Interfacing SWIR Vision Systems Devices to Custom Applications

Important Notes Before You Begin

This application note is provided under the assumption that the user understands the USB3 Vision and GigE Vision standards. SWIR Vision Systems devices are USB3 Vision and GigE Vision compliant. This makes it simple to control image streaming and camera features when using third-party software or libraries that support USB3 Vision and GigE Vision devices. SWIR Vision Systems devices utilize a Pleora Technologies (NTX-GigE or NTX-U3) embedded video interface. The Pleora eBUS SDK is installed with the Pleora drivers when SWIR Vision Systems' SVSImagIR software is installed. For most custom applications, the eBUS SDK is the best way to control communicate with the device. The documentation that was used to create this tutorial can be found at: **C:\Program Files\Pleora Technologies Inc\eBUS SDK**. The eBUS SDK includes very useful example projects that are written in C++ and C#.

First Operation of Camera

Before developing your own custom application, reference the **Acuros 1.1 Quick Start Guide** to install the appropriate SWIR Vision Systems software, power up the camera, and perform the first operation of camera. You should understand the software and image controls as described in the **Acuros 1.1 User Manual**. First, we recommend learning to use your product in the SVSImagIR software. Connecting to SVSImagIR is a simple way to ensure that your device is supplied with power and properly connected to the PC. The SVSImagIR software should be closed while running any custom application that utilizes the eBUS SDK.

This Tutorial

This tutorial describes how to utilize Pleora's eBUS SDK in a C++ project to:

- **Connect to the camera**
- **Control GenICam features**
- **Get image data**

Pleora provides an example project called **ImageProcessing** that covers all three of these topics. The **ImageProcessing** project is discussed in this guide. The example project can be found at:
C:\Program Files\Pleora Technologies Inc\eBUS SDK\Samples\ImageProcessing. Pleora provides excellent documentation for the eBUS SDK. All of the Pleora classes and functions that are referenced in this tutorial are described in detail at:
C:\Program Files\Pleora Technologies Inc\eBUS SDK\Documentation\eBUS SDK C++ API.chm.

****The following guide should serve only as a supplement to a thorough study of the eBUS SDK example projects and documentation.****

Connecting to the Camera

The code at the top of the **ImageProcessing()** function demonstrates how to use the eBUS SDK to connect to a Pleora interface and create a **PvDevice** object (Fig. 1). Once connected , the **PvDevice** object can be used to get parameters from the camera and modify camera parameters. All camera functionality originates from the **PvDevice** object or an object that stems from the **PvDevice** object. The **DeviceFinding()** function of the **DeviceFinder** example presents a more robust method for device finding that does not require any user input at connection time. This example can be found at:
C:\Program Files\Pleora Technologies Inc\eBUS SDK\Samples\DeviceFinder.

```
156     PvResult lResult;
157
158     //Get the selected device information.
159     PvString lConnectionID;
160     if ( !PvSelectDevice( &lConnectionID ) )
161     {
162         cout << "No device selected." << endl;
163         return false;
164     }
165
166     // Connect to the GigE Vision or USB3 Vision device
167     cout << "Connecting to device" << endl;
168     PvDevice *lDevice = PvDevice::CreateAndConnect( lConnectionID, &lResult );
169     if ( !lResult.IsOK() )
170     {
171         cout << "Unable to connect to device" << endl;
172         PvDevice::Free( lDevice );
173         return false;
174     }
175
```

Figure 1: Before streaming video or adjusting camera settings, the **ImageProcessing()** function establishes a connection to the camera. The PvDevice object is created when the program successfully connects to a Pleora USB3 or GigE interface.

Controlling GenICam Features

After connecting to the device, most of the code in the **ImageProcessing()** function pertains to streaming imagery. There are GenICam parameters that must be considered when setting up the streaming. This example project uses a couple of GenICam stream parameters to monitor streaming statistics. In addition, it uses the **AcquisitionStart** GenICam device parameter, which is a **Command** parameter, to begin streaming imagery. While all other GenICam parameter types can be set and queried, **Command** parameters may only be executed using the **PvGenParameterArray::ExecuteCommand()** function (Fig. 2).

```
208     PvGenParameterArray *lDeviceParams = lDevice->GetParameters();
209     PvGenParameterArray *lStreamParams = lStream->GetParameters();
210
211     // Get stream parameters/stats.
212     PvGenInteger *lBlockCount = dynamic_cast<PvGenInteger *>( lStreamParams->Get( "BlockCount" ) );
213     PvGenFloat *lFrameRate = dynamic_cast<PvGenFloat *>( lStreamParams->Get( "AcquisitionRate" ) );
214     PvGenFloat *lBandwidth = dynamic_cast<PvGenFloat *>( lStreamParams->Get( "Bandwidth" ) );
215
216     // Enables stream before sending the AcquisitionStart command.
217     cout << "Enable streaming on the controller." << endl;
218     lDevice->StreamEnable();
219
220     // The buffers are queued in the stream, we just have to tell the device
221     // to start sending us images.
222     cout << "Sending StartAcquisition command to device" << endl;
223     lDeviceParams->ExecuteCommand( "AcquisitionStart" );
```

Figure 2: This section of the **ImageProcessing()** function sets up the objects that control the **AcquisitionStart** GenICam device parameter and the **BlockCount**, **AcquisitionRate**, and **Bandwidth** GenICam stream parameters. This function executes the **AcquisitionStart** GenICam command to begin streaming video. The GenICam stream parameters are used later in the function to monitor streaming statistics while the camera is collecting images.

