

Basler Components



Migrating from BCAM to Pylon 2.0

APPLICATION NOTES

Document Number: AW000717

Version: 02 Language: 000 (English)

Release Date: 30 July 2008

Contacting Basler Support Worldwide

Europe:

Basler AG
An der Strusbek 60 - 62
22926 Ahrensburg
Germany
Tel.: +49-4102-463-500
Fax.: +49-4102-463-599
bc.support.europe@baslerweb.com

Americas:

Basler, Inc.
855 Springdale Drive, Suite 160
Exton, PA 19341
U.S.A.
Tel.: +1-877-934-8472
Fax.: +1-610-280-7608
bc.support.usa@baslerweb.com

Asia:

Basler Asia Pte. Ltd
8 Boon Lay Way
03 - 03 Tradehub 21
Singapore 609964
Tel.: +65-6425-0472
Fax.: +65-6425-0473
bc.support.asia@baslerweb.com

www.baslerweb.com

**Copyright 2008 Basler Vision Technologies.
All material in this publication is subject to change without notice.**

1 Introduction

When starting to write a new application for a Basler IIDC 1394a/b camera, you must choose between the "classic" BCAM API or the new pylon V2.0 API. Because it is GenICam compliant, because it provides user friendly access to the camera's Smart Features, and because it supports both Basler IEEE 1394 and Basler GigE Vision cameras, we strongly recommend using the newer pylon V2.0 API.

Also, if you are already using the BCAM API, it might become necessary for you to migrate an existing application from BCAM to pylon (for example, because your application should also be able to handle GigE Vision cameras).

This migration guide explains which BCAM classes and functions correlate with which pylon V2.0 classes and functions in order to simplify the migration process.

The basic architecture of both APIs is quite similar. If you want to grab images from the camera, you must proceed as follows:

- Enumerate camera devices/create a camera object
- Open the camera
- Configure the camera (shutter, gain, brightness, ...)
- Create and allocate image buffers
- Queue in the buffers
- Start exposure
- Wait for the image buffer to be returned
- Close camera

Both APIs are available as a complete SDK with many examples and with documentation that details how to build a BCAM or pylon based application, what libraries to link, etc.

The information on the following pages is just an abstract listing of classes/methods that are somehow related (when grabbing images asynchronously) and is not intended to supersede the official API documentation.

In contrast to BCAM, which supports IIDC1394 cameras only, with the pylon API you must decide whether you will write code with the "native" C++ API (for an IIDC1394 camera or a GigEVision camera) or a "generic" C++ application that should be able to handle both types of camera.

"Native" in this sense means, that the API provides C++ getter methods and setter methods for accessing the camera's features directly. For example, setting the camera's color mode with:

```
pCamera->PixelFormat.SetValue (PixelFormat_Mono8);
```

This is a comfortable approach, but in this case the type of pCamera must be an instance of a native C++ class for, for example, a GigEVision camera (CBaslerGigECamera) or a IIDC1394 camera (CBasler1394Camera). This means that you must decide what kind of camera interface you would like to deal with at compile time.

"Generic" means, that you must access the camera's features via a so called node map, which is a list of the available camera features created dynamically at run time. In this case, you must ask the node map for a given feature by its name such as "PixelFormat". For example, setting the camera's color mode would look like this:

```
// get Node Map for device configuration
INodeMap *pCameraNodeMap = pCamera->GetNodeMap ();

// set color mode
CEnumerationPtr ptrPixelFormat = pCameraNodeMap->GetNode ("PixelFormat");
ptrPixelFormat->FromString ("Mono8");
```

This is less comfortable, but the advantage is that the type of pCamera is not bound to a certain interface such as GigE Vision or IIDC1394. Rather it can handle all possible GenICam compliant camera interfaces, and you can decide at run time to what kind of camera interface pCamera refers. Because the camera features are obtained at runtime, you can also access camera features that are not yet accessible via native C++ getter methods and setter methods.

2 Comparison Tables

The following tables provide a side-by-side comparison of the steps required with the pylon API vs. the BCAM API to operate a camera.

1. Namespaces

BCAM	Pylon
<code>namespace Bcam</code>	<p>Pylon Native</p> <pre>namespace Pylon // for IIDC1394 aware applications namespace Basler_IIDC1394CameraParams namespace Basler_IIDC1394StreamParams // for GigEVision aware applications namespace Basler_GigECameraParams namespace Basler_GigEStreamParams</pre> <p>Pylon Generic</p> <pre>namespace Pylon namespace GenApi namespace GenICam</pre>

2. Camera class declarations

BCAM	Pylon
<pre>class CBcam</pre>	<p>Pylon Native</p> <pre>// for IIDC1394 aware applications class CBasler1394Camera class CBasler1394Camera::StreamGrabber_t // for GigEVision aware applications class CBaslerGigECamera class CBaslerGigECamera::StreamGrabber_t</pre> <p>Pylon Generic</p> <pre>class IPylonDevice class IStreamGrabber</pre>

