imaWorx CXP-12 Quad

Applet Feature Reference Manual for

Acq_QuadCXP12Line

Functional Description
For pylon or GenTL Usage

Document Number: AW001882
Part Number: 000 (English)
Document Version: 02
Release Date: 25 February 2025
Applet Version 2.5.9.0

Contacting Basler Support Worldwide

Europe, Middle East, Africa
Tel. +49 4102 463 515
support.europe@baslerweb.com

The Americas
Tel. +1 610 280 0171
support.usa@baslerweb.com

Asia-Pacific
Tel. +65 6367 1355
support.asia@baslerweb.com

Singapore
Tel. +65 6367 1355
support.asia@baslerweb.com

Taiwan
Tel. +886 3 558 3955
support.asia@baslerweb.com

China
Tel. +86 10 6295 2828
support.asia@baslerweb.com

Korea
Tel. +82 31 714 3114
support.asia@baslerweb.com

Japan
Tel. +81 3 6672 2333
support.asia@baslerweb.com

https://www.baslerweb.com/en/sales-support/support-contact

Supplemental Information

Acquisition Card Documentation:
https://docs.baslerweb.com/acquisition-cards
Frame Grabber Documentation:
https://docs.baslerweb.com/frame-grabbers
Framegrabber SDK Documentation:
https://docs.baslerweb.com/frame-grabbers/framegrabber-sdk-overview.html

Table of Contents

Chapter 1. Introduction

This document provides you with detailed information on applet "Acq_QuadCXP12Line" for imaWorx CXP-12 Quad frame grabber.

In the following, you will find a full description of the applet's functionality and features.

For information on the hardware or for a general introduction on how to configure the CXP-12 Interface Card using the pylon API, the pylon Viewer, or the gpioTool check the document which can be found in https://docs.baslerweb.com/pc-cards.

All applet-specific parameters described in this document are as represented in the GenTL interface.

For a general explanation of the GenTL interface, check the Basler GenTL interface documentation (https://www.baslerweb.com/en/sales-support/downloads/document-downloads/cxp-gentl-producer-feature-documentation/).

For information on camera features, check the respective camera documentation.

For information on Basler pylon features and for API documentation, check the pylon documentation.

1.1. Features of Applet Acq_QuadCXP12Line

"Acq_QuadCXP12Line" is a quad-camera applet. Up to four individual cameras can be used. The features of this applet are fully available for all camera ports. You can configure the CoaXPress camera interface for CoaXPress cameras version 1.1.1 and 2.0, transferring grayscale (monochrome), BiColor pattern according to PFNC , or color pixels. Allowed pixel formats are Gray (Mono8, Mono10, Mono12, Mono14, Mono16), Color (RGB8, RGB10, RGB12, RGB14, RGB16), biColor (BiColorRGBG8, BiColorRGBG10, BiColorRGBG12, BiColorGRGB8, BiColorGRGB10, BiColorGRGB12, BiColorBGRG8, BiColorBGRG10, BiColorBGRG12, BiColorGBGR8, BiColorGBGR10, BiColorGBGR12) and YCbCr422_8. You can only use single link CoaXPress cameras with this applet. The maximum link speed is CXP-12. A multi-functional line trigger is included in the applet. This allows you to control the camera or external devices using frame grabber generated, external or software generated trigger pulses. Line scan cameras up to a width of 32768 pixels can be processed. The trigger system will generate images of a maximum height of 8388607 pixels. The applet is processing data at a bit depth of 16 bits. An image selector at the camera port facilitates the selection of one image out of a parameterizable sequence of images. This enables the distribution of the images to multiple frame grabber and PCs. For reverse operation, you can mirror the image in x-direction and y-direction before cutting the ROI. Acquired images are buffered in frame grabber memory. You can select a region of interest (ROI) for further processing. The stepsize of the ROI width is 8 pixel. The ROI stepsize for the image height is 1 line. This applets includes the special color interpolation filter for bilinear color linescan cameras. The first line is blue red, the second line is green only. A color converter automatically converts the input pixel formats to the output formats. In this applet conversions from monochrome, RGB or BiColor to monochrome and RGB can be performed. You can configure the 14 bit full resolution lookup table either by using a user defined table, or by using a processor. The processor gives you the opportunity to use pre-defined functions such as offset, gain, invert to enhance the image quality. The color components are processed individually. A gamma correction is possible.

Processed image data are output by the applet via high speed DMA channels. You can select the pixel format of the output. The pixel format can either be 8 bit, 10 bit packed, 12 bit packed, 14 bit packed, or 16 bits per pixel (or per pixel component if you work with a color format).

Table 1.1. Feature Summary of Acq_QuadCXP12Line

Feature	Applet Property
Applet Name	 Acq_QuadCXP12Line
Type of Applet	AcquisitionApplets
Board	imaWorx CXP-12 Quad
No. of Cameras	4 , asynchronous or synchronous
Camera Type	CoaXPress, link aggregation max. 1, maximum speed CXP-12, Version 1.1.1 and 2.0
Sensor Type	Line Scan
Camera Format	Monochrome, BiColor or RGB
Pixel Format	Gray (Mono8, Mono10, Mono12, Mono14, Mono16), Color (RGB8, RGB10, RGB12, RGB14, RGB16), biColor (BiColorRGBG8, BiColorRGBG10, BiColorRGBG12, BiColorGRGB8, BiColorGRGB10, BiColorGRGB12, BiColorBGRG8, BiColorBGRG10, BiColorBGRG12, BiColorGBGR8, BiColorGBGR10, BiColorGBGR12) and YCbCr422_8.
Processing Bit Depth	16 Bit per color component
Sensor Correction / Tap Sorting	no
Maximum Images Dimensions	32768 * 8388607
ROI Stepsize	x: 8, y: 1
Tap Geometry Sorting	1X-1Y only
Mirroring	Yes, horizontal and vertical (set the parameter *VantagePoint*)
Image Selector	Yes
Noise Filter	No
Shading Correction	No
Dead Pixel Interpolation	No
Color Array Filter	Two lines. First Blue and Red, second Green. (or swapped)
Color White Balancing	Yes
Color Converter	yes, Mono, RGB or BiColor to Mono or RGB
Lookup Table	Full Resolution Input bits = 14, Output bits = 16 Lookup table can be disabled.
DMA	Full Speed
DMA Image Output Format	All grayscale and color formats. See description above.
Event Generation	yes
Overflow Control	yes

1.1.1. Parameterization Order

We recommend to configure the functional blocks which are responsible for sensor setup/correction first. This will be the camera settings, shading correction, and dead pixel interpolation (if available). Afterwards, you can configure other image enhancement functional blocks such as white balancing, noise filter, and lookup table. By default, all presets are configured for receiving images directly.

1.2. Bandwidth

The maximum bandwidths of applet Acq_QuadCXP12Line are listed in the following table.

Table 1.2. Bandwidth of Acq_QuadCXP12Line

Description	Bandwidth
Max. CXP Speed	CXP-12
Peak Bandwidth per Camera	1200 MPixel/s
Mean Bandwidth per Camera	1200 MPixel/s
DMA Bandwidth	7200 MByte/s (depends on PC mainboard)

The peak bandwidth defines the maximum allowed bandwidth for each camera at the camera interface. If the camera's peak bandwidth is higher than the mean bandwidth, the frame grabber on-board buffer will fill up as the data can be buffered, but not be processed at that speed.

The mean bandwidth per camera describes the maximally allowed mean bandwidth for each camera at the camera interface. It is the product of the framerate and the image pixels. For example, with 1-megapixel images at a framerate of 100 frames per second, the mean bandwidth will be 100 MPixel/s. In case of 8bit per pixel as output format, this would be equal to 100 MB per second.

The required output bandwidth of an applet can differ from the input bandwidth. A region of interest (ROI) and the output format can change the required output bandwidth and the maximum mean bandwidth. Moreover, this applet is a Bayer applet. The required output bandwidth will be three times higher than the input bandwidth. (This applies only when debayering is switched to ON.) Mind that the DMA bandwidth is the total bandwidth. The sum of all camera channel bandwidths has to be less than the maximum DMA bandwidth to avoid overflows.

Regard the relation between MPixel/s and MByte/s: The MByte/s depend on the applet and its parameterization concerning the pixel format. It is possible to acquire more than 8 bit per pixel or to convert from one bit depth to another. 1 MByte is 1,000,000 Byte.

Bandwidth Varies

The exact maximum DMA bandwidth depends on the used PC system and its chipset. The camera bandwidth depends on the image size and the selected frame rate. The given values of 7200 MByte/s for the possible DMA bandwidth might be lower due to the chipset and its configuration. Additionally, some PCIe slots do not support the required number of lanes to transfer the requested or expected bandwidth. In these cases, have a look at the mainboard specification. A behaviour like multiplexing between several PCIe slots can be seen in rare cases. Some mainboard manufacturers provide a BIOS feature where you can select the PCIe payload size: Always try to set this to its maximum value or simply to automatic. This can help in specific cases.

1.3. Requirements

In the following, the requirements on software, hardware and frame grabber license are listed.

1.3.1. Software Requirements

To run this applet, a supporting runtime environment is required. This can be either Basler pylon, or the Basler Framegrabber SDK providing the GenTL interface.

1.3.2. Hardware Requirements

To run applet "Acq_QuadCXP12Line", a Basler imaWorx CXP-12 Quad frame grabber is required.

For PC system requirements, check the frame grabber hardware documentation. The applet itself does not require any additional PC system requirements.

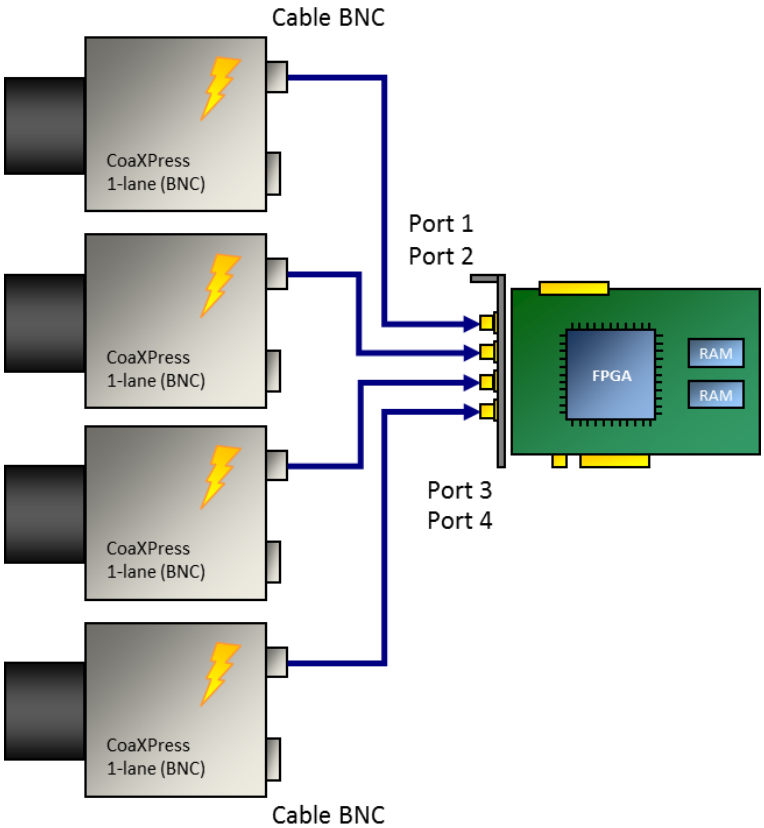
1.3.3. License

This applet is of type AcquisitionApplets. For applets of this type, no license is required. All compatible frame grabbers can run the applet using the Basler Framegrabber SDK.

1.4. Camera Interface

Applet "Acq_QuadCXP12Line" supports 4 CXP cameras. The frame grabber has 4 connectors. Connect one camera cable of each camera to the ports of the frame grabber. The mapping of the ports between the camera and the frame grabber is not important. You can chose any order.

Figure 1.1. Camera Interface and Camera Cable Setup



1.5. Frame ID

For CoaXPress linescan cameras the CXP Source Tag is not used as it is constand throughout the acquisition. Instead an internal counter is used to represent frame IDs. This applet will output each frame to the host PC

attached with this frame ID. Moreover, overflow events will also include this frame ID. By this, the exact mapping of a given frame in the host PC to the frame the frame grabber's image trigger is possible.

Check chapter Chapter 10, '*BufferStatus*' for more information about overflow conditions and the overflow event data structure including the frame ID.

The frame ID is processed together with the images in the host PC. Check the Basler GenTL documentation to learn on how to extract the frame ID from the buffer.

1.6. Image Transfer to PC Memory

The image transfer between frame grabber and PC is performed via DMA transfers. In this applet, 4 DMA channels exist for transferring image data. One channel for each camera. The DMA channels have the same indices as the cameras, starting with 0. The applet output format can be set via the parameters of the output format module. See Chapter 16, '*OutputFormat*'. All outputs are little-endian coded.

Chapter 2. CoaXPress

This applet can be used with up to 4 line scan cameras. To receive correct image data from your camera, it is crucial that the camera output format matches the selected frame grabber input format. The following parameters configure the frame grabber's camera interface to match with the individual camera pixel format. Most cameras support different operation modes. Consult the manual of your camera to obtain the necessary information how to configure the camera to the desired pixel format.

Ensure that the lines transferred by the camera do not exceed the maximum allowed line length for this applet (32768).

With the following parameters you can define the way trigger packets are sent from the frame grabber to the camera on the CXP link.

2.1. SystemmonitorStreamPacketSize

Returns the stream packet size in bytes. Range: between 4 and 65535 bytes in steps of 4 bytes.

Table 2.1. Parameter properties of SystemmonitorStreamPacketSize

Property	Value
Name	**SystemmonitorStreamPacketSize**
Display Name	**Systemmonitor Stream Packet Size**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 2.1. Usage of SystemmonitorStreamPacketSize

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorStreamPacketSizeSelector = i;
    value_ = SystemmonitorStreamPacketSize;
}
```

2.2. SystemmonitorCxpStandard

Returns the version of the used CXP standard.

Table 2.2. CXP Standard Version

CXP Standard Version		
CXP_1_0		
CXP_1_1_1		
CXP_2_0		
Unknown		

Table 2.3. Parameter properties of SystemmonitorCxpStandard

Property	Value
Name	**SystemmonitorCxpStandard**
Display Name	**Systemmonitor CXP Standard**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Beginner**

Example 2.2. Usage of SystemmonitorCxpStandard

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorCxpStandardSelector = i;
    value_ = SystemmonitorCxpStandard;
}
```

2.3. CxpStreamPacketCount

This parameter counts the amount of received stream packets. Bits [29:0] count the number of packets. Bit [30] is set when a counter overflow occurs. Range: 0 to 4294967295 (32 bit).

Table 2.4. Parameter properties of CxpStreamPacketCount

Property	Value
Name	**CxpStreamPacketCount**
Display Name	**CXP Stream Packet Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 2.3. Usage of CxpStreamPacketCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpStreamPacketCountSelector = i;
    value_ = CxpStreamPacketCount;
}
```

2.4. PixelFormat

This parameter specifies the data format of the connected camera.

The formats defined in the following list can be selected. Choose the pixel format which best matches with your camera.

In this applet, the processing data bit depth is 16 bit. The camera interface automatically performs a conversion to the 16 bit format using bit shifting independently from the selected camera format. If the camera bit depth is greater than the processing bit depth, bits will be right shifted to meet the internal bit depth. If the camera bit depth is less than the processing bit depth, bits will be left shifted to meet the internal bit depth. In this case, the lower bits are fixed to zero.

This applet performs a Bayer de-mosaicing. The Bayer pattern is derived from the pixel format.

GenTL Controls the Pixel Format

The GenTL interface has a built in automatic adaptation of the pixel format to the camera settings. Changing the applet pixel format might be overwritten by the GenTL on acquisition start. You can only set the pixel format if the automatic setting is disabled. See the GenTL documentation parameter **`AutomaticFormatControl`** for more details.

Table 2.5. Parameter properties of PixelFormat

Property	Value	
Name	**`PixelFormat`**	
Display Name	**`Pixel Format`**	
Interface	**`IEnumeration`**	
Access policy	**`Read/Write/Change`**	
Visibility	**`Beginner`**	
Allowed values	**`BiColorRGBG8`**	BiColor RG BG 8
	`BiColorRGBG10`	BiColor RG BG 10
	`BiColorRGBG12`	BiColor RG BG 12
	`BiColorBGRG8`	BiColor BG RG 8
	`BiColorBGRG10`	BiColor BG RG 10
	`BiColorBGRG12`	BiColor BG RG 12
	`Mono8`	Mono 8
	`Mono10p`	Mono 10p
	`Mono12p`	Mono 12p
	`Mono14p`	Mono 14p
	`Mono16`	Mono 16p
	`RGB8`	RGB 8
	`RGB10p`	RGB 10p
	`RGB12p`	RGB 12p
	`RGB14p`	RGB 14p
	`RGB16`	RGB 16
	`YCbCr422_8`	YCbCr422_8
Default value	**`Mono8`**	

Example 2.4. Usage of PixelFormat

```
/* Set */ PixelFormat = Mono8;
/* Get */ value_ = PixelFormat;
```

2.5. SystemmonitorUsedCxpConnections

The currently used number of CXP ports used in this process.

Table 2.6. Parameter properties of SystemmonitorUsedCxpConnections

Property	Value
Name	**SystemmonitorUsedCxpConnections**
Display Name	**System Monitor Used Cxp Connections**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 1** **Maximum 4** **Stepsize 1**

Example 2.5. Usage of SystemmonitorUsedCxpConnections

```
/* Get */ value_ = SystemmonitorUsedCxpConnections;
```

2.6. SystemmonitorCxpImageLineMode

This parameter informs on the current transfer mode, used by the camera. The transfer can be an areascan (= 0) or linescan (= 1) image.

Table 2.7. Parameter properties of SystemmonitorCxpImageLineMode

Property	Value
Name	**SystemmonitorCxpImageLineMode**
Display Name	**System Monitor Cxp Image Line Mode**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 1** **Stepsize 1**

Example 2.6. Usage of SystemmonitorCxpImageLineMode

