Stream Control

Using eBUS™ Universal Pro and User Mode Data Receiver in eBUS SDK Version 3.1 (and Later)

Application Note

The eBUS™ SDK provides high performance and low latency GigE Vision block data streaming through the eBUS Universal Pro driver and the user mode data receiver. In most cases, the system performs well using the default stream configuration settings. If you are experiencing streaming errors, this guide will help you to understand and address the errors. This guide also provides guidelines that help you achieve maximum system performance.

The following topics are covered in this application note:

This application note is for systems that use the eBUS Universal Pro driver and the user mode data receiver in eBUS SDK version 2.1.0 (and later). The content of this application note was last validated with eBUS SDK version 6.2.5.

This guide does not apply to systems that use the legacy eBUS Universal or Optimal drivers in any pre-2.1.0 version of the eBUS/PureGEV SDK or the Vision SDK. Although some parameter names are shared between the new and legacy drivers/data receivers, the definitions and their effects could be different.

Pleora Technologies

Changes to the eBUS SDK

This guide is based on eBUS SDK 6.2.5. The following eBUS SDK changes have been introduced in subsequent releases. Please keep these changes in mind as you use this guide:

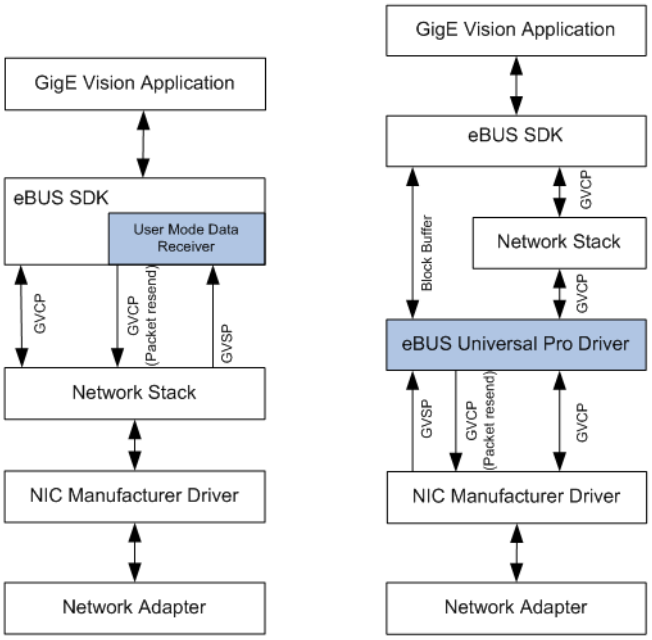
- eBUS SDK 3.0.0. Removed the PacketTimeout parameter, as legacy eBUS drivers have been removed from the eBUS SDK.
- eBUS SDK 3.1.17. Added the **OnlyStartOnFirstPacket** parameter. See "OnlyStartOnFirstPacket" on page 17.
- eBUS SDK 4.x. Added the **ExtraBlockCompletion** parameter. Removed the **ForceMissingPacketsAtNextBlock** parameter.
- eBUS SDK 6.0.1. Added the NO_LICENSE error.

About the Data Receiver

The term data receiver refers to the SDK components that handle the GigE Vision Stream Protocol (GVSP). A data receiver can be either of the following components:

- User mode data receiver (sometimes referred to as the network stack or manufacturer driver). Uses only the driver provided by the network interface card (NIC) manufacturer, not a Pleora driver. The data receiver resides only in the user mode of the operating system.
- eBUS Universal Pro driver. Uses the Pleora eBUS Universal Pro filter driver on top of the NIC's manufacturer driver. Portions of the data receiver reside in both the kernel and user mode of the operating system.

The following illustration shows the data receiver in the networking data path in a computer.



Checking the Stream Using the eBUS Player Application

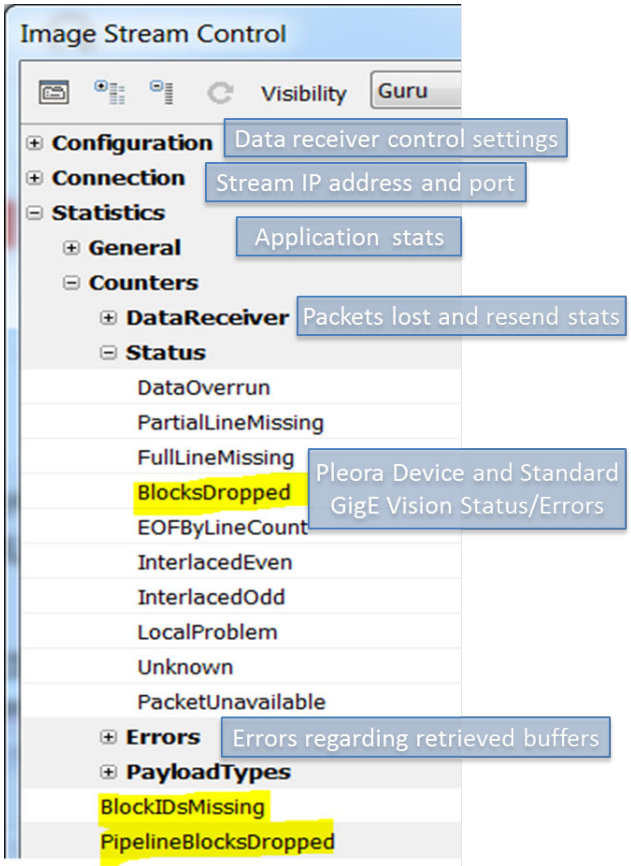
To ensure that your system (that is, the computer, camera, and network/switch) is streaming properly, we recommend that you start with the eBUS Player application, or the PvStreamSample or PvPipelineSample applications of eBUS SDK and verify that images are streaming as expected. If images are not streaming as expected, you can adjust the settings and see the results immediately. After you determine the best settings, you can modify your application's code to match these settings.