3. Enumerate devices - create a camera object - open the camera

BCAM	Pylon
<pre> CBcam bcam; // Get the device name of the first camera CString DeviceName = *(CBcam::DeviceNames ().begin ()); // Open the driver bcam.Open (DeviceName); </pre>	<p>Pylon Native</p> <pre> CBasler1394Camera *pCamera; CBasler1394Camera::StreamGrabber_t *pStreamGrabber // for GigE it must be // CBaslerGigECamera *pCamera; // CBaslerGigECamera::StreamGrabber_t *pStreamGrabber // Automatically call PylonInitialize and PylonTerminate to // ensure the pylon runtime system is initialized during // lifetime of this object. Pylon::PylonAutoInitTerm autoInitTerm; // Get the transport layer factory CTlFactory &TlFactory = CTlFactory::GetInstance (); // Create the transport layer object needed to enumerate or // create a camera object ITransportLayer *pTl = TlFactory.CreateTl (Camera_t::DeviceClass ()); // Get all attached cameras DeviceInfoList_t devices; pTl->EnumerateDevices (devices); // Create the camera object of the first available camera // The camera object is used to set and get all available // camera features. pCamera = pTl->CreateDevice (devices[0]); // Get the first stream grabber object of the selected camera pStreamGrabber = pCamera->GetStreamGrabber (0); // Open the camera pCamera->Open (); // Open the stream grabber pStreamGrabber->Open (); </pre>

Pylon Generic

```
IPylonDevice *pCamera;
IStreamGrabber *pStreamGrabber

// Automatically call PylonInitialize and PylonTerminate to
// ensure the pylon runtime system is initialized during
// lifetime of this object.
Pylon::PylonAutoInitTerm autoInitTerm;

// Create the transport layer object needed to enumerate or
// create a camera object
CTlFactory *pTlFactory = &CTlFactory::GetInstance ();

// Get all attached cameras
DeviceInfoList_t devices;
pTlFactory->EnumerateDevices (devices);

// Create the camera object of the first available camera
// The camera object is used to set and get all available
// camera features.
pCamera = pTlFactory->CreateDevice (devices[0]);

// Get the first stream grabber object of the selected camera
pStreamGrabber = pCamera->GetStreamGrabber (0);

// Open the camera
pCamera->Open ();

// Open the stream grabber
pStreamGrabber->Open ();
```

4. Configure the camera

BCAM	Pylon
<pre> // Parameter set for the fastest monochrome format bcam.SetVideoMode (DCS_Format7, DCS_Mode0); bcam.FormatSeven[DCS_Mode0].ColorCoding = DCSColor_Mono8; // Set Area Of Interest to maximum aoiSize = m_Bcam.FormatSeven[DCS_Mode0].MaxSize(); bcam.FormatSeven[DCS_Mode0].Position = CPoint (0, 0); bcam.FormatSeven[DCS_Mode0].Size = aoiSize; </pre>	<p>Pylon Native</p> <pre> // Set image format pCamera->PixelFormat.SetValue (PixelFormat_Mono8); pCamera->OffsetX.SetValue (0); pCamera->OffsetY.SetValue (0); pCamera->Width.SetValue (pCamera->Width.GetMax ()); pCamera->Height.SetValue (pCamera->Height.GetMax ()); </pre> <p>Pylon Generic</p> <pre> // get Node Map for device configuration INodeMap *pCameraNodeMap= pCamera->GetNodeMap(); // set color mode CEnumerationPtr ptrPixelFormat= pCameraNodeMap->GetNode ("PixelFormat"); ptrPixelFormat->FromString ("Mono8"); // Set AOI CIntegerPtr ptrOffsetX = pCameraNodeMap->GetNode ("OffsetX"); CIntegerPtr ptrOffsetY = pCameraNodeMap->GetNode ("OffsetY"); CIntegerPtr ptrAoiWidth = pCameraNodeMap->GetNode ("Width"); CIntegerPtr ptrAoiHeight = pCameraNodeMap->GetNode ("Height"); ptrOffsetX->SetValue (0); ptrOffsetY->SetValue (0); ptrAoiWidth->SetValue (ptrAoiWidth->GetMax ()); ptrAoiHeight->SetValue (ptrAoiHeight->GetMax ()); </pre>

5. Create and allocate image buffer

BCAM	Pylon
<pre> // Get the image buffer size imageSize = bcam.FormatSeven[VideoMode].BytePerFrame (); // Set full speed unsigned long BytePerPacketMax = bcam.FormatSeven[VideoMode].BytePerPacket.Max (); bcam.FormatSeven[VideoMode].BytePerPacket = BytePerPacketMax; // create a buffer unsigned char *pImageBuffer = new unsigned char [imageSize]; // Allocate Resources bcam.AllocateResources (1, imageSize); </pre>	<p>Pylon Native</p> <pre> // Get the image buffer size const size_t imageSize= (size_t) (pCamera->PayloadSize.GetValue()); // We won't use image buffers greater than ImageSize pStreamGrabber->MaxBufferSize.SetValue(imageSize); // We won't queue more than one image buffer at a time pStreamGrabber->MaxNumBuffer.SetValue(1); // Allocate all resources for grabbing. Critical parameters // like image size now must not be changed until FinishGrab() // is called. pStreamGrabber->PrepareGrab(); // Buffers used for grabbing must be registered at the stream // grabber. The Registration returns a handle to be used for // queuing the buffer create a buffer uchar8_t *pImageBuffer= new uchar8_t [imageSize]; const StreamBufferHandle hBuffer= pStreamGrabber->RegisterBuffer (pImageBuffer, imageSize); </pre>