```
/* Get */ value_ = SystemmonitorCxpImageLineMode;
```

Chapter 3. Camera

This applet Acq_QuadCXP12Line for the imaWorx CXP-12 Quad acquires the sensor data of a line scan camera. When this is performed some sensor dimension depending information can be used to register an event based callback function.

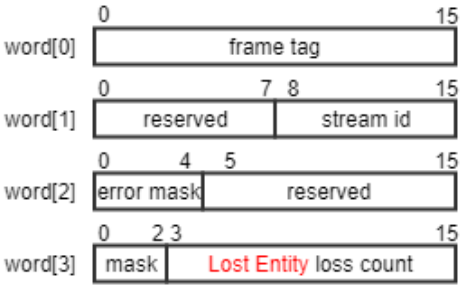
3.1. CameraEvents

In programming or runtime environments, a callback function is a piece of executable code that is passed as an argument, which is expected to call back (execute) exactly that time an event is triggered. This applet can generate some software callback events based on applet-events as explained in the following section. These events are not related to a special camera functionality. Other event sources are described in additional sections of this document.

The Basler Framegrabber SDK enables an application to get these event notifications about certain state changes at the data flow from camera to RAM and the image and trigger processing as well. Please consult the Basler Framegrabber SDK documentation for more details concerning the implementation of this functionality.

3.1.1. CameraStreamStatus

When the operator detects that the received reconstructed frame is larger or smaller than what was promoted by the camera in the CXP image header, a safety circuit gets activated. The operator then cuts off exceeding pixels and lines, so that the subsequent processing pipeline always sees the frame size which was defined in the image header. If the received frame is smaller in its dimensions than what was specified in the image header, the operator fills up the received frame with undefined data to achieve the specified frame dimensions which were defined in the image header. Filling up a smaller frame can cause the follow-up frames to get lost. The loss is then reported per event to the runtime software (Framegrabber SDK)(see the following paragraph). The size mismatch causes an event, too.



The event payload is provided as four 16-bit data words. The event format is defined as follows:

- word [0]
  - bits [0:15]: CXP image tag in which the event occurred.
- word [1]
  - bits [8:15]: Stream ID in which the event occurred.
  - bits [0:7]: Reserved, treat as don't care.
- word [2]
  - bit [0]: CRC error occurred.
  - bit [1]: Stream marker error detected in the image header.

- bit [2]: An error in the image header was detected which could not be corrected.
- bit [3]: A frame size error was detected, i.e. the image size defined in the CXP image header isn't matching the reconstructed frame size from the transmitted packets. This happens when the camera puts one info into the image header but transmits different amount of data as promoted in the header.
- bits [4:15]: Reserved, treat as don't care.

- word [3]
  - bit [0]: Event type, 0 = Corrupted Entity , 1 = Lost Entity.
    - **Corrupted Entity** means that the error happens within a frame and that this frame is already sourced into the VisualApplets pipeline.
    - **Lost Entity** means that the error occurred before the frame was forwarded to the following operators and the frame was discarded by the camera operator.
    - When a corrupted entity is observed, the operator will fill up the frame according to the CXP image header definition so that the following operators will not cause undefined behavior. During this fill-up, a new frame may arrive and will then get lost. The lost entity event will also be raised when the camera sends data with a gap according to the frame tag.
  - bit [1]: An event loss for type **Corrupted Entity** occurred. This means that preceding events of type **Corrupted Entity** got lost. This happens when the runtime software is not reacting to events and the internal event queues ran full.
  - bit [2]: An event loss for type **Lost Entity** occurred. This means that preceding events of type **Lost Entity** got lost. This happens when the runtime software (Framegrabber SDK) is not reacting to events and the internal event queues ran full.
  - bits [3:15]: amount of lost **Lost Entity** events.

There are two types of events: events for corrupted entities and events for lost entities. Bit 0 of word 3 describes which kind of event occurred. If the event buffers are full, it might happen that events get lost. When an event gets lost that marks a corrupted entity, bit 1 of word 3 will be set. When an event gets lost that marks a lost entity, bit 2 of word 3 will be set and bit 3 to 15 will provide the number of lost events indicating a lost frame. If bit 2 is set but the counter is 0, it means that a counter overflow happened.

Every event causes a software interrupt. To reduce the number of events, several events with the same frame tag might be merged together. In that case some error flags are combined. If an event was lost, the event before the lost event contains the information about the lost event and cannot be merged with further events with the same frame tag.

The events caused due to CRC errors report a frame tag, which may not be exactly related to the frame in which the CRC errors happen. The frame tag can be that of the preceding or following frame. This can only happen, when a camera sends a CXP packet, which contains a transition between 2 or more frames. The CRC computation is finished at the end of the packet, but the stream data is reconstructed on-the-fly. This means that a situation can happen, in which a CRC error is detected only after the preceding frame was already sent by the operator. In normal situations, in which the camera packets don't contain data both of the end of the ongoing frame and the beginning of the next frame, the frame tag during CRC error will always be correct. For all other cases as long as the complete frame stream data is less than the maximal packet size of 8k, there might be only 1 frame overlap within 1 packet. In that case, the software application should consider the preceding frame with the frame tag - 1 and the following frame with the frame tag + 1 as potentially corrupted as well.

Differentiating Error Events Between Taps

The error handling and event system are common to both CXP tap streams. Use the stream ID field to relate the received event to the appropriate tap. Normally, tap 0 will get a lower stream ID, typically 0. Tap 1 will get a stream ID, which is larger than the one of tap 0.

Table 3.1. Event parameters of CameraStreamStatus

Name	Interface	Description
EventCameraStream	IInteger	FrameId	Frame ID in which the event occurred.
EventCameraStrear	IInteger	Stream ID in which the event occurred.
EventCameraStream	IBoolean	CrcError	CRC error occurred.
EventCameraStrear	IBoolean	Stream marker error in image header occurred.
EventCameraStream	IBoolean	UncorrectableHeaderError	Uncorrectable error in image header occurred.
EventCameraStrear	IBoolean	Frame size error occurred.
EventCameraStream	IEnumeration	EventType	Event type (Corrupted or Lost).
EventCameraStrear	IBoolean	Event loss for type Corrupted occurred.
EventCameraStream	IBoolean	Lost	Event loss for type Lost occurred.
EventCameraStrear	IInteger	Amount of lost Lost events.

3.1.2. FrameTransferStart

3.1.3. FrameTransferEnd

3.1.4. LineTransferStart

This event is generated when the first pixel of camera line arrives at the framegrabber. Keep in mind that a high linerate can cause a critical high interrupt rate which might slow down the overall PC system. Even if the trigger setup will not use this line for a generated frame output this event will occour. This event can only occur if the acquisition is running.

3.1.5. LineTransferEnd

This event is generated when the last pixel of camera line has arrives at the framegrabber. Keep in mind that a high linerate can cause a critical high interrupt rate which might slow down the overall PC system. This event can only occur if the acquisition is running.

Chapter 4. SensorGeometry

Some operations, for example mirroring or tap sorting, require knowledge on the sensor dimension and orientation of the camera. The following parameters supply this kind of information.

4.1. VantagePoint

This parameter defines the vantage point. Use this parameter to mirror the image. Note that when using this parameter for mirroring, the received sensor image is mirrored and not the selected ROI in the frame grabber. Therefore, to mirror the ROI in the frame grabber, ensure to set the correct offsets in the frame grabber.

If a horizontal mirroring is active, the parameter *SensorWidth* limits the maximum width. The parameter dependency will then be *OffsetX* + *Width* <= *SensorWidth*.

If a vertical mirroring is active , the parameter *SensorHeight* limits the maximum height. The parameter dependency will then be *OffsetY* + *Height* <= *SensorHeight*.

Table 4.1. Parameter properties of VantagePoint

Property	Value	
Name	**VantagePoint**	
Display Name	**Vantage Point**	
Interface	**IEnumeration**	
Access policy	**Read/Write**	
Visibility	**Beginner**	
Allowed values	**TopLeft**	Top Left
	TopRight	Top Right
	BottomLeft	Bottom Left
	BottomRight	Bottom Right
Default value	**TopLeft**	

Example 4.1. Usage of VantagePoint

```
/* Set */ VantagePoint = TopLeft;
/* Get */ value_ = VantagePoint;
```

4.2. SensorWidth

To mirror the incoming data correctly, the parameter *SensorWidth* is required. The value of *SensorWidth* is ignored, if *VantagePoint* = Top-**Left** or Bottom-**Left**. If also a vertical mirroring is used, the available DRAM and sensor height limit the maximum sensor width. This is so, because the sensor image needs to fit twice into the DRAM, because double buffering is used.

If No Mirroring Is Active, the Value of *SensorWidth* Is Not Used

If no mirroring is active, the value of the parameter *SensorWidth* is not used. Instead, the sum of *OffsetX* and *Width* is used. This makes the use of the module easier as an extra configuration is avoided, if defaults are used.

Table 4.2. Parameter properties of SensorWidth

Property	Value
Name	**SensorWidth**
Display Name	**Sensor Width**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Beginner**
Allowed values	**Minimum 8** **Maximum 32768** **Stepsize 8**
Default value	**1024**
Unit of measure	**pixel**

Example 4.2. Usage of SensorWidth

```
/* Set */ SensorWidth = 1024;
/* Get */ value_ = SensorWidth;
```

4.3. SensorHeight

For vertical mirroring or tap geometry sorting in vertical direction, the applet needs to be parameterized with the exact height transferred from the camera to the frame grabber. If you have set a region of interest in the camera, the parameter *SensorHeight* needs to be set to the ROI size, otherwise use the sensor height.

If Only One Y-Zone Is Used and No Vertical Mirroring Is Active, the Value of *SensorHeight* Is Not Used

If no vertical mirroring is configured the value of the parameter *SensorHeight* is not used. Instead, the sum of *OffsetY* and *Height* is used. This makes the use of the module easier as an extra configuration is avoided, if defaults are used.

Table 4.3. Parameter properties of SensorHeight

Property	Value
Name	**SensorHeight**
Display Name	**Sensor Height**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Beginner**
Allowed values	**Minimum 1** **Maximum 8388607** **Stepsize 1**
Default value	**1024**
Unit of measure	**pixel**

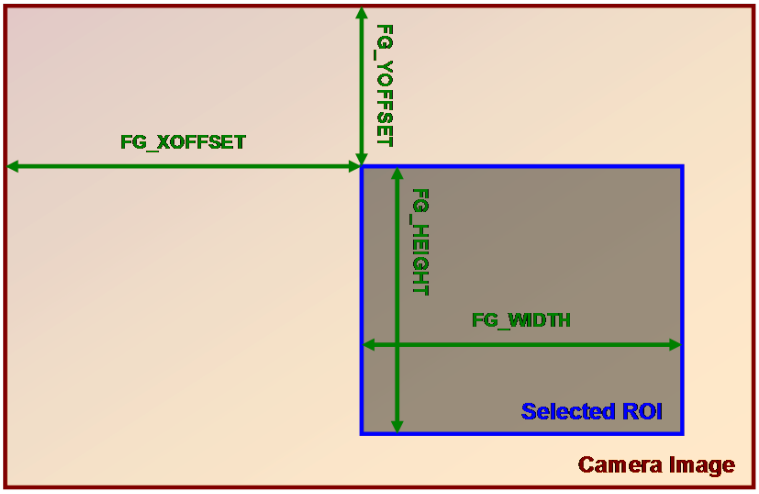
Example 4.3. Usage of SensorHeight

```
/* Set */ SensorHeight = 1024;
/* Get */ value_ = SensorHeight;
```

Chapter 5. ROI

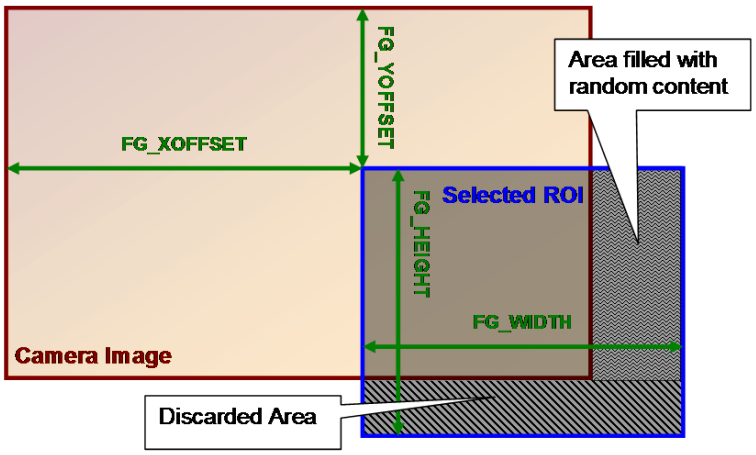
This module allows the definition of a region of interest (ROI), also called area of interest (AOI). A ROI allows the selection of a smaller subset pixel area from the input image. It is defined by using parameters *OffsetX*, *Width*, *OffsetY* and *Height*. The following figure illustrates the parameters.

Figure 5.1. Region of Interest



As can be seen, the region of interest lies within the input image dimensions. Thus, if the image dimension provided by the camera is greater or equal to the specified ROI parameters, the applet will fully cut-out the ROI subset pixel area. However, if the image provided by the camera is smaller than the specified ROI, lines will be filled with random pixel content and the image height might be cut or filled with random image lines as illustrated in the following.

Figure 5.2. Region of Interest Selection Outside the Input Image Dimensions



Furthermore, mind that the image sent by the camera must not exceed the maximum allowed image dimensions. This applet allows a maximum image width of 32768 pixels and a maximum image height of 8388607 lines. The chosen ROI settings can have a direct influence on the maximum bandwidth of the applet as they define the image size and thus, define the amount of data.

The parameters have dynamic value ranges. For example an x-offset cannot be set if the sum of the offset and the image width will exceed the maximum image width. To set a high x-offset, the image width has to be reduced, first. Hence, the order of setting the parameters for this module is important. The return values of the function calls in the SDK should always be evaluated to check if changes were accepted.

Mind the minimum step size of the parameters. This applet has a minimum step size of 8 pixel for the width and the x-offset, while the step size for the height and the y-offset is 1.

The settings made in this module will define the display size and buffer size if the applet is used in microDisplay. If you use the applet in your own programs, ensure to define a sufficient buffer size for the DMA transfers in your PC memory.

All ROI parameters can only be changed if the acquisition is not started i.e. stopped.

Automatic Adaptation to Camera Width and Height with the GenTL Adaptor

The GenTL adaptor can automatically copy the image width and height from the camera to the applet settings so that the user does not have to set these values. Changing the *Width* and *Height* of the applet might get overwritten by the Gen TL on acquisition start. You can only set the width and height if this automatic adaptation is disabled. See the GenTL documentation parameter **`AutomaticROIControl`** for more details.

ROI Setting Defines GenTL Buffer Info

The parameters define the DMA output size and therefore the GenTL buffer info values to inform the consumer about the used output image width and height of the interface. See the GenTL documentation parameter **`AutomaticROIControl`** for more details.

Influence on Bandwidth

A ROI might cause a strong reduction of the required bandwidth. If possible, the camera frame dimension should be reduced directly in the camera to the desired size instead of reducing the size in the applet. This will reduce the required bandwidth between the camera and the frame grabber.

5.1. Width

The parameter specifies the width of the ROI. The values of parameters *Width* + *OffsetX* must not exceed the maximum image width of 32768 pixels. If a horizontal mirroring is active the sensor width limits the maximum width (Width + XOffset). If furthermore vertical mirroring is active the maximum width is limited by the DRAM and sensor height (the sensor dimension needs to fit into the DRAM).

Maximum image width is reduced for horizontal mirrored images

Limitations of the available BRAM in the FPGA allow only to store smaller lines and there for the images that can be mirrored needs to be smaller. A mirrored image can only have width of 8192, the not mirrored image can have the full width of 32768.

Table 5.1. Parameter properties of Width

Property	Value
Name	**`Width`**
Display Name	**`Width`**
Interface	**`IInteger`**
Access policy	**`Read/Write`**
Visibility	**`Expert`**
Allowed values	**`Minimum 8`** **`Maximum 32768`** **`Stepsize 8`**
Default value	**`1024`**
Unit of measure	**`pixel`**

Example 5.1. Usage of Width

```
/* Set */ Width = 1024;
/* Get */ value_ = Width;
```

5.2. Height

The parameter specifies the height of the ROI. The values of parameters *Height* + *OffsetY* must not exceed the maximum image height of 8388607 pixels. If a vertical mirroring is active the sensor height limits the maximum height (Height + YOffset). Furthermore he maximum height is limited by the DRAM and the sensor width (the sensor dimension needs to fit into the DRAM).

Table 5.2. Parameter properties of Height

Property	Value
Name	**Height**
Display Name	**Height**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Expert**
Allowed values	**Minimum 1** **Maximum 8388607** **Stepsize 1**
Default value	**1024**
Unit of measure	**pixel**

Example 5.2. Usage of Height

```
/* Set */ Height = 1024;
/* Get */ value_ = Height;
```

5.3. OffsetX

The x-offset is defined by this parameter. If a horizontal mirroring is active the sensor width limits the maximum width (Width + XOffset). If furthermore vertical mirroring is active the maximum width is limited by the DRAM and the sensor height (the sensor dimension needs to fit into the DRAM).

Table 5.3. Parameter properties of OffsetX

Property	Value
Name	**OffsetX**
Display Name	**Offset X**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 32760** **Stepsize 8**
Default value	**0**
Unit of measure	**pixel**

Example 5.3. Usage of OffsetX

```
/* Set */ OffsetX = 0;
/* Get */ value_ = OffsetX;
```

5.4. OffsetY

The y-offset is defined by this parameter. If a vertical mirroring is active the sensor height limits the maximum height (Height + YOffset). Furthermore the maximum height is limited by the DRAM and the sensor width (the sensor dimension needs to fit into the DRAM).

Table 5.4. Parameter properties of OffsetY

Property	Value
Name	**OffsetY**
Display Name	**Offset Y**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Expert**
Allowed values	**Minimum** **0** **Maximum** **8388606** **Stepsize** **1**
Default value	**0**
Unit of measure	**pixel**

Example 5.4. Usage of OffsetY

```
/* Set */ OffsetY = 0;
/* Get */ value_ = OffsetY;
```

Chapter 6. DigitalIO

The frame grabber provides digital inputs and digital outputs for triggering, light synchronization, hardware control etc. This imaWorx CXP-12 Quad frame grabber has

- 8 general purpose digital inputs (GPIs) using the extension board connector of the frame grabber.
- 8 digital outputs on the GPO connector
- trigger over CXP cable function

This AcquisitionApplets allows an arbitrary mapping of the inputs to the trigger processing modules of the frame grabber. The same applies for the outputs: Any signal source from the trigger modules or digital inputs can be selected.

- **GND**: Value set to GND, zero. For digital outputs check for possibly inverted outputs.
- **VCC**: Value set to VCC, one. For digital outputs check for possibly inverted outputs.
- **SignalExsync**: The Exsync signal. Usually the line trigger signal used to trigger the camera. Check Chapter 7, '*LineTriggerExSync*' for more information.
- **SignalExsync2**: The Exsync 2 signal a delayed exsync signal. Check *LineTriggerDelay* for more information.
- **SignalFlash**: The flash signal. It is generated once at the start of each frame generated by the trigger module. Check Chapter 8, '*ImageTriggerFlash*' for more information.
- **SignalLineValid**: The line valid signal of the received camera or simulator image data. The signal is high for the duration of the line data transfer.
- **SignalFrameValid**: The frame valid signal after the trigger module. The signal is high for the duration of the frame data transfer. Depending on the image trigger mode, the image dimension and timing the signal can vary. See Chapter 8, '*ImageTriggerFlash*' for more information.
- Multi camera applet signal sources: The above signal source are available for all camera processes. Thus you can arbitrary select each signal. For example, you can use the same internal exsync signal to trigger all cameras. This allows a 100% synchronization of the cameras.
- **SignalGPI0** to **SignalGPI7**: Direct mapping of the digital input signal after debouncing.
- **SignalLineStart**: Line start pulse. Use for events and signal analyzer.
- **SignalLineEnd**: Line end pulse. Use for events and signal analyzer.
- **SignalFrameStart**: Frame start pulse. Use for events and signal analyzer.
- **SignalFrameEnd**: Frame end pulse. Use for events and signal analyzer.

6.1. CameraTriggerSource

For CoaXPress triggering, packets are sent to the camera instead of signals. A trigger signal usually consists of a pulse of a certain pulse length defining, for example, the duration time of the exposure. The start of the pulse, i.e. the rising edge, defines the start of the exposure. For most cameras the moment of this rising edge of the pulse is used to send a CXP trigger on CXP LinkTigger0. At the time of the falling edge, the CXP LinkTrigger1 is used by many cameras to end the exposure in a trigger controlled mode.

Thus, you need to select the source signals for the CXP link triggers and define whether you want to use the rising or falling edge. You can do this with the following parameter. Note that the camera must match with these settings.

6.1.1. CxpLinkTrigger0Source

Table 6.1. Parameter properties of CxpLinkTrigger0Source

Property	Value	
Name	**CxpLinkTrigger0Source**	
Display Name	**CXP Link Trigger 0 Source**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**GND**	GND
	VCC	VCC
	SignalExsync	Signal Exsync
	SignalExsync2	Signal Exsync2
	SignalFlash	Signal Flash
	SignalLineValid	Signal Line Valid
	SignalFrameValid	Signal Frame Valid
	SignalCam1Exsync	Signal Cam1 Exsync
	SignalCam1Exsync2	Signal Cam1 Exsync2
	SignalCam1Flash	Signal Cam1 Flash
	SignalCam1LineValid	Signal Cam1 Line Valid
	SignalCam1FrameValid	Signal Cam1 Frame Valid
	SignalCam2Exsync	Signal Cam2 Exsync
	SignalCam2Exsync2	Signal Cam2 Exsync2
	SignalCam2Flash	Signal Cam2 Flash
	SignalCam2LineValid	Signal Cam2 Line Valid
	SignalCam2FrameValid	Signal Cam2 Frame Valid
	SignalCam3Exsync	Signal Cam3 Exsync
	SignalCam3Exsync2	Signal Cam3 Exsync2
	SignalCam3Flash	Signal Cam3 Flash
	SignalCam3LineValid	Signal Cam3 Line Valid
	SignalCam3FrameValid	Signal Cam3 Frame Valid
	SignalGPI0	Signal GPI 0
	SignalGPI1	Signal GPI 1
	SignalGPI2	Signal GPI 2
	SignalGPI3	Signal GPI 3
	SignalGPI4	Signal GPI 4
	SignalGPI5	Signal GPI 5
	SignalGPI6	Signal GPI 6
	SignalGPI7	Signal GPI 7
	SignalFrontGPI0	Signal Front GPI 0
	SignalFrontGPI1	Signal Front GPI 1
	SignalFrontGPI2	Signal Front GPI 2
	SignalFrontGPI3	Signal Front GPI 3
Default value	**SignalExsync**	

Example 6.1. Usage of CxpLinkTrigger0Source

