

PLEORA TECHNOLOGIES INC.



Python API Quick Start Guide



Table of Contents

1 About this Guide.....	3
1.1 What this Guide Provides	3
1.2 Related Documents	3
2 Introducing the eBUS SDK Python API	5
2.1 About the eBUS SDK Python API	5
2.2 eBUS SDK Licenses	5
2.3 System Requirements	6
3 eBUS SDK Python Packages	9
4 Installing the eBUS SDK Python and its dependency packages	10
5 Using the Sample Code	19
5.1 Overview: System Components	19
5.2 Description of Samples	19
6 Code Walkthrough: Acquiring Images with the eBUS SDK.....	23
6.1 Accessing the Python Sample Code	23
6.2 Classes Used in the PvStreamSample	23
6.3 Module Imports	24
6.4 PvStreamSample	25
6.5 The connect_to_device Function	25
6.6 The open_stream Function	26
6.7 The configure_stream Function.....	27
6.8 The configure_stream_buffers Function	27
6.9 The acquire_images Function.....	28
7 Troubleshooting	34
7.1 Troubleshooting Tips	34

Document Number and Revisions

Document ID: QSG-0027

Revision	Reason for Revision	Date
1.0	Initial Release with eBUS SDK 7.0.0	January 2026
2.0	Release with eBUS SDK 7.0.1	June 2026

1 About this Guide

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

1.1 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 Python API) and an overview of the system requirements.

You can use the Python sample applications to see how the Python 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 Python sample code to verify that your system is working properly (that is, determine whether there is a problem with your code or your equipment).

You can also consult the eBUS SDK Python API Help files for further information regarding the Python API itself.

These help files are HTML-based and these files are co-located with the standard eBUS SDK files:

On Linux:

/opt/pleora/ebus/<distribution_targeted>/share/doc/sdk_python/index.html

On Windows:

C:\Program Files\Pleora Technologies Inc\eBUS\Documentation\python\index.html

For troubleshooting information and technical support, refer to the “Troubleshooting” section.

1.2 Related Documents

eBUS SDK Related Documents

Guides are complemented by the following Pleora Technologies documents, which are available on the Pleora Technologies Support Center (supportcenter.pleora.com):

- [C++ API Quick Start Guide](#)
This guide provides you with the information you need to install the eBUS SDK (which lets you use the eBUS SDK C++ API) and an overview of the system requirements.
- [Docker Support for eBUS User Guide](#)
This guide provides instruction on how to get started with using eBUS SDK in a Docker environment. It also provides basic instruction on how to set up Docker on a Ubuntu-based system.
- [Getting Started with eBUS Edge](#)
The eBUS Edge API (introduced in eBUS SDK 7.0) allows developers to create a software-based GigE Vision transmitter device. eBUS Edge is fully compliant with GigE Vision and GenICam and will work with any GigE Vision and GenICam compliant third-party image processing systems or software.
- [eBUS SDK 7.0 \(and later\) Licensing](#)
This knowledge base article explains the eBUS SDK license structure, explains how to obtain a license, and provides activation instructions. If you are experiencing difficulty activating your license, please review the troubleshooting steps at the end of this publication.
- [Linux Quick Start Guide](#)
This guide provides you with the steps to use the eBUS SDK on the Linux operating system, on a

Linux x86_64, or ARM platform. This guide is intended for novice Linux users, although advanced Linux users may be interested in some of the eBUS SDK-specific elements of this guide.

- [.NET API Quick Start Guide](#)
This guide provides you with instructions for compiling and using the sample code are provided, along with an overview of the basic calls that can be used to build custom applications using the eBUS SDK .NET API.
- [eBUS Player User Guide](#)
This guide provides in-depth details about setting up and using the eBUS Player software application to control your GigE Vision or USB3 Vision compliant video transmitters (cameras) and receivers.
- [eBUS Player Quick Start Guide](#)
This quick start guide provides you with the information you need to start using the eBUS Player application, which lets you control the parameters of your GigE Vision or USB3 Vision compliant device and lets you view imaging video and data.
- [Python API Quick Start Guide](#)
This guide provides you with the information you need to install the eBUS SDK (which lets you use the eBUS SDK Python API) and an overview of the system requirements.
- [Supported Software Ecosystem](#)
This support summary provides an overview of the supported protocols, operating systems, ARM platforms, development environments, and drivers. Pleora is currently supporting eBUS SDK 7.0 and later.

2 Introducing the eBUS SDK Python API

This chapter describes the eBUS SDK Python 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 Python.

2.1 About the eBUS SDK Python API

The **eBUS SDK for Python** is available through three API modules: **eBUS Base Python**, **eBUS Edge Python**, and **eBUS Receive Python**. By utilizing these APIs, users can reliably receive video over **GigE**, **10 GigE**, and **USB 3.0**, with cross-platform portability across **Windows** and **Linux** operating systems.

With an **eBUS SDK Seat License**, developers can create production-ready software applications in the same environment as their end users. This enables quick and easy modification of applications for different media types, while avoiding the complexity of supporting multiple APIs from various vendors.

Compared to SDKs provided by camera vendors, eBUS liberates developers from being tied to a specific camera. Instead, they can choose the device that best suits their application.

2.1.1 eBUS Base

eBUS Base provides a common runtime API, designed to be used in conjunction with the eBUS Edge and eBUS Receive modules.

2.1.2 eBUS Edge for Sensor Devices

eBUS Edge is a software implementation of a full device-level GigE Vision transmitter, without requiring any additional hardware. Adding eBUS Edge to a CPU's software stack turns it into a fully compliant GigE Vision device that supports image transmission and enables the device to respond to control requests from a host controller. eBUS Edge is GigE Vision and GenICam-compliant, meaning end-users can use any standards-compliant third-party image processing system. eBUS Edge currently supports the GigE Vision standard.

2.1.3 eBUS Receive for Host Applications

eBUS Receive 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.

2.2 eBUS SDK Licenses

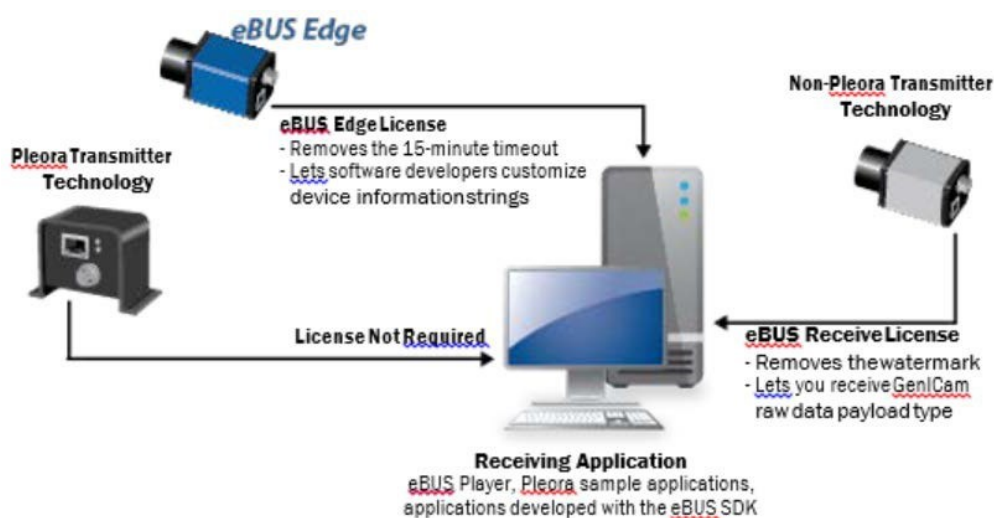
Some components of the eBUS SDK require a Pleora license to remove transmit and receive limitations.