Pylon Generic

```
// Get the image buffer size
CIntegerPtr ptrPayloadSize=
    pCameraNodeMap->GetNode ("PayloadSize");
const size_t imageSize = (size_t) (ptrPayloadSize->GetValue ());

// get Node Map for streamgrabber configuration
INodeMap *pGrabberNodeMap= pStreamGrabber->GetNodeMap ();

// We won't use image buffers greater than ImageSize
CIntegerPtr ptrMaxBufferSize
    = pGrabberNodeMap->GetNode ("MaxBufferSize");
ptrMaxBufferSize->SetValue(imageSize);

// We won't queue more than one image buffer at a time
CIntegerPtr ptrMaxNumBuffer
    = pGrabberNodeMap->GetNode("MaxNumBuffer");
ptrMaxNumBuffer->SetValue (1);

// Allocate all resources for grabbing. Critical parameters
// like image size now must not be changed until FinishGrab()
// is called.
pStreamGrabber->PrepareGrab ();

// Buffers used for grabbing must be registered at the stream
// grabber. The Registration returns a handle to be used for
// queuing the buffer create a buffer
uchar8_t *pImageBuffer = new uchar8_t [imageSize];
const StreamBufferHandle hBuffer =
    pStreamGrabber->RegisterBuffer (pImageBuffer, imageSize);
```

6. Queue in the buffer - start exposure

BCAM	Pylon
<pre data-bbox="205 352 968 480"> // Enqueue Grab command, use ContinuousShot for grabbing bcam.GrabImageAsync (pImageBuffer ,imageSize, 0, false); // Switch on the camera bcam.ContinuousShot= true; </pre>	<pre data-bbox="1018 352 1871 608"> Pylon Native // Enqueue buffer pStreamGrabber->QueueBuffer (hBuffer, NULL); // use ContinuousShot for grabbing pCamera->AcquisitionMode.SetValue (AcquisitionMode_Continuous); // Start acquisition on the camera pCamera->AcquisitionStart.Execute(); </pre> <pre data-bbox="1018 632 1696 1023"> Pylon Generic // Enqueue buffer pStreamGrabber->QueueBuffer (hBuffer, NULL); // use ContinuousShot for grabbing CEnumerationPtr ptrAcquisitionMode= pCameraNodeMap->GetNode ("AcquisitionMode"); ptrAcquisitionMode->FromString ("Continuous"); // Start acquisition on the camera CCommandPtr ptrAcquisitionStart = pCameraNodeMap->GetNode ("AcquisitionStart"); ptrAcquisitionStart->Execute (); </pre>

7. Wait for the image buffer to be returned

BCAM	Pylon
<pre>FunctionCode_t FunctionCode; void *pContext; // Wait (forever) for things to show up at the driver's //completion port e.g. until a buffer has been exposed bcam.WaitForCompletion (&FunctionCode, &ErrorCode, &pContext, INFINITE);</pre>	<p>Pylon Native/Generic</p> <pre>void *pContext; // grab result GrabResult result; // Wait (forever) for things to show up at the driver's //completion port e.g. until a buffer has been exposed pStreamGrabber->GetWaitObject().Wait (INFINITE); pStreamGrabber->RetrieveResult (result); pContext = result.Context ();</pre>

5. Close the camera

BCAM	Pylon
<pre> // switch off camera bcam.ContinuousShot = false; // Cancel pending I/O requests. bcam.Cancel(); // Release bandwidth, isoch channels and isoch transfer // contexts bcam.FreeResources(); // Close driver bcam.Close(); </pre>	<pre> Pylon Native/Generic // It is necessary to deregister buffer before freeing the memory // hBuffer is of type StreamBufferHandle pStreamGrabber->DeregisterBuffer (hBuffer); // Free all resources used for grabbing pStreamGrabber->FinishGrab (); // Close stream grabber pStreamGrabber->Close (); // Close camera pCamera->Close (); Pylon Native // Delete camera object // pTl is of type ITransportLayer // Not required! // This is done by PylonAutoInitTerm Pylon Generic // Delete camera object // pTlFactory is of type CTlFactory pTlFactory->DestroyDevice (m_pCamera); </pre>

Revision History

Doc. ID Number	Date	Changes
AW00071701000	24 Jul 2008	Initial release of this document.
AW00071702000	30 Jul 2008	Updated contact addresses and phone numbers.