```
/* Set */ CxpLinkTrigger0Source = SignalExsync;
/* Get */ value_ = CxpLinkTrigger0Source;
```

6.1.2. CxpLinkTrigger0SourceEdge

Table 6.2. Parameter properties of CxpLinkTrigger0SourceEdge

Property	Value
Name	**CxpLinkTrigger0SourceEdge**
Display Name	**CXP Link Trigger 0 Source Edge**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**RisingEdge** Rising Edge **FallingEdge** Falling Edge
Default value	**RisingEdge**

Example 6.2. Usage of CxpLinkTrigger0SourceEdge

```
/* Set */ CxpLinkTrigger0SourceEdge = RisingEdge;
/* Get */ value_ = CxpLinkTrigger0SourceEdge;
```

6.1.3. CxpLinkTrigger1Source

Table 6.3. Parameter properties of CxpLinkTrigger1Source

Property	Value	
Name	**CxpLinkTrigger1Source**	
Display Name	**CXP Link Trigger 1 Source**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**GND**	GND
	VCC	VCC
	SignalExsync	Signal Exsync
	SignalExsync2	Signal Exsync2
	SignalFlash	Signal Flash
	SignalLineValid	Signal Line Valid
	SignalFrameValid	Signal Frame Valid
	SignalCam1Exsync	Signal Cam1 Exsync
	SignalCam1Exsync2	Signal Cam1 Exsync2
	SignalCam1Flash	Signal Cam1 Flash
	SignalCam1LineValid	Signal Cam1 Line Valid
	SignalCam1FrameValid	Signal Cam1 Frame Valid
	SignalCam2Exsync	Signal Cam2 Exsync
	SignalCam2Exsync2	Signal Cam2 Exsync2
	SignalCam2Flash	Signal Cam2 Flash
	SignalCam2LineValid	Signal Cam2 Line Valid
	SignalCam2FrameValid	Signal Cam2 Frame Valid
	SignalCam3Exsync	Signal Cam3 Exsync
	SignalCam3Exsync2	Signal Cam3 Exsync2
	SignalCam3Flash	Signal Cam3 Flash
	SignalCam3LineValid	Signal Cam3 Line Valid
	SignalCam3FrameValid	Signal Cam3 Frame Valid
	SignalGPI0	Signal GPI 0
	SignalGPI1	Signal GPI 1
	SignalGPI2	Signal GPI 2
	SignalGPI3	Signal GPI 3
	SignalGPI4	Signal GPI 4
	SignalGPI5	Signal GPI 5
	SignalGPI6	Signal GPI 6
	SignalGPI7	Signal GPI 7
	SignalFrontGPI0	Signal Front GPI 0
	SignalFrontGPI1	Signal Front GPI 1
	SignalFrontGPI2	Signal Front GPI 2
	SignalFrontGPI3	Signal Front GPI 3
Default value	**SignalExsync**	

Example 6.3. Usage of CxpLinkTrigger1Source

```
/* Set */ CxpLinkTrigger1Source = SignalExsync;
/* Get */ value_ = CxpLinkTrigger1Source;
```

6.1.4. CxpLinkTrigger1SourceEdge

Table 6.4. Parameter properties of CxpLinkTrigger1SourceEdge

Property	Value
Name	**CxpLinkTrigger1SourceEdge**
Display Name	**CXP Link Trigger 1 Source Edge**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**RisingEdge** Rising Edge **FallingEdge** Falling Edge
Default value	**FallingEdge**

Example 6.4. Usage of CxpLinkTrigger1SourceEdge

```
/* Set */ CxpLinkTrigger1SourceEdge = FallingEdge;
/* Get */ value_ = CxpLinkTrigger1SourceEdge;
```

6.1.5. CxpLinkTrigger2Source

Table 6.5. Parameter properties of CxpLinkTrigger2Source

Property	Value	
Name	**CxpLinkTrigger2Source**	
Display Name	**CXP Link Trigger 2 Source**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**GND**	GND
	VCC	VCC
	SignalExsync	Signal Exsync
	SignalExsync2	Signal Exsync2
	SignalFlash	Signal Flash
	SignalLineValid	Signal Line Valid
	SignalFrameValid	Signal Frame Valid
	SignalCam1Exsync	Signal Cam1 Exsync
	SignalCam1Exsync2	Signal Cam1 Exsync2
	SignalCam1Flash	Signal Cam1 Flash
	SignalCam1LineValid	Signal Cam1 Line Valid
	SignalCam1FrameValid	Signal Cam1 Frame Valid
	SignalCam2Exsync	Signal Cam2 Exsync
	SignalCam2Exsync2	Signal Cam2 Exsync2
	SignalCam2Flash	Signal Cam2 Flash
	SignalCam2LineValid	Signal Cam2 Line Valid
	SignalCam2FrameValid	Signal Cam2 Frame Valid
	SignalCam3Exsync	Signal Cam3 Exsync
	SignalCam3Exsync2	Signal Cam3 Exsync2
	SignalCam3Flash	Signal Cam3 Flash
	SignalCam3LineValid	Signal Cam3 Line Valid
	SignalCam3FrameValid	Signal Cam3 Frame Valid
	SignalGPI0	Signal GPI 0
	SignalGPI1	Signal GPI 1
	SignalGPI2	Signal GPI 2
	SignalGPI3	Signal GPI 3
	SignalGPI4	Signal GPI 4
	SignalGPI5	Signal GPI 5
	SignalGPI6	Signal GPI 6
	SignalGPI7	Signal GPI 7
	SignalFrontGPI0	Signal Front GPI 0
	SignalFrontGPI1	Signal Front GPI 1
	SignalFrontGPI2	Signal Front GPI 2
	SignalFrontGPI3	Signal Front GPI 3
Default value	**GND**	

Example 6.5. Usage of CxpLinkTrigger2Source

```
/* Set */ CxpLinkTrigger2Source = GND;
/* Get */ value_ = CxpLinkTrigger2Source;
```

6.1.6. CxpLinkTrigger2SourceEdge

Table 6.6. Parameter properties of CxpLinkTrigger2SourceEdge

Property	Value
Name	**CxpLinkTrigger2SourceEdge**
Display Name	**CXP Link Trigger 2 Source Edge**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**RisingEdge** Rising Edge **FallingEdge** Falling Edge
Default value	**RisingEdge**

Example 6.6. Usage of CxpLinkTrigger2SourceEdge

```
/* Set */ CxpLinkTrigger2SourceEdge = RisingEdge;
/* Get */ value_ = CxpLinkTrigger2SourceEdge;
```

6.1.7. CxpLinkTrigger3Source

Table 6.7. Parameter properties of CxpLinkTrigger3Source

Property	Value	
Name	**CxpLinkTrigger3Source**	
Display Name	**CXP Link Trigger 3 Source**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**GND**	GND
	VCC	VCC
	SignalExsync	Signal Exsync
	SignalExsync2	Signal Exsync2
	SignalFlash	Signal Flash
	SignalLineValid	Signal Line Valid
	SignalFrameValid	Signal Frame Valid
	SignalCam1Exsync	Signal Cam1 Exsync
	SignalCam1Exsync2	Signal Cam1 Exsync2
	SignalCam1Flash	Signal Cam1 Flash
	SignalCam1LineValid	Signal Cam1 Line Valid
	SignalCam1FrameValid	Signal Cam1 Frame Valid
	SignalCam2Exsync	Signal Cam2 Exsync
	SignalCam2Exsync2	Signal Cam2 Exsync2
	SignalCam2Flash	Signal Cam2 Flash
	SignalCam2LineValid	Signal Cam2 Line Valid
	SignalCam2FrameValid	Signal Cam2 Frame Valid
	SignalCam3Exsync	Signal Cam3 Exsync
	SignalCam3Exsync2	Signal Cam3 Exsync2
	SignalCam3Flash	Signal Cam3 Flash
	SignalCam3LineValid	Signal Cam3 Line Valid
	SignalCam3FrameValid	Signal Cam3 Frame Valid
	SignalGPI0	Signal GPI 0
	SignalGPI1	Signal GPI 1
	SignalGPI2	Signal GPI 2
	SignalGPI3	Signal GPI 3
	SignalGPI4	Signal GPI 4
	SignalGPI5	Signal GPI 5
	SignalGPI6	Signal GPI 6
	SignalGPI7	Signal GPI 7
	SignalFrontGPI0	Signal Front GPI 0
	SignalFrontGPI1	Signal Front GPI 1
	SignalFrontGPI2	Signal Front GPI 2
	SignalFrontGPI3	Signal Front GPI 3
Default value	**GND**	

Example 6.7. Usage of CxpLinkTrigger3Source

```
/* Set */ CxpLinkTrigger3Source = GND;
/* Get */ value_ = CxpLinkTrigger3Source;
```

6.1.8. CxpLinkTrigger3SourceEdge

Table 6.8. Parameter properties of CxpLinkTrigger3SourceEdge

Property	Value
Name	**CxpLinkTrigger3SourceEdge**
Display Name	**CXP Link Trigger 3 Source Edge**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**RisingEdge** Rising Edge **FallingEdge** Falling Edge
Default value	**RisingEdge**

Example 6.8. Usage of CxpLinkTrigger3SourceEdge

```
/* Set */ CxpLinkTrigger3SourceEdge = RisingEdge;
/* Get */ value_ = CxpLinkTrigger3SourceEdge;
```

6.2. GPO

6.2.1. TriggerOutGPO0Source et al.

Note

This description applies also to the following parameters: TriggerOutGPO1Source, TriggerOutGPO2Source, TriggerOutGPO3Source, TriggerOutGPO4Source, TriggerOutGPO5Source, TriggerOutGPO6Source, TriggerOutGPO7Source

Select the signal source of the General Purpose Output (GPO). For further explanation of the available sources see Chapter 6, '*DigitalIO*'.

You can change the polarity using parameter *TriggerOutGPO0Polarity*.

Table 6.9. Parameter properties of TriggerOutGPO0Source

Property	Value	
Name	**TriggerOutGPO0Source**	
Display Name	**Trigger Out GPO 0 Source**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**GND**	GND
	VCC	VCC
	SignalExsync	Signal Exsync
	SignalExsync2	Signal Exsync2
	SignalFlash	Signal Flash
	SignalLineValid	Signal Line Valid
	SignalFrameValid	Signal Frame Valid
	SignalCam1Exsync	Signal Cam1 Exsync
	SignalCam1Exsync2	Signal Cam1 Exsync2
	SignalCam1Flash	Signal Cam1 Flash
	SignalCam1LineValid	Signal Cam1 Line Valid
	SignalCam1FrameValid	Signal Cam1 Frame Valid
	SignalCam2Exsync	Signal Cam2 Exsync
	SignalCam2Exsync2	Signal Cam2 Exsync2
	SignalCam2Flash	Signal Cam2 Flash
	SignalCam2LineValid	Signal Cam2 Line Valid
	SignalCam2FrameValid	Signal Cam2 Frame Valid
	SignalCam3Exsync	Signal Cam3 Exsync
	SignalCam3Exsync2	Signal Cam3 Exsync2
	SignalCam3Flash	Signal Cam3 Flash
	SignalCam3LineValid	Signal Cam3 Line Valid
	SignalCam3FrameValid	Signal Cam3 Frame Valid
	SignalGPI0	Signal GPI 0
	SignalGPI1	Signal GPI 1
	SignalGPI2	Signal GPI 2
	SignalGPI3	Signal GPI 3
	SignalGPI4	Signal GPI 4
	SignalGPI5	Signal GPI 5
	SignalGPI6	Signal GPI 6
	SignalGPI7	Signal GPI 7
	SignalFrontGPI0	Signal Front GPI 0
	SignalFrontGPI1	Signal Front GPI 1
	SignalFrontGPI2	Signal Front GPI 2
	SignalFrontGPI3	Signal Front GPI 3
Default value	**SignalFlash**	

Example 6.9. Usage of TriggerOutGPO0Source

```
/* Set */ TriggerOutGPO0Source = SignalFlash;
/* Get */ value_ = TriggerOutGPO0Source;
```

6.2.2. TriggerOutGPO0Polarity et al.

Note

This description applies also to the following parameters: TriggerOutGPO1Polarity, TriggerOutGPO2Polarity, TriggerOutGPO3Polarity, TriggerOutGPO4Polarity, TriggerOutGPO5Polarity, TriggerOutGPO6Polarity, TriggerOutGPO7Polarity

Select the output polarity the General Purpose Output (GPO). For further explanation of the available sources see Chapter 6, '*DigitalIO*'.

Table 6.10. Parameter properties of TriggerOutGPO0Polarity

Property	Value
Name	**TriggerOutGPO0Polarity**
Display Name	**Trigger Out GPO 0 Polarity**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**LowActive** Low Active **HighActive** High Active
Default value	**HighActive**

Example 6.10. Usage of TriggerOutGPO0Polarity

```
/* Set */ TriggerOutGPO0Polarity = HighActive;
/* Get */ value_ = TriggerOutGPO0Polarity;
```

6.2.3. TriggerOutFrontGPO0Source et al.

Note

This description applies also to the following parameters: TriggerOutFrontGPO1Source

Select the signal source of the Front General Purpose Output (Front GPO). For further explanation of the available sources see Chapter 6, '*DigitalIO*'.

You can change the polarity using parameter *TriggerFrontOutGPO0Polarity*.

Table 6.11. Parameter properties of TriggerOutFrontGPO0Source

Property	Value	
Name	**TriggerOutFrontGPO0Source**	
Display Name	**Trigger Out Front GPO 0 Source**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**GND**	GND
	VCC	VCC
	SignalExsync	Signal Exsync
	SignalExsync2	Signal Exsync2
	SignalFlash	Signal Flash
	SignalLineValid	Signal Line Valid
	SignalFrameValid	Signal Frame Valid
	SignalCam1Exsync	Signal Cam1 Exsync
	SignalCam1Exsync2	Signal Cam1 Exsync2
	SignalCam1Flash	Signal Cam1 Flash
	SignalCam1LineValid	Signal Cam1 Line Valid
	SignalCam1FrameValid	Signal Cam1 Frame Valid
	SignalCam2Exsync	Signal Cam2 Exsync
	SignalCam2Exsync2	Signal Cam2 Exsync2
	SignalCam2Flash	Signal Cam2 Flash
	SignalCam2LineValid	Signal Cam2 Line Valid
	SignalCam2FrameValid	Signal Cam2 Frame Valid
	SignalCam3Exsync	Signal Cam3 Exsync
	SignalCam3Exsync2	Signal Cam3 Exsync2
	SignalCam3Flash	Signal Cam3 Flash
	SignalCam3LineValid	Signal Cam3 Line Valid
	SignalCam3FrameValid	Signal Cam3 Frame Valid
	SignalGPI0	Signal GPI 0
	SignalGPI1	Signal GPI 1
	SignalGPI2	Signal GPI 2
	SignalGPI3	Signal GPI 3
	SignalGPI4	Signal GPI 4
	SignalGPI5	Signal GPI 5
	SignalGPI6	Signal GPI 6
	SignalGPI7	Signal GPI 7
	SignalFrontGPI0	Signal Front GPI 0
	SignalFrontGPI1	Signal Front GPI 1
	SignalFrontGPI2	Signal Front GPI 2
	SignalFrontGPI3	Signal Front GPI 3
Default value	**SignalFlash**	

Example 6.11. Usage of TriggerOutFrontGPO0Source

```
/* Set */ TriggerOutFrontGPO0Source = SignalFlash;
/* Get */ value_ = TriggerOutFrontGPO0Source;
```

6.2.4. TriggerFrontOutGPO0Polarity et al.

Note

This description applies also to the following parameters: TriggerFrontOutGPO1Polarity

Select the output polarity the Front General Purpose Output (Front GPO). For further explanation of the available sources see Chapter 6, '*DigitalIO*'.

Table 6.12. Parameter properties of TriggerFrontOutGPO0Polarity

Property	Value
Name	**TriggerFrontOutGPO0Polarity**
Display Name	**Trigger Front Out GPO 0 Polarity**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**LowActive** Low Active **HighActive** High Active
Default value	**HighActive**

Example 6.12. Usage of TriggerFrontOutGPO0Polarity

```
/* Set */ TriggerFrontOutGPO0Polarity = HighActive;
/* Get */ value_ = TriggerFrontOutGPO0Polarity;
```

6.3. GPIState

6.3.1. DigitalInput

Parameter *DigitalInput* is used to monitor the digital inputs of the frame grabber. This AcquisitionApplets has 12 digital inputs. You can read the current state of these inputs using parameter *DigitalInput*. Bit 0 of the read value represents input 0, bit 1 represents input 1 and so on. For example, if you obtain the value 37 or hexadecimal 0x25, the frame grabber will have high level on its digital inputs 0, 2 and 5.

Table 6.13. Parameter properties of DigitalInput

Property	Value
Name	**DigitalInput**
Display Name	**Digital Input**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum** 0 **Maximum** 4095 **Stepsize** 1
Unit of measure	

Example 6.13. Usage of DigitalInput