2.2.1 eBUS Receiver License

When a license is not present and you are receiving images from third-party GigE Vision or USB3 Vision transmitter technology, an embossed Pleora watermark will appear on the images that you receive. In addition, you will not be able to receive the GigE Vision or USB3 Vision raw data payload type from the transmitting device.

2.2.2 GigE Vision Devices Created with the eBUS Edge API

When an eBUS Edge license is not present on the eBUS Edge application side, the eBUS receiver application will automatically disconnect from the eBUS Edge device after 15 minutes. In addition, the eBUS Edge device will report hard-coded device information (such as the device model name), instead of your organization's customized information.



2.2.3 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.

2.3 System Requirements

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



You can access installation files from the Pleora Support Center at supportcenter.pleora.com.

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 and 2022.
- All C++ sample project files (.vcxproj) are compatible with Visual Studio 2019 (and later). When you open them with Visual Studio for the first time, you have the option of upgrading.
- All C# sample project files (.csproj) are compatible with Visual Studio 2022.



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 manages 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.

One of the following operating systems:

- For Microsoft Windows
 - Microsoft Windows 11, 64-bit
 - Microsoft Windows 10, 64-bit
- For the x86_64 Linux platform:
 - Ubuntu 24.04 LTS 64-bit
 - Ubuntu 22.04 LTS 64-bit
 - Ubuntu 20.04 LTS 64-bit
 - RHEL 9.7, 64-bit
 - CentOS Stream 9, 64-bit
- For the Linux for ARM platform:
 - NVIDIA Jetson AGX Orin, Jetson Orin NX, Jetson Orin Nano running Jetpack 6.2. (Ubuntu 22.04)
 - NVIDIA Jetson AGX Xavier, Jetson Xavier NX, Jetson AGX Orin, Jetson Orin NX running Jetpack 5.1.4. (Ubuntu 20.04)
 - Raspberry Pi 4B, Raspberry Pi 5 running Raspberry Pi OS (64 bits). (Debian aarch64 GNU/Linux 13 Trixie)
 - ST Micro STM32 MP2
 - Yocto 4.0 (Kirkstone) and Yocto 5.0 (Scarthgap) (on NXP i.MX8, Raspberry Pi 4B / 5, MediaTek Genio 510)

According to the Linux/ARM platform, we provide eBUS Python for the stock Python version of the OS:

- Ubuntu 24.04 Desktop (64-bit), eBUS Python for Python 3.12 is installed with the SDK
- Ubuntu 24.04 for ARM (64-bit), eBUS Python for Python 3.12 is installed with the SDK
- Ubuntu 22.04 Desktop (64-bit), eBUS Python for Python 3.10 is installed with the SDK
- Ubuntu 22.04 for ARM (64-bit), eBUS Python for Python 3.10 is installed with the SDK

- Ubuntu 20.04 Desktop (64-bit), eBUS Python for Python 3.8 is installed with the SDK
- Ubuntu 20.04 for ARM (64-bit), eBUS Python for Python 3.8 is installed with the SDK
- Raspberry Pi Desktop (64-bit), eBUS Python for Python 3.13 is installed with the SDK.



For non-default Python versions for different Linux ARM/x86_64 platforms, consult your Pleora support representative.



Ensure that you uninstall any previous versions of eBUS SDK from your machine before installing eBUS SDK 7.x.

3 eBUS SDK Python Packages

eBUS SDK Python is included in both Windows and Linux eBUS SDK installers. It can be configured in the following ways:

- **eBUS SDK:** Combines both **eBUS Edge Python** and **eBUS Receive Python** modules for comprehensive functionality.
- **eBUS Edge Runtime:** Uses the **eBUS Edge Python** module exclusively.
- **eBUS Receive Runtime:** Uses the **eBUS Receive Python** module exclusively.

4 Installing the eBUS SDK Python and its dependency packages

On Linux:

For full details about installing the eBUS SDK Python on Linux, see the eBUS SDK for Linux Quick Start Guide, available at the Pleora Support Center (supportcenter.pleora.com¹).

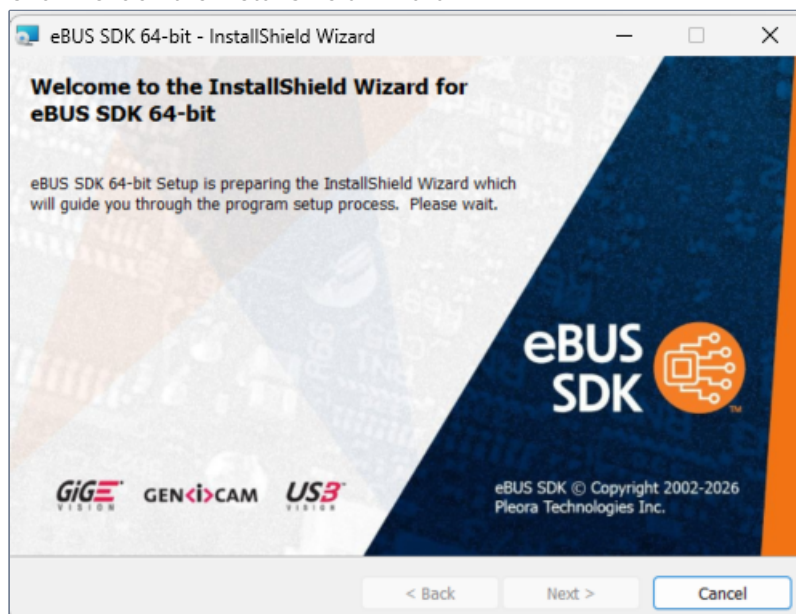
On Windows:

Users must select Python during the eBUS SDK installation. The installer will search for a supported Python version to install the wheel files. Follow the guide below. If users wish to install the Python wheels manually, they can be found in "C:\Program Files\Common Files\Pleora\eBUS".

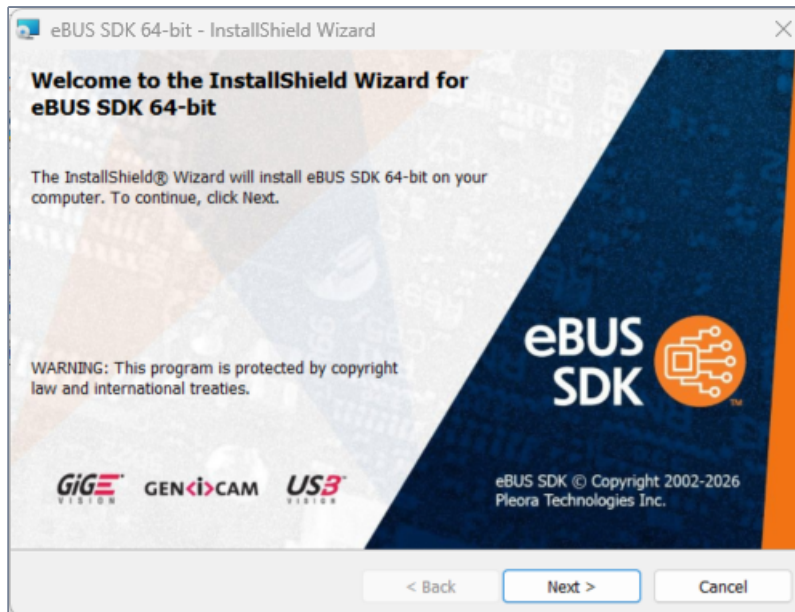
Steps to install eBUS SDK for Windows

- i • Ensure that you uninstall any previous versions of eBUS SDK from your machine.
- Starting with 7.0.0, all eBUS components use a unified Windows installation directory structure, replacing the product-specific folders used in earlier versions. `C:\Program Files\Pleora Technologies Inc\eBUS\`. With this, the license file will now be located in C:\Program Files\Pleora Technologies Inc\eBUS\Licenses