Please note that when you click **Save Preferences** on the **Tools** menu in eBUS Player, the application does not save the stream configuration settings (instead, this option saves the eBUS Player preferences as your default eBUS Player settings).

To ensure the same changes appear each time eBUS Player connects to the GVSP transmitter during your testing, we recommend that you save the changes to a configuration file (.pvcfg) and load this file each time you connect. You can save a configuration file by clicking **File** > **Save**.

Run the eBUS Player application and start the streaming, then click the Image stream control button.

The Image Stream Control window opens (note that the **Visibility** should be set to **Guru** to display the full list).



All of the statistic counters are accumulated starting from the **AcquisitionStart** command in the **Device Control** window or by selecting the **Reset** command under the **General** category in the **Image Stream Control** windows.

Blocks Missing

In the image above, the three highlighted status and counters capture the missing blocks information.

BlocksIDsDropped indicate that the Pleora video interface dropped blocks at the pixel buss and the dropped block was not grabbed to the onboard memory. The number of blocks dropped is not counted and the BlocksDropped status bit is set to the next grabbed block. The BlockIDs of the grabbed frames before and after the dropped frames are consecutive. This occurs when the video interface is unable to send out the block data over Ethernet fast enough to keep up with the block data that is coming from the camera head. This issue can happen when the data rate from the camera head is close to the link bandwidth and the application does not use jumbo packets. This issue can be corrected by using jumbo packets, for example setting the **GevSCPSPacketSize** to 8976 (keeping in mind that the NIC and all of the network equipment along the data path must support jumbo packets up to 9K).

GevSCPSPacketSize is deprecated in Standard Features Naming Convention (SFNC) version 2.1.
It is mapped to **DeviceStreamChannelPacketSize**.

Each block grabbed to the onboard memory of the Pleora video interface has its own BlockID and the BlockIDs for subsequent blocks are consecutive. Blocks are transferred out in FIFO mode. If the BlockIDs that are retrieved are not consecutive, blocks could be missing at one of the following locations:

- **BlockIDsMissing**: The data receiver does not receive those blocks. This can happen when the NIC could not handle the traffic and the computer is running low on CPU resources or when the buffers are not queued/ retrieved from PvStream.
- **PipeLineBlocksDropped**: The PvPipeline has no free buffer to queue to the driver.

There are Blocks Missing related counters under the Errors category in the Image Stream Control window. The Errors category indicates the errors identified for received blocks.

Checking the Stream in Your Application

Accessing the Stream Control Configuration Settings and Statistic Counters

From your application, you can access all of the stream control configuration settings and statistic counters using the same method. The following code provides an example of how to retrieve the **FullLineMissing** counter value.

```
uint64_t lFullLineMissingVal = 0;
PvGenParameterArray *lStreamParams = lStream.GetParameters();
lStreamParams->GetIntegerValue( "FullLineMissing", lFullLineMissingVal);
```

The statistic counters accessed from PvStream class are accumulated.

The **PvStreamRaw** class (available in eBUS SDK version 2.x and 3.x) does not use a dynamic GenICam interface for configuration and statistics access. Instead, the **PvStreamRaw** class accesses the stream configuration settings and statistic counters through direct API calls. For example, **PvStreamRaw::GetFirstPacketTimeout()**, **PvStreamRaw::GetStatistics()**, and **PvStreamRaw::GetResultTimeout()**. For all of the settings and counters you can find the corresponding API name in the **PvStreamRaw** and **PvStatistics** classes. The **PvStreamRaw** and **PvStatistics** classes contain all of the API calls, some of which are for the deprecated parameters and counters. If you use a deprecated API call in the Microsoft Windows® operating system, a warning message will appear in the Microsoft Visual Studio® console output panel at build time.

The **PvStreamRaw** class is removed in eBUS SDK 4.x (and later).

Checking the Errors and Statistic Counters for Each Buffer

You should always check the returned result for errors when retrieving a block.

1. When **PvPipeline** is used:

```
PvResult lResult = PvPipeline::RetrieveNextBuffer( PvBuffer** aBuffer,
                                    PvUInt32   aTimeout = 0xFFFFFFFF,
                                               PvResult*  aOperationResult);
```

2. When **PvStream** and **PvBuffer** are used directly through the **PvStream::QueueBuffer**() call:

```
PvResult lResult = PvStream::RetrieveBuffer( PvBuffer** aBuffer,
                                                    PvResult*  aOperationResult,
                                    PvUInt32   aTimeout = 0xFFFFFFFF);
```

The following table provides details about the IResult Result codes.