```
/* Get */ value_ = DigitalInput;
```

6.4. EventSource

6.4.1. CustomSignalEvent0Source

Select the source for the custom signal event.

Property	Value	
Name	**CustomSignalEvent0Source**	DigitalIO
Display Name	**Custom Signal Event 0 Source**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**GND**	GND
	VCC	VCC
	SignalExsync	Signal Exsync
	SignalExsync2	Signal Exsync2
	SignalFlash	Signal Flash
	SignalLineValid	Signal Line Valid
	SignalFrameValid	Signal Frame Valid
	SignalLineStart	Signal Line Start
	SignalLineEnd	Cam0 Line Transfer End
	SignalFrameStart	Signal Frame Start
	SignalFrameEnd	Signal Frame End
	SignalCam1Exsync	Signal Cam1 Exsync
	SignalCam1Exsync2	Signal Cam1 Exsync2
	SignalCam1Flash	Signal Cam1 Flash
	SignalCam1LineValid	Signal Cam1 Line Valid
	SignalCam1FrameValid	Signal Cam1 Frame Valid
	SignalCam1LineStart	Signal Cam1 Line Start
	SignalCam1LineEnd	Signal Cam1 Line End
	SignalCam1FrameStart	Signal Cam1 Frame Start
	SignalCam1FrameEnd	Signal Cam1 Frame End
	SignalCam2Exsync	Signal Cam2 Exsync
	SignalCam2Exsync2	Signal Cam2 Exsync2
	SignalCam2Flash	Signal Cam2 Flash
	SignalCam2LineValid	Signal Cam2 Line Valid
	SignalCam2FrameValid	Signal Cam2 Frame Valid
	SignalCam2LineStart	Signal Cam2 Line Start
	SignalCam2LineEnd	Signal Cam2 Line End
	SignalCam2FrameStart	Signal Cam2 Frame Start
	SignalCam2FrameEnd	Signal Cam2 Frame End
	SignalCam3Exsync	Signal Cam3 Exsync
	SignalCam3Exsync2	Signal Cam3 Exsync2
	SignalCam3Flash	Signal Cam3 Flash
	SignalCam3LineValid	Signal Cam3 Line Valid
	SignalCam3FrameValid	Signal Cam3 Frame Valid
	SignalCam3LineStart	Signal Cam3 Line Start
	SignalCam3LineEnd	Signal Cam3 Line End
	SignalCam3FrameStart	Signal Cam3 Frame Start
	SignalCam3FrameEnd	Signal Cam3 Frame End
	SignalGPI0	Signal GPI 0
	SignalGPI1	Signal GPI 1
	SignalGPI2	Signal GPI 2
	SignalGPI3	Signal GPI 3
	SignalGPI4	Signal GPI 4
	SignalGPI5	Signal GPI 5
	SignalGPI6	Signal GPI 6
	SignalGPI7	Signal GPI 7
	SignalFrontGPI0	Signal Front GPI 0
	SignalFrontGPI1	Signal Front GPI 1
	SignalFrontGPI2	Signal Front GPI 2
	SignalFrontGPI3	Signal Front GPI 3
Default value	**SignalExsync**	

Table 6.14: Parameter properties of CustomSignalEvent0Source

Example 6.14. Usage of CustomSignalEvent0Source

```
/* Set */ CustomSignalEvent0Source = SignalExsync;
/* Get */ value_ = CustomSignalEvent0Source;
```

6.4.2. CustomSignalEvent0Polarity

Select the polarity for the custom signal event.

Table 6.15. Parameter properties of CustomSignalEvent0Polarity

Property	Value
Name	**CustomSignalEvent0Polarity**
Display Name	**Custom Signal Event 0 Polarity**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**LowActive** Low Active **HighActive** High Active
Default value	**HighActive**

Example 6.15. Usage of CustomSignalEvent0Polarity

```
/* Set */ CustomSignalEvent0Polarity = HighActive;
/* Get */ value_ = CustomSignalEvent0Polarity;
```

6.4.3. CustomSignalEvent1Source

Select the source for the custom signal event.

Property	Value	
Name	**CustomSignalEvent1Source**	DigitalIO
Display Name	**Custom Signal Event 1 Source**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**GND**	GND
	VCC	VCC
	SignalExsync	Signal Exsync
	SignalExsync2	Signal Exsync2
	SignalFlash	Signal Flash
	SignalLineValid	Signal Line Valid
	SignalFrameValid	Signal Frame Valid
	SignalLineStart	Signal Line Start
	SignalLineEnd	Cam0 Line Transfer End
	SignalFrameStart	Signal Frame Start
	SignalFrameEnd	Signal Frame End
	SignalCam1Exsync	Signal Cam1 Exsync
	SignalCam1Exsync2	Signal Cam1 Exsync2
	SignalCam1Flash	Signal Cam1 Flash
	SignalCam1LineValid	Signal Cam1 Line Valid
	SignalCam1FrameValid	Signal Cam1 Frame Valid
	SignalCam1LineStart	Signal Cam1 Line Start
	SignalCam1LineEnd	Signal Cam1 Line End
	SignalCam1FrameStart	Signal Cam1 Frame Start
	SignalCam1FrameEnd	Signal Cam1 Frame End
	SignalCam2Exsync	Signal Cam2 Exsync
	SignalCam2Exsync2	Signal Cam2 Exsync2
	SignalCam2Flash	Signal Cam2 Flash
	SignalCam2LineValid	Signal Cam2 Line Valid
	SignalCam2FrameValid	Signal Cam2 Frame Valid
	SignalCam2LineStart	Signal Cam2 Line Start
	SignalCam2LineEnd	Signal Cam2 Line End
	SignalCam2FrameStart	Signal Cam2 Frame Start
	SignalCam2FrameEnd	Signal Cam2 Frame End
	SignalCam3Exsync	Signal Cam3 Exsync
	SignalCam3Exsync2	Signal Cam3 Exsync2
	SignalCam3Flash	Signal Cam3 Flash
	SignalCam3LineValid	Signal Cam3 Line Valid
	SignalCam3FrameValid	Signal Cam3 Frame Valid
	SignalCam3LineStart	Signal Cam3 Line Start
	SignalCam3LineEnd	Signal Cam3 Line End
	SignalCam3FrameStart	Signal Cam3 Frame Start
	SignalCam3FrameEnd	Signal Cam3 Frame End
	SignalGPI0	Signal GPI 0
	SignalGPI1	Signal GPI 1
	SignalGPI2	Signal GPI 2
	SignalGPI3	Signal GPI 3
	SignalGPI4	Signal GPI 4
	SignalGPI5	Signal GPI 5
	SignalGPI6	Signal GPI 6
	SignalGPI7	Signal GPI 7
	SignalFrontGPI0	Signal Front GPI 0
	SignalFrontGPI1	Signal Front GPI 1
	SignalFrontGPI2	Signal Front GPI 2
	SignalFrontGPI3	Signal Front GPI 3
Default value	**SignalFlash**	

Table 6.16: Parameter properties of CustomSignalEvent1Source

Example 6.16. Usage of CustomSignalEvent1Source

```
/* Set */ CustomSignalEvent1Source = SignalFlash;
/* Get */ value_ = CustomSignalEvent1Source;
```

6.4.4. CustomSignalEvent1Polarity

Select the polarity for the custom signal event.

Table 6.17. Parameter properties of CustomSignalEvent1Polarity

Property	Value
Name	**CustomSignalEvent1Polarity**
Display Name	**Custom Signal Event 1 Polarity**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**LowActive** Low Active **HighActive** High Active
Default value	**HighActive**

Example 6.17. Usage of CustomSignalEvent1Polarity

```
/* Set */ CustomSignalEvent1Polarity = HighActive;
/* Get */ value_ = CustomSignalEvent1Polarity;
```

6.5. Events

In programming or runtime environments, a callback function is a piece of executable code that is passed as an argument, which is expected to call back (execute) exactly that time an event is triggered. This applet can generate some software callback events based on trigger inputs as explained in the following section. These events are not related to a special camera functionality. Other event sources are described in additional sections of this document.

Basler Framegrabber SDK enables an application to get these event notifications about certain state changes at the data flow from camera to RAM and the image and trigger processing as well. Please consult the Basler Framegrabber SDK documentation for more details concerning the implementation of this functionality.

6.5.1. Line0RisingEdge

This event is generated for each rising signal edge at trigger input 0. Except for the timestamp, the event has no additional data included. Keep in mind that fast changes of the input signal can cause high interrupt rates which might slow down the system. This event can occur independent of the acquisition status.

6.5.2. Line0FallingEdge

This event is generated for each falling signal edge at trigger input 0. Except for the timestamp, the event has no additional data included. Keep in mind that fast changes of the input signal can cause high interrupt rates which might slow down the system. This event can occur independent of the acquisition status.

6.5.3. CustomSignalEvent0

The event defined by *CustomSignalEvent0Source* and *CustomSignalEvent0Polarity*.

6.5.4. CustomSignalEvent1

The event defined by *CustomSignalEvent1Source* and *CustomSignalEvent1Polarity*.

Chapter 7. LineTriggerExSync

The line trigger function block uses signals to control the line scan acquisition of the specific camera. A external synchronization signal or internal generated puls with fixed frequency being sent to the line scan camera is called ExSync. With the help of this signal it is possible to control the exposure of the connected camera.

The camera needs to be configured accordingly to use the ExSync as control signal. Furthermore the camera might expect the ExSync at a particular CC signal and/or polarity.

For CoaXPress the the exposure control is sent in two independent packets. A single start- and a single end-packet. The time in between is interpreted as pulse width. The timing of these is very precise.

An sensor exposure control based on pulse length/duration is very common. Please make sure that the exposure time is less than the period of the expected maximum line frequency. Consult the camera's manual for more details because these are device specific. More details concerning ExSync can be found in the parameter description of *ExSyncOn*.

Basically two different generation modes for the ExSync signals are available,

- a simple periodical and
- an externally triggered generation.

Additionally, two variants of these are available,

- the first is independent from the image gate,
- and the second is gated by the image gate, which creates ExSync signals only during the actual acquisition.

All details can be found in the parameter description of *LineTriggerMode*.

For the mapping of the ExSync signals to the digital outputs check Chapter 6, '*DigitalIO*'.

7.1. LineTriggerMode

Please choose one of the line trigger modes described here. Make sure that the operation modes of the frame grabber and the camera are the same.

Image independent ExSync modes:

- **Grabber Controlled**

  For the grabber controlled line trigger, the ExSync signal is a simple periodical signal. Its period defines the line frequency and its active time is used by many cameras to define the exposure time.

- **External Trigger**

  The external trigger mode for ExSync generates a single ExSync pulse when the external trigger source becomes active. The ExSync defines the exposure time for the camera. During the exposure time is not possible to re-trigger the ExSync. If the camera needs an additional setup time, it is possible to extend the deadtime of the trigger - the time where no re-trigger is possible - beyond the exposure time. If you want to trigger fewer lines than pulses available at the trigger input, it is possible to downscale the trigger input, e.g. a downscaler of 2 will generate an ExSync every 2nd input pulse, a downscaler of 3 only every third of the input pulses, and so on.

Image gate dependent ExSync modes:

- **Grabber Controlled Gated**

For the grabber controlled gated line trigger, the ExSync signal is generated the very same way as for the grabber controlled mode described above. However, the generator for the ExSync is starting the rising image gate and stops with the image gate becoming inactive. This gives a smaller jitter for the time from the start of the image gate and the generation of the first ExSync, especially for very long ExSync periods.

- **External Trigger Gated**

  For the external trigger gated controlled line trigger, the ExSync signal is generated the very same way as for the external trigger mode described above. However, the generator for the ExSync is starting the rising image gate and stops with the image gate becoming inactive. For this mode two downscalers are available. The first is the downscaler from the beginning of the image gate to the first ExSync, it is called phase. The second is downscaling all succeeding input triggers and is the same as the downscaler used in external trigger mode described above. The options downscale and phase allow further adjustment of the camera trigger with respect to its external source, the trigger input. The value downscale determines the divisor of the input frequency, e.g. a downscale of 16 will produce an ExSync every 16 * n of the input trigger. Furthermore, the phase gives the possibility to shift the camera trigger. A phase shift of 90° is achieved when setting phase to 4, which produces a camera trigger at times 16 * n + 4 of the input trigger signal.

Table 7.1. Parameter properties of LineTriggerMode

Property	Value	
Name	**`LineTriggerMode`**	
Display Name	**`Line Trigger Mode`**	
Interface	**`IEnumeration`**	
Access policy	**`Read/Write`**	
Visibility	**`Beginner`**	
Allowed values	**`GrabberControlled`**	Grabber Controlled
	`AsyncExternalTrigger`	Async External Trigger
	`GrabberControlledGated`	Grabber Controlled Gated
	`AsyncGatedTrigger`	Async Gated Trigger
Default value	**`GrabberControlled`**	

Example 7.1. Usage of LineTriggerMode

```
/* Set */ LineTriggerMode = GrabberControlled;
/* Get */ value_ = LineTriggerMode;
```

7.2. ExSyncOn

This parameter enables the transmission of ExSync signals to the camera.

Please take care to first start the acquisition before setting this ExSyncOn parameter to On (**On**) if you want to acquire all lines being generated by the camera. The signal will be sent as soon as the ExSync has been started. As soon as the acquisition is started the used timeout parameter becomes valid independent of the ExSyncOn parameter being On (**On**) or Off (**Off**). By switching this parameter On (**On**) and Off (**Off**) during an acquisition you can check if the camera is configured to use this external signal for exposure start.

Whether the ExSync is really used by the camera is based on the settings of the camera. Consult the camera's manual for more details because these are device specific.

Table 7.2. Parameter properties of ExSyncOn

Property	Value
Name	**ExSyncOn**
Display Name	**Ex Sync On**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**On** On **Off** Off
Default value	**On**

Example 7.2. Usage of ExSyncOn

```
/* Set */ ExSyncOn = On;
/* Get */ value_ = ExSyncOn;
```

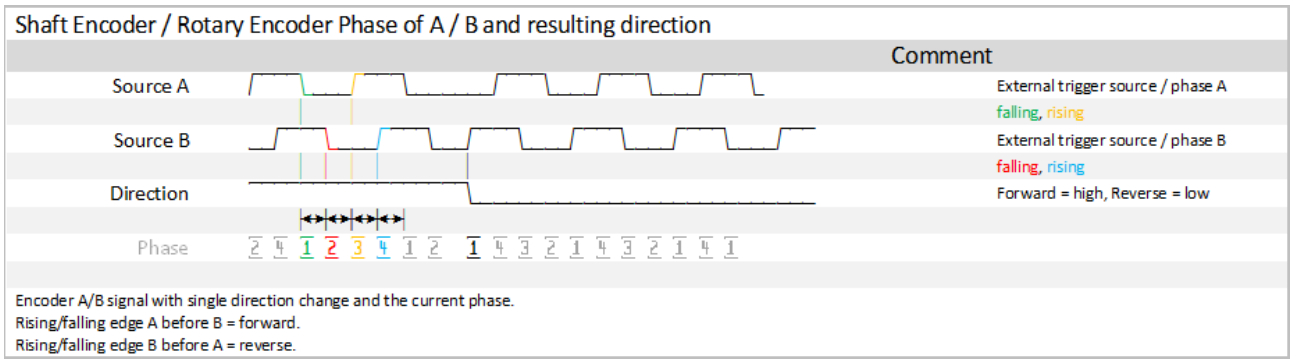
7.3. LineTriggerInput

In the line trigger input category of the line trigger module, the applet is configured for a possible external line trigger input. Here, debouncing times, downscales, polarities and a shaft encoder input are configured.

The external peripheral line trigger source will be in most cases a shaft encoder, also called a rotary encoder. These devices convert the objects movement over an angular motion into relative incremental pulses. The angular motion is taken from the motor axis or a wheel being connected to the translational motion of the scanned object. For most line scan applications it is relevant to get exact feedback of the relative motion between camera and object. By this a certain number of incremental pulses per distance is given to the frame grabber trigger input interface. Depending on the used incremental shaft encoders a certain number (500, 1000, ...) of incremental pulses per rotation is produced.

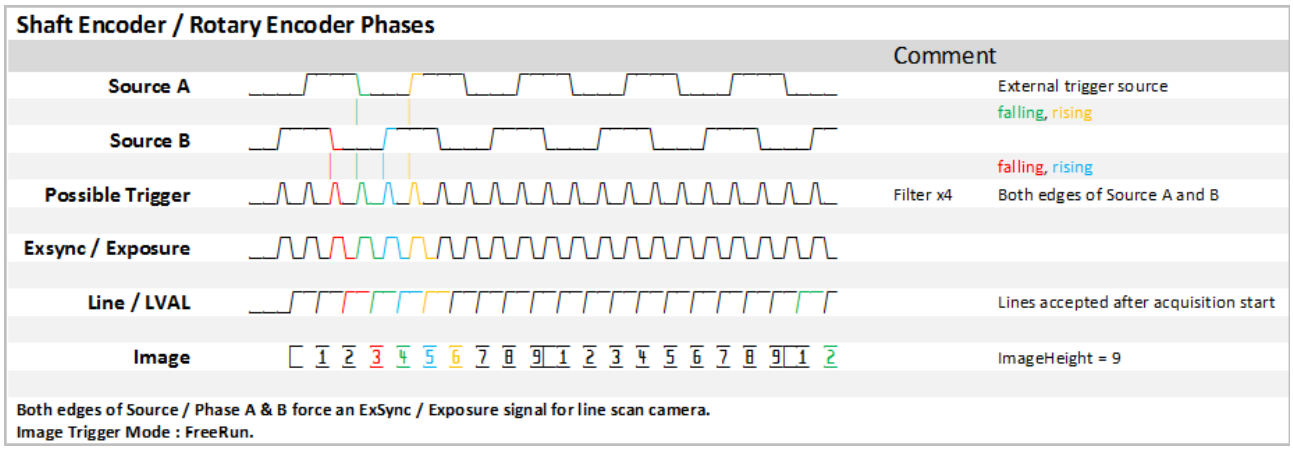
Most incremental shaft encoders provide 2 signals that are called A & B. By using these two signals the relative increments can be seen at the edges of these signals and a direction. In one direction the A-phase high state rises before the B-phase in the other direction, i.e. vice versa. If we do not need a direction for our application, only the A-phase is necessary. A combination of A & B may provide a higher resolution. Please see *ShaftEncoderMode* and *ShaftEncoderOn* for this.

Figure 7.1. Shaft Encoder, A & B phase, direction



During an acquisition the shaft encoder signals trigger the ExSync signals and force the sensor to perform an exposure. After the sensor exposure the line is read-out and transfered. The time between exposure and transfer is for most line scan cameras very short.

Figure 7.2. Shaft Encoder, A & B signal, acquisition



The different phases are defined as seen in the following table. A positive phase increment is forward direction, a negative means reverse. This induces rising/falling edge A before B equals forward direction and rising/falling edge B before A means reverse.

Table 7.3. Phases of an A/B Shaft Encoder

Phase	A-state	B-state
1	low	high
2	low	low
3	high	low
4	high	high

Some shaft encoders provide a third signal that is pulsed for each full rotation which is called Z or index. This signal Z could become interesting for an image trigger mode. For more details see Chapter 8, '*ImageTriggerFlash*'.

For most applications and several camera or line scan sensor types it is necessary to have the same resolution in X and Y direction of an image. Due to this the number of pixels per mm in sensor- and motion-direction needs to be the same. In case of an 1024 pixel line scan sensor looking at 10 cm we have 10.24 pixel per mm orthogonal to the web direction. In order to reach an 1:1 scaling we need 10.24 ExSync signals per mm. If a perfectly round object is scanned with an 1:1 scaling then it is exactly round in the image too. When the result becomes elliptic, the scaling is not perfect and some line scan sensor architectures (Bi/Tri-Linear, Dual-Line, ...) will show some additional artefacts.

7.3.1. LineTriggerInSource

This parameter specifies the digital signal source for phase A, which is used to trigger the ExSync signal. If an A/B shaft encoder is used, configure source B at *ShaftEncoderInputSource*, too. For more details consult the Framegrabber SDK manual.

It is possible to use the shaft encoder A phase only if the direction of scanning is not of interest in the target application. Concerning more details to the shaft encoder please consider the introduction of Section 7.3, 'LineTriggerInput'.

Table 7.4. Parameter properties of LineTriggerInSource

Property	Value	
Name	**LineTriggerInSource**	
Display Name	**Line Trigger In Source**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**GPITriggerSource0**	GPI Trigger Source 0
	GPITriggerSource1	GPI Trigger Source 1
	GPITriggerSource2	GPI Trigger Source 2
	GPITriggerSource3	GPI Trigger Source 3
	GPITriggerSource4	GPI Trigger Source 4
	GPITriggerSource5	GPI Trigger Source 5
	GPITriggerSource6	GPI Trigger Source 6
	GPITriggerSource7	GPI Trigger Source 7
	TriggerInSourceFrontGPI0	Trigger In Source Front GPI 0
	TriggerInSourceFrontGPI1	Trigger In Source Front GPI 1
	TriggerInSourceFrontGPI2	Trigger In Source Front GPI 2
	TriggerInSourceFrontGPI3	Trigger In Source Front GPI 3
Default value	**GPITriggerSource1**	

Example 7.3. Usage of LineTriggerInSource