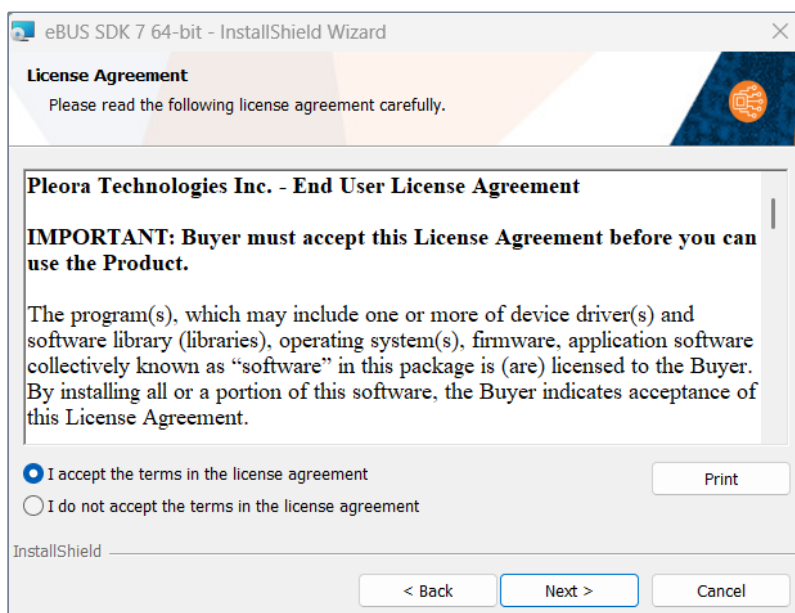
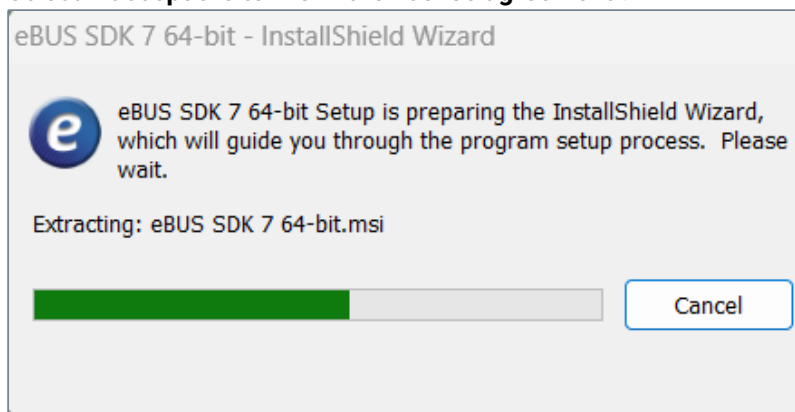
1. Double click on the eBUS SDK .exe (eBUS SDK 64-bit.X.X.X.XXXX.exe) file/icon that was provided to you or you'll have to download it.
2. Click **Next** on the InstallShield Wizard