Table 1: **IResult** Result Codes

Return value PvResult::	Description	Reasons
OK	Buffer is returned. Check the **aOperationResult**	N/A
TIMEOUT	No buffer is returned. Call the retrieve buffer function again.	The aTimeout expired and no buffer was returned from the data receiver. When aTimeout = 0 and there is no buffer ready to be retrieved, the API call returns with this error immediately. When aTimeout = 0xFFFFFFFF, the **DefaultBlockTimeout** determines the timeout value. We recommend that you set aTimeout to be longer than the **RequestTimeout**. To enable the infinite timeout for the retrieve buffer function calls, set aTimeout =0xFFFFFFFF and DefaultBlockTimeout = 0xFFFFFFFF.
NOT_CONNECTED	No buffer is returned.	**PvStream::Open**() has not been called. This code does not come from PvPipeline::RetrieveNextBuffer()
NO_MORE_ITEM	No buffer is returned.	A buffer has not been queued into the data receiver. This code does not come from the PvPipeline::RetrieveNextBuffer()

The errors described in Table 1 are retrieve buffer API errors. They are not generated by the data receiver and have no corresponding stream statistic **Errors** counters.

As of eBUS SDK version 2.1.0 (and later), **DefaultBlockTimeout** is deprecated and set to **invisible** from the stream configuration settings. For backwards compatibility, it can still be accessed programmatically through the **PvStream** parameter array. A future eBUS SDK release will remove this parameter, so the timeout is directly determined by the value of the *aTimeout* argument.

The arguments **aOperationResult** and **aBuffer** are only valid and should only be used when **IResult** returns with **OK**.

Data Receiver Result and Error Code Descriptions

Table 2: **aOperationResult**/Data Receiver Result Codes

***Data receiver* result code PvResult:: /Stream statistic error counter**	**Buffer valid?**	**Descriptions and possible reasons**
OK	Yes	N/A
NOT_INITIALIZED / ResultNotInitialized	No	An empty buffer is returned. This happens when: • $Block_n$ is received and filled in to $buffer_n$ and the first packet of $block_{n+2}$ is received and filled in to $buffer_{n+2}$. If the **ForceMissingPacketsAtNextBlockStart** parameter is enabled, then $buffer_{n+1}$ (which is reserved for $block_{n+1}$) is set with this result code. • Or, when the **IResult** returned from the retrieve buffer function call is not OK, and you check the **aOperationResult**.
TIMEOUT / ResultTimeout	No	The **FirstPacketTimeout** or **RequestTimeout** has elapsed and no packet has been received for the retrieved buffer. **1.** The value of **FirstPacketTimeout** or the **RequestTimeout** is too short. **2.** A firewall is blocking packets. **3.** The anti-virus software is unable to forward the packet, which causes the network stack to drop the packet. **4.** The device is not sending packets. In this case the **Network activity** LED on the RJ45 connector blinks slowly.

Table 2: **aOperationResult**/Data Receiver Result Codes (Continued)

***Data receiver* result code PvResult:: /Stream statistic error counter**	**Buffer valid?**	**Descriptions and possible reasons**
MISSING_PACKETS / ResultMissingPackets	Potentially	The **InterPacketTimeout** or **RequestTimeout** has expired or the **ForceMissingPacketsAtNextBlockStart** or **ExtraBlockCompletion** takes effect and some packets are received for the retrieved buffer. • **RequestMissingPackets** is set to false. **RequestMissingPackets** is set to true but: • The GVSP transmitter does not support packet resends. • The NIC's manufacturer driver could not handle the traffic. Upgrading the manufacturer driver may help. • The system is operating at maximum resources and a glitch on the operating system has produced a variation in the packet timing that results in a dropped packet. • The number of receive buffers (also referred to as receive descriptors) or the corresponding setting on the NIC's manufacturer driver is too small. To change this setting, open the Windows Device Manager, access the NIC's properties, and increase the value. • Some network components (Ethernet cable or network switch) are faulty. Replace or remove the components one by one to see which part caused the problem. • **GevSCPSPacketSize** is set to a jumbo packet size while the NIC does not (or is not configured to) support the jumbo packet size.
BUFFER_TOO_SMALL / ResultBufferTooSmall	Potentially	The buffer size is not large enough to contain the entire block. This could happen when the application is acting as a data receiver only and does not know what block size to expect. When **PvPipeline** is used, this error leads the **PvPipeline** to automatically re-allocate the buffer size. eBUS Player uses **PvPipeline**. When eBUS Player works as a data receiver only and the incoming block size is larger than the default size, this error occurs multiple times until all of the buffers that are queued into the **PvPipeline** are returned and resized. In your application, if **PvStream::QueueBuffer()** is used directly, you need to resize each retrieved buffer to the size that is returned from **PvBuffer::GetRequiredSize()**.