```
/* Set */ LineTriggerInSource = GPITriggerSource1;
/* Get */ value_ = LineTriggerInSource;
```

7.3.2. LineTriggerInPolarity

The parameter defines the polarity of the external input trigger signal encoder source A and source B. When set to LowActive, the ExSync generator starts on a falling edge of the signal specified by the parameter *LineTriggerInSource*. Otherwise, the ExSync generation starts on a rising edge. This is only relevant if the *LineTriggerMode* is set to an external trigger.

Table 7.5. Parameter properties of LineTriggerInPolarity

Property	Value	
Name	**LineTriggerInPolarity**	
Display Name	**Line Trigger In Polarity**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**LowActive**	Low Active
	HighActive	High Active
Default value	**HighActive**	

Example 7.4. Usage of LineTriggerInPolarity

```
/* Set */ LineTriggerInPolarity = HighActive;
/* Get */ value_ = LineTriggerInPolarity;
```

7.3.3. LineTriggerDebouncing

This parameter specifies the debouncing time. This is the time for which the input line trigger signals must keep the same value to be detected as such. Fast signal changes within the debouncing time will be filtered out.

Table 7.6. Parameter properties of LineTriggerDebouncing

Property	Value
Name	**LineTriggerDebouncing**
Display Name	**Line Trigger Debouncing**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0032** **Maximum 26.0** **Stepsize 0.0032**
Default value	**0.112**
Unit of measure	**µs**

Example 7.5. Usage of LineTriggerDebouncing

```
/* Set */ LineTriggerDebouncing = 0.112;
/* Get */ value_ = LineTriggerDebouncing;
```

7.3.4. Downscale

7.3.4.1. LineDownscale

Sets the value after how many pulses of the input trigger signal a single one is passed through as ExSync. For example, a value of 2 creates an ExSync pulse at each 2nd input trigger signal. This is only relevant if the *LineTriggerMode* is set to an external trigger mode. The parameter *LineDownscaleInit* selects an initial delay of incoming pulses.

Figure 7.3. Downscale and Init phase behaviour

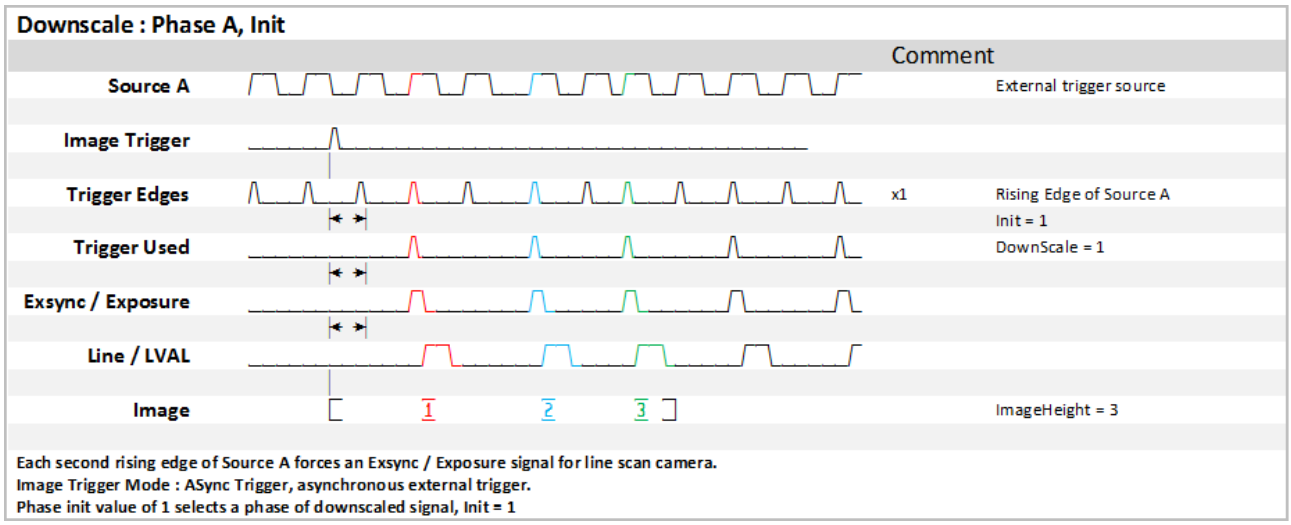


Table 7.7. Parameter properties of LineDownscale

Property	Value
Name	**LineDownscale**
Display Name	**Line Downscale**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Beginner**
Allowed values	**Minimum 1** **Maximum 255** **Stepsize 1**
Default value	**1**
Unit of measure	**pulses**

Example 7.6. Usage of LineDownscale

```
/* Set */ LineDownscale = 1;
/* Get */ value_ = LineDownscale;
```

7.3.4.2. LineDownscaleInit

In addition to the downscale value this parameter sets a phase position. This parameter specifies the number of external input trigger signals, which are needed to generate the first ExSync of a frame. This is only relevant if the *LineTriggerMode* is set to an image gate dependent ExSync mode. This value is applied after the image start pulse. The parameter *LineDownscale* represents the number of possible steps and an explaining figure is found in its description (Init=1).

Table 7.8. Parameter properties of LineDownscaleInit

Property	Value
Name	**LineDownscaleInit**
Display Name	**Line Downscale Init**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Beginner**
Allowed values	**Minimum 1** **Maximum 255** **Stepsize 1**
Default value	**1**
Unit of measure	**pulses**

Example 7.7. Usage of LineDownscaleInit

```
/* Set */ LineDownscaleInit = 1;
/* Get */ value_ = LineDownscaleInit;
```

7.4. ShaftEncoderABFilter

With the support of signal A/B for shaft encoders it is possible to detect the rotary direction of an attached encoder and filter the encoder signals accordingly. Also a compensation is performed for up to 16,777,216 reverse encoder signals. A brief description about this feature is found in the shaft encoder documentation.

7.4.1. ShaftEncoderOn

Switch the shaft encoder filter On or Off. This is only relevant if the *LineTriggerMode* is set to an external trigger mode. The functionalities of *ShaftEncoderMode*, *ShaftEncoderInputSource*, *ShaftEncoderLeading*, *ShaftEncoderCompensationEnable*, *ShaftEncoderCompensationCount* become relevant in the case this parameter is set to On = **On**. When enabling the shaft encoder, a reset of the encoder compensation is performed. If this filter is switched on an correct A & B encoder signal is expected and necessary for correct functionality. Please be aware that the input signal at *ShaftEncoderInputSource* is interpreted as phase B and the input signal at *LineTriggerInSource* as phase A. A sketch of the signal can be found in the description of parameter *LineTriggerInSource*.

Table 7.9. Parameter properties of ShaftEncoderOn

Property	Value
Name	**ShaftEncoderOn**
Display Name	**Shaft Encoder On**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**On** On **Off** Off
Default value	**Off**

Example 7.8. Usage of ShaftEncoderOn

```
/* Set */ ShaftEncoderOn = Off;
/* Get */ value_ = ShaftEncoderOn;
```

7.4.2. ShaftEncoderMode

The shaft encoder mode can be run in three operation modes. Please choose the according operation mode for your application. This feature can be used if *ShaftEncoderOn* is switched on. It enables you to adjust the number of increments per rotation of the shaft encoder. Together with the parameter *LineDownscale* you can adjust the increment re-scaling.

The following modes are available:

- Filter x1

  ExSync is generated for a forward rotation of the shaft encoder in single resolution, i.e. a trigger pulse for rising edge of Source A.

- Filter x2

  ExSync is generated for a forward rotation of the shaft encoder in double resolution, i.e. a trigger pulse for a rising and falling edge of Source A, edges of Source B are not used.

- Filter x4

  ExSync is generated for a forward rotation of the shaft encoder in quad resolution, i.e. a trigger pulse for a rising and falling edge of Source A and a rising and falling edge of Source B.

Figure 7.4. Shaft Encoder Mode : Filter x4, x2, x1

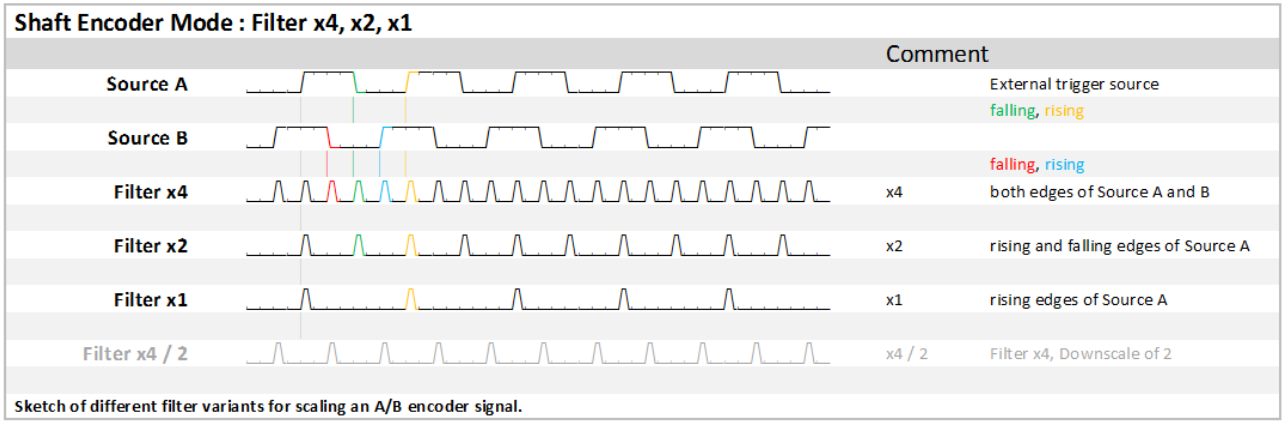


Table 7.10. Parameter properties of ShaftEncoderMode

Property	Value
Name	**ShaftEncoderMode**
Display Name	**Shaft Encoder Mode**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**FilterX1** Filter X1 **FilterX2** Filter X2 **FilterX4** Filter X4
Default value	**FilterX1**

Example 7.9. Usage of ShaftEncoderMode

```
/* Set */ ShaftEncoderMode = FilterX1;
/* Get */ value_ = ShaftEncoderMode;
```

7.4.3. ShaftEncoderInputSource

Specifies the input signal source / phase B for the shaft encoder filter. Signal source B of the shaft encoder is 90 degree phase shifted to source / phase A. In this document you can get more explanations regarding the input pins in the context of parameter *LineTriggerInSource* and concerning the shaft encoder in the introduction of Section 7.3, 'LineTriggerInput'. Check the hardware documentation of the microEnable trigger board and the Framegrabber SDK manual for more details.

Table 7.11. Parameter properties of ShaftEncoderInputSource

Property	Value	
Name	**ShaftEncoderInputSource**	
Display Name	**Shaft Encoder Input Source**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**GPITriggerSource0**	GPI Trigger Source 0
	GPITriggerSource1	GPI Trigger Source 1
	GPITriggerSource2	GPI Trigger Source 2
	GPITriggerSource3	GPI Trigger Source 3
	GPITriggerSource4	GPI Trigger Source 4
	GPITriggerSource5	GPI Trigger Source 5
	GPITriggerSource6	GPI Trigger Source 6
	GPITriggerSource7	GPI Trigger Source 7
	TriggerInSourceFrontGPI0	Trigger In Source Front GPI 0
	TriggerInSourceFrontGPI1	Trigger In Source Front GPI 1
	TriggerInSourceFrontGPI2	Trigger In Source Front GPI 2
	TriggerInSourceFrontGPI3	Trigger In Source Front GPI 3
Default value	**GPITriggerSource2**	

Example 7.10. Usage of ShaftEncoderInputSource

```
/* Set */ ShaftEncoderInputSource = GPITriggerSource2;
/* Get */ value_ = ShaftEncoderInputSource;
```

7.4.4. ShaftEncoderLeading

This parameter defines the leading signal (= direction) of the shaft encoder filter. This induces rising/falling edge A before B equals forward direction and rising/falling edge B before A means reverse. The default setting is A as the leading signal. Flipping the input pins or their polarity will have the same effect as changing this to B as the leading signal. It simply defines the valid direction of the scan. An explanation of the direction detection based on an encoder A / B signal is found in Section 7.3, 'LineTriggerInput'.

Table 7.12. Parameter properties of ShaftEncoderLeading

Property	Value	
Name	**ShaftEncoderLeading**	
Display Name	**Shaft Encoder Leading**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**SourceA**	Source A
	SourceB	Source B
Default value	**SourceA**	

Example 7.11. Usage of ShaftEncoderLeading

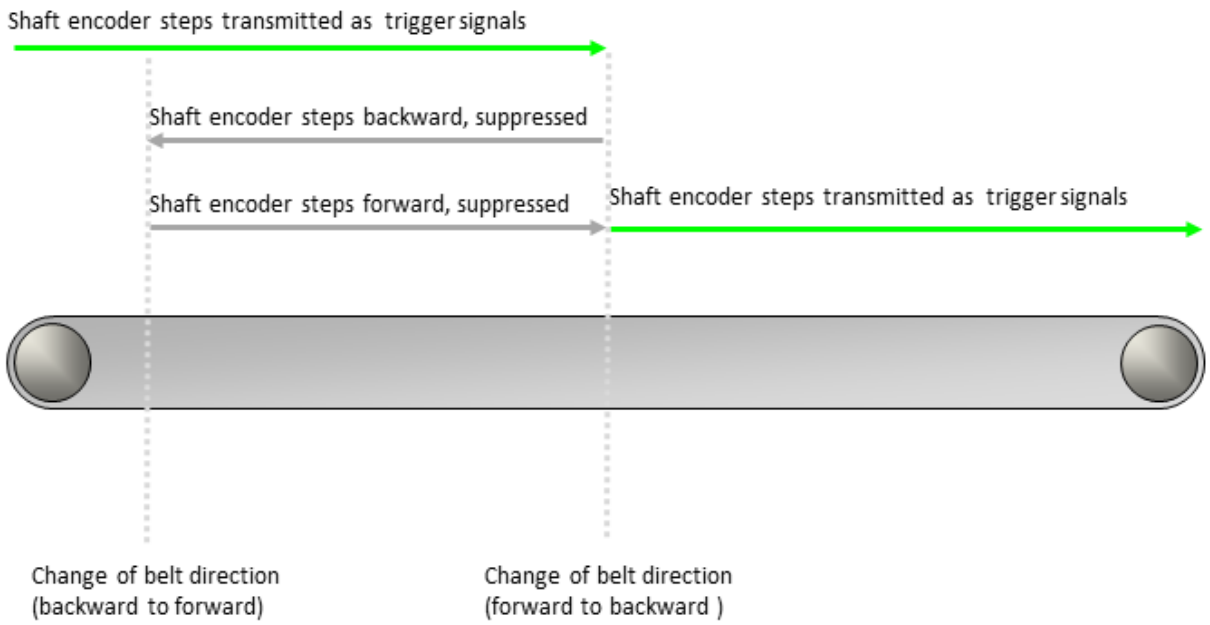
```
/* Set */ ShaftEncoderLeading = SourceA;
/* Get */ value_ = ShaftEncoderLeading;
```

7.4.5. ShaftEncoderCompensationEnable

The shaft encoder analyzer includes a rollback compensation. In case the rollback compensation is enabled, the module will compensate the reverse movement so that no object is scanned twice. The module will count the number of reverse pulses and will suppress all reverse and forward pulses until position of maximum progress is reached again. If switched to ON, in case of shaft encoder backward movement, the operator counts how many shaft encoder steps the shaft encoder moves backwards. When the shaft encoder moves forwards again, this number of shaft encoder steps (now forward direction) is not transmitted as external trigger signals. Only after the transportation belt is back to the place where the backward movement started, the shaft encoder steps (forward direction) are transmitted as external trigger signals again.

Parameter *ShaftEncoderCompensationEnable* switched ON:

Figure 7.5. Shaft Encoder Compensation Enable = ON



In case the rollback compensation is disabled, the shaft encoder analyzer will only suppress reverse pulses but use all forward pulses. If switched to OFF, the operator simply doesn't transmit any trigger signals as long as the transportation belt moves backwards. As soon as the transport belt starts to move forwards again, the operator transmits the shaft encoder steps (forward direction) as trigger signals.

Parameter *ShaftEncoderCompensationEnable* switched OFF:

Figure 7.6. Shaft Encoder Compensation Enable = OFF

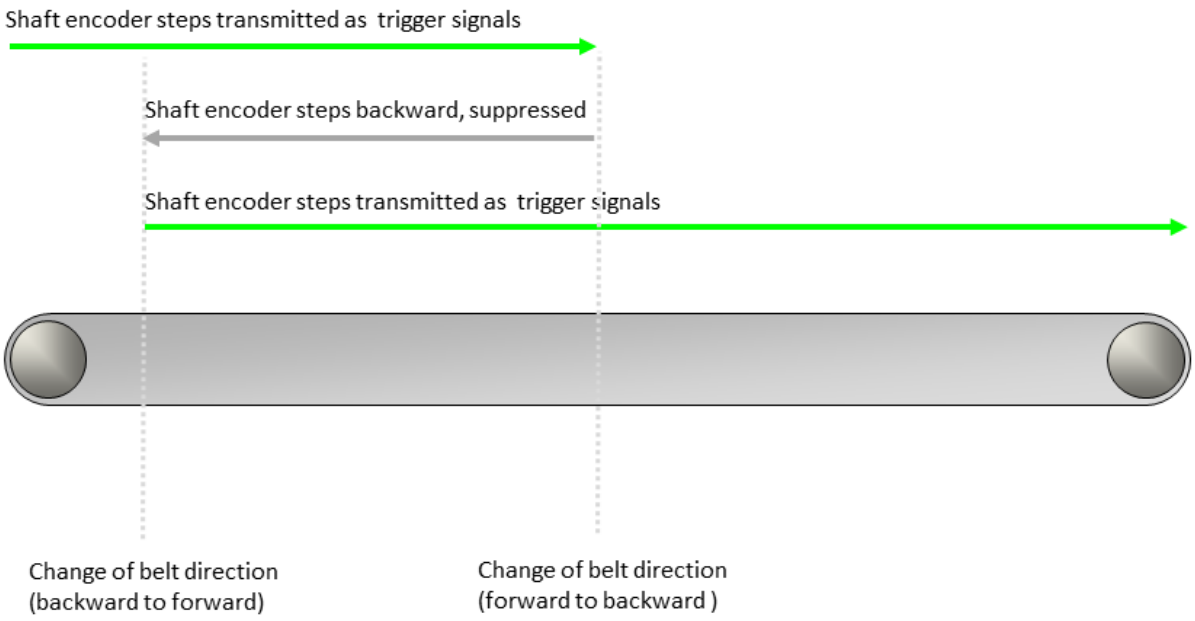


Table 7.13. Parameter properties of ShaftEncoderCompensationEnable

Property	Value
Name	**ShaftEncoderCompensationEnable**
Display Name	**Shaft Encoder Compensation Enable**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**On** On **Off** Off
Default value	**On**

Example 7.12. Usage of ShaftEncoderCompensationEnable

