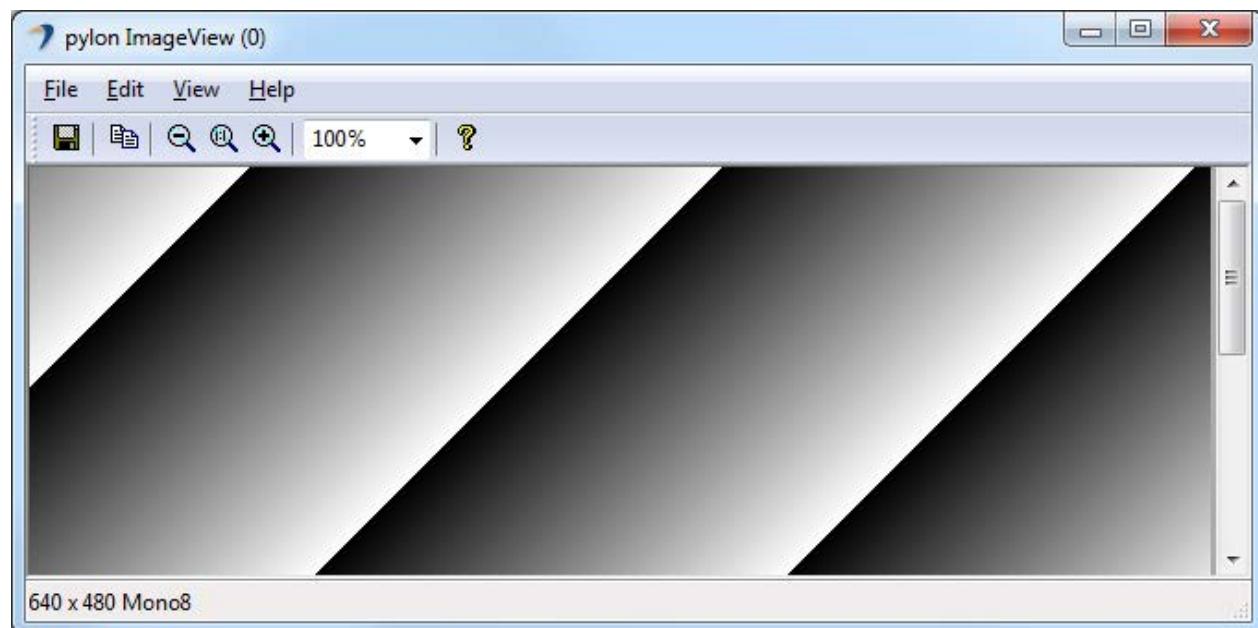
SizeX: 640
SizeY: 480
Gray value of first pixel: 134