Table 2: **aOperationResult**/Data Receiver Result Codes (Continued)

***Data receiver* result code PvResult:: /Stream statistic error counter**	**Buffer valid?**	**Descriptions and possible reasons**
IMAGE_ERROR / ResultImageError	Potentially	Data overrun, full line, or partial line missing occur when the GVSP transmitter receives image data from the camera head. To help determine which issue is occurring, call **PvImage::IsFullLineMissing()**, **::IsPartialLineMissing()**, and **::IsDataOverrun()**. The PartialLIneMIssing and/or FullLIneMising can occur in certain instances, for example if the camera is switched to binning mode without the **Width** and **Height** settings being changed, or if the **Width** and/or **Height** settings do not match the camera output defined by the camera signal LVAL and/or FVAL. The buffer is valid if **PvBuffer::GetLostPacketCount**() returns zero and **PvBuffer::GetAcquiredSize**() matches the current binning mode.
ABORTED /ResultAborted	No	After calling **PvStream::AbortQueuedBuffers()**, all of the buffers queued in the data receiver are set with this result code.
TOO_MANY_CONSECUTIVE_RESENDS / ResultTooManyConsecutive Resends	Potentially	The number of sequential packets in one block that are missing is larger than the threshold set by **MaximumResendGroupSize**.
TOO_MANY_RESENDS / ResultTooManyResends	Potentially	The total number of packet resend requests for one block exceeds the threshold set by **MaximumPendingResends**. For example, **GevSCPSPacketSize** is set to a jumbo packet size while the NIC does not (or is not configured to) support that jumbo packet size.
RESENDS_FAILURE / ResultResendsFailure	Potentially	The packet resend fails when the threshold set by **MaximumResendRequestRetryByPacket** is reached, or the data receiver receives the standard GigE Vision status code **GEV_STATUS_PACKET_UNAVAILABLE** from the GVSP transmitter.
INVALID_DATA_FORMAT / ResultInvalidDataFormat	No	The data receiver receives an unknown or unsupported GVSP block payload type. For a list of supported payload types, see the description of the **PvPayloadType** enumeration eBUS SDK C++ API Help file.
AUTO_ABORTED / ResultAutoAborted	No	The block buffer is aborted as the setting of **ResetOnIdle** or **AutoResetOnLackOfResources** takes effect.
NO_LICENSE	No	No license was available and watermarking could not be used; the buffer was not an image or was too small for watermarking.

Table 2 describes the data receiver result codes and the possible causes of these errors, and also describes their relationship to the stream control configuration settings (as applicable). Some error codes, such as **IMAGE_ERROR**, **BUFFER_TOO_SMALL** and **INVALID_DATA_FORMAT**, are not related to the data receiver's behavior. Other errors are related to the data receiver's behavior, which is controlled by the stream control configuration settings.

Checking the Blocks Missing in Your Application

For each application retrieved buffer using **lBuffer->GetImage**()->**IsImageDropped**() you can check whether blocks are missing between the current and previous grabbed blocks on the Pixel bus of Pleora Video interface.

For each retrieved buffer getting the **BlockID** from **lBuffer->GetBlockID**(), you can compare the BlockIDs of consecutive retrieved buffers to determine whether there are blocks missing.

How Buffers are Managed in the Data Receiver

To help understand the stream configuration settings and their impact, this section provides a summary of how the data receiver manages buffers.

The block buffers are locked by the data receiver between the application call **PvStream::QueueBuffer**() (**PvPipeline** internally calls this function) and **PvStream::RetrieveBuffer**() (**PvPipeLine:: RetrieveNextBuffer**()). Inside the data receiver the block buffers can be in one of three states: **Empty**, **Receiving**, and **Ready for Processing**.

- **Empty.** The data receiver can fill block data in to the buffer.
- **Receiving.** The data receiver is filling block data in to the block buffer. The data receiver can fill multiple buffers simultaneously. This ensures the process can continue, even if out-of-order or missing packets occur.
- **Ready for Processing.** The data receiver changes the receiving block buffer (or buffers) to this state and sets the result code when events occur or errors are detected, as outlined in Table 3.

Table 3: Events and Errors that Trigger the Data Receiver to Change from "Receiving" to "Ready for Processing"

Event or error	Result code
Receives the whole block data.	OK
The **FirstPacketTimeout** timeout expires.	TIMEOUT
The **InterPacketTimeout** timeout expires.	MISSING_PACKETS
The **RequestTimeout** timeout expires.	TIMEOUT. No packets are received. MISSING_PACKETS. Some packets are received.
Not enough space to copy all of the payload packets in to the buffer.	BUFFER_TOO_SMALL
PvStream::AbortQueuedBuffer() is called.	ABORTED
The threshold set by **MaximumResendGroupSize** is exceeded.	TOO_MANY_CONSECUTIVE_RESENDS
The threshold set by **MaximumPendingResends** is exceeded.	TOO_MANY_RESENDS
The packet resend fails (the **ResendRequestTimeout** expires) **MaximumResendRequestRetryByPacket** times.	RESENDS_FAILURE
Receives the standard GigE Vision status code **GEV_STATUS_PACKET_UNAVAILABLE**.	RESENDS_FAILURE
INVALID_DATA_FORMAT error detected.	INVALID_DATA_FORMAT
The parameter **AutoResetOnLackOfResources** takes effect.	AUTO_ABORTED
The parameter **ResetOnIdle** takes effect.	AUTO_ABORTED

Table 3: Events and Errors that Trigger the Data Receiver to Change from “Receiving” to “Ready for Processing”

Event or error	Result code
The parameter **ForceMissingPacketsAtNextBlockStart** takes effect.	NOT_INITIALIZED. No packets are received. MISSING_PACKETS. Some packets are received.
The parameter **ExtraBlocCompletion** (**OnBlockTrailer** or **OnNextBlockStart**) takes effect	MISSING_PACKETS

The result code is only set once based on the event or error that happens first. All of the incoming packets received for a buffer in the **Ready for Processing** state are discarded.

