PLEORA TECHNOLOGIES INC.

eBUS SDK .NET API

eBUS SDK Version 6.2

Quick Start Guide

HongKe 虹科

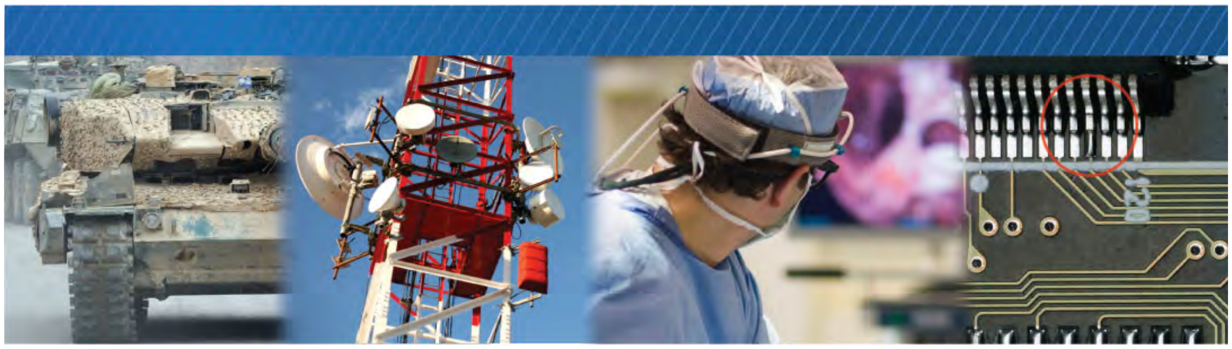
Pleora Technologies

Document Number

EX001-017-0007,Version 14.0 3/4/21

Table of Contents

What’s New?



What’s New In Version 6.2?

eBUS SDK is updated with the following new features and enhancements in Release 6.2.

- “Enhanced 3D Linescan Support for eBUS Tx-Based Applications Targeting Embedded Devices”
- “Ubuntu 20.04 LTS (64-bit) Support”
- “Installation Package Updates”
- “Microsoft Visual Studio 2019 Support”
- “eBUS Daemon Removal on Linux Operating System”
- “Support for Jetpack 4.3”

Enhanced 3D Linescan Support for eBUS Tx-Based Applications Targeting Embedded Devices

eBUS SDK is improved in Release 6.2 to reduce CPU consumption, thereby enabling additional on-board processing capabilities to be leveraged for eBUS Tx applications deployed on embedded devices for 3D Linescan applications. These enhancements ensure that sufficient resources are available on the embedded system for accurate triggering in Linescan applications, as well as ensuring that all 3D data analysis (for example, point cloud calculation) can be performed on the embedded device.

Ubuntu 20.04 LTS (64-bit) Support

eBUS SDK introduces support for Ubuntu 20.04 LTS (64-bit) on x86 platforms in Release 6.2.

You must install the following library before installing eBUS SDK on Ubuntu 20.04:

```
sudo apt-get install libavcodec58
```

Installation Package Updates

eBUS SDK Release 6.2 updates the naming for eBUS SDK installation packages for the following platforms:

- Ubuntu 14.04 (32-bit and 64-bit)
- Ubuntu 16.04 (32-bit and 64-bit)
- Ubuntu 18.04 (64-bit)
- RHEL and CentOS 7

Package names are changed to add the Ubuntu version, for example:

`eBUS_SDK_Ubuntu-14.04-i686-6.2.`*`<sub-minor version>`*`-`*`<build number>`*

The RHEL and CentOS 7 installation package is renamed to remove the OS version. In this instance, “7” is removed and the package is renamed as follows:

`eBUS_SDK_RHEL-CentOS-x86_64-6.2.`*`<sub-minor version>`*`-`*`<build number>`*

Upgrading directly to the new eBUS SDK software for these platforms is not supported. Pleora recommends uninstalling the previous eBUS SDK before installing the new software package. Also, note that the installation path is changed to include the distribution targeted.

Installation prerequisites have also been added for the following platforms in eBUS SDK Release 6.2:

- Ubuntu 14.04 (32-bit and 64-bit)
- Ubuntu 16.04 (32-bit and 64-bit)
- Ubuntu 18.04 (64-bit)
- RHEL and CentOS 7
- ARM64 (aarch64) on NVIDIA Jetson platforms

See the “Installing eBUS SDK for Linux” section in the *eBUS SDK for Linux Quick Start Guide* for details.

Microsoft Visual Studio 2019 Support

eBUS SDK Release 6.2 adds Visual Studio 2019 to the list of supported integrated development environments for Windows application development.

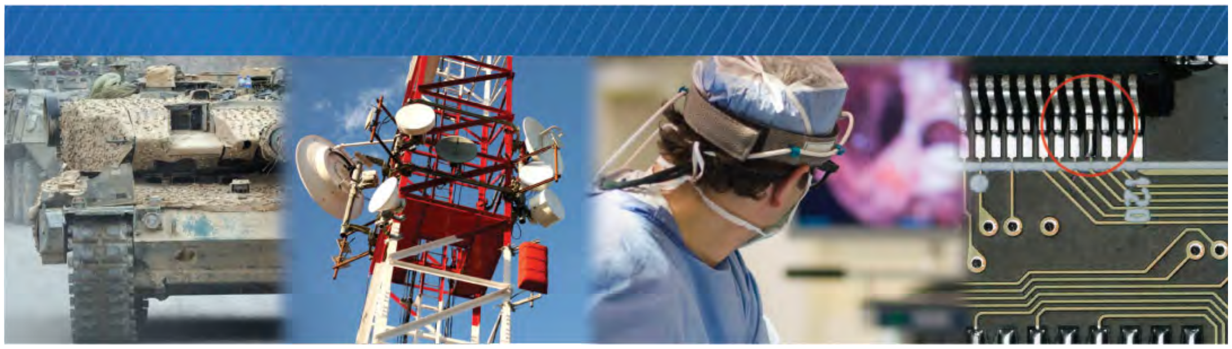
eBUS Daemon Removal on Linux Operating System

In previous eBUS SDK versions (version 4.0 to 6.1) stopping and restarting the eBUS Daemon was required when activating a license on a Linux OS. eBUS SDK is enhanced in Release 6.2 to remove the eBUS Daemon for the Linux operating system. As of eBUS SDK Release 6.2, you must simply close and re-launch your application after applying a license on the Linux OS for the license to be activated.

Support for Jetpack 4.3

eBUS SDK Release 6.2 introduces support for Jetpack 4.3 on Jetson platforms.

Chapter 1eBUS SDK .NET API



About this Guide

This chapter describes the purpose and scope of this guide, and provides a list of complementary guides.

The following topics are covered in this chapter:

- "What this Guide Provides" on page 2
- "Related Documents" on page 2

What this Guide Provides

This guide provides you with the information you need to install the eBUS SDK (which lets you use the eBUS SDK .NET API) and an overview of the system requirements.

You can use the sample applications to see how the API classes and methods work together for device configuration and control, unicast and multicast communication, image and data acquisition, image display, and diagnostics. You can also use the sample code to verify that your system is working properly (that is, determine whether there is a problem with your code or your equipment).