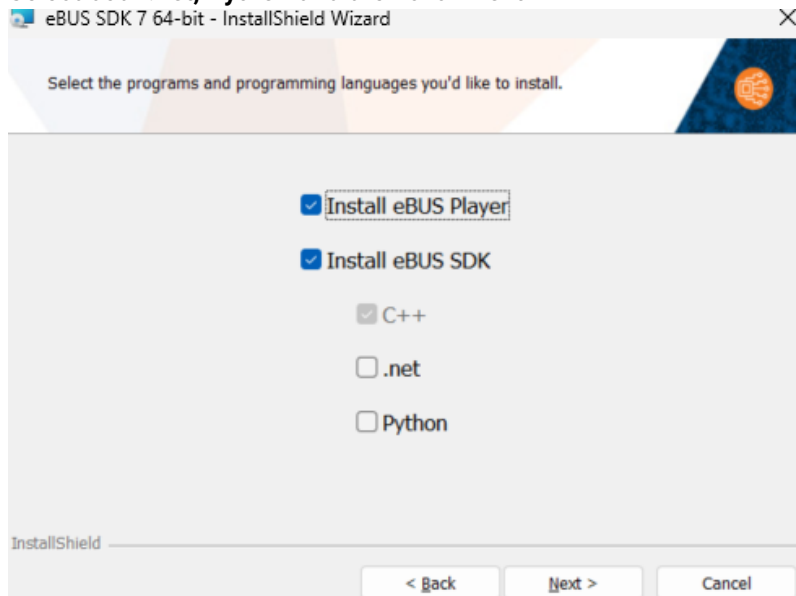
1. <http://supportcenter.pleora.com>

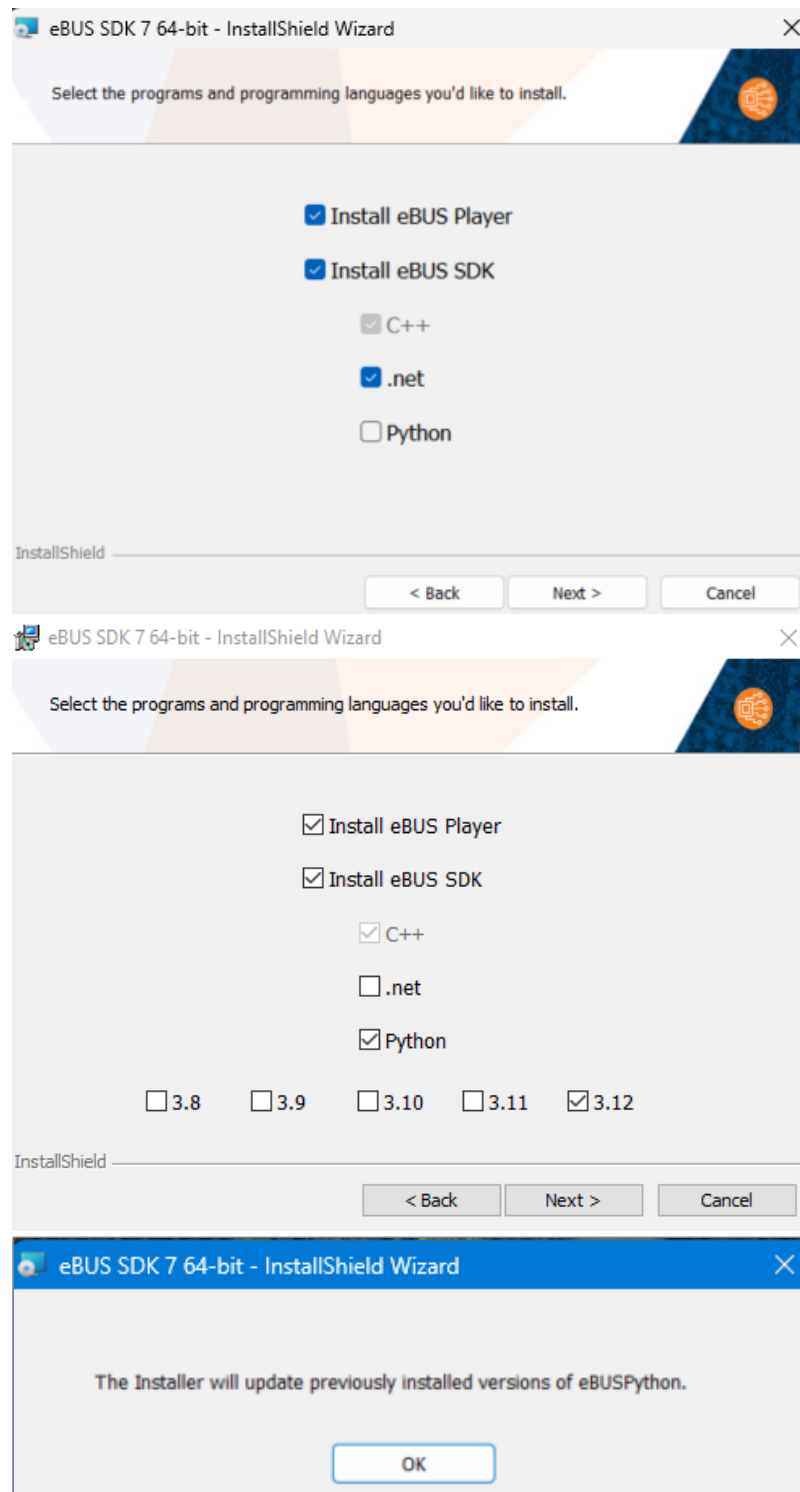


3. Select "I accept the terms in the license agreement".

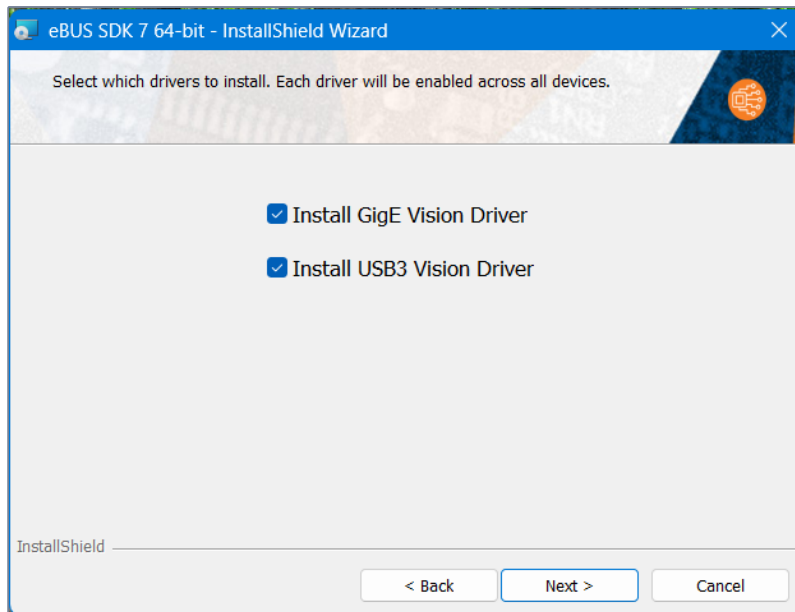


4. Select both **.net**, **Python** and then click **Next**

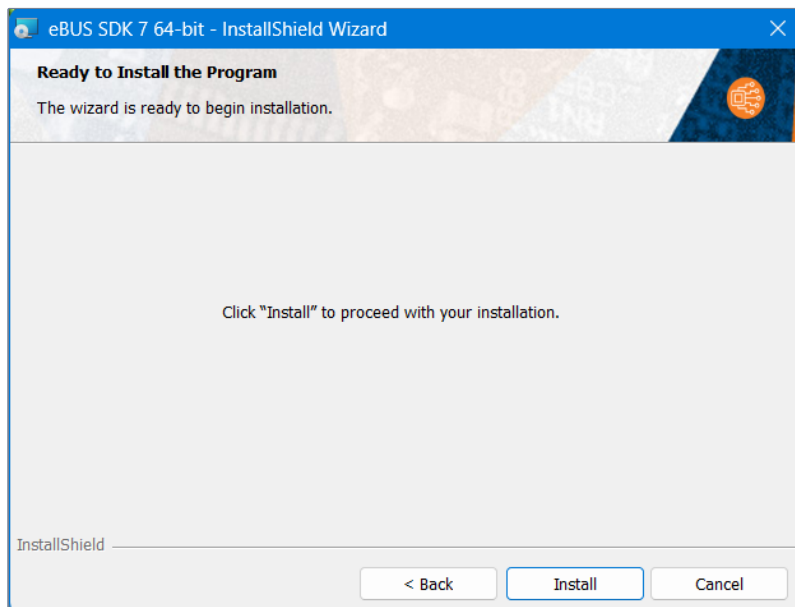


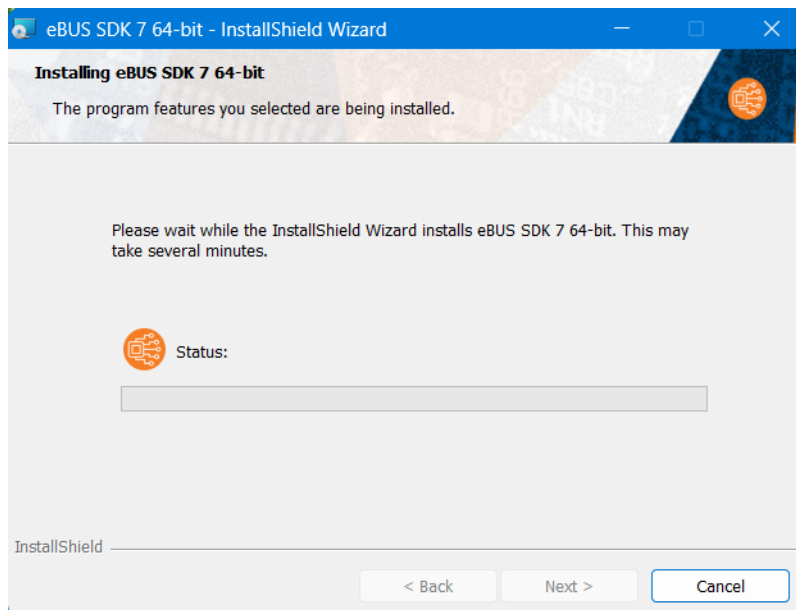


5. Select both **Install GigE Vision Driver** & **Install USB3 Vision Driver**

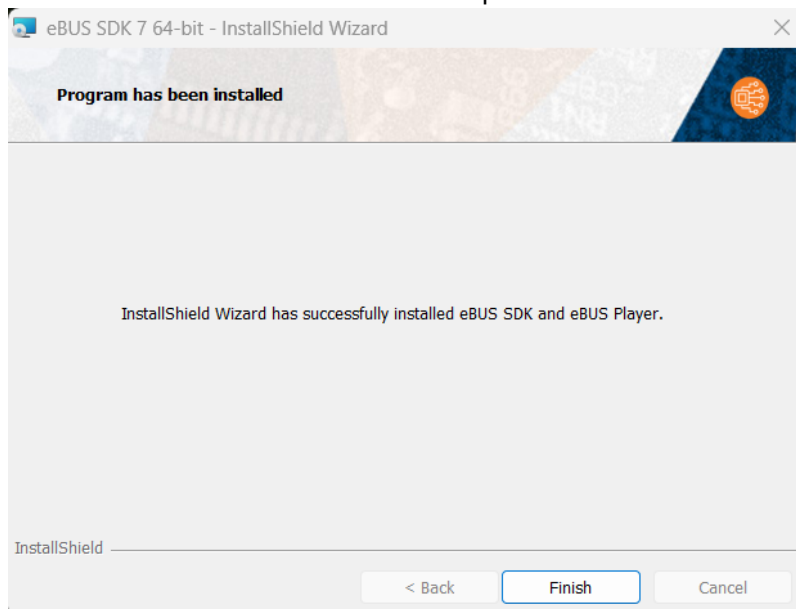


6. Click **Install**

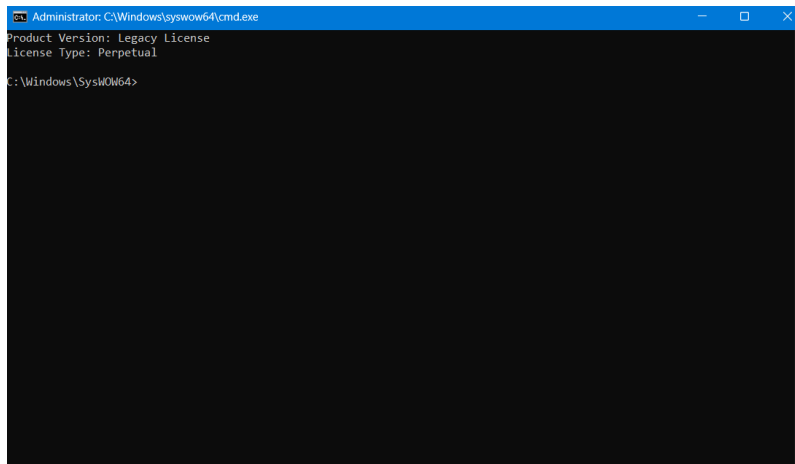




7. Click **Finish** and now the install has completed



8. Close the Command Prompt that will appear with information about Product Version, License Type and where it was installed on your system.



Dependencies:

Install the following:

1. pip
2. numpy < 2
3. opencv-python (optional for some samples)

How to upgrade pip on Windows

From a terminal, run the following command:

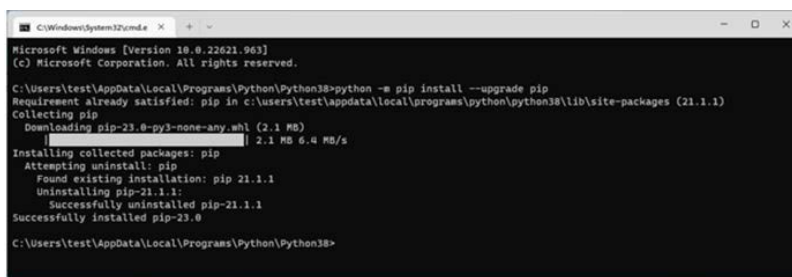
```
python -m pip install --upgrade pip
```

If you have only one Python package installed and you have added the Python to your PATH, you can call python everywhere.

If you have several Python packages installed, you should launch python from the installed location.

From a terminal, run the following command if you use python **3.8**:

```
C:\Users\<username>\AppData\Local\Programs\Python\Python38>python -m pip install --upgrade pip
```



How to install numpy on Windows

From a terminal, run the following command:

```
python -m pip install "numpy<2"
```

If you have several Python packages installed, you need to specify the python executable.

From a terminal run the following command with the default Python path, if you use python **3.8**:

```
C:\Users\<username>\AppData\Local\Programs\Python\Python38>python -m pip install "numpy<2"
```

```
C:\Users\test\AppData\Local\Programs\Python\Python311>python -m pip install "numpy<2"
Collecting numpy<2