```
/* Set */ ShaftEncoderCompensationEnable = On;
/* Get */ value_ = ShaftEncoderCompensationEnable;
```

7.4.6. ShaftEncoderCompensationCount

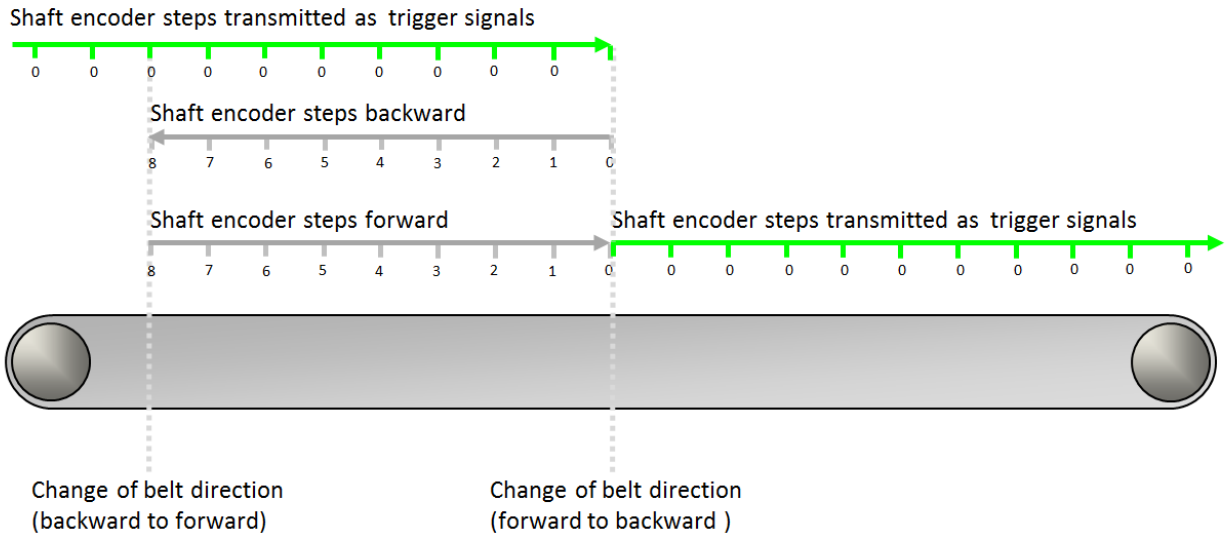
Using this parameter you can read and write the current shaft encoder rollback compensation counter. A compensation value zero indicates that currently no compensation is made. Therefore, you can reset the compensation by writing value zero to this parameter. Any other value will set a new compensation value. By knowing the distance / resolution for every encoder pulse, the compensation distance can be set. Concerning the shaft encoder find some more details in the introduction of Section 7.3, 'LineTriggerInput'.

It is based on a 20bit counter enabling a backward movement of up to 1048575 encoder pulses. An overflow of this value will not occur since it will skip all additional pulses for a compensation state of more than 1048575. By this the count of the rollback compensation is limited by 2 to the power of 20 pulses, what is enough for most applications in practice. As an example we could use a pretty high resolution of 20 pulses per mm, what is already sufficient for a maximum rollback distance of more than 50 meters.

Basic Conditions

If parameter *ShaftEncoderCompensationEnable* is set to ON, an internal counter counts the shaft encoder steps the transportation belt moves backwards. This is necessary to be able to compensate the exact number of shaft encoder steps when the transportation belt starts moving forwards again:

Figure 7.7. Shaft Encoder Compensation Enable = ON



The internal counter counts forwards as long as the transportation belt moves backwards. (In figure 7.7, from 0 to 8.)

The internal counter counts backwards while the transportation belt moves forwards. (In figure 7.7, from 8 to 0.)

When the internal counter holds the value 0, the shaft encoder steps are transmitted as trigger signals.

The value the internal counter holds at a given moment is the value of parameter *ShaftEncoderCompensationCount*. Only if this value is 0, encoder steps are transmitted as trigger signals. If the value of parameter *ShaftEncoderCompensationCount* is not 0, the shaft encoder steps are not transmitted as trigger signals and the value keeps changing with every encoder step until it reaches the value 0 again.

Reading the Parameter

The parameter *ShaftEncoderCompensationCount* is a read/write parameter. Therefore, at any given moment, you can always read out the value the counter holds at a given moment.

Defining an Offset

On the other hand, you can always modify the parameter value since you have write access during acquisition. If you need to define an offset to the standard encoder compensation, you can use this parameter to enter the number of steps you need the offset to be.

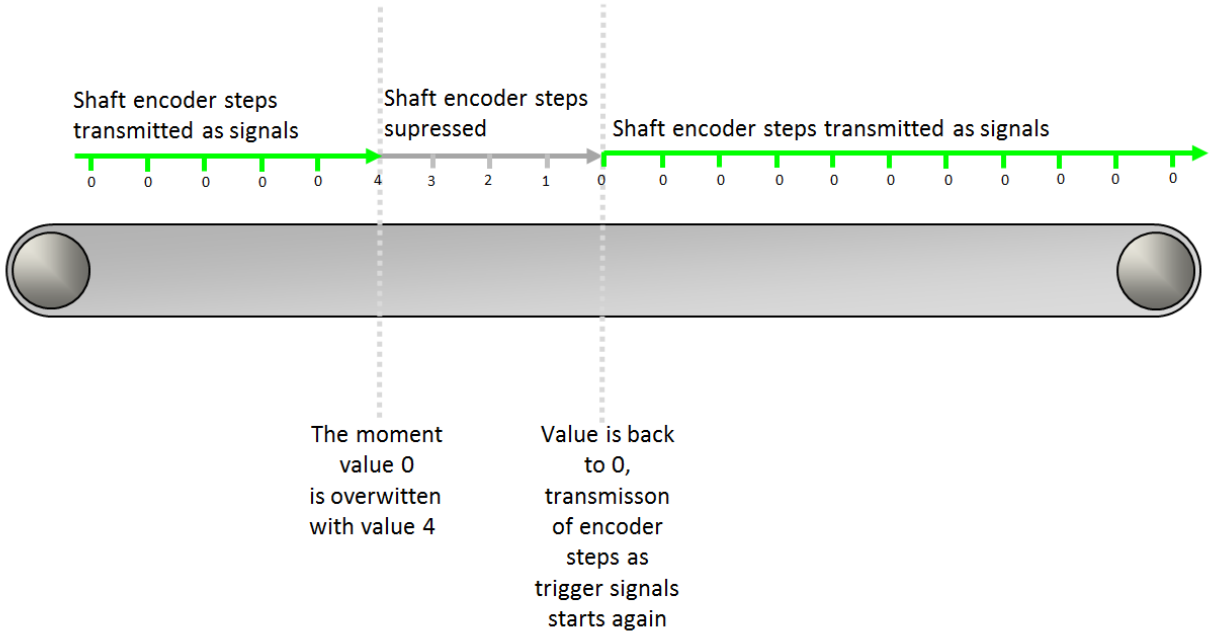
As soon as you enter a value for *ShaftEncoderCompensationCount*, this value overwrites the value the parameter holds before.

In the following let's look at some examples for overwriting the current value of *ShaftEncoderCompensationCount*:

Example 1:

The transportation belt is moving forward, the shaft encoder steps are transmitted as trigger signals, and the value of *ShaftEncoderCompensationCount* is 0. Then, the value 0 of *ShaftEncoderCompensationCount* is overwritten by value 4. Result: 4 shaft encoder steps are not transmitted as trigger signals.

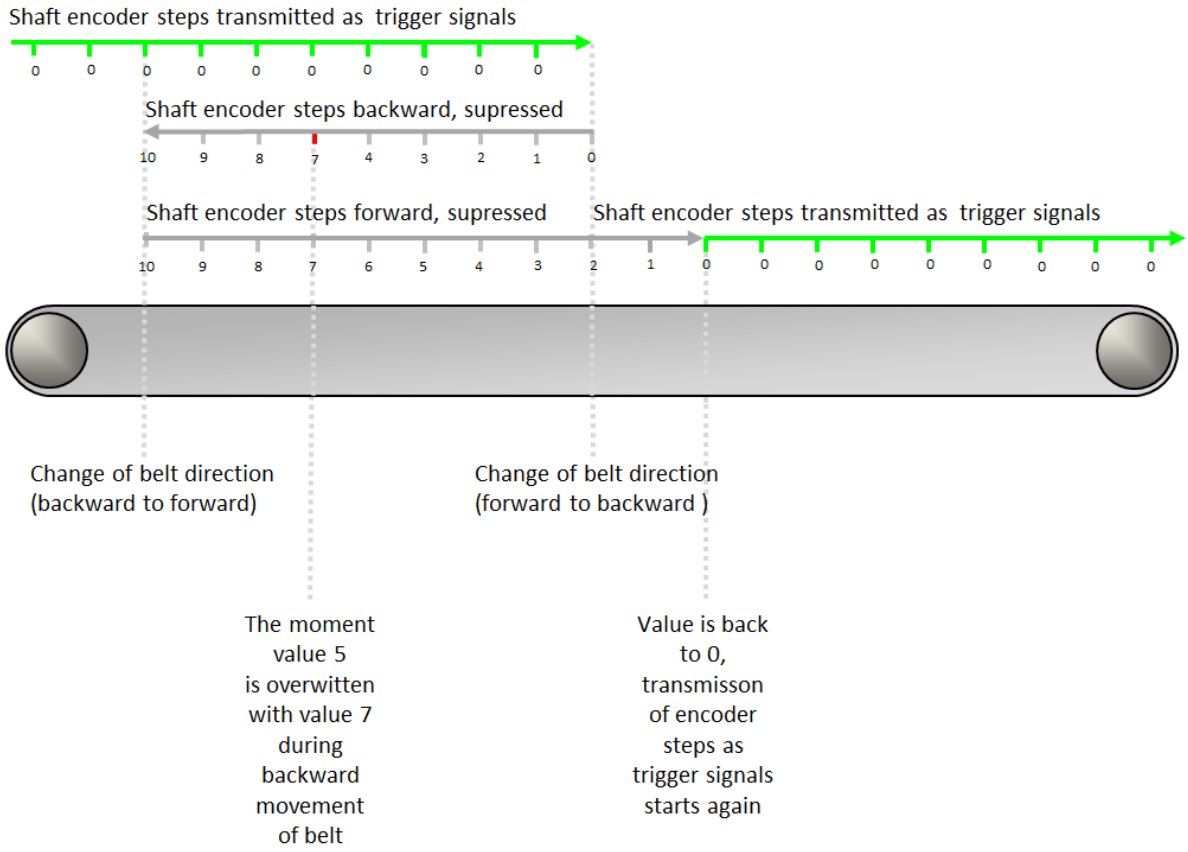
Figure 7.8. Shaft Encoder Compensation Count Example 1



Example 2:

The transportation belt is moving backward, the (backward) shaft encoder steps are suppressed, and the value of *ShaftEncoderCompensationCount* is not 0. Then, during backward movement of the transportation belt, the value 5 of *ShaftEncoderCompensationCount* is overwritten by value 7. Result: Offset of 2 shaft encoder steps.

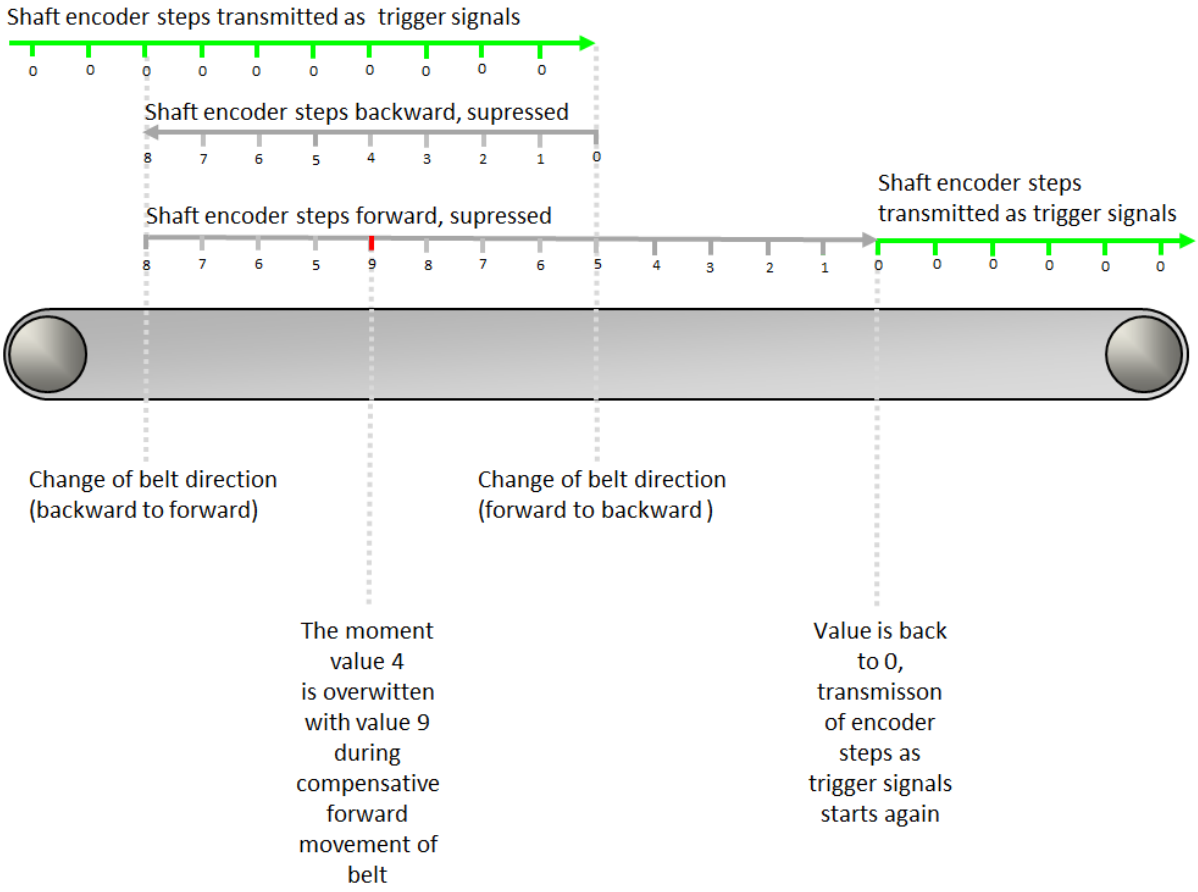
Figure 7.9. Shaft Encoder Compensation Count Example 2



Example 3:

The transportation belt is moving forward during compensation, the (forward) shaft encoder steps are suppressed, and the value of *ShaftEncoderCompensationCount* is not 0. Then, during compensative forward movement of the transportation belt, the value 4 of *ShaftEncoderCompensationCount* is overwritten with value 9. Result: Offset of 5 shaft encoder steps.

Figure 7.10. Shaft Encoder Compensation Count Example 3



Example 4:

The transportation belt is moving forward during compensation, the (forward) shaft encoder steps are suppressed, and the value of *ShaftEncoderCompensationCount* is not 0. Then, during compensative forward movement of the transportation belt, the value 4 of *ShaftEncoderCompensationCount* is overwritten with a smaller value, in our case with value 3. Result: Negative offset of -1 shaft encoder step.

Figure 7.11. Shaft Encoder Compensation Count Example 4

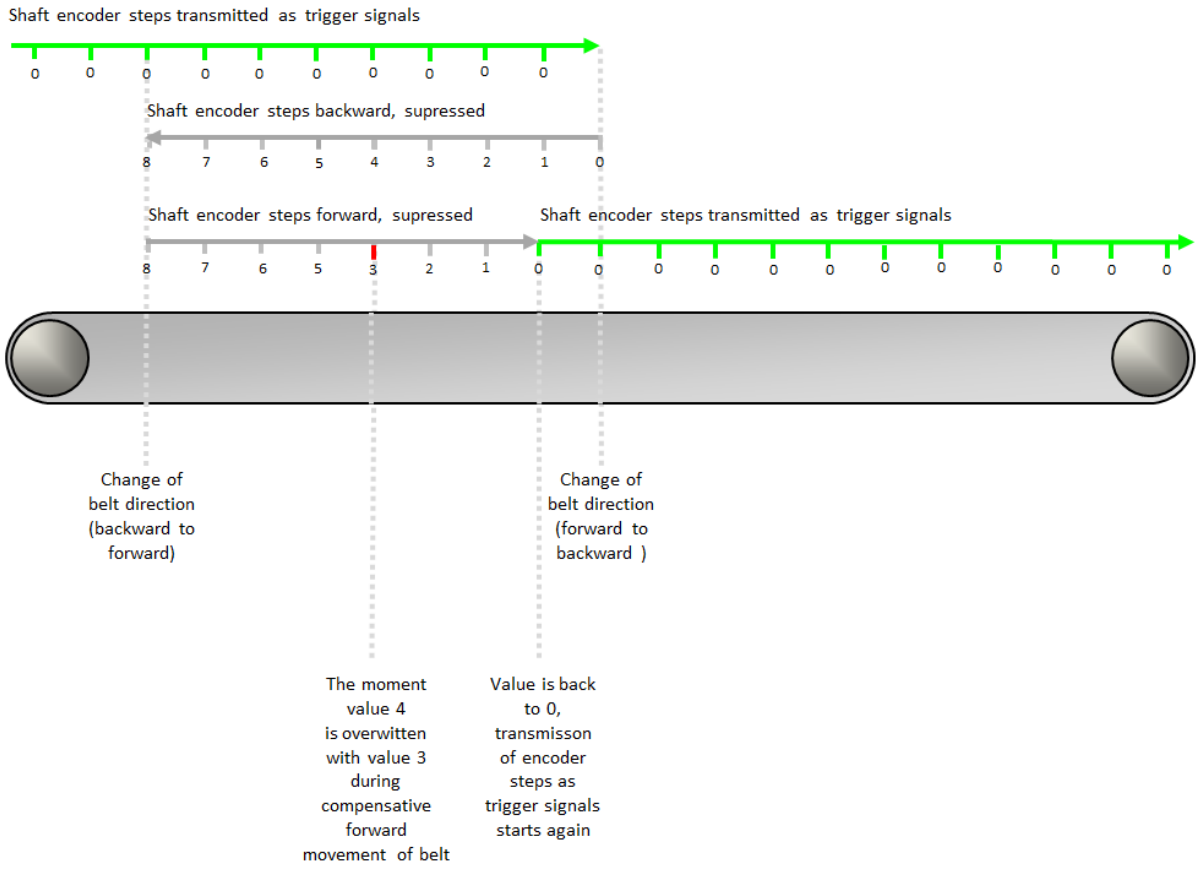


Table 7.14. Parameter properties of ShaftEncoderCompensationCount

Property	Value
Name	**ShaftEncoderCompensationCount**
Display Name	**Shaft Encoder Compensation Count**
Interface	**IInteger**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 1048575** **Stepsize 1**
Default value	**0**
Unit of measure	**pulses**

Example 7.13. Usage of ShaftEncoderCompensationCount

```
/* Set */ ShaftEncoderCompensationCount = 0;
/* Get */ value_ = ShaftEncoderCompensationCount;
```

7.5. ExSyncOutput

This category includes parameters to specify and parameterize the generated ExSync output signals.

7.5.1. LinePeriod

This parameter specifies the period of the ExSync signal. Therefore, it defines the line frequency when using the grabber controlled mode to trigger the connected camera. This period is of interest if a grabber controlled line trigger mode is used; more details for this can be found at *LineTriggerMode*. The line period is not allowed to be shorter than the minimum period - maximum line frequency - being supported by the camera, or in other words:

Please do not try to trigger the camera at a higher frequency than possible.

This maximum frequency is limited by the exposure time and the line scan sensor maximum speed. Please consider the camera manual for more details.

The following equations are mentioned in order to support the setup process if no period for *LinePeriod* is mentioned:

- **Frequency**

  The period **T** is the duration of time of one cycle in a repeating event, so the period is the reciprocal of the frequency **f**.

Equation 7.1. Frequency to Period

$$T = \frac{1}{f}$$

Equation 7.2. Example: 17.6 kHz to Period

$$\begin{aligned} T &= \frac{1}{F} = \frac{1}{17.6kHz} = \frac{1}{17600Hz} \\ T &= 0.0000568s = 0.0568ms = 56.8\mu s \end{aligned}$$

- **Velocity and Pixel / mm**

  The period **T** is the duration of time of one cycle in a repeating event. At a velocity **v** and a given number **n** of pixels / mm together with the number **n** of pixels / mm being based on the resolution count **r** of the line scan sensor pixels and the width of view **w** in mm the following equations are valid.

Equation 7.3. Velocity and Resolution to Period

$$\begin{aligned} n &= \frac{r}{w} \\ v &= \frac{distance}{time} \\ f &= v * n \\ T &= \frac{1}{f} \end{aligned}$$

Equation 7.4. Example: v = 53.4 m/min, r = 4096 pixels, w = 19.2 cm Wide Web to Period

$$\begin{aligned} n &= \frac{r}{w} = \frac{4096}{19.2cm} = \frac{4096}{192mm} = \frac{21.33}{mm} \\ v &= \frac{distance}{time} = \frac{53.4m}{min} = \frac{53.4m}{60s} = 0.89\frac{m}{s} \\ f &= v * n = 0.89\frac{m}{s} * \frac{21.33}{mm} = 890\frac{mm}{s} * \frac{21.33}{mm} \\ &= \frac{890 * 21.33}{s} = \frac{18983.7}{s} = 18983.7Hz = 18.9837kHz \\ T &= \frac{1}{f} \\ &= \frac{1}{18983.7Hz} = 52.68\mu s \end{aligned}$$

Table 7.15. Parameter properties of LinePeriod

Property	Value
Name	**LinePeriod**
Display Name	**Line Period**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.2048** **Maximum 838.8576** **Stepsize 0.0032**
Default value	**200.0**
Unit of measure	**µs**

Example 7.14. Usage of LinePeriod