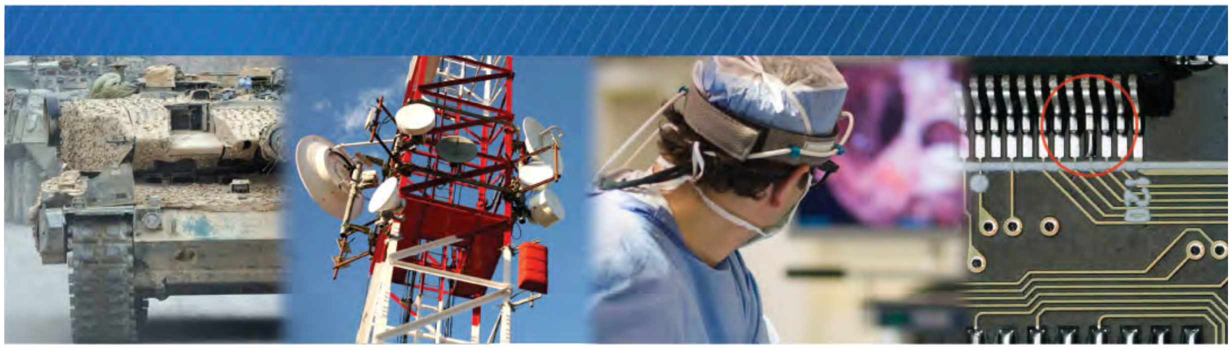
For troubleshooting information and technical support contact information for Pleora Technologies, see the last few chapters of this guide.

Related Documents

The *eBUS SDK .NET API Quick Start Guide* is complemented by the following Pleora Technologies documents, which are available on the Pleora Technologies Support Center (supportcenter.pleora.com):

- *eBUS Player Quick Start Guide*
- *eBUS Player User Guide*
- *eBUS SDK .NET API Help File*
- *eBUS SDK for Linux Quick Start Guide*
- *eBUS SDK Licensing Overview Knowledge Base Article*
- *Configuring Your Computer and Network Adapters for Best Performance Knowledge Base Article*

Chapter 2



Introducing the eBUS SDK .NET API

This chapter describes the eBUS SDK .NET API, which is a feature of the eBUS SDK that allows you to develop custom vision systems to acquire and transmit images and data using C#.NET or Visual Basic® .NET (VB.NET).

The following topics are covered in this chapter:

- "About the eBUS SDK .NET API" on page 4
- "eBUS SDK Licenses" on page 5

About the eBUS SDK .NET API

eBUS SDK is built on a single API to receive video over GigE, 10 GigE, and USB 3.0 that is portable across Windows, Linux, and macOS operating systems. With a Developer Seat License, designers can develop production-ready software applications in the same environment as their end-users, and quickly and easily modify applications for different media, while avoiding supporting multiple APIs from various vendors. Compared to camera vendor provided SDKs, eBUS frees developers from being tied to a specific camera, and instead they can choose the device that is best for the application.

eBUS Rx for Host Applications

eBUS Rx manages high-speed reception of images or data into buffers for hand-off to the end application for further analysis. Developers can write applications that run on a host computer to seamlessly control and configure an unlimited number of GigE Vision or USB3 Vision and GenICam compliant sensors.

The eBUS Universal Pro driver reduces CPU usage when receiving images or data, leaving more processing power for analysis and inspection applications while helping meet latency and throughput requirements for real-time applications. The eBUS Universal Pro driver is easily integrated into third-party processing software to bring performance advantages to end-user applications.

eBUS SDK Licenses

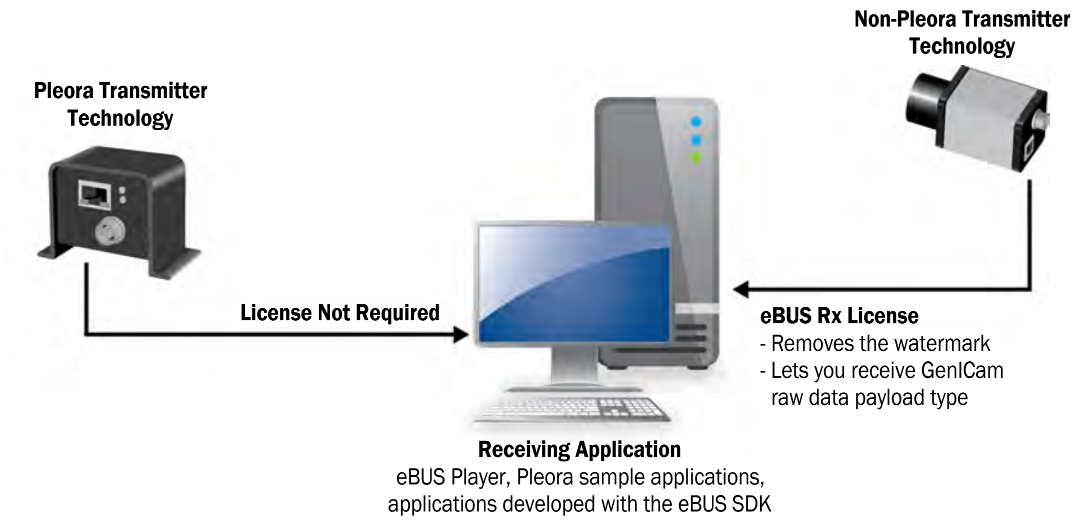
Some components of the eBUS SDK require a Pleora license to remove transmit and recieve limitations.

eBUS Player, Sample Applications, and Software Applications Created with the eBUS SDK

When a license is not present and you are receiving images from third-party transmitter technology, an embossed Pleora watermark will appear on the images that you receive. In addition, you will not be able to receive the GenICam raw data payload type from the transmitting device.

GigE Vision Devices Created with the eBUS Tx API

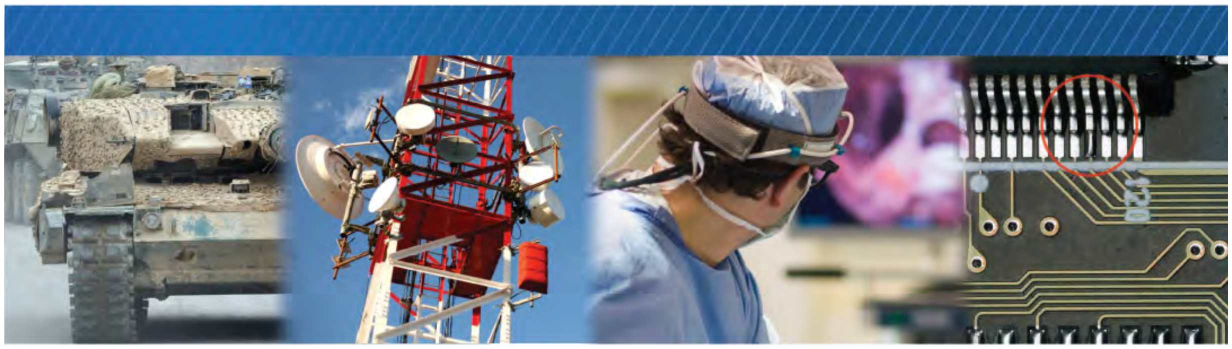
When a license is not present, GigE Vision receivers will be disconnected from the GigE Vision device after 15 minutes. In addition, your device will report hard-coded device information (such as the device model name), instead of your organization's customized information.


Pleora Transmitter Technology
Non-Pleora Transmitter Technology
License Not Required
eBUS Rx License
- Removes the watermark
- Lets you receive GenICam raw data payload type
Receiving Application
eBUS Player, Pleora sample applications, applications developed with the eBUS SDK

Activating an eBUS SDK License

For detailed information about licensing, including details on activating a license, see the *eBUS SDK Licensing Overview Knowledge Base Article* available on the Pleora Technologies Support Center at supportcenter.pleora.com.

Chapter 3



Installing the eBUS SDK

The eBUS SDK .NET API is installed on your computer during the installation of the eBUS SDK.