  Using cached numpy-1.26.4-cp311-cp311-win_amd64.whl.metadata (61 kB)
Using cached numpy-1.26.4-cp311-cp311-win_amd64.whl (15.8 MB)
Installing collected packages: numpy
Successfully installed numpy-1.26.4
```

(optional) How to install opencv-python on Windows

From a terminal, run the following command:

```
python -m pip install opencv-python
```

If you have several Python packages installed, you need to specify the python executable. From a terminal, run the following command with the default Python installation path, if you use python **3.8**:

```
C:\Users\<username>\AppData\Local\Programs\Python\Python38>py -3.10 -m pip install "numpy<2"
opencv-python
```

This is to make sure opencv version installation supports numpy<2.

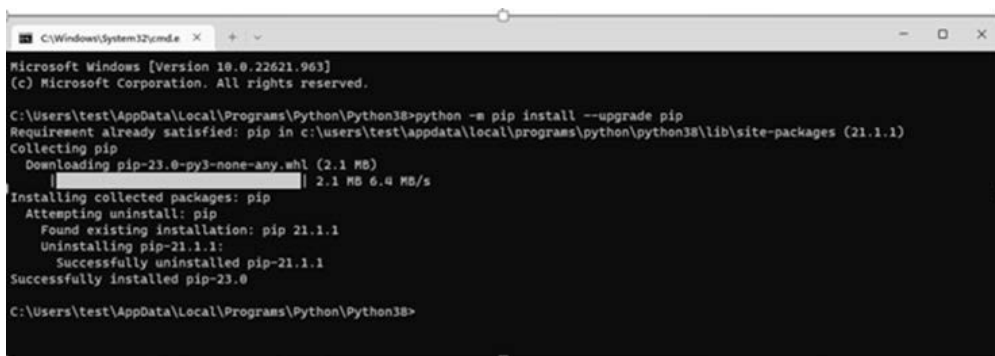
How to install eBUS-Python on Windows

From a terminal, run the following command:

```
python -m pip install ebus_base-7.0.0-<build number>-py<py version>-none-
win_amd64.whl
python -m pip install ebus_edge7.0.0-<build number>-py<py version>-none-win_amd64.whl
python -m pip install ebus_receive-7.0.0-<build number>-py<py version>-none-
win_amd64.whl
```

If you have several Python packages installed, you need to specify the python executable. From a terminal, run the following command with the default Python installation path, if you use python **3.8**:

```
C:\Users\<username>\AppData\Local\Programs\Python\Python38>python -m pip install <path of the
package>\ebus_python-6.3.0-<build number>-py38-none-win_amd64.whl
```



Location of installed eBUS Python on Windows

```
C:\Users\<username>\AppData\Local\Programs\Python\Python<Python
version>\Lib\site-packages\ebus-base
```

```
C:\Users\<username>\AppData\Local\Programs\Python\Python<Python
```

```
version>\Lib\site-packages\ebus-edge
```

```
C:\Users\<username>\AppData\Local\Programs\Python\Python<Python
```

```
version>\Lib\site-packages\ebus-receive
```

For example, the default installation path if you use python 3.8 with eBUS Edge will be:

C:\Users\<username>\AppData\Local\Programs\Python\Python**38**\Lib\site-packages\ebus-edge

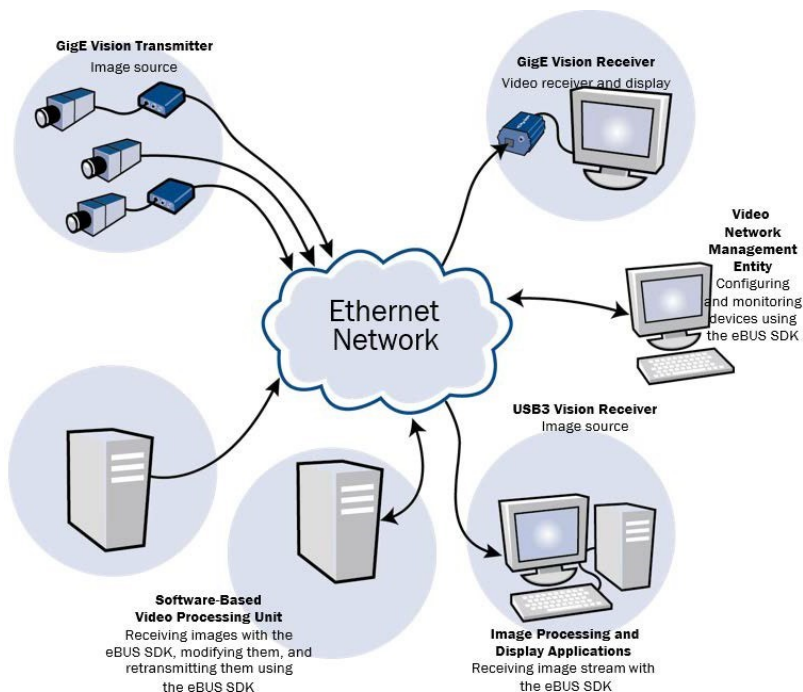
5 Using the Sample Code

To illustrate how you can use the eBUS SDK Python API to acquire and transmit images, the SDK includes sample code that you can use. This chapter provides a description of the sample code.

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

5.1 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.



5.2 Description of Samples

The following samples are available as Python samples in this release.



The following table lists the Python sample applications that are available. For more information about the C++ samples that are also available, see the *eBUS SDK C++ API Quick Start Guide*. For information about the C# and VB.NET samples that are also available, see the *eBUS SDK .NET API Quick Start Guide*.

Table 2: Sample Code

Sample code	Function	Type of application that is created
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. All platforms
Image Streaming		
ReceiveMultiPart	This sample shows how to connect to a GigE Vision device and receive a stream of multi-part payload type. The images that are acquired are displayed on screen using different windows. The SoftDeviceGEVMultiPart sample can be used to instantiate a software GigE Vision Device with a multi-part interface	Command line. All platforms
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. All platforms
MultiSource	This command line sample for GigE Vision devices shows you how to receive images from a GigE Vision device that has multiple streaming sources.	Command line. - All platforms - For GigE Vision devices only
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. All platforms
Discovery and Connection		
DeviceFinder	This sample shows how to detect and enumerate GigE Vision and USB3 Vision devices on the network.	Command line. All platforms

ConnectionRecovery	This sample shows how to automatically recover from connectivity issues, such as accidental disconnects and power interruptions, to build more robustness into your eBUS SDK application.	Command line. All platforms
Configuration and Event Monitoring		
DeviceSerialPort	This sample shows how to send commands to a camera or other device that accepts serial input commands through a compatible Pleora iPORT video interface using the Pleora device's General Purpose Input/Output (GPIO) signals, including UART or BULK.	Command line. All platforms
GenlCamParameters	This sample shows how to enumerate and display the GenlCam features and settings of a GenlCam-compatible device by discovering and accessing the features of the device's node map. The node map is built programmatically from the device's GenlCam XML file.	Command line. All platforms
EventSample	This sample shows how to receive and handle events sent from USB3 Vision and GigE Vision devices including Pleora eBUS Edge through EVENT_CMD and EVENTDATA_CMD.	Command line. All platforms
Device Update		
FirmwareUpdater	This sample shows how to use the PvGenFUUpdater class to perform a firmware update on a device capable of GenlCam firmware update.	Command line. All platforms
eBUS Edge Code Samples		
SoftDeviceGEVSimpl e	This sample shows how to create a basic software GigE Vision device with one streaming source and a single pixel type. A sample test pattern is generated as a streaming source.	Command line. All platforms
SoftDeviceGEV	This sample shows how to create a fully functioning software GigE Vision device with multiple streaming sources and fixed width and height pixel types. A sample test pattern is generated as a streaming source. This sample also illustrates how to implement custom GenApi features and device registers, as well as how to access the GVCP messaging channel to send events and chunk data.	Command line. All platforms

SoftDeviceGEVMultiPart	This sample shows how to use PvSoftDeviceGEV to create a software GigE Vision multi-part transmitter device.	Command line. • All platforms
SoftDeviceGEVChunkData	This sample shows how to create and run a Software GigE Vision device sending data using a payload of type ChunkData.	Command line. • All platforms
SoftDeviceGEVGenDC	This sample shows how to implement GenDC payload type in PvSoftDeviceGEV.	Command line. • All platforms
SoftDeviceGEVTrigger	This sample shows how to use PvSoftDeviceGEV to create different types of triggers in a software GigE Vision transmitter device.	Command line. • All platforms

6 Code Walkthrough: Acquiring Images with the eBUS SDK

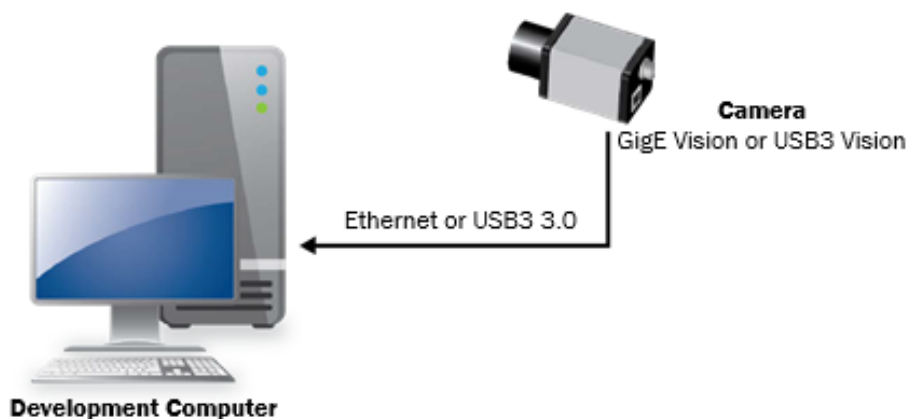
This section walks you through the code contained in **PvStreamSample**. This sample illustrates how to detect available devices, connect to a device, and start an image stream.

6.1 Accessing the Python Sample Code

- On Linux: /opt/pleora/ebus/<distribution_targeted>/share/samples/python_samples/ebus
(Note: You must copy the samples to a user-writable directory before use.)
- On Windows: C:\Program Files\Pleora Technologies Inc\eBUS\Samples\python
(Note: Copy the samples to a non-system directory before use.)

6.1.1 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.