```
/* Set */ LinePeriod = 200.0;
/* Get */ value_ = LinePeriod;
```

7.5.2. LineExposure

This parameter specifies the pulse width of the ExSync signal, which can be used by many cameras to specify the exposure time. It is possible to adjust the exposure time via software, even while grabbing. The value is set in microseconds and may not exceed the period time of the ExSync *LinePeriod*. In order to check the polarity simply increase this value and the resulting frame should become brighter. If this behaves in an opposite way check the polarity using *ExSyncPolarity*.

Table 7.16. Parameter properties of LineExposure

Property	Value
Name	**LineExposure**
Display Name	**Line Exposure**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.2048** **Maximum 419.4272** **Stepsize 0.0032**
Default value	**19.0**
Unit of measure	**µs**

Example 7.15. Usage of LineExposure

```
/* Set */ LineExposure = 19.0;
/* Get */ value_ = LineExposure;
```

7.5.3. ExSyncPolarity

The parameter adjusts the polarity of the ExSync signal generator. Use Low Active, if the camera opens the shutter on a falling edge, otherwise use High Active. For the mapping of the ExSync signals to the digital outputs check Chapter 6, '*DigitalIO*'.

Table 7.17. Parameter properties of ExSyncPolarity

Property	Value
Name	**ExSyncPolarity**
Display Name	**Ex Sync Polarity**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**LowActive** Low Active **HighActive** High Active
Default value	**HighActive**

Example 7.16. Usage of ExSyncPolarity

```
/* Set */ ExSyncPolarity = HighActive;
/* Get */ value_ = ExSyncPolarity;
```

7.5.4. LineTriggerDelay

This parameter specifies the delay between the generated ExSync and ExSync2 signals with respect to an external trigger input. Therefore, the ExSync2 signal is a delayed clone of the ExSync (polarity, period, etc. are the same as for ExSync). For the mapping of the ExSync signals to the digital outputs check Chapter 6, '*DigitalIO*'.

Please note that the line trigger delay needs to be less than the line trigger period. You might need to increase the line period first before increasing the line delay. This constraint also applies for external line trigger modes.

Table 7.18. Parameter properties of LineTriggerDelay

Property	Value
Name	**LineTriggerDelay**
Display Name	**Line Trigger Delay**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum** **0.0** **Maximum** **419.4272** **Stepsize** **0.0032**
Default value	**0.0**
Unit of measure	**µs**

Example 7.17. Usage of LineTriggerDelay

```
/* Set */ LineTriggerDelay = 0.0;
/* Get */ value_ = LineTriggerDelay;
```

Chapter 8. ImageTriggerFlash

The image trigger for line-scan cameras is in charge to generate an internal signal called image gate. Lines sent by the camera are only accepted if this image gate is active = open. Therefore, with help of the Image Gate it is possible to define frames by grouping all lines that belong to the same image gate into one frame.

This AcquisitionApplets supports three distinct operation modes of the image trigger:

- Free run

  In free run mode the image gate basically remains active all time. Therefore, all lines sent by the camera are grabbed. Moreover, it cuts the input lines into frames of the height specified by parameter *Height* of the display module. Also, offsets defined by *OffsetY* are covered and removed from the camera transfers for each image.

- Async Trigger

  For the external trigger mode of the image trigger, the image gate is inactive = closed until an external trigger signal activates the image gate for *Height* + *OffsetY* lines. Therefore, for each external trigger event, the frame grabber records a frame of the specified height.

- Async Trigger Multi Buffer

  For the external trigger mode of the image trigger, the image gate is inactive = closed until an external trigger signal activates the image gate. In contrast to the **AsyncExternalTrigger** mode, the gate is open for *ImageTriggerAsyncHeight* lines while this image is split into smaller chunks of *Height* lines. Therefore, for each external trigger event, the frame grabber records a frame of a large specified height and split the large image into smaller chunks. The purpose of the mode is to start processing in PC while the image is still recorded.

  The parameter value of *OffsetY* is without influence in this mode.

- Gated, Trigger

  For the external gated mode of the image trigger, the image gate is active as long as the external trigger source is active, but is becoming inactive when *Height* + *OffsetY* lines have been grabbed. Therefore, during an external trigger phase the frame grabber records a frame with a height depending on the duration of active time of the external trigger signal, but is not exceeding an image height of *Height* + *OffsetY* lines.

- Gated Multi Buffer, Triggered

  Equal to the 'Gated Trigger' mode, for the 'Gated Multi Buffer Trigger' the image gate is active as long as the external trigger source is active. In contrast, it does not limit the height to *Height* lines. It will cut the image after *Height* lines and start a new frame. Thus, for each gate, multiple frames are generated when a gate is active for more lines than defined by *Height*.

  All images of a generated sequence will have a height of *Height* lines. However, the last image of each sequence might have a lower number of lines in the image.

  To detect the last image of a sequence in your software. Parameter **FG_IMAGE_TAG** can be used. This parameter is of type unsigned 32 bit integer. The most significant bit i.e. bit 31 includes a flag which is set to one if the respective image is the last image of a multi buffer sequence.

```
uint32_t imageTag = 0;
int returnCode = Fg_getParameterEx(fg, FG_IMAGE_TAG, &imageTag, 0, pmem0, imageNumber);
bool isLastImageOfSequence = imageTagRAW >> 31;
```

  All other bits of parameter **FG_IMAGE_TAG** are fixed to value 0. The image tag parameter does not output the image number as available for older AcquisitionApplets.

Note that the value of parameter *OffsetY* is not considered if the 'Gated Multi Buffer Trigger' mode is used. An y-offset cannot be set in the applet.

8.1. ImageTriggerMode

Choose one of the image trigger modes described above. Please make sure that the operation mode of frame grabber and camera is the same.

Table 8.1. Parameter properties of ImageTriggerMode

Property	Value	
Name	**ImageTriggerMode**	
Display Name	**Image Trigger Mode**	
Interface	**IEnumeration**	
Access policy	**Read/Write**	
Visibility	**Beginner**	
Allowed values	**freeRun**	Free Run
	AsyncExternalTrigger	Async External Trigger
	AsyncExternalTriggerMultiframe	Async External Trigger Multiframe
	AsyncGatedTrigger	Async Gated Trigger
	AsyncGatedTriggerMultiframe	Async Gated Trigger Multiframe
Default value	**freeRun**	

Example 8.1. Usage of ImageTriggerMode

```
/* Set */ ImageTriggerMode = freeRun;
/* Get */ value_ = ImageTriggerMode;
```

8.2. ImageTriggerOn

The generation of image triggers can be switched on or off by use of this parameter. When the image trigger is disabled and the image trigger is not running in free-run mode, the image acquisition is terminated. If the image trigger is enabled, the acquisition will start immediately.

Table 8.2. Parameter properties of ImageTriggerOn

Property	Value	
Name	**ImageTriggerOn**	
Display Name	**Image Trigger On**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**On**	On
	Off	Off
Default value	**On**	

Example 8.2. Usage of ImageTriggerOn

```
/* Set */ ImageTriggerOn = On;
/* Get */ value_ = ImageTriggerOn;
```

8.3. FlashOn

To enable the flash output use this parameter.

For the mapping of the flash signal to the digital IO check Chapter 6, '*DigitalIO*'.

Table 8.3. Parameter properties of FlashOn

Property	Value
Name	**FlashOn**
Display Name	**Flash On**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**On** On **Off** Off
Default value	**On**

Example 8.3. Usage of FlashOn

```
/* Set */ FlashOn = On;
/* Get */ value_ = FlashOn;
```

8.4. ImageTriggerAsyncHeight

This parameter only has influence in the image trigger mode *ImageTriggerMode* Async Trigger Multi Frame **AsyncExternalTriggerMultiframe**. The value is used to define the image height of the frame after the trigger pulse. Whereas parameter *Height* defines the chunk height.

If the value of *ImageTriggerAsyncHeight* is less than *Height*, the frame is not split into multiple frames an will result in a smaller output frame.

Table 8.4. Parameter properties of ImageTriggerAsyncHeight

Property	Value
Name	**ImageTriggerAsyncHeight**
Display Name	**Image Trigger Async Height**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Beginner**
Allowed values	**Minimum** 1 **Maximum** 16777216 **Stepsize** 1
Default value	**1024**
Unit of measure	**lines**

Example 8.4. Usage of ImageTriggerAsyncHeight

```
/* Set */ ImageTriggerAsyncHeight = 1024;
/* Get */ value_ = ImageTriggerAsyncHeight;
```

8.5. ImageTriggerIsBusy

The image trigger is busy if the current requested frame from the camera has not been completely transferred to the grabber. This parameter can be used to check if the camera can accept a new software trigger pulse.

Table 8.5. Parameter properties of ImageTriggerIsBusy

Property	Value
Name	**ImageTriggerIsBusy**
Display Name	**Image Trigger is Busy**
Interface	**IEnumeration**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Busy** Busy **NotBusy** Not Busy

Example 8.5. Usage of ImageTriggerIsBusy

```
/* Get */ value_ = ImageTriggerIsBusy;
```

8.6. ImageTriggerInput

This category includes parameters to specify and control the image trigger inputs. The input can either be input pins of the frame grabber's trigger connector or trigger pulses generated by software register accesses.

8.6.1. ImageTriggerInputSource

This parameter specifies the signal source, which is used to trigger the image acquisition gate. If a software image trigger has to be used select option **SoftwareTrigger**.

Table 8.6. Parameter properties of ImageTriggerInputSource

Property	Value
Name	**ImageTriggerInputSource**
Display Name	**Image Trigger Input Source**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**GPITriggerSource0** GPI Trigger Source 0 **GPITriggerSource1** GPI Trigger Source 1 **GPITriggerSource2** GPI Trigger Source 2 **GPITriggerSource3** GPI Trigger Source 3 **GPITriggerSource4** GPI Trigger Source 4 **GPITriggerSource5** GPI Trigger Source 5 **GPITriggerSource6** GPI Trigger Source 6 **GPITriggerSource7** GPI Trigger Source 7 **TriggerInSourceFrontGPI0** Trigger In Source Front GPI 0 **TriggerInSourceFrontGPI1** Trigger In Source Front GPI 1 **TriggerInSourceFrontGPI2** Trigger In Source Front GPI 2 **TriggerInSourceFrontGPI3** Trigger In Source Front GPI 3 **SoftwareTrigger** Software Trigger
Default value	**GPITriggerSource0**

Example 8.6. Usage of ImageTriggerInputSource

```
/* Set */ ImageTriggerInputSource = GPITriggerSource0;
/* Get */ value_ = ImageTriggerInputSource;
```

8.6.2. ImageTriggerInputPolarity

The parameter defines the polarity of the external input trigger signal.

Table 8.7. Parameter properties of ImageTriggerInputPolarity

Property	Value
Name	**ImageTriggerInputPolarity**
Display Name	**Image Trigger Input Polarity**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**LowActive** Low Active **HighActive** High Active
Default value	**HighActive**

Example 8.7. Usage of ImageTriggerInputPolarity

```
/* Set */ ImageTriggerInputPolarity = HighActive;
/* Get */ value_ = ImageTriggerInputPolarity;
```

8.6.3. ImageTriggerGateDelay

With this parameter, a delay of lines can be configured before the activation of the image gate. This delays the start of the image acquisition. The parameter y-offest (as in free run mode) rejects the first lines from the camera. Delay and y-offset seem to have the same effect, however the difference is, that y-offset doesn't affect the image gate, which is relevant while using the gated line trigger mode.

Table 8.8. Parameter properties of ImageTriggerGateDelay

Property	Value
Name	**ImageTriggerGateDelay**
Display Name	**Image Trigger Gate Delay**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Beginner**
Allowed values	**Minimum** 0 **Maximum** 65535 **Stepsize** 1
Default value	**0**
Unit of measure	**lines**

Example 8.8. Usage of ImageTriggerGateDelay

```
/* Set */ ImageTriggerGateDelay = 0;
/* Get */ value_ = ImageTriggerGateDelay;
```

8.6.4. ImageTriggerDebouncing

This parameter specifies the debouncing time. This is the time for which the input image trigger signal must keep the same value to be detected as such. Fast signal changes within the debounce time will be filtered out.

Table 8.9. Parameter properties of ImageTriggerDebouncing

Property	Value
Name	**ImageTriggerDebouncing**
Display Name	**Image Trigger Debouncing**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0032** **Maximum 26.0** **Stepsize 0.0032**
Default value	**0.112**
Unit of measure	**µs**

Example 8.9. Usage of ImageTriggerDebouncing

```
/* Set */ ImageTriggerDebouncing = 0.112;
/* Get */ value_ = ImageTriggerDebouncing;
```

8.6.5. StrobePulseDelay

This parameter specifies the delay of the generated flash signal with respect to an external trigger input. Therefore, it is possible to synchronize the flash to the external trigger input. The delay is set in image line ticks.

Table 8.10. Parameter properties of StrobePulseDelay

Property	Value
Name	**StrobePulseDelay**
Display Name	**Strobe Pulse Delay**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 65535** **Stepsize 1**
Default value	**0**
Unit of measure	**lines**

Example 8.10. Usage of StrobePulseDelay

```
/* Set */ StrobePulseDelay = 0;
/* Get */ value_ = StrobePulseDelay;
```

8.6.6. Flash

8.6.6.1. FlashPolarity

The polarity of the generated flash signal can be changed with this parameter. For the mapping of the flash signal to the digital outputs check Chapter 6, '*DigitalIO*'.

Table 8.11. Parameter properties of FlashPolarity

Property	Value
Name	**FlashPolarity**
Display Name	**Flash Polarity**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**LowActive** Low Active **HighActive** High Active
Default value	**HighActive**

Example 8.11. Usage of FlashPolarity

```
/* Set */ FlashPolarity = HighActive;
/* Get */ value_ = FlashPolarity;
```

8.6.7. SoftwareTrigger

For the image trigger it is possible to use a software generated trigger signal to replace the external trigger input.

The software trigger control modules allows the to either generate a software trigger pulse or allows to set the state of the software trigger signal to generate a gate i.e. for gated image trigger mode.

To enable the software trigger set parameter *ImageTriggerInputSource* to software trigger.

8.6.7.1. SendSoftwareTrigger

A software trigger pulse can be sent by use of this parameter. Ensure to enable the software trigger by *ImageTriggerInputSource*.

Table 8.12. Parameter properties of SendSoftwareTrigger

Property	Value
Name	**SendSoftwareTrigger**
Display Name	**Send Software Trigger**
Interface	**ICommand**
Access policy	**Write/Change**
Visibility	**Beginner**

Example 8.12. Usage of SendSoftwareTrigger

```
/* Set */ SendSoftwareTrigger();
```

8.6.7.2. SetSoftwareTrigger

The software trigger state can be set to zero = inactive = low or one = active = high. Ensure to enable the software trigger by *ImageTriggerInputSource*.

Table 8.13. Parameter properties of SetSoftwareTrigger

Property	Value
Name	**SetSoftwareTrigger**
Display Name	**Set Software Trigger**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**LowActive** Low Active **HighActive** High Active
Default value	

Example 8.13. Usage of SetSoftwareTrigger

```
/* Set */ SetSoftwareTrigger = ;
/* Get */ value_ = SetSoftwareTrigger;
```

Chapter 9. SignalAnalyzer

The signal analyzer module computes some information on a signal source. These are

- Pulse Count
- Period (current, min, max)
- Difference between two pulse counters

The module is used to detect unexpected behaviors of the trigger system. For example a bouncing encode signal resulting in overtriggering of the camera. Another example is the detection of trigger lost signals or corrupted camera data which can result in extra lines.

Simply select the analyzer source signal and polarity. The measurement values can be obtained using read-only parameters. All measurements can be cleared synchronously.

Note that the module is available only once for the applet. All cameras share the same module. The camera/DMA index in the setParameter and getParameter functions has no influence.

9.1. SignalAnalyzer0Source et al.

Note

This description applies also to the following parameters: SignalAnalyzer1Source

Select the source signal for the trigger analyzer. For further explanation of the available sources see Chapter 6, '*DigitalIO*'. In addition, the line/frame start/end pulses can be used as signal sources, too.

Property	Value	
Name	**SignalAnalyzer0Source**	SignalAnalyzer
Display Name	**Signal Analyzer 0 Source**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**GND**	GND
	VCC	VCC
	SignalExsync	Signal Exsync
	SignalExsync2	Signal Exsync2
	SignalFlash	Signal Flash
	SignalLineValid	Signal Line Valid
	SignalFrameValid	Signal Frame Valid
	SignalLineStart	Signal Line Start
	SignalLineEnd	Cam0 Line Transfer End
	SignalFrameStart	Signal Frame Start
	SignalFrameEnd	Signal Frame End
	SignalCam1Exsync	Signal Cam1 Exsync
	SignalCam1Exsync2	Signal Cam1 Exsync2
	SignalCam1Flash	Signal Cam1 Flash
	SignalCam1LineValid	Signal Cam1 Line Valid
	SignalCam1FrameValid	Signal Cam1 Frame Valid
	SignalCam1LineStart	Signal Cam1 Line Start
	SignalCam1LineEnd	Signal Cam1 Line End
	SignalCam1FrameStart	Signal Cam1 Frame Start
	SignalCam1FrameEnd	Signal Cam1 Frame End
	SignalCam2Exsync	Signal Cam2 Exsync
	SignalCam2Exsync2	Signal Cam2 Exsync2
	SignalCam2Flash	Signal Cam2 Flash
	SignalCam2LineValid	Signal Cam2 Line Valid
	SignalCam2FrameValid	Signal Cam2 Frame Valid
	SignalCam2LineStart	Signal Cam2 Line Start
	SignalCam2LineEnd	Signal Cam2 Line End
	SignalCam2FrameStart	Signal Cam2 Frame Start
	SignalCam2FrameEnd	Signal Cam2 Frame End
	SignalCam3Exsync	Signal Cam3 Exsync
	SignalCam3Exsync2	Signal Cam3 Exsync2
	SignalCam3Flash	Signal Cam3 Flash
	SignalCam3LineValid	Signal Cam3 Line Valid
	SignalCam3FrameValid	Signal Cam3 Frame Valid
	SignalCam3LineStart	Signal Cam3 Line Start
	SignalCam3LineEnd	Signal Cam3 Line End
	SignalCam3FrameStart	Signal Cam3 Frame Start
	SignalCam3FrameEnd	Signal Cam3 Frame End
	SignalGPI0	Signal GPI 0
	SignalGPI1	Signal GPI 1
	SignalGPI2	Signal GPI 2
	SignalGPI3	Signal GPI 3
	SignalGPI4	Signal GPI 4
	SignalGPI5	Signal GPI 5
	SignalGPI6	Signal GPI 6
	SignalGPI7	Signal GPI 7
	SignalFrontGPI0	Signal Front GPI 0
	SignalFrontGPI1	Signal Front GPI 1
	SignalFrontGPI2	Signal Front GPI 2
	SignalFrontGPI3	Signal Front GPI 3
Default value	**SignalExsync**	

Table 9.1: Parameter properties of SignalAnalyzer0Source

Example 9.1. Usage of SignalAnalyzer0Source