The instructions in this chapter are based on the Windows 7 operating system. The steps may vary depending on your computer's operating system.

The following topics are covered in this chapter:

- "System Requirements" on page 8
- "Installing the eBUS SDK" on page 8

System Requirements

Ensure the computer on which you install the eBUS SDK meets the following recommended requirements:

- At least one Gigabit Ethernet NIC (if you are using GigE Vision devices) or at least one USB 3.0 port (if you are using USB3 Vision devices).
- An appropriate compiler or integrated development environment (IDE):
  - Visual Studio 2019, 2017, 2015, 2013, 2012, and 2010.

One of the following operating systems:

- Microsoft Windows 10, 32-bit or 64-bit
- Microsoft Windows 8.1, 32-bit or 64-bit
- Microsoft Windows 7 with Service Pack 1 (or later), 32-bit or 64-bit

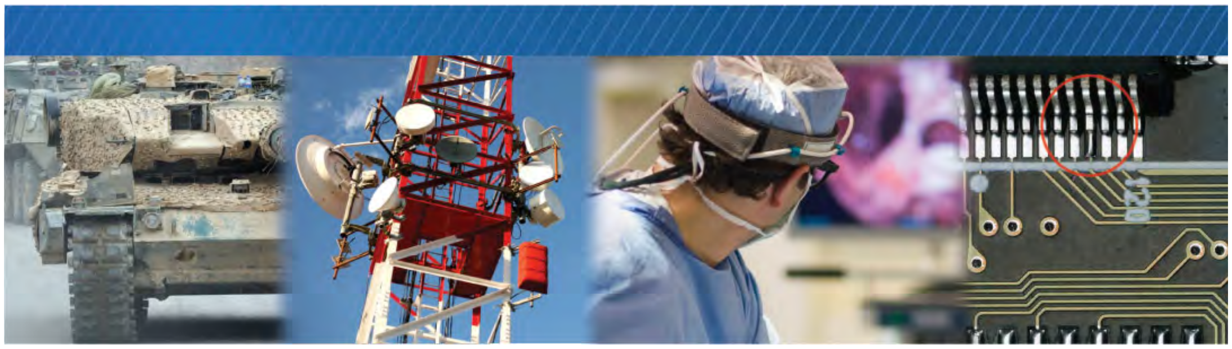
For supported USB 3.0 host controller chipsets, consult the eBUS SDK Release Notes, available on the Pleora Support Center.

Depending on the incoming and outgoing bandwidth requirements, as well as the performance of each NIC, you may require multiple NICs. For example, even though Gigabit Ethernet is full duplex (that is, it manage 1 Gbps incoming and 1 Gbps outgoing), the computer’s PCIe bus may not have enough bandwidth to support this. This means that while your NIC can – in theory – accept four cameras at 200 Mbps each incoming, and output a 750 Mbps stream on a single NIC, the NIC you choose may not support this level of performance.

Installing the eBUS SDK

Because the eBUS SDK .NET API is part of the eBUS SDK, it is included in the eBUS SDK installation package. You can download the eBUS SDK from the Pleora Support Center at supportcenter.pleora.com.

Chapter 4



Using the Sample Code

To illustrate how you can use the eBUS SDK .NET API to acquire and transmit images, the SDK includes sample code that you can use. This chapter provides a description of the sample code and provides general information about accessing the code to create sample applications.

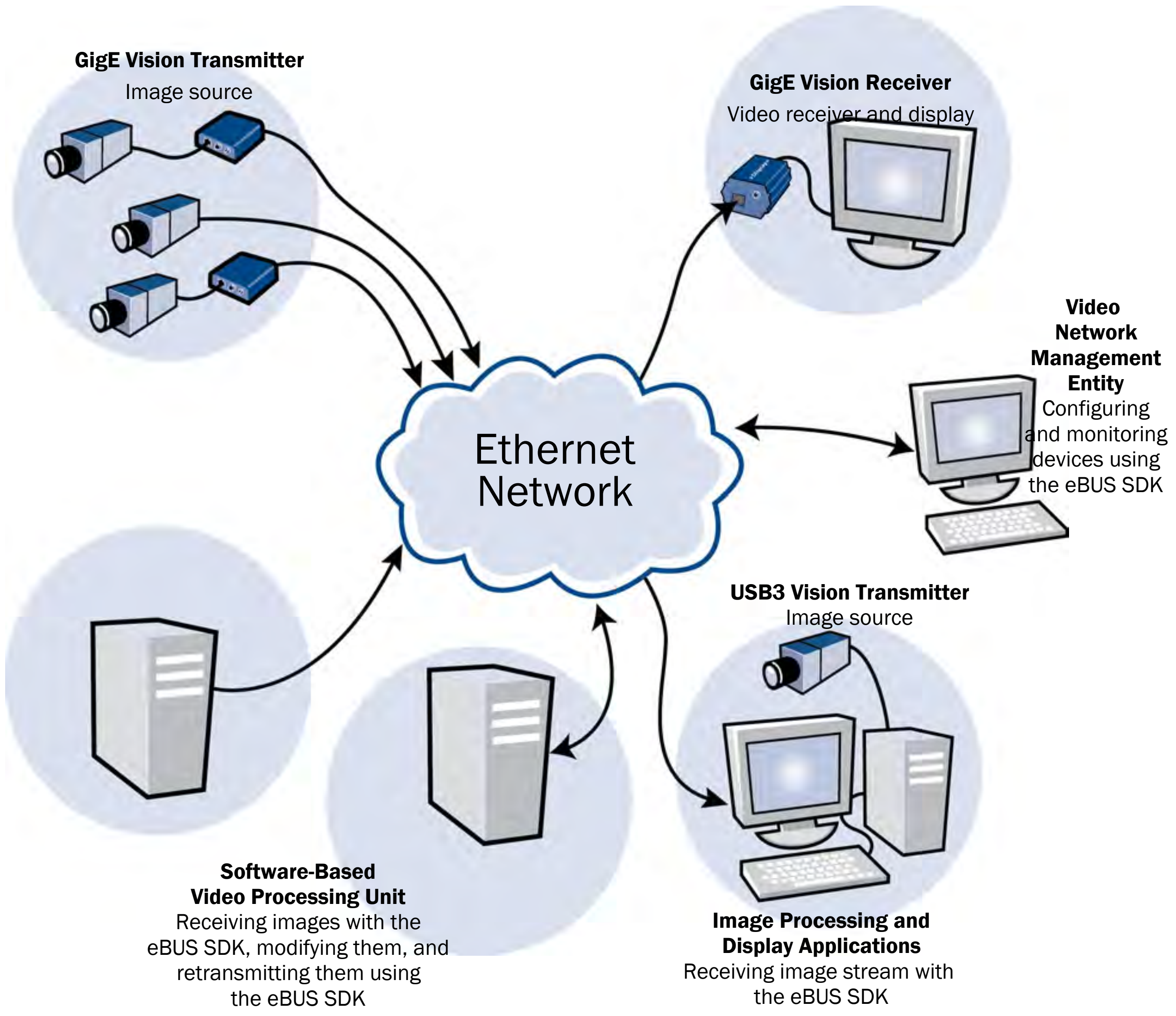
Most of the code samples are UI-based. However, it is possible to perform any activity demonstrated in the samples programmatically.

The following topics are covered in this chapter:

- "Overview: System Components" on page 10
- "Description of Samples" on page 11

Overview: System Components

The following illustration shows the components that are used, illustrating the relationship between the eBUS SDK, GigE Vision receivers, GigE Vision transmitters, and USB3 Vision transmitters.