SizeX: 640
SizeY: 480
Gray value of first pixel: 135

SizeX: 640
SizeY: 480
Gray value of first pixel: 136

SizeX: 640
SizeY: 480
Gray value of first pixel: 137

Press enter to exit.
```



## Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [Configuration](#) class is used to set the acquisition mode to free running continuous acquisition when the camera is opened.

The [PLCCameraInstance](#) class provides a list of all parameter names available for the Camera class instance. It is used to set the parameter MaxNumBuffer that controls the amount of buffers allocated for grabbing.

The [ImageWindow](#) class is used to display the grabbed image on the screen.

## Applicable Interfaces

GigE Vision, USB3 Vision, CXP

[Back to top](#)

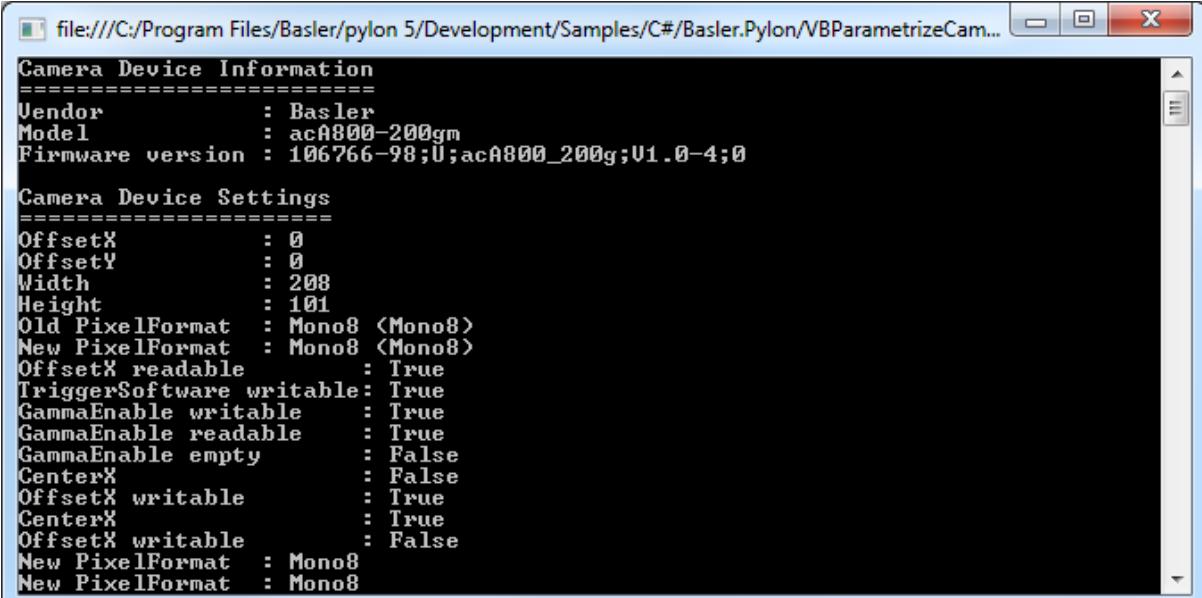
## 5.27 VBParametrizeCamera

This sample illustrates how to read and write different camera parameter types.

For camera configuration and for accessing other parameters, the pylon API uses the technologies defined by the GenICam standard (<http://www.genicam.org>). The standard also defines a format for camera description files.

These files describe the configuration interface of GenICam compliant cameras. The description files are written in XML and describe camera registers, their interdependencies, and all other information needed to access high-level features. This includes features such as Gain, Exposure Time, or Pixel Format. The features are accessed by means of low level register read and write operations.

The elements of a camera description file are represented as parameter objects. For example, a parameter object can represent a single camera register, a camera parameter such as Gain, or a set of parameter values.



The screenshot shows a window titled "Camera Device Information". The content is a list of parameters and their values:

```
Vendor : Basler
Model : acA800-200gm
Firmware version : 106766-98;U;acA800_200g;V1.0-4;0

Camera Device Settings
=====
OffsetX : 0
OffsetY : 0
Width : 208
Height : 101
Old PixelFormat : Mono8 <Mono8>
New PixelFormat : Mono8 <Mono8>
OffsetX readable : True
TriggerSoftware writable : True
GammaEnable writable : True
GammaEnable readable : True
GammaEnable empty : False
CenterX : False
OffsetX writable : True
CenterX : True
OffsetX writable : False
New PixelFormat : Mono8
New PixelFormat : Mono8
```

### Code

The [Camera](#) class is used to create a camera object that opens the first camera device found. This class also provides other constructors for selecting a specific camera device, e.g., based on the device name, or serial number.

The [PLCamera](#) class is used to demonstrate the configuration of different camera features such as Width, Height, OffsetX, OffsetY, PixelFormat, etc.

The [PLUsbCamera](#) class is used to configure features compatible with the SFNC version 2.0, e.g., the feature Gain available on USB3 Vision cameras.

### Applicable Interfaces

GigE Vision, USB3 Vision, Camera Link, CXP

[Back to top](#)

## Revision History

Document Number	Date	Changes
AW00148801000	14 Jan 2019	Initial release version of this document.
AW00148802000	20 Aug 2019	Updated to version 6.0 of the pylon Camera Software Suite. Added the Grab_UsingBufferFactory sample in the .NET Samples chapter.
AW00148803000	21 Jan 2020	Added the following samples: <ul style="list-style-type: none"><li>• Utility_ImageDecompressor</li><li>• Utility_InstantInterface</li><li>• Utility_IpConfig</li><li>• Utility_AnnounceRemoteDevice</li><li>• Utility_ImageDecompressor</li><li>• Utility_IpConfig</li></ul>
AW00148804000	2 Feb 2021	Added the GUI_SampleMultiCam sample in the C++ Samples chapter. Added the ParametrizeCamera_SerialCommunication sample in the C++ Samples chapter. Added the ImageDecompressor sample in the C samples chapter. Added the GUI_SampleMultiCam sample in the .NET Samples chapter.
AW00148805000	7 Jun 2022	Updated to version 7.x of the pylon Camera Software Suite. Added chapter 2 Data Processing API for C++ (Windows, Linux) with the following samples: <ul style="list-style-type: none"><li>• Barcode</li><li>• Camera</li><li>• Composite Data Types</li></ul>
AW00148806000	19 Jul 2022	Added the GUI_QtMultiCam sample in the C++ Samples chapter.