Since most of the types of GenICam parameters are not covered in the **ImageProcessing** sample, this tutorial includes an additional code sample that demonstrates how to set and get GenICam parameters of the **Enumeration**, **Boolean**, **Integer**, and **Float** types (Fig. 3). In addition to the common GenICam parameters like **Width**, **Height**, **Binning**, **ExposureTime**, and **Pixel Format**, there are a number of GenICam parameters that are specific to SWIR Vision Systems cameras (Fig. 4). At startup, these GenICam parameters are recalled and used unless expressly changed. Changes will persist a power cycle only if you execute the **SaveSettings** feature after changing the parameters and before cycling the device's power. In Pleora's eBUS Player, these specific features can be found in the **Device Control** dialog and are labeled **CameraHeadFeatures**. Pleora provides another sample project that reads and displays all of the GenICam parameters from the camera's XML file. It can be found at: **C:\Program Files\Pleora Technologies Inc\eBUS SDK\Samples\GenICamParameters**.

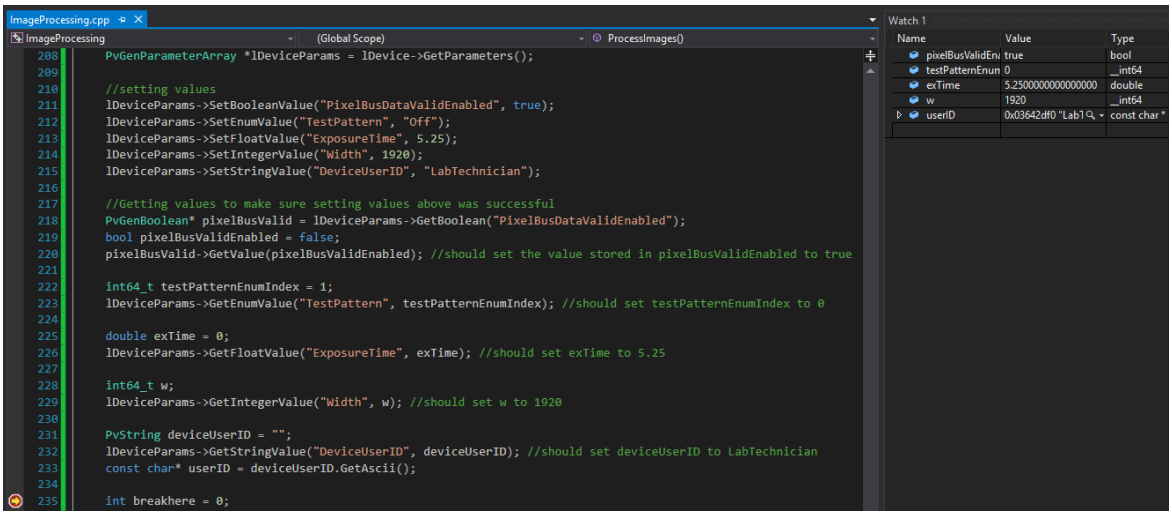

Figure 3: This code is not included in the **ImageProcessing** example project. This code merely demonstrates how to set and get the five types of GenICam parameters: Boolean, Enumeration, Float, Integer, and String. Note that the values that are displayed in **Watch 1** are consistent with the values that are set in the sample code.

Getting Image Data

Although the **ImageProcessing()** function demonstrates how to intialize an image stream and retrieve an image buffer, it does not go as far as accessing pixel data from the imagery. With a couple additional lines of code, it is simple to get a **PvImage** from the image buffer and use the **PvImage** to access the pixel data directly (Fig. 4). With direct access to the pixel data in a custom application, it should be convenient to pass the pixel values into any object required to utilize an image processing or image visualization library.