GigE Vision Transmitter
Image source
GigE Vision Receiver
Video receiver and display
Ethernet Network
Video Network Management Entity
Configuring and monitoring devices using the eBUS SDK
USB3 Vision Transmitter
Image source
Software-Based Video Processing Unit
Receiving images with the eBUS SDK, modifying them, and retransmitting them using the eBUS SDK
Image Processing and Display Applications
Receiving image stream with the eBUS SDK

Description of Samples

The following samples are available as C# samples in this release. Some samples are also available as VB.NET samples.

Table 1: Sample Code

Sample code	Function	Type of application that is created
eBUS Demo Application		
SimpleApplication, SimpleApplicationVB	For developers using .NET or Visual Basic.NET, this sample showcases some of the functionality of the eBUS Rx SDK, which is used to detect, connect, and configure GigE Vision and USB3 Vision devices, and display and streaming images.	UI-based: C# .NET, and VB .NET
Getting Started		
PvStreamSample	This “Hello World” sample shows you how to connect to a GigE Vision or USB3 Vision device, receive an image stream, stop streaming, and disconnect from the device.	• Command line
Image Streaming		
PvPipelineSample	This sample extends the "Hello World" PvStreamSample by showing how buffers are managed internally by the PvPipeline class. This removes some of the complexity of buffer management from the application when compared to the PvStream sample.	• Command line
DualSource	This GUI-based sample for GigE Vision devices extends the MultiSource sample by allowing you to view image streams from a GigE Vision device that has two streaming sources.	• UI-based
ImageProcessing	This sample illustrates how to acquire an image and process it using an external buffer to interface with a non-Pleora library. This is useful when you want to interface the eBUS SDK to popular third-party SDKs for image processing or machine learning, such as OpenCV.	• Command line
MulticastMaster for GigE Vision devices	This sample shows how to connect to a GigE Vision device and initiate a multicast stream to allow multiple slave devices to receive and process the image stream simultaneously. This sample is used in conjunction with the **MulticastSlave** sample, which listens to the multicast stream.	• Command line

Table 1: Sample Code (Continued)

Sample code	Function	Type of application that is created
MulticastSlave for GigE Vision devices	This sample shows how to configure the eBUS SDK to receive an image stream from a GigE Vision device that is configured for multicast mode. This sample is used in conjunction with the **MulticastMaster** sample, which initiates the multicast stream.	• Command line
DirectShowDisplay	This sample shows how to integrate the eBUS SDK DirectShow source filter and how to build a filter graph that receives images from a GigE Vision or USB3 Vision device. This functionality is recommended for existing DirectShow applications to enable quick integration of GigE Vision or USB3 Vision devices. Note: This sample assumes good knowledge of DirectShow and COM.	• UI based
Configuration and Event Monitoring		
GenICamParameters	This sample shows how to enumerate and display the GenICam features and settings of a GenICam-compatible device by discovering and accessing the features of the device's node map. The node map is built programmatically from the device's GenICam XML file.	• Command line

For developers who have worked with earlier releases of the eBUS SDK, please note that the Video Server API and the following sample applications are not recommended for new designs (NRND): **TransmitTestPattern**, and **TransmitTiledImages**.

Using H.264 Encoding

You can use eBUS Player (C++ sample application) to encode images in H.264 video format and save them on your computer in an MP4 file.

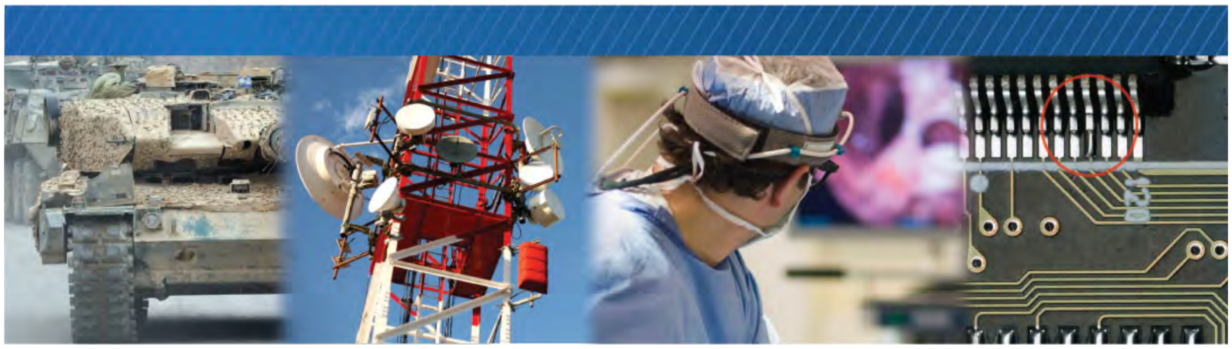
If you are interested in adding H.264 encoding to your application, you can refer to the Mp4WriterWin32 class in the eBUS Player C++ sample application for an example. At this time, Pleora does not supply sample code to save H.264 video in an mp4 file for .NET applications. However you can use the `mp4writerwin32` C++ class as a template to migrate the code to .NET.

This feature can be used when you start eBUS Player from the Windows Start menu. However, it is important to note that if you are using the eBUS Player sample application, this feature is not enabled by default.

Video Encoder Information

When saving H.264 video in an MP4 container, the eBUS SDK uses the Microsoft Media Foundation H.264 video encoder. The default video encoder settings are used, with the exception of the frame rate and average bit rate. The video is encoded at 30 frames per second. And the average bit rate is the value set by the `PvMp4Writer.AvgBitrate` property.

Chapter 5



Code Walkthrough: Acquiring Images with the eBUS SDK

This section walks you through the code contained in **PvStreamSample**. This sample illustrates how to detect available devices, connect to a device, and start an image stream.

The following topics are covered in this chapter:

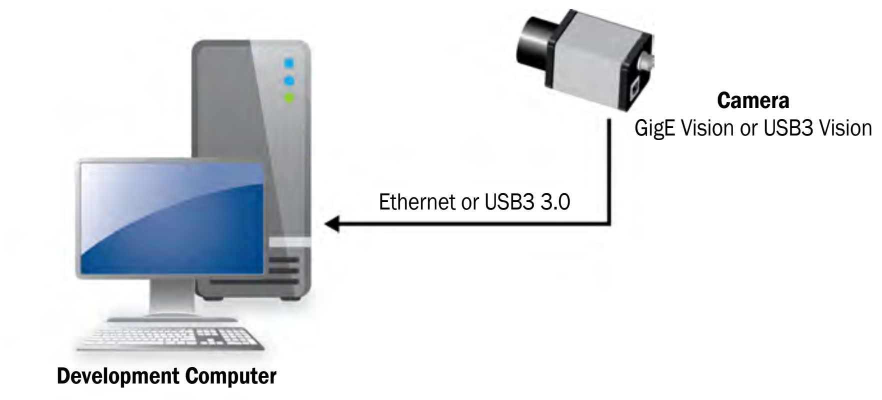
Accessing the Sample Code

The sample code is available in the following location: C:\Program Files\Pleora Technologies Inc\eBUS SDK\Samples

You must copy the sample code to a location on your computer (such as your C: drive) before you open the sample code in your IDE. Access to these directories is restricted.

Required Items

The sample code requires that you have a GigE Vision device connected to a NIC on your computer or a USB3 Vision device connected to a USB 3.0 port on your computer.



Classes Used in the PvStreamSample

PvStreamSample uses the classes listed in the following table.