The block buffer queue is handled in FIFO mode. The packets from the new block are filled in to the oldest block buffer in the queue, and the oldest block buffer is the first to be retrieved back to the application. The following two extreme cases show how the FIFO mode rule is followed:

- **Case#1.** (Block buffer)*n* is filled with (or is filling with) packets from block*n*. Then, the first packet of block*n*+2 comes in and this packet is filled in (block buffer)*n*+2. The (block buffer)*n*+1 is reserved for block*n*+1 and its status changes to **Receiving**.
- **Case#2.** (Block buffer)*n* is still waiting for some packets. (Block buffer)n+1 has filled with full block data and changes to **Ready for Processing**. (Block buffer) *n*+1 cannot be retrieved until the (block buffer)*n* changes to **Ready for Processing**.

Understanding the Stream Control Configuration Settings and Their Impact on the Data Receiver's Behavior

By design, timeouts are only checked after the stream is started. The data receiver stream start is defined when:

- There are block buffers in the **Empty** state
- One GVSP packet has been received

Please note that the timer for timeout does not start when **PvStream::QueueBuffer**() or **PvPipeline::Start**() is called.

FirstPacketTimeout

Type	Default	Units	Notes
Integer	0	Milliseconds	This parameter is disabled when it is set to 0.

This parameter defines the time that the data receiver waits for the first packet of a block.

The timer starts after the last packet of the previous block is received.

When this timeout expires, the block buffer is set with error code **TIMEOUT**.

InterPacketTimeout

Type	Default	Units	Notes
Integer	0	Milliseconds	This parameter is disabled when it is set to 0.

This parameter defines the time that the data receiver waits for subsequent packets to arrive for the newest receiving block buffer.

For example, if (block buffer)*n* is waiting for the last few packets and (block buffer)n+1 receives its first packet, the timeout is only checked for (block buffer)*n*+1 and not for (block buffer)*n*.

When this timeout expires, the block buffer is set with error code **MISSING_PACKETS**. Note that the error code cannot be **TIMEOUT**, as this parameter takes effect after at least one packet has been received for the given block buffer.

RequestTimeout

Type	Default	Units	Notes
Integer	5000	Milliseconds	This parameter is disabled when it is set to 0 or infinite.

This parameter defines the time that the data receiver waits for all packets to arrive for a given block buffer.

The timer starts after the last packet of the previous block arrives or stream start is detected.

In most systems, this timeout should be the block time plus some extra time for possible packet resend multiple retries at the end of the block. If the block is generated in trigger mode, the value of this parameter should be long enough to cover the longest possible trigger intervals.

When this timeout expires, the block buffer is set with error code **MISSING_PACKETS** (if some packets have been received) and **TIMEOUT** (if no packets have been received for the given block).

PreemptiveResendTimeout

Type	Default	Units	Notes
Integer	0	Milliseconds	This parameter is disabled when it is set to 0.

This parameter defines the time that the data receiver waits for subsequent packets to arrive before preemptively issuing the packet resend command for future packets.

The timer starts after the first packet of the current receiving block is received, and resets with each new arriving packet.

This parameter is useful when:

- You are using **SingleFrame** or are receiving the last frame in **MultiFrame** acquisition,

  - And-

- The next frame is delayed and the last few packets (including the trailer) of the current frame are lost.

If this parameter is disabled, the data receiver waits for the **RequestTimeout** to expire.

When this timeout expires, the data receiver sends a packet resend request for the remaining packets of the current receiving block. If the SDK does not receive the remaining packets of the current receiving block, the **ResendRequestTimeout** and **MaximumResendRequestRetryByPacket** properties take effect.

If **InterpacketTimeout** is shorter than this timeout, this parameter will never take effect.

MaximumPendingResends

Type	Default	Units	Notes
Integer	45	Resend Requests	From 0–4096. This parameter is disabled when: • The value is set to 0. • **RequestMissingPackets** is set to **false**.

This parameter defines the threshold of the total number of packet resends that the data receiver can request for a given block.

When the threshold is crossed, the block buffer is set with error code **TOO_MANY_RESENDS**.

RequestMissingPacket

Type	Default	Units	Notes
Boolean	True	N/A	None

This parameter defines whether the data receiver should request packet resends for lost packets.

This parameter was introduced in eBUS SDK version 2.1.0 to replace the **IgnoreMissingPackets** parameter in older SDK releases. These two parameters disable and enable the same eBUS SDK packet resend feature with opposite logic. The **IgnoreMissingPackets** parameter is deprecated and set to **invisible** from the stream configuration settings in eBUS SDK version 2.1.0.

Some GVSP transmitters, such as the Video Server API, do not support packet resend. In this case, we recommend that you set this parameter to **false** to prevent the data receiver from sending unnecessary packet resend commands.

LatencyLevel

Type	Default	Units	Notes
Integer	0		Only available for the eBUS Universal Pro driver. Values range from 0–3. For the lowest latency, use **0**. The latency level increases when you increase the value of this parameter.

This parameter allows you to modify the latency level.

The lower the latency level, the greater the number of CPU resources used.

AutoResetOnLackOfResources

Type	Default	Units	Notes
Integer	1	PvBuffers	This parameter is disabled when it is set to 0, which can have negative side effects on your system.