Parameter Name	Parameter Type	Values
Mirror	Enumeration	(0) Horizontal Off/Vertical Off, (1) Horizontal On/Vertical Off, (2) Horizontal Off/Vertical On, (3) Horizontal On/Vertical On
NucTable	Enumeration	0, 1, 2, 3
NucEnable	Enumeration	(0) Offset Off / Gain Off, (1) Offset On / Gain Off, (2) Offset Off / Gain On, (3) Offset On / Gain On,
BadPixelReplace-ment	Enumeration	(0) Off, (1) On
MasterClock	Enumeration	(0) Low, (1) High
TriggerMode	Enumeration	(0) Internal, (1) External, (2) External PWM, (3) Freerun
RoicGain	Enumeration	(0) Low, (1) Medium, (2) High
BinningHorizReg	Int	0,1,2,4
BinningVertReg	Int	0,1,2,4
WidthMaxReg	Int	Device Dependent
HeightMaxReg	Int	Device Dependent
FrameTime	Float	Value in milliseconds
LoadFactoryDefaults	Command	
SaveSettings	Command	

Figure 4: A table of GenICam parameters that are specific to SWIR Vision Systems devices. It is important to note the parameter type, because different parameter types use different set and query functions in the eBUS SDK. The valid settings for each of the parameters are discussed in the **Software Command Protocol** section of the **Acuros 1.1 User Manual**.

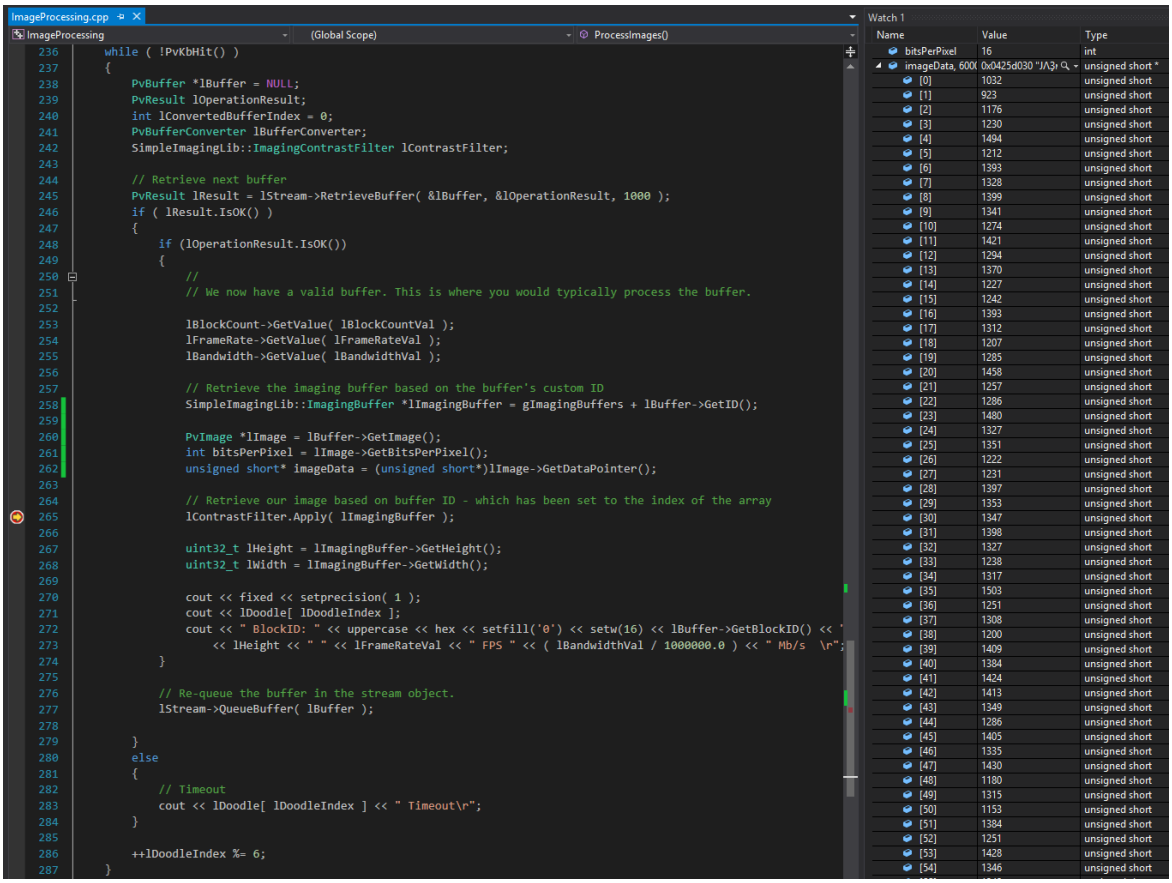

Figure 5: The code at lines 260-262 is not included in the **ImageProcessing** example project. The **ImageProcessing** example utilizes the **SimpleImagingLib** code to handle images from the stream in an abstracted way. Lines 260-262 were added to this example to demonstrate a place where the user can extract actual pixel data from the image stream. Note that the camera's **PixelFormat** GenICam parameter is set to **PvPixelMono14** in this example. 14 bits per pixel is the default for SWIR Vision Systems cameras. The array of unsigned shorts labeled **imageData** that is displayed in **Watch 1** is consistent with what we would expect for 14-bit pixel data. It makes sense that **bitsPerPixel** is equal to 16, because there is no 14-bit data type in C++. The smallest data type that can represent 14-bit data is 16 bits deep.