Table 2: Classes Used in the Sample

Class	Description
PvDeviceInfo	Used to access information about a device, such as its manufacturer information, protocol (either GigE Vision or USB3 Vision), serial number, ID, and version.
PvDevice	Used to connect to and control a device and initiate the image stream. Protocol and interface-specific functionality is available in two subclasses, PvDeviceGEV and PvDeviceU3V.
PvStream	Provides access to the image stream. Like PvDevice, there are protocol and interface-specific subclasses, PvStreamGEV and PvStreamU3V.

Table 2: Classes Used in the Sample

Class	Description
PvBuffer	Represents a block of data from the device, such as an image.
PvResult	A simple class that represents the result of various eBUS SDK methods.

Namespaces

The `PvDotNet` and `PvGUIDotNet` namespaces are required for applications that use the eBUS SDK. If you are developing an application that does not use GUI components, you can exclude `PvGUIDotNet`.

C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using PvDotNet;
using PvGUIDotNet;
```

When you include `PvGUIDotNet`, eBUS SDK GUI components are available to drag from the Visual Studio toolbox into your application.

To add the display, browser, and status controls to the Visual Studio toolbox

1. In Visual Studio, right-click the **Toolbox** and then click **Choose Items**.
2. On the **.NET Framework Components** tab, select the following controls: **PvDisplayControl**, **PvGenBrowserControl**, and **PvStatusControl**.

Timer

The sample uses a timer to sequence through the methods of the sample. Every time that the timer ticks (every 1 second), the sample executes the next method. The use of the timer helps demonstrate concepts within the eBUS SDK, and you do not need to include it in your application.

The timer is only enabled when one of the methods completes.

FormClosing Event

On form closing, if a stream has been started, the stream stops. If there is a connection with the device, the application disconnects from the device.

Step1Connecting Method

Step1Connecting calls methods that allow the user to select a device from a dialog box, connect to a device, open the image stream, and set the browser parameters.

This sample demonstrates the display of available devices to the user, using a device finder window. Another option is to select a device programmatically, using the PvSystem class to iterate through all PvInterface objects (network adapters or USB host controllers) and within those, iterate through PvDeviceInfo objects. Alternatively, you can hard-code the device identifier (the IP address, MAC address, or USB GUID) or read it from a configuration file.

The first section of the sample provides the user with a way to see all of the available GigE Vision and USB3 Vision devices, and select one. A device finder dialog box presents all of the devices that are available to the user. If the user selects a GigE Vision or USB3 Vision device and clicks the **OK** button, then we proceed to connect to the device and open the image stream, as discussed in the next paragraph.

C#

```
        private void Step1Connecting()
               {
                   oneLabel.Enabled = true;

                   // Pop device finder, let user select a device.
                   PvDeviceFinderForm lForm = new PvDeviceFinderForm();
                   if (lForm.ShowDialog() == DialogResult.OK)
                   {
                       try
                       {
                           PvDeviceInfo lDeviceInfo = lForm.Selected;
```

Next, the sample connects to the selected device. GigE Vision and USB3 Vision devices are represented by different classes (PvDeviceGEV and PvDeviceU3V) and they share a parent class (PvDevice) that abstracts most of the differences. When possible, you should use a PvDevice object instead of a protocol-specific object to reduce code duplication. To create a PvDevice object from a PvDeviceInfo object without explicitly checking the protocol of the device, use the CreateAndConnect static factory method from the PvDevice class, which abstracts the device type.

If it is not important for your application to support both GigE Vision and USB3 Vision devices (for example, your organization only uses devices of a particular type), you can call the GetType method of the PvDeviceInfo object (aDeviceInfo) to determine whether the device is GigE Vision or USB3 Vision. Then you could create a new PvDeviceGEV or PvDeviceU3V object and call the Connect method directly.

When the sample initiates the image stream, it uses a static factory method (CreateAndOpen) to create and open the PvStream object, which allows your application to support both GigE Vision and USB3 Vision devices.

Finally, it retrieves a reference to an array of GenICam features that represent the stream statistics and parameters and stores them in the UI element on the form.

The eBUS SDK .NET API uses exceptions for error management. For more information, see "Error Management" on page 33.

C#

```
                    // Connect device.
                   mDevice = PvDevice.CreateAndConnect(lDeviceInfo);

                   // Open stream.
                   mStream = PvStream.CreateAndOpen(lDeviceInfo.ConnectionID);
                   if (mStream == null)
                   {
                       MessageBox.Show("Unable to open stream.", Text);
                       return;
                   }

                   // Set browser parameters.
                   browser.GenParameterArray = mStream.Parameters;
               }
               catch (PvException ex)
               {
                   Step6Disconnecting();
                   MessageBox.Show(ex.Message, Text, MessageBoxButtons.OK, MessageBoxIcon.Error);
                   Close();
               }
           }
           else
           {
               Close();
           }
       }
```

Step2Configuring Method

For most of this sample, there is no need to distinguish between GigE Vision or USB3 Vision devices, as the eBUS SDK classes abstract the device type. However, when using a GigE Vision device, you must set a destination IP address for the image stream. In this sample, the destination is automatically set to be the IP address of the network interface card on the PC used to interface with the device (which is the most common configuration).

Also, for optimal performance over Gigabit Ethernet, it is necessary to determine the largest possible packet size for the connection (ideally the link would use jumbo frames — typically about 9000 bytes). This is the only place in the application where we check the device type.

Jumbo frames are configured on your computer's network interface card (NIC). For more information, see the operating system documentation or the *Configuring Your Computer and Network Adapters for Best Performance Knowledge Base Article*, available on the Pleora Support Center at supportcenter.pleora.com.

When developing your application, you may prefer to hard-code the packet size based on your target system, instead of using `PvDeviceGEV.NegotiatePacketSize`.

First, we use a dynamic cast to determine if the `PvDevice` object represents a GigE Vision device. If it is a GigE Vision device, we do the required configuration. If it is a USB3 Vision device, no stream configuration is required for this sample.

C#

```
private void Step2Configuring()
       {
           oneLabel.Enabled = false;
           twoLabel.Enabled = true;
           try
           {
               // Perform GigE Vision only configuration
               PvDeviceGEV lDGEV = mDevice as PvDeviceGEV;
               if (lDGEV != null)
               {
                   // Negotiate packet size
                   lDGEV.NegotiatePacketSize();

                   // Set stream destination.
                   PvStreamGEV lSGEV = mStream as PvStreamGEV;
                   lDGEV.SetStreamDestination(lSGEV.LocalIPAddress, lSGEV.LocalPort);
               }
```

Next, the sample allocates memory for the received images.

`PvStream` contains two buffer queues: an "input" queue and an "output" queue. First, we add `PvBuffer` objects to the input queue of the `PvStream` object by calling `PvStream.QueueBuffer` once per buffer. As images are received, `PvStream` populates the `PvBuffers` with images and moves them from the input queue to the output queue. The populated `PvBuffers` are removed from the output queue by the application (using `PvStream.RetrieveBuffer`), processed, and returned to the input queue (using `PvStream.QueueBuffer`). The queuing and retrieval of buffers is managed by a worker thread that executes the ThreadProc method, which appears near the end of the sample code and is described in "The Worker Thread" on page 23.

The memory allocated for PvBuffer objects is based on the resolution of the image and the bit depth of the pixels (the payload) retrieved from the device using PvDevice.GetPayloadSize. The device returns the number of bytes required to hold one buffer, based on the configuration of the device.

When designing applications that deal with higher frame rate streams or that run on slower platforms, it may be necessary to increase the BUFFER_COUNT (to give you some margin for performance dips when you cannot process buffers fast enough for a short period). This allows the application to avoid a scenario where all buffers are in the output queue awaiting retrieval, and none are available in the input queue to store newly-received images.

C#