This parameter defines the threshold value for the number of **Empty** block buffers in the data receiver.

When the threshold is exceeded, all **Receiving** block buffers change to the **Ready for Processing** state and are set with the error code **AUTO_ABORTED**. For example, for the default value **1**, the threshold is exceeded when the last Ready buffer starts to receive packets.

This parameter lets the data receiver automatically abort uncompleted blocks when the **RequestTimeout** is too long and **Ready** block buffer resources are too low.

MaximumResendRequestRetryByPacket

Type	Default	Units	Notes
Integer	3	Resend request	This parameter is disabled when: • The value is set to 0. • **RequestMissingPacket** is set to false.

This parameter defines the maximum number of times that the data receiver can request a packet resend for the same packet.

Keep in mind that the value of **RequestTimeout** should be long enough to allow the packet resend retry to occur for the number of times defined by this parameter, especially if the missing packets are the last few packets.

If the number of retries has been reached and the missing packet has still not been retrieved, the block buffer is set with error code **RESENDS_FAILURE**.

MaximumResendGroupSize

Type	Default	Units	Notes
Integer	15	Packets	This parameter is disabled when: • The value is set to 0. • **RequestMissingPacket** is set to **false**.

This parameter defines the threshold number of sequential packets that the data receiver can request for resend in one packet resend command.

The packet resend shares the bandwidth with new incoming packets. In bandwidth-sensitive systems or low latency critical systems, this parameter allows the data receiver to abandon packet resends for the block when too many sequential packets are lost.

When the threshold is crossed, the block buffer is set with the error code **TOO_MANY_CONSECUTIVE_RESENDS**.

ResendRequestTimeout

Type	Default	Units	Notes
Integer	5000	Microseconds	This parameter is disabled when **RequestMissingPacket** is set to **false**.

This parameter defines the maximum amount of time that the data receiver waits to receive the requested resend packet before determining if the current resend request has failed.

When the request fails, the data receiver issues the resend request again if the number of retries is less than the value of **MaximumResendRequestRetryByPacket**. Otherwise, the data receiver sets the block buffer with the error code **RESENDS_FAILURE**.

ResendDelay

Type	Default	Units	Notes
Integer	0	Microseconds	—

This parameter defines the time that the data receiver waits before issuing a resend request for missing packets. This is used when the **PacketOutOfOrder** counter under **Statistic > Counters > DataReceiver** is non-zero.

ResetOnIdle

Type	Default	Units	Notes
Integer	200	Milliseconds	This parameter is disabled when it is set to 0.

This parameter defines the amount of time that the data receiver waits for GVSP packets (when no GVSP packets are being received) before it resets itself.

The reset sets:

- All of the **Receiving** blocks to **Ready for Processing** states with error code **AUTO_ABORTED**.
- The data receiver back to the pre-stream start state.

This parameter allows the data receiver to better handle cases where the data receiver receives no packets for a long time due to a connection loss or a large number of packets that have been dropped on the network. Without this parameter, the data receiver may not be able to recover quickly, as it may try to request several packet resends.

This parameter must be carefully set to avoid triggering undesirable resets while streaming. In general, the value of this parameter should be larger than the value of **RequestTimeout**, or set to zero to disable this parameter.

ForceMissingPacketsAtNextBlockStart

Type	Default	Units	Notes
Boolean	False	N/A	This parameter is deprecated in eBUS SDK version 4.0.6. The new **ExtraBlockCompletion = OnNextBlockStart** parameter has the same effect as **ForceMissingPacketsAtNextBlockStart = True**. This parameter was removed in eBUS SDK version 4.0.

This parameter specifies whether the data receiver should change the current **Receiving** buffer to the **Ready to retrieve** status as soon as the first packet from the next or subsequent block arrives.

In cases where the application needs to have the block data as soon as possible (even if there are missing packets) or the GigE Vision device does not support packet resends, this parameter should be turned on.

When this parameter takes effect, the data receiver sets the buffer with result code **MISSING_PACKETS** if at least one packet is received, and **NOT_INITIALIZED** if no packets are received.

EnableMissingPacketsList

Availability	Type	Default	Units	Notes
eBUS User Mode Data Receiver eBUS Universal Pro	Boolean	False	N/A	Added in eBUS SDK version 2.1.

This parameter specifies whether the data receiver should keep track of missing packets.

When this parameter is enabled, the data receiver keeps a list of missing packets and the list can be retrieved using the **PvBuffer::GetMissingPacketIdsCount**() and **PvBuffer::GetMissingPacketIds**() calls. Please note that keeping the list takes extra resources and, for this reason, this feature is turned off by default.

MaximumPreQueuedBuffer

Type	Default	Units	Notes
Integer	1	PvBuffer	Only available for the eBUS Universal Pro driver. Values range from 1–32.

This parameter defines the number of buffers to transfer at one time from the user space to the kernel space of the operating system (for example, batch transfer).

One of the more resource-intensive operations in the Windows operating system is the transition between the user space and the kernel space, and vice versa. At very high frame rates this operation may require significant processing power. Transferring multiple buffers at once between the user and kernel space reduces the amount of processing resources that are required.