6.1.2 Windows and Linux Support

PvStreamSample can be used on the Windows and Linux operating systems. Platform-specific code is abstracted by **PvSampleUtils.py**.

6.2 Classes Used in the PvStreamSample

PvStreamSample uses the classes listed in the following table.

Table 3: Classes Used in the Sample

Class	eBUS Python Module	Description
PvDeviceInfo	eBUSReceive	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	eBUSReceive	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	eBUSReceive	Provides access to the image stream. Like PvDevice , there are protocol and interface-specific subclasses, PvStreamGEV and PvStreamU3V .
PvBuffer	eBUSBase eBUSReceive (for receive related API)	Represents a block of data from the device, such as an image.
PvResult	eBUSBase	A simple class that represents the result of various eBUS SDK functions and methods.

6.3 Module Imports

1. The following modules are required by the sample. Please note that the PvSampleUtils.py module provides some basic helper and multi-platform routines:

```
import numpy as np
import lib.PvSampleUtils as psu
import lib.PvGenDCUtils as gendc
from ebus_base.eBUSBase import *
from ebus_receive.eBUSReceive import *
import lib.PvSampleUtils as psu
```

2. Then call the main class to utilize eBUS SDK classes. See python docs to know where the classes can be called.

6.4 PvStreamSample

The **PvStreamSample** allows the user to select a device, connect to a device (**connect_to_device**), start the image stream (**open_stream**, **configure_stream**, **configure_stream_buffers**), and process the image stream (**acquire_images**). To perform these tasks, the following objects are required:

- A **PvDeviceInfo** object, which indicates the device that the user has selected for streaming.
 - A **PvDevice** object, which allows the user to control the selected device.
 - A **PvStream** object, which is used to receive the image stream for the selected device.
- kb.start()**, **kb.getch()** and **kb.kbhit()** are platform-independent helper functions. These functions are provided in the **PvSampleUtils** module.

6.4.1 Python

```
print("PvStreamSample:")

connection_ID = psu.PvSelectDevice()
if connection_ID:
    device = connect_to_device(connection_ID)
    if device:
        stream = open_stream(connection_ID)
        if stream:
            configure_stream(device, stream)
            buffer_list = configure_stream_buffers(device, stream)
            acquire_images(device, stream)
            buffer_list.clear()

            # Close the stream
            print("Closing stream")
            stream.Close()
            PvStream.Free(stream);

            # Disconnect the device
            print("Disconnecting device")
            device.Disconnect()
            PvDevice.Free(device)

print("<press a key to exit>")
kb.start()
kb.getch()
kb.stop()
```

6.5 The connect_to_device Function

connect_to_device establishes a connection with the 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.

i It is important that objects allocated with **CreateAndConnect** be freed with **PvDevice.Free**, as shown in **PvStreamSample** routine.

i 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.

A **PvDevice** object is returned and can now be used to control the device and initiate streaming.

6.5.1 Python

```
def connect_to_device(connection_ID):
    # Connect to the GigE Vision or USB3 Vision device
    print("Connecting to device.")
    result, device = PvDevice.CreateAndConnect(connection_ID)
    if device == None:
        print(f"Unable to connect to device: {result.GetCodeString()}")
    ({result.GetDescription()})
    return device
```

6.6 The open_stream Function

open_stream initiates the image stream. Again, the sample 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.

A pointer to the **PvStream** object is returned and can now be used to receive images as **PvBuffer** objects.

```
def open_stream(connection_ID):
    # Open stream to the GigE Vision or USB3 Vision device
    print("Opening stream from device.")
    result, stream = PvStream.CreateAndOpen(connection_ID)
    if stream == None:
        print(f"Unable to stream from device. {result.GetCodeString()}")
    ({result.GetDescription()})
```

```
return stream
```

6.7 The `configure_stream` Function

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.

i 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.

i 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 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.

6.7.1 Python

```
def configure_stream(device, stream):
    # If this is a GigE Vision device, configure GigE Vision specific streaming
    # parameters
    if isinstance(device, PvDeviceGEV):
        # Negotiate packet size
        device.NegotiatePacketSize()
        # Configure device streaming destination
        device.SetStreamDestination(stream.GetLocalIPAddress(),
        stream.GetLocalPort())
```

6.8 The `configure_stream_buffers` Function

configure_stream_buffers 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 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.

6.8.1 Python

```
def configure_stream_buffers(device, stream):
    buffer_list = []
    # Reading payload size from device
    size = device.GetPayloadSize()

    # Use BUFFER_COUNT or the maximum number of buffers, whichever is smaller
    buffer_count = stream.GetQueuedBufferMaximum()
    if buffer_count > BUFFER_COUNT:
        buffer_count = BUFFER_COUNT

    # Allocate buffers
    for i in range(buffer_count):
        # Create new pvbuffer object
        pvbuffer = PvBuffer()
        # Have the new pvbuffer object allocate payload memory
        pvbuffer.Alloc(size)
        # Add to external list - used to eventually release the buffers
        buffer_list.append(pvbuffer)

    # Queue all buffers in the stream
    for pvbuffer in buffer_list:
        stream.QueueBuffer(pvbuffer)
    print(f"Created {buffer_count} buffers")
    return buffer_list
```

6.9 The `acquire_images` Function

In the **acquire_images** function method, we acquire images from the device.

First, the sample retrieves an array of GenICam features that will be used to control the device. These features are defined in the GenICam XML file that is present on all GigE Vision and USB3 Vision devices. Then, it maps two GenICam commands from the array to local variables that will be used later to start and stop the stream.

Next, it retrieves an array of GenICam features that represent the stream parameters. It maps two GenICam floating-point values that represent stream statistics, which will later be used to display the data rate and bandwidth during image acquisition.

6.9.1 Python

```
def acquire_images(device, stream):
    # Get device parameters need to control streaming
    device_params = device.GetParameters()

    # Map the GenICam AcquisitionStart and AcquisitionStop commands
    start = device_params.Get("AcquisitionStart")
    stop = device_params.Get("AcquisitionStop")

    # Get stream parameters
    stream_params = stream.GetParameters()

    # Map a few GenICam stream stats counters
    frame_rate = stream_params.Get("AcquisitionRate")
    bandwidth = stream_params[ "Bandwidth" ]
```

...

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



For GigE Vision devices, **StreamEnable** sets the **TLPParamsLocked** feature, which prevents changes to the streaming-related parameters during image acquisition. For USB3 Vision devices, it sets the **TLPParamsLocked** feature, configures the USB driver for streaming, and sets the stream enable bit on the device.

6.9.2 Python

```
# Enable streaming and send the AcquisitionStart command
print("Enabling streaming and sending AcquisitionStart command.")
device.StreamEnable()
start.Execute()
```

In the next section, we set up a doodle that will indicate to the user that images are being acquired. The doodle will animate every time a buffer is returned using **RetrieveBuffer** (regardless of whether we got

an image or a timeout) until the user presses a key. We also initialize variables to access GenICam statistics (block count, acquisition rate, and bandwidth) that were retrieved earlier.

Next, we start the loop, 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 (**result**) to determine that a buffer has been retrieved. If a buffer has been retrieved, then it checks the value of the **PvResult** object (**operational_result**) to verify the acquisition operation (for example, it checks if the operation timed out, had too many resends, or was aborted.)