```
            // Read payload size, preallocate buffers of the pipeline.
            Int64 lPayloadSize = mDevice.PayloadSize;

            // Get minimum buffer count, creates and allocates buffer.
            UInt32 lBufferCount = (mStream.QueuedBufferMaximum < cBufferCount) ?
                         mStream.QueuedBufferMaximum : cBufferCount;
            PvBuffer[] lBuffers = new PvBuffer[lBufferCount];
            for (UInt32 i = 0; i < lBufferCount; i++)
            {
                lBuffers[i] = new PvBuffer();
                lBuffers[i].Alloc((UInt32)lPayloadSize);
            }

            // Queue all buffers in the stream.
            for (UInt32 i = 0; i < lBufferCount; i++)
            {
                mStream.QueueBuffer(lBuffers[i]);
            }
        }
        ...
```

Step3StartingStream Method

In this method, we acquire images from the device.

`Step3StartingStream` creates a new thread, which executes the `ThreadProc` method that manages the queueing and retrieval of buffers. While it is not necessary to queue and retrieve buffers on a separate thread, we use this approach in the sample to allow the user interface to be updated independently of buffer management.

To start the image stream, we enable streaming on the device (`PvDevice.StreamEnable`) and execute the GenICam `AcquisitionStart` command (`lStart`).

For GigE Vision devices, `StreamEnable` sets the `TLParamsLocked` feature, which prevents changes to the streaming related parameters during image acquisition.

For USB3 Vision devices, it sets the `TLParamsLocked` feature, configures the USB driver for streaming, and sets the stream enable bit on the device.

C#

```
        private void Step3StartingStream()
               {
                   twoLabel.Enabled = false;
                   threeLabel.Enabled = true;

                   // Start display thread.
                   mThread = new Thread(new ParameterizedThreadStart(ThreadProc));
                   MainForm lP1 = this;
                   object[] lParameters = new object[] { lP1 };
                   mIsStopping = false;
                   mThread.Start(lParameters);

                   // Enables streaming before sending the AcquisitionStart command.
                   mDevice.StreamEnable();

                   // Start acquisition on the device
                   mDevice.Parameters.ExecuteCommand("AcquisitionStart");
               }
```

The Worker Thread

As the acquisition thread runs, we retrieve the first `PvBuffer`, and check the results. When we retrieve the `PvBuffer` object, we remove it temporarily from the `PvStream` output buffer queue and process it. When processing is complete, we add the `PvBuffer` object back into the input buffer queue.

To verify that a buffer has been retrieved successfully from the stream object and to verify the acquisition of an image, we examine the two values supplied by `RetrieveBuffer`. First, we check the value of a `PvResult` object (`lResult`) to determine that a buffer has been retrieved. If a buffer has been retrieved, then it checks the value of the `PvResult` object (`lOperationResult`) to verify the acquisition operation (for example, it checks if the operation timed out, had too many resends, or was aborted.)

C#

```
private static void ThreadProc(object aParameters)
    {
        object[] lParameters = (object[])aParameters;
        MainForm lThis = (MainForm)lParameters[0];

        for (;;)
        {
            if (lThis.mIsStopping)
            {
                // Signaled to terminate thread, return.
                return;
            }

            PvBuffer lBuffer = null;
            PvResult lOperationResult = new PvResult(PvResultCode.OK);

            // Retrieve next buffer from acquisition pipeline
            PvResult lResult = lThis.mStream.RetrieveBuffer(ref lBuffer, ref lOperationResult, 100);
            if (lResult.IsOK)
            {
                // Operation result of buffer is OK, display.
                if (lOperationResult.IsOK)
                {
                    lThis.displayControl.Display(lBuffer);
                }

                // We have an image - do some processing (...) and VERY IMPORTANT,
                // re-queue the buffer in the stream object.
                lThis.mStream.QueueBuffer(lBuffer);
            }
        }
    }
```

Now that we have obtained a PvBuffer with an image, we display the image in the application. This is also the point at which your application would typically process the buffer. After we display the image, we requeue the PvBuffer object in the input queue by calling PvStream.QueueBuffer.

If lOperationalResult returns something other than OK, a PvBuffer object has been retrieved, but it is not valid (for example, only part of the image could be retrieved or a timeout occurred). In this case, we also re-queue the PvBuffer object back to the PvStream object so it can be used again.

If lResult returns something other than OK, a PvBuffer object was not retrieved and therefore there is no PvBuffer to requeue.

C#

```
        private void Step5StoppingStream()
               {
                   stopButton.Enabled = false;
                   fourLabel.Enabled = false;
                   fiveLabel.Enabled = true;

                   try
                   {
                       PvBuffer lBuffer = null;
                       PvResult lOperationResult = new PvResult(PvResultCode.OK);
                       PvResult lResult = new PvResult(PvResultCode.OK);
       ...
```

Step5StoppingStream Method

The remainder of the sample is used to stop acquisition and clean up resources when the user presses the **Stop** button. First, we execute the GenICam `AcquisitionStop` command (`lStop`). Then, we disable the stream and stop the worker thread.

For GigE Vision devices, `StreamDisable` resets the `TLParamsLocked` feature, which allows changes to the streaming related parameters to occur.

For USB3 Vision devices, `StreamDisable` resets the `TLParamsLocked` feature and sets the stream enable bit on the device.

C#

```
                        // Stop acquisition.
                        mDevice.Parameters.ExecuteCommand("AcquisitionStop");

                        // Disable streaming after sending the AcquisitionStop command.
                        mDevice.StreamDisable();

                        // Stop display thread.
                        mIsStopping = true;
                        mThread.Join();
                        mThread = null;
```

Now that streaming has stopped, we mark all of the buffers in the input queue as aborted (using `PvStream.AbortQueuedBuffers`), which moves the buffers to the output queue.

For `PvStreamGEV` objects, before resuming streaming after a pause, you should flush the queue using `PvStreamGEV.FlushPacketQueue`, which removes all unprocessed UDP packets from the data receiver.

If your application does not abort queued buffers, your application will receive timeout errors when you restart `PvStream`, since the buffers in the input queue will have exceeded the timeout value.

While our sample does not necessarily require that we abort and remove the buffers from the queue (because we do not restart `PvStream` in this sample), it is included in this sample to illustrate the concept of clearing buffers.

Finally, we remove all of the buffers from the queue (using `PvStream.RetrieveBuffer`) so they can be requeued the next time the stream is enabled.

C#