If you set this parameter to 3 for example, each call of **PvStream::QueueBuffer**() in the application passes the given buffer from the application to the data receiver, while the data receiver waits until the third call of **PvStream::QueueBuffer**(). After the third call, the data receiver then passes three buffers together to the kernel space. Buffers must be in kernel space to be in the Ready state. In this example, a second batch transfer of buffers happens after the sixth call of **PvStream::QueueBuffer**(). The **PvPipeline** class calls **PvStream::QueueBuffer**() internally and the same batch transfer happens inside the data receiver.

To ensure there are always enough Ready buffers in the data receiver, the buffer queue size should be at least larger than (**AutoResetOnLackOfResource + MaximumPrequeuedBuffer + 1**). This parameter is mainly used for high frame rate cases that typically require a very large buffer queue size. For example, for a stable system with a buffer queue size of 256, you can optimize processing power usage by setting this parameter to 16 and increasing the buffer size to 256 + 16.

To transfer buffers back from kernel to user space, the batch transfer and low latency criteria are followed. At the retrieve buffer function call (if there are buffers in user space), the oldest buffer will be returned immediately. If there are no buffers in user space, the data receiver transfers all **Ready for Processing** buffers (up to 32) at that moment together from kernel to user space. Then, the oldest buffer is returned to the application and the other buffers stay in the user space of the data receiver waiting to be retrieved by the subsequent retrieve buffer function calls.

OnlyStartOnFirstPacket

Type	Default	Units	Notes
Boolean	True	N/A	Introduced in eBUS SDK 3.1.17.

This parameter specifies at stream start whether the data receiver discards packets until a GVSP block leader is received.

When set to true, GVSP packets are discarded until a block leader is received. When it is set to false, the data receiver starts processing the first received packet and checks whether it is the block leader. If it is not the block leader, the data receiver sends a resend request for previous packet IDs for the current block.

Setting this feature to false can cause multiple unwanted resend requests on multicast systems.

ExtraBlockCompletion

Type	Default	Units	Notes
Enumeration	Disabled	N/A	Introduced in eBUS SDK 4.0.7. It has three options: **Disable**, **OnNextBlockStart**, and **OnBlockTrailer**

This parameter specifies whether the data receiver should change the current **Receiving buffer** to the **Ready to retrieve** status as soon as:

- The trailer packet of the current block is received (**OnBlockTrailer**).

  - Or -

- The first packet from the next or subsequent block is received (**OnNextBlockStart**).

In cases where the application needs to have the block data as soon as possible (even if there are missing packets) or the GigE Vision device does not support packet resends, this parameter should be used.

When this parameter takes effect, the data receiver sets the buffer with result code **MISSING_PACKETS** if at least one packet has been received and **NOT_INITIALIZED** if no packets have been received.

Understanding the DataReceiver Statistic Counters

The counters in the **DataReceiver** category contain information about the system's performance. For a very stable system, most of the counters should be zero or small values. For an unstable system, some of the counters could have large values. Understanding the information in the counters can help you determine the optimal stream configuration settings for your system.

If the counter values under the DataReceiver increase quickly, this is an indication that the system is experiencing performance issues.

Table 4: DataReceiver Statistics Counters

Counters	Description and possible reasons
PacketsRecovered	The total number of lost packets recovered by packet resend. To recover one lost packet, packet resend commands could be sent from the data receiver to the GVSP transmitter one or several times. If this counter keeps increasing, the network communication is marginally stable and you should address the issue. If this counter suddenly increases and then stops, the network communication was momentarily unstable and has since recovered.
PacketsRecoveredSingleResend	The total number of lost packets recovered by a single packet resend command sent from the data receiver. If this counter value is smaller than the value of **PacketsRecovered**, the **ResendRequestTimeout** may be too short to allow the resent packets to be received before the resend request is repeated.
UnexpectedResend	Only applies to legacy drivers/data receivers.
ResendGroupRequested	The total number of packet resend commands sent from the data receiver. One command could ask for one or multiple sequential packet resend requests.
ResendPacketRequested	The total number of packet resend requests, which is the sum of the number of packet resend requests in all packet resend commands.

Table 4: DataReceiver Statistics Counters (Continued)

Counters	Description and possible reasons
LostPacket	The total number of missing packets that have not been recovered. This does not include packets where the entire block is missing. If this counter value is not zero, then some of the following error counters may not be zero: • ResultMissingPackets • ResultTooManyResends • ResultTooManyConsecutiveResends • ResultResendsFailure
IgnoredPacket	This is the total number of packets that the data receiver discarded. It includes: **1.** GVSP packets that do not belong to any **Receiving** block buffers. **2.** Non-GVSP packets that reached the socket address to which the data receiver is listening. **3.** For case 1, the **RequestTimeout** value may not be correct for the application and you should incrementally increase the **RequestTimeout**. For case 2, you should remove unnecessary network traffic.
RedundantPacket	The total number of related GVSP packets received more than once when the block buffer is still in **Receiving status**. These packets could be: Resend packets that came in too late and multiple packet resends are requested for the same packet. Resend packets requested by other data receivers in the same multicasting group. For case 1, increasing the **ResendRequestTimeout** should help. For case 2, turning off the **RequestMissingPackets** for some data receivers in the same multicast group should help.
PacketOutOfOrder	The total number of packets that arrived out of order and were successfully re-ordered by the data receiver without any packet resend requests. Only the eBUS Universal Pro driver is able to re-order packets without packet resend requests by setting a non-zero value for the **ResendDelay** parameter.