6.9.3 Python

```
print("\n<press a key to stop streaming>") kb.start()
while not kb.is_stopping():
    # Retrieve next pvbuffer
    result, pvbuffer, operational_result = stream.RetrieveBuffer(1000)
    if result.IsOK():
        if operational_result.IsOK():
            #
            # We now have a valid pvbuffer. This is where you would typically
            process the pvbuffer.
            #
            - .....
            .....
            # ...

            result, frame_rate_val = frame_rate.GetValue()
            result, bandwidth_val = bandwidth.GetValue()

            print(f"{doodle[doodle_index]} BlockID: {pvbuffer.GetBlockID()}",
end='')

            payload_type = pvbuffer.GetPayloadType()
```

Now that we have obtained a **PvBuffer** with an image, we display some general statistics retrieved from the device, including block ID, width, height, and bandwidth. This is the point at which your application would typically process the buffer. Then, we discard the image and requeue the **PvBuffer** object in the input queue by calling **PvStream.QueueBuffer**.

It is important to note that the stream may not contain an image, so we use the **PvPayloadType** enumeration to check that an image is included. For example, **PvPayloadType** can be **PvPayloadTypeImage**, **PvPayloadTypeUndefined** (an undefined or non-initialized payload type), or **PvPayloadTypeChunkData**, **PvPayloadTypeRawData**, or **PvPayloadTypeMultiPart**.

6.9.4 Python

```
if payload_type == PvPayloadType.PvPayloadTypeImage.value:
    image = buffer_for_processing.GetImage()
```

```

image_data = image.GetDataPointer()
print(f" W: {image.GetWidth()} H: {image.GetHeight()} ", end='')

if opencv_is_available:
    if image.GetPixelFormat() == PvPixelFormat.PvPixelFormatMono8.value:
        display_image = True
    if image.GetPixelFormat() == PvPixelFormat.PvPixelFormatRGB8.value:
        image_data = cv2.cvtColor(image_data, cv2.COLOR_RGB2BGR)
        display_image = True

    if display_image:
        cv2.imshow("stream", image_data)
    else:
        if not warning_issued:
            # display a message that video only display for
            Mono0 / RGB8 images
            displayed", end='\r')

            print(f" ")
            print(f" Currently only Mono8 / RGB8 images are

            print(f"")
            warning_issued = True

        if cv2.waitKey(1) & 0xFF != 0xFF:
            break

    elif payload_type == PvPayloadType.PvPayloadTypeChunkData.value:
        print(f" Chunk Data payload type with
{buffer_for_processing.GetChunkCount()} chunks", end='')

    elif payload_type == PvPayloadType.PvPayloadTypeRawData.value:
        print(f" Raw Data with
{buffer_for_processing.GetRawData().GetPayloadLength()} bytes", end='')

    elif payload_type == PvPayloadType.PvPayloadTypeMultiPart.value:
        print(f" Multi Part with
{buffer_for_processing.GetMultiPartContainer().GetPartCount()} parts", end='')

    else:
        print(" Payload type not supported by this sample", end='')

        print(f" {frame_rate_val:.1f} FPS {bandwidth_val / 1000000.0:.1f}
Mb/s ", end='\r')
    else:
        # Non OK operational result
        print(f"{doodle[ doodle_index ]} {operational_result.GetCodeString()}
", end='\r')
        # Re-queue the pvbuffer in the stream object
        stream.QueueBuffer(pvbuffer)
    else:
        # Retrieve pvbuffer failure
        print(f"{doodle[ doodle_index ]} {result.GetCodeString()} ",
end='\r')

```

If **operational_result** 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, an error

message is presented and we also re-queue the **PvBuffer** object back to the **PvStream** object so it can be used again.

6.9.5 Python

```

else:
    # Non OK operational result
    print(f"{doodle[ doodle_index ]} {operational_result.GetCodeString()}"
", end='\r')
    # Re-queue the pvbuffer in the stream object
    stream.QueueBuffer(pvbuffer)

...

```

If **result** returns something other than **OK**, a **PvBuffer** object was not retrieved and therefore there is no **PvBuffer** to requeue. In this case, the error message is also presented to the user.

6.9.6 Python

```

else:
    # Retrieve pvbuffer failure
    print(f"{doodle[ doodle_index ]} {result.GetCodeString()}"
", end='\r')

doodle_index = (doodle_index + 1) % 6

```

The remainder of the sample is used to stop acquisition and clean up resources when the user presses a key. First, we execute the GenICam **AcquisitionStop** command (**stop**). Then, we disable the stream.

-  For GigE Vision devices, **StreamDisable** resets the **TLPParamsLocked** feature, which allows changes to the streaming-related parameters to occur.
- For USB3 Vision devices, **StreamDisable** resets the **TLPParamsLocked** feature and sets the stream enable bit on the device.

6.9.7 Python

```

kb.stop()
if opencv_is_available:
    cv2.destroyAllWindows()

# Tell the device to stop sending images.
print("\nSending AcquisitionStop command to the device")
stop.Execute()

```

```
# Disable streaming on the device
print("Disable streaming on the controller.")
device.StreamDisable()
```

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.

i 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.

6.9.8 Python

```
# Abort all buffers from the stream and dequeue
print("Aborting buffers still in stream")
stream.AbortQueuedBuffers()
while stream.GetQueuedBufferCount() > 0:
    result, pvbuffer, lOperationalResult = stream.RetrieveBuffer()
```

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.

i 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 queued the next time the stream is enabled.

7 Troubleshooting

This chapter provides you with troubleshooting tips and recommended solutions for issues that can occur when using the eBUS SDK Python 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.

7.1 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 4: Troubleshooting Tips

Symptom	Possible cause	Resolution
ModuleNotFoundError: No module named 'ebus_<module>	Missing eBUS Python modules	Install eBUS Python packages. Can be installed using eBUS SDK Installer. If eBUS SDK is fully installed and still missing: Windows: Navigate to C:\Program Files\Common Files\Pleora\eBUS\ and you can find and install eBUS Python wheel files. Linux: Navigate to /opt/pleora/ebus_sdk/<platform>/lib/ eBUSPython/ and you can find and install eBUS Python wheel files. <pre>python3 -m pip install ebus_<module>_*.whl</pre>

ImportError: DLL load failed while importing _ebus_<module>: The specified module could not be found.	Missing required dependencies	For complete SDK Libraries, install eBUS SDK. For runtime application only, install eBUS Edge Runtime or eBUS Receive Runtime.
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. Retry 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.

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.
Dropped packets: eBUS Player, 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 <i>Configuring Your Computer and Network Adapters for Best Performance Application Note</i>, available on the Pleora Support Center.

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.

Copyright Information

Copyright © 2026 Pleora Technologies Inc.

These products are not intended for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Pleora Technologies Inc. (Pleora) customers using or selling these products for use in such applications do so at their own risk and agree to indemnify Pleora for any damages resulting from such improper use or sale.

Trademarks

VIVOEPlayer, PureGEV, eBUS, iPORT, vDisplay, AutoGEV, AutoGen, AI Gateway, eBUS Studio, Vaira and all product logos are trademarks of Pleora Technologies. Third party copyrights and trademarks are the property of their respective owners.

Notice of Rights

All information provided in this manual is believed to be accurate and reliable. No responsibility is assumed by Pleora for its use. Pleora reserves the right to make changes to this information without notice. Redistribution of this manual in whole or in part, by any means, is prohibited without obtaining prior permission from Pleora.

Document ID: QSG-0027