```
                    // Abort all buffers in the stream and dequeue
                    mStream.AbortQueuedBuffers();
                    for (int i = 0; i < mStream.QueuedBufferCount; i++)
                    {
                        lResult = mStream.RetrieveBuffer(ref lBuffer, ref lOperationResult);
                        if (lResult.IsOK)
                        {
                            lBuffer = null;
                        }
                    }
                }
                catch (PvException lExc)
                {
                    MessageBox.Show(lExc.Message, Text);
                }
            }
```

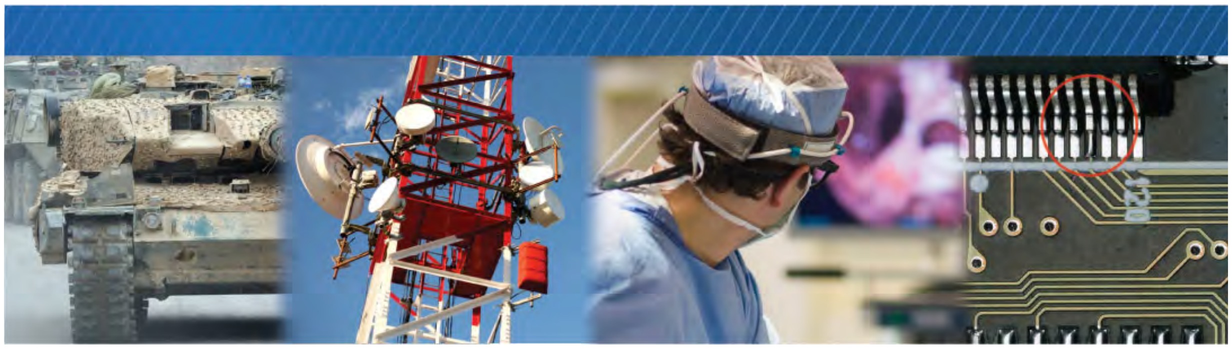
Step6Disconnecting Method

`Step6Disconnecting` releases the browser parameter array reference. The stream is closed (if it was opened) and then the connection to the device is disconnected.

C#

```
        private void Step6Disconnecting()
            {
                fiveLabel.Enabled = false;
                sixLabel.Enabled = true;

                // Release browser parameters reference.
                browser.GenParameterArray = null;

                if (mStream != null)
                {
                    // Close and release stream
                    mStream.Close();
                    mStream = null;
                }

                if (mDevice != null)
                {
                    // Disconnect and release device
                    mDevice.Disconnect();
                    mDevice = null;
                }
            }
```

Chapter 6



Troubleshooting

This chapter provides you with troubleshooting tips and recommended solutions for issues that can occur when using the eBUS SDK C++ API, GigE Vision, and USB3 Vision devices.

Not all scenarios and solutions are listed here. You can refer to the Pleora Technologies Support Center at supportcenter.pleora.com for additional support and assistance. Details for creating a customer account are available on the Pleora Technologies Support Center.

Refer to the product release notes that are available on the Pleora Technologies Support Center for known issues and other product features.

Troubleshooting Tips

The scenarios and known issues listed in this chapter are those that you might encounter during the setup and operation of your device. Not all possible scenarios and errors are presented. The symptoms, possible causes, and resolutions depend upon your particular setup and operation.

If you perform the resolution for your issue and the issue is not corrected, we recommend you review the other resolutions listed in this table. Some symptoms may be interrelated.

Table 3: Troubleshooting Tips

Symptom	Possible cause	Resolution
SDK cannot detect or connect to the Pleora device	Power not supplied to the device, or inadequate power supplied	Both the detection and connection to the device will fail if adequate power is not supplied to the device. Verify that the Network LED is active. For information about the LEDs, see the documentation accompanying the device. Re-try the connection to the device with your application.
	The GigE Vision device is not connected to the network	Verify that the network LED is active. If this LED is illuminated, check the LEDs on your network switch to ensure the switch is functioning properly. If the problem continues, connect the device directly to the computer to verify its operation. For information about the LEDs, see the documentation accompanying the device.
	The GigE Vision device and computer are not on the same subnet	Images might not appear in your application if the GigE Vision device and the computer running your application are not on the same subnet. Ensure that these devices are on the same subnet. In addition, ensure that these devices are connected using valid gateway and subnet mask information. You can view the IP address information in the **Available Devices** list in your application. A red icon appears beside the device if there is an invalid IP configuration.
SDK cannot detect the API or transmitter	NIC that is receiving and NIC that is transmitting are on different subnets	Ensure the transmitting and receiving NICs are on the same subnet.
Errors appear	For GigE Vision devices, the drivers for your NIC may not be the latest version	Ensure you have installed the latest drivers from the manufacturer of your NIC.

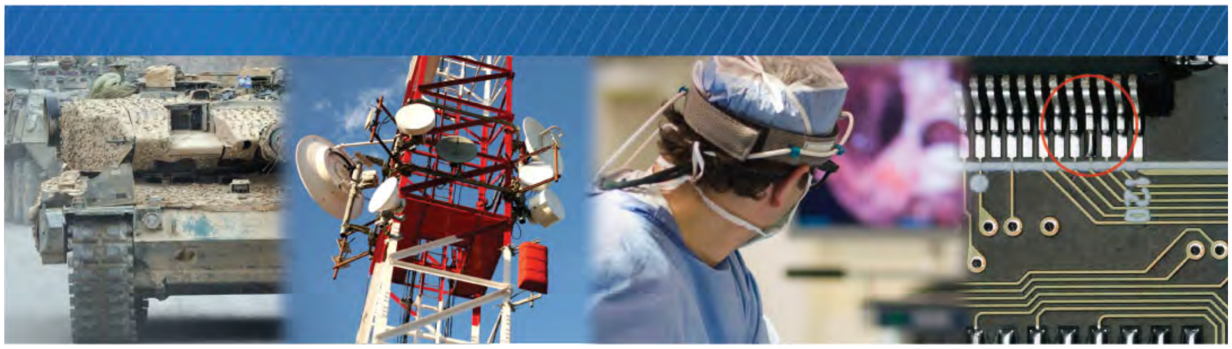
Table 3: Troubleshooting Tips (Continued)

Symptom	Possible cause	Resolution
SDK is able to connect, but no images appear in your application. In a multicast GigE Vision configuration, images appear on a display monitor connected to a vDisplay HDI-Pro External Frame Grabber but do not appear in your application.	In a multicast configuration, the device may not be configured correctly	Images only appear on the display if you have configured the device for a multicast network configuration. The device and all multicast receivers must have identical values for both the **GevSCDA** and **GevSCPHostPort** features in the **TransportLayerControl** section. For more information, see the documentation accompanying the device.
	In a multicast configuration, your computer's firewall may be blocking your application	Ensure that your application is allowed to communicate through the firewall.
	Anti-virus software or firewalls blocking transmission	Images might not appear in your application because of anti-virus software or firewalls on your network. Disable all virus scanning software and firewalls, and re-attempt a connection to the device with your application.
	Ensure jumbo packets are properly configured for the NIC	Enable jumbo packet support for the NIC and network switch (as required). If the NIC or network switch does not support jumbo packets, disable jumbo packets for the transmitter.

Table 3: Troubleshooting Tips (Continued)

Symptom	Possible cause	Resolution
Dropped packets: eBUS Player, NetCommand, or applications created using the eBUS SDK	Insufficient computer performance	The computer being used to receive images from the device may not perform well enough to handle the data rate of the image stream. The GigE Vision driver reduces the amount of computer resources required to receive images and is recommended for applications that require high throughput. Should the application continue to drop packets even after the installation of the GigE Vision driver, a computer with better performance may be required.
	Insufficient NIC performance	The NIC being used to receive images from the GigE Vision device may not perform well enough to handle the data rate of the image stream. For example, the bus connecting the NIC to the CPU may not be fast enough, or certain default settings on the NIC may not be appropriate for reception of a high-throughput image stream. Examples of NIC settings that may need to be reconfigured include the number of Rx Descriptors and the maximum size of Ethernet packets (jumbo packets). Additionally, some NICs are known to not work well in high-throughput applications. For information about maximizing the performance of your system, see the *Configuring Your Computer and Network Adapters for Best Performance Application Note*, available on the Pleora Support Center.

Chapter 7



Reference: C++ eBUS SDK and .NET SDK Comparison

This chapter outlines the differences between the C++ version of the eBUS SDK and the C# version.

The following topics are covered in this chapter:

The *eBUS SDK .NET API Help File* provides detailed information about the .NET classes available in the eBUS SDK .NET API. The Help file is available from the Windows **Start** menu (**All Programs** > **Pleora Technologies Inc.** > **eBUS SDK** > **eBUS SDK .NET API**).

Types

The eBUS SDK types used in C++ are replaced by native .NET types. For example, PvString (which is used to input and output strings to/from the eBUS SDK) is replaced with a string.

Table 4: .NET Equivalent of eBUS SDK C++ Types

C++ type	Replaced by the following .NET type
PvString	string
int_64	Int64
uint64_t	UInt64
int32_t	Int32
uint32_t	UInt32
int16_t	Int16
uint16_t	UInt16
uint8_t	UInt8

Properties

The Get and Set methods in C++ are replaced by properties in the .NET version of the eBUS SDK. A property internally defines Get and Set methods but it is only accessed through its property name. This is because Get and Set methods without properties are not commonly used in .NET.

Exceptions to this rule are the Get and Set methods that require additional parameters (for example, accessing values in the GenICam node tree). Because these methods cannot be replaced with properties, they remain the same as their C++ counterparts.

C++