System Considerations When Packet Resends Are Enabled

Multicast Systems

Each data receiver can send packet resend commands to the GVSP transmitter. The GigE Vision standard defines that the GVSP transmitter should respond to all resend requests. When multiple data receivers request the same packet resend, the transmitter device retransmits (multicasts) the same packet multiple times (one for each request).

If your system is currently using almost all of the available bandwidth, the extra bandwidth required by the packet resend may have an impact on streaming and can result in an image drop on the device side. So you may consider setting **RequestMissingPacket** to **false** for some of the data receivers in the multicast group.

High Frame Rate Systems

In systems with high frame rates, packet resends take a significant amount of time to be serviced when compared with the transfer time of the block. Resends can introduce jitter and latency and more buffers should be queued to the data receiver.

High Data Rate Systems

In systems with high data rates, the stream payload data rate is close to the maximum bandwidth of the Ethernet link. Packet resends could lead the total bandwidth to the limit of the Ethernet link. To have a reliable system you should:

- Use jumbo packets. Set **GevSCPSPacketSize** to **8164**.

  IMPORTANT: See the note about **GevSCPSPacketSize** deprecation on page 4.
- Turn **RequestMissingPacket** off.
- Limit the amount of allowed packet resends by reducing the values of **MaximumPendingResends** and **MaximumResendGroupSize**.

In systems with high burst (or peak) data rates but low average data rates, increasing the interpacket delay (**GevSCPD**) will increase the system reliability.

Connection Lost Error

The **Connection lost** dialog box that appears in eBUS Player does not indicate streaming failure, but can be a related issue. This failure occurs when the SDK does not receive an ACK from the device for heartbeat packets or commands. These GVCP packets could be lost when the system is streaming close to the maximum bandwidth.

Firewalls, Anti-Virus Software, and Third-Party Filters

Firewalls, anti-virus software, and third-party filter drivers can affect the performance of your system because they may all monitor incoming packets at different network stack layers. For powerful systems or a low bandwidth stream, the impact of these filters can be negligible. For other systems, the impact can be significant and needs to be handled appropriately. For full details on the firewall's impact on the system, see the application note titled *Correcting Firewall Issues With 3rd Party GigE Vision Devices*, available at the Pleora Support Center.

eBUS Universal Pro

The eBUS Universal Pro is a filter driver that extracts the GVSP stream from the network stack as early as possible. This filter driver is inserted before Windows Firewall in the network stack. Therefore, Windows Firewall should have no impact on the stream's packets. The GVCP packets are still forwarded to the firewall.

If your anti-virus software uses NDIS filter technologies to protect your computer and is inserted before the eBUS Universal Pro filter, major performance issues can occur. These anti-virus filters inspect the incoming packets and (as

a result) slow down the system. This can also result in packet drops, as they are unable to keep up with the data rate on slower systems. For this reason, we suggest that you remove this component if it impacts the system.

Third-party filters can also use the same NDIS technologies. Pleora's drivers can impact a third-party filter, depending on the internal order of the hooks. For the same level of hook, the order is determined by the installation order. For this reason, we recommend that you do not install side-by-side GigE Vision compliant SDKs from different providers, even if you have no issues while running these SDKs.

eBUS User Mode Data Receiver

When using the eBUS User Mode Data Receiver, the data receiver is working on top of the normal network stack. This makes the packet transition through more layers (on the operating system's network stack) than when using the eBUS Universal Pro. This increases the system's sensitivity to timing problems and disruptions by other pieces of software.

The firewall assesses each packet of the stream before determining an action, which can slow down the packets. For anti-virus software that uses an NDIS filter (or any third-party SDK that installed a filter driver on the NIC) it will inspect the packets. For this reason, the timing of the packets is impacted before the data receiver gets them, and therefore affects the system's performance. We suggest that you always install the eBUS Universal Pro driver.

Monitoring Traffic to the Data Receiver

Wireshark®/WinPcap® captures the packets at a higher level in the network stack layer than the eBUS Universal Pro filter driver. As a result, GVSP packets extracted by the eBUS Universal Pro filter driver cannot be captured by Wireshark. For the same reason, in Windows XP the networking performance monitoring tool in Windows Task Manager does not report the portion of the bandwidth used by the GVSP stream packets. In Windows 7 the networking performance monitoring tool does report the portion of bandwidth used by the GVSP stream packets.

To capture the GVSP packets when using the eBUS Universal Pro filter driver, you can use a network switch with the port mirroring feature. Run Wireshark over a NIC without the eBUS Universal Pro driver installed, and connect to the monitor port on the network switch. The NIC with the eBUS Universal Pro driver installed connects to the port being monitored on the network switch. You can refer to the application note Using Wireshark to Capture Ethernet Activity on GigE Vision Devices to learn how to use Wireshark. You can also use **ebPCAPRecorder**, a utility built with eBUS SDK, to monitor the traffic.

When the user mode data receiver is used, the GVSP stream packets can be captured by Wireshark over the same NIC, without any need to enable port mirroring on a switch. Please note that Wireshark copies the GVSP packets before the user mode data receiver and as a result the packet's timing is impacted.

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

Document Number

EX003-017-0006 Version 5.0