```
PvSerialPort lPort;
PvUInt32 lSize = lPort.GetRxBufferSize();
lPort.SetRxBufferSize( lSize );
```

C#

```
PvSerialPort lPort = new PvSerialPort();
UInt32 lSize = lPort.RxBufferSize;
lPort.RxBufferSize = lSize;
```

Enumeration Types

Typed native .NET enumerations (not to be confused with GenICam enumerations) are used to wrap C++ enumerations. The same underlying C++ values are used when defining the .NET enumerations.

.NET enumerations are different from C++ enumerations in the following ways:

- They throw an exception when you attempt to set them to an unsupported integer value.
- They can be converted to strings and the strings can be parsed and turned into an enumeration value.
- They can be enumerated in statements, such as foreach statements.

Error Management

The .NET version of the eBUS SDK uses exceptions for error management, whereas the C++ version uses methods that return error codes. For example, the eBUS SDK uses a void return type and throws a PvException object when an error occurs, whereas the C++ version returns a PvResult object.

A PvException object contains a Result property, which is the PvResult that the method would have returned. The ToString and Description members of PvException are adapted to return a string-based representation of the PvResult containing the error code and description.

The only exception to this rule is streaming methods that manage buffers in the PvStreamBase, PvPipeline, and PvTransmitterRaw classes. For performance reasons try/catch and exception management are not used to handle these errors. For these methods, there is no change from the C++ approach — these methods return a PvResult object.

C++ eBUS SDK methods that have a PvResult return value and take a reference to a parameter to return the true return value, now simply return the return value (or are implemented as properties). A good example is the GetValue method of the PvGenParameter class.

C++

```
PvInt64 lValue = 0;
PvResult lResult = lInteger.GetValue( lValue );
if (!lResult.IsOK())
{
   // ...
}
```

C#

```
try
{
   Int64 lValue = lInteger.Value;
}
catch (PvException lEx)
{
   // ...
}
```

C++

```cpp
PvResult lResult = lDevice.Connect( "192.168.138.164" );
if ( !lResult.IsOK() )
{
    // ...
}

lResult = lStream.Open( "192.168.138.164" );
if ( !lResult.IsOK() )
{
    // ...
}

lResult = lPipeline.Start();
if ( !lResult.IsOK() )
{
    // ...
}
```

C#

```csharp
try
{
   lDevice.Connect( "192.168.138.164" );
   lStream.Open( "192.168.138.164" );
   lPipeline.Start();
}
catch (PvException lEx)
{
    // ...
}
```

Enumerators

Some eBUS SDK classes are containers for other objects that can be enumerated. .NET interfaces are implemented for these classes to allow the program to go through all of the objects with a foreach statement:

- The PvInterface class enumerates PvDeviceInfo objects.
- The PvSystem class enumerates PvInterface objects.
- The PvGenEnum class enumerates PvEnumEntry objects.
- The PvGenParameterArray enumerates PvGenParameter objects.

C# (Without an Enumerator)

```
UInt32 lCount = lArray.Count;
for (UInt32 i = 0; i < lCount; i++)
{
   PvGenParameter lP =
      lParameterArray.Get(i);
   // …
}
```

C# (With an Enumerator)

```
foreach (PvGenParameter lP in lArray)
{
   // ...
}
```

Collections

The C++ version of the eBUS SDK has collection classes such as PvStringList, PvPropertyList, and PvParameterList. In .NET, native List templates are used instead, such as List<string>, List<PvProperty>, and List<PvParameter>.

Callbacks

All C++ eBUS SDK callbacks or virtual methods used as callbacks are implemented in PvDotNet as .NET events of typed delegates.

Registering to an event is done in .NET using the += operator next to an event. The IntelliSense® feature of Visual Studio then automatically generates an event handler in the .NET code.

This rule ensures that the callback mechanism is as expected by .NET users. It also leverages the automated code generation performed by Visual Studio to help developers quickly and effortlessly hook up events to their own code.

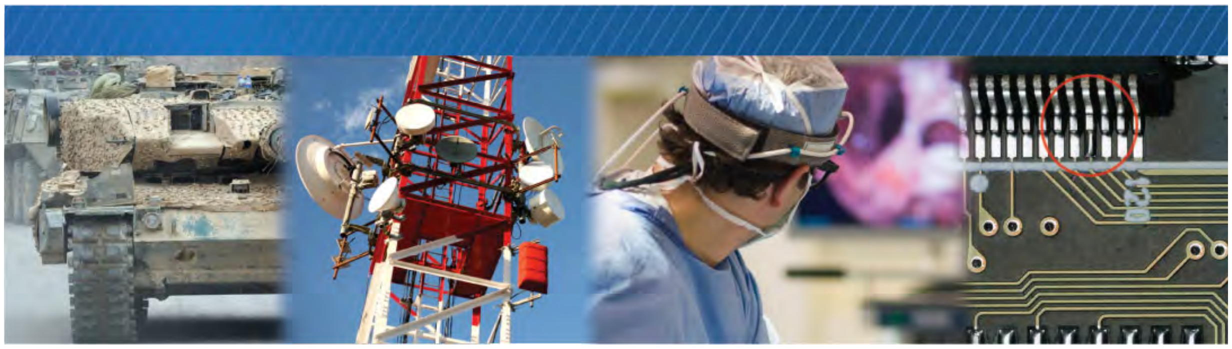
C++

```
class MyClass : PvDeviceEventSink
{
   void OnLinkDisconnected( PvDevice *aDevice )
   {
      // ...
   }
}

MyClass lMyObject;
lDevice.RegisterEvenSink( &lMyObject );
// ...
lDevice.UnregisterEventSink( &lMyObject );
```

C#

```
private void Device_OnLinkDisconnected( PvDevice aDevice)
{
   // ...
}

lDevice.OnLinkDisconnected += new OnLinkDisconnectedEvent(Device_OnLinkDisconnected);
   // ...
lDevice.OnLinkDisconnected -= new OnLinkDisconnectedEvent(Device_OnLinkDisconnected);
```

Chapter 8



Technical Support

On the Pleora Support Center, you can:

- Download the latest software and firmware.
- Log a support issue.
- View documentation for current and past releases.
- Browse for solutions to problems other customers have encountered.
- Read knowledge base articles for information about common tasks.

To visit the Pleora Support Center

- Go to supportcenter.pleora.com.

  Most material is available without logging in to a Support Center account. To access software and firmware downloads, in addition to other content, log in to the Support Center. If you do not have an account, click **Request Account**.

  Accounts are usually validated within one business day.