```
/* Set */ SignalAnalyzer0Source = SignalExsync;
/* Get */ value_ = SignalAnalyzer0Source;
```

9.2. SignalAnalyzer0Polarity et al.

Note

This description applies also to the following parameters: SignalAnalyzer1Polarity

Select the polarity for the signal analyzer of the selected source. With this parameter you can invert the signal. The signal analyzer module will only measure on rising edges.

Table 9.2. Parameter properties of SignalAnalyzer0Polarity

Property	Value
Name	**SignalAnalyzer0Polarity**
Display Name	**Signal Analyzer 0 Polarity**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**LowActive** Low Active **HighActive** High Active
Default value	**HighActive**

Example 9.2. Usage of SignalAnalyzer0Polarity

```
/* Set */ SignalAnalyzer0Polarity = HighActive;
/* Get */ value_ = SignalAnalyzer0Polarity;
```

9.3. SignalAnalyzer0CurrentPeriod et al.

Note

This description applies also to the following parameters: SignalAnalyzer1CurrentPeriod

This read-only parameter returns the last measured period of the selected signal source. Keep in mind that the module requires two rising edges to obtain a measurement result. Selecting a new source or changing the acquisition states can result in very long periods.

Table 9.3. Parameter properties of SignalAnalyzer0CurrentPeriod

Property	Value
Name	**SignalAnalyzer0CurrentPeriod**
Display Name	**Signal Analyzer 0 Current Period**
Interface	**IFloat**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum** 0.0 **Maximum** 1.3743895344E7 **Stepsize** 0.0032
Unit of measure	**ns**

Example 9.3. Usage of SignalAnalyzer0CurrentPeriod

```
/* Get */ value_ = SignalAnalyzer0CurrentPeriod;
```

9.4. SignalAnalyzer0MaxPeriod et al.

Note

This description applies also to the following parameters: SignalAnalyzer1MaxPeriod

This read-only parameter returns the maximum measured period after the last reset. Keep in mind that selecting a new source or changing the acquisition states can result in very long periods.

Table 9.4. Parameter properties of SignalAnalyzer0MaxPeriod

Property	Value
Name	**SignalAnalyzer0MaxPeriod**
Display Name	**Signal Analyzer 0 Max Period**
Interface	**IFloat**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 1.3743895344E7** **Stepsize 0.0032**
Unit of measure	**ns**

Example 9.4. Usage of SignalAnalyzer0MaxPeriod

```
/* Get */ value_ = SignalAnalyzer0MaxPeriod;
```

9.5. SignalAnalyzer0MinPeriod et al.

Note

This description applies also to the following parameters: SignalAnalyzer1MinPeriod

This read-only parameter returns the minimum measured period after the last reset.

Table 9.5. Parameter properties of SignalAnalyzer0MinPeriod

Property	Value
Name	**SignalAnalyzer0MinPeriod**
Display Name	**Signal Analyzer 0 Min Period**
Interface	**IFloat**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0.0032** **Maximum 1.3743895344E7** **Stepsize 0.0032**
Unit of measure	**ns**

Example 9.5. Usage of SignalAnalyzer0MinPeriod

```
/* Get */ value_ = SignalAnalyzer0MinPeriod;
```

9.6. SignalAnalyzer0PulseCount et al.

Note

This description applies also to the following parameters: SignalAnalyzer1PulseCount

Returns the counter value of the selected source. For each rising edge the counter is increased. This, after the first pulse, the counter value will be one. On counter overflow, it will start from 0 again.

Table 9.6. Parameter properties of SignalAnalyzer0PulseCount

Property	Value
Name	**SignalAnalyzer0PulseCount**
Display Name	**Signal Analyzer 0 Pulse Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 4294967295** **Stepsize 1**
Unit of measure	**pulses**

Example 9.6. Usage of SignalAnalyzer0PulseCount

```
/* Get */ value_ = SignalAnalyzer0PulseCount;
```

9.7. SignalAnalyzerPulseCountDifference

Use this read only parameter to check the difference of the signal analyzer 0 and 1 pulse counter values (Analyzer 0 - Analyzer 1 value). This can be used to check for trigger lost signals if analyzer 0 will count the exsyc pulses and analyzer 1 the returned camera lines. In this case the difference is between 0 and 1 for single line cameras with no extra delay. If the difference exceeds 1, the camera did not return a line for all trigger pulses i.e. a trigger is lost or ignored due to overtriggering. If the difference is less than 0 an additional camera line was generated and received by the frame grabber. The reason for this can be a noisy trigger cable which added extra spikes or a corrupted data transfer which split the data into several parts.

Table 9.7. Parameter properties of SignalAnalyzerPulseCountDifference

Property	Value
Name	**SignalAnalyzerPulseCountDifference**
Display Name	**Signal Analyzer Pulse Count Difference**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum -4294967296** **Maximum 4294967295** **Stepsize 1**
Unit of measure	**pulses**

Example 9.7. Usage of SignalAnalyzerPulseCountDifference

```
/* Get */ value_ = SignalAnalyzerPulseCountDifference;
```

9.8. SignalAnalyzerClear

To clear all signal analyzer measurement results and counters use this parameter. All counters will be reset synchronously and are ready to restart immediately.

Table 9.8. Parameter properties of SignalAnalyzerClear

Property	Value
Name	**SignalAnalyzerClear**
Display Name	**Signal Analyzer Clear**
Interface	**ICommand**
Access policy	**Write/Change**
Visibility	**Beginner**

Example 9.8. Usage of SignalAnalyzerClear

```
/* Set */ SignalAnalyzerClear();
```

Chapter 10. BufferStatus

The applet processes image data as fast as possible. Any image data sent by the camera is immediately processed and sent to the PC. The latency is minimal. In general, only one concurrent image line is stored and processed in the frame grabber. However, the transfer bandwidth to the PC via DMA channel can vary caused by interrupts, other hardware and the current CPU load. Furthermore, if operated in **selective mode**, it is possible to queue buffer slower than the camera offers new images and therefore generate an overflow condition on the frame grabber. Also, the camera frame rate can vary due to an fluctuating trigger. For these cases, the applet is equipped with a memory to buffer the input frames. The fill level of the buffer can be obtained by reading from parameter *FillLevel*.

In normal operation conditions the buffer will always remain almost empty. For fluctuating camera bandwidths or for short and fast acquisitions, the buffer can easily fill up quickly. Of course, the input bandwidth must not exceed the maximum bandwidth of the applet. Check Section 1.2, 'Bandwidth' for more information.

If the buffer's fill level reaches 100%, the applet is in overflow condition, as no more data can be buffered and camera data will be discarded. This can result in two different behaviors:

- Corrupted Frames:

  The transfer of a current frame is interrupted by an overflow. This means, the first pixels or lines of the frame were transfered into the buffer, but not the full frame. The output of the applet i.e. the DMA transfer will be shorter. The output image will not have it's full height. These images will be marked incomplete. Check the Basler GenTL documentation to learn on how to identify incompleted buffers (https://www.baslerweb.com/en/sales-support/downloads/document-downloads/cxp-gentl-producer-feature-documentation/).

- Lost Frames:

  A full camera frame was discarded due to a full buffer memory. No DMA transfer will exist for the discarded frame. This means the number of applet output images can differ from the number of applet input images.

The buffer overflow threshold *OverflowOnThreshold* and *OverflowSyncOnThreshold* default ensures that under normal conditions frames can be completed or will be fully dropped so that corrupted frames are avoide

A way to detect the overflows is to read parameter *Overflow* or check for event *Overflow*. Reading from the parameter will provide information about an overflow condition. As soon as the parameter is read, it will reset. Using the parameter an overflow condition can be detect, but it is not possible to obtain the exact image number and the moment. For this, the overflow event can be used.

10.1. FillLevel

The fill-level of the frame grabber buffers used in this applet can be read-out by use of this parameter. The value allows to check if the mean input bandwidth of the camera is to high to be processed with the applet.

Table 10.1. Parameter properties of FillLevel

Property	Value
Name	**FillLevel**
Display Name	**Fill Level**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 100** **Stepsize 1**
Unit of measure	**%**

Example 10.1. Usage of FillLevel

```
/* Get */ value_ = FillLevel;
```

10.2. Overflow

If the applet runs into overflow, a value "1" can be read by the use of this parameter. Note that an overflow results in loss of images. To avoid overflows reduce the mean input bandwidth.

The parameter is reset at each readout cycle. The program microDisplayX will continuously poll the value, thus the occurrence of an overflow might not be visible in microDisplayX.

A more effective and robust way is to detect overflows is the use of the event system.

Table 10.2. Parameter properties of Overflow

Property	Value
Name	**Overflow**
Display Name	**Buffer overflow**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 1** **Stepsize 1**

Example 10.2. Usage of Overflow

```
/* Get */ value_ = Overflow;
```

10.3. OverflowOffThreshold

The Overflow state will be deactivated once the buffer Fillevel (*FillLevel*) will fall below this value. As long as the applet remains in overflow state all images arriving will be discarded. This will result in Overflow events with a set "lost" flag.

Table 10.3. Parameter properties of OverflowOffThreshold

Property	Value
Name	**OverflowOffThreshold**
Display Name	**Overflow Off Threshold**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 100.0** **Stepsize 0.5**
Default value	**50.0**

Example 10.3. Usage of OverflowOffThreshold

```
/* Set */ OverflowOffThreshold = 50.0;
/* Get */ value_ = OverflowOffThreshold;
```

10.4. OverflowOnThreshold

The applet will enter Overflow state once the buffer Fillevel exceeds this filllevel (*FillLevel*). If the overflow state is active images will be stopped imidiately. This may lead to an incomplete frame. Incomplete frames are marked incomplete in the image Tag and an overflow event can be generated.

Table 10.4. Parameter properties of OverflowOnThreshold

Property	Value
Name	**OverflowOnThreshold**
Display Name	**Overflow On Threshold**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 100.0** **Stepsize 0.5**
Default value	**99.5**

Example 10.4. Usage of OverflowOnThreshold

```
/* Set */ OverflowOnThreshold = 99.5;
/* Get */ value_ = OverflowOnThreshold;
```

10.5. OverflowSyncOnThreshold

The applet will enter Overflow state once the buffer fillevel (*FillLevel*) exceeds this filllevel and the currently arriving frame is stored to the buffer. If the applet remains in overflow state frames might be droped. If the buffer falls below this fillevel frames are accepted again. There is no hysteresis for this threshold.

Table 10.5. Parameter properties of OverflowSyncOnThreshold

Property	Value
Name	**OverflowSyncOnThreshold**
Display Name	**Overflow Sync On Threshold**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 100.0** **Stepsize 0.5**
Default value	**80.0**

Example 10.5. Usage of OverflowSyncOnThreshold

```
/* Set */ OverflowSyncOnThreshold = 80.0;
/* Get */ value_ = OverflowSyncOnThreshold;
```

10.6. OverflowEventSelect

The *Overflow* Event. Allows to generate events if one of the following conditions is meet.

Table 10.6. Event select for *Overflow*

Value	Description
Incomplete	Each incomplete frame will generate an Event containing the information that the frame is incomplete and the frameID
Lost	Each lost frame will generate an Event containing the information that the frame is lost and the frameID
IncompleteLost	Each lost or incomplete frame will generate an Event containing the information that the frame is lost/ incomplete and the frameID
OK	Each correct frame will generate an Event containing the information that the frame is transfered correct and the frameID of the frame
IncompleteOK	Each incomplete or correct frame will generate an Event containing the information that the frame is correct or incomplete and the frameID
LostOK	Each lost or correct frame will generate an Event containing the information that the frame is correct or lost and the frameID
All	Each frame will generate an Event containing the status(lost, incomplete or correct) of the frame and the frameID

Table 10.7. Parameter properties of OverflowEventSelect

Property	Value	
Name	**`OverflowEventSelect`**	
Display Name	**`Overflow Event Select`**	
Interface	**`IEnumeration`**	
Access policy	**`Read/Write/Change`**	
Visibility	**`Beginner`**	
Allowed values	**`Incomplete`**	Incomplete
	`Lost`	Lost
	`IncompleteLost`	Incomplete Lost
	`OK`	OK
	`IncompleteOK`	Incomplete OK
	`LostOK`	Lost OK
	`All`	All
Default value	**`IncompleteLost`**	

Example 10.6. Usage of OverflowEventSelect

```
/* Set */ OverflowEventSelect = IncompleteLost;
/* Get */ value_ = OverflowEventSelect;
```

10.7. OverflowEvents

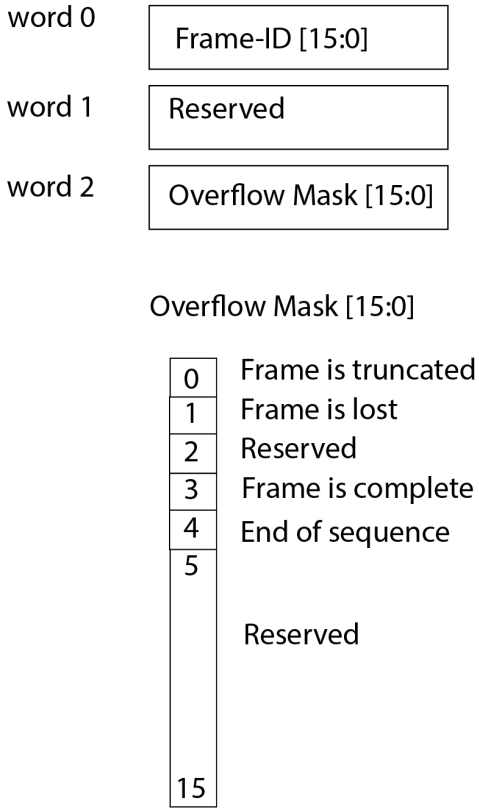
In programming or runtime environments, a callback function is a piece of executable code that is passed as an argument, which is expected to call back (execute) exactly that time an event is triggered. This applet can generate some software callback events based on the memory overflow condition as explained in the following section. These events are not related to a special camera functionality. Other event sources are described in additional sections of this document.

The Basler Framegrabber SDK and pylon SDK via GenTL enables an application to get these event notifications about certain state changes at the data flow from camera to RAM and the image and trigger processing as well. Please consult the Basler Framegrabber SDK, pylon SDK or GenTL documentation for more details concerning the implementation of this functionality.

10.7.1. Overflow

Overflow events are generated for each truncated, lost or complete frame. The selection can be done using *OverflowEventSelect*. The overflow event contains data, namely the type of overflow, the image number and the timestamp. The following figure illustrates the event data. Data is contained in a 64-bit data packet. The first 16 bits contain the frame-ID from the camera. Bits 32 to 47 provide an overflow mask.

Figure 10.1. Illustration of Overflow Data Packet



Note that the frame-ID is taken from the camera stream. See Section 1.5, 'Frame ID' for more information. The frame-ID is a 16-bit value. If its maximum is reached, the frame-ID starts at zero again. If the **frame truncated** flag is set, the frame with the frame-ID in the event is truncated i.e. it doesn't have its full length but is still transferred via DMA channel. If the **frame lost** flag is set, the frame with the frame-ID in the event was fully discarded. No DMA transfer exists for this frame. The **truncated frame** flag and the **frame lost** flag never occur for the same event.

Table 10.8. Event parameters of Overflow

Name	Interface	Description
EventOverflowFrameID	IInteger	Camera frame-ID for area scan applets or grabber frame-ID for line scan applets.
EventOverflowIsTru	IBoolean	Frame is truncated.
EventOverflowIsLost	IBoolean	Frame is lost.
EventOverflowIsCor	IBoolean	Frame is complete.
EventOverflowIsEndOfSequence	IBoolean	Marks the end of a sequence.

Chapter 11. ImageSelector

The Image Selector allows the user to cut out a period of *p* images from the image stream and select a particular image *n* from it.

The following example will explain the settings of *p* and *n* which represent the frame grabber parameters *ImageSelectPeriod* and *ImageSelect*. Suppose two frame grabbers being connected to a camera signal multiplexer, providing all camera images to both devices. Grabber 0 is required to process all even frames, while grabber 1 is required to process all odd frames. The settings will then be:

1. Grabber 0:
   - *ImageSelectPeriod* = 2

     ImageSelect = 0
2. Grabber 1:
   - *ImageSelectPeriod* = 2

     ImageSelect = 1

Ensure that both grabbers are used synchronously. This is possible with a triggered camera. To do so, initialize and configure both frame grabbers. Configure the camera for external trigger and the trigger system of master grabber which is directly connected to the camera.

11.1. ImageSelectPeriod

This parameter specifies the period length *p*. The parameter can be changed at any time. However, changing during acquisition can result in an asynchronous switching which will result in the loss of a synchronous grabbing. It is recommended to change the parameter only when the acquisition is stopped.

The parameter's value has to be greater than *ImageSelect*.

Table 11.1. Parameter properties of ImageSelectPeriod

Property	Value
Name	**ImageSelectPeriod**
Display Name	**Image Select Period**
Interface	**IInteger**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 1** **Maximum 256** **Stepsize 1**
Default value	**1**
Unit of measure	**image**

Example 11.1. Usage of ImageSelectPeriod

```
/* Set */ ImageSelectPeriod = 1;
/* Get */ value_ = ImageSelectPeriod;
```

11.2. ImageSelect

The parameter *ImageSelect* specifies a particular image from the image set defined by *ImageSelectPeriod*. This parameter can be changed at any time. However, changing during acquisition can result in an asynchronous switching which will result in the loss of a synchronous grabbing. It is recommended to change the parameter only when the acquisition is stopped.

The parameter's value has to be less than *ImageSelectPeriod*.

Table 11.2. Parameter properties of ImageSelect

Property	Value
Name	**ImageSelect**
Display Name	**Image Select**
Interface	**IInteger**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 255** **Stepsize 1**
Default value	**0**
Unit of measure	**image**

Example 11.2. Usage of ImageSelect

```
/* Set */ ImageSelect = 0;
/* Get */ value_ = ImageSelect;
```

Chapter 12. WhiteBalance

The applet enables a spectral adaptation of the image to the lighting situation of the application. The color values for the red, green and blue components can be individually enhanced or reduced by a scaling factor to adjust the spectral sensibility of the camera sensor.

The applet Acq_QuadCXP12Line performs a Bayer de-mosaicing. The white balancing is performed prior to the Bayer de-mosaicing, to ensure the correction of the raw data and avoid subsequent faults during processing.

12.1. ScalingFactorGreen

Table 12.1. Parameter properties of ScalingFactorGreen

Property	Value
Name	**ScalingFactorGreen**
Display Name	**Scaling Factor Green**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 3.9990234375** **Stepsize 9.765625E-4**
Default value	**1.0**

Example 12.1. Usage of ScalingFactorGreen

```
/* Set */ ScalingFactorGreen = 1.0;
/* Get */ value_ = ScalingFactorGreen;
```

12.2. ScalingFactorRed

Table 12.2. Parameter properties of ScalingFactorRed

Property	Value
Name	**ScalingFactorRed**
Display Name	**Scaling Factor Red**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 3.9990234375** **Stepsize 9.765625E-4**
Default value	**1.0**

Example 12.2. Usage of ScalingFactorRed

```
/* Set */ ScalingFactorRed = 1.0;
/* Get */ value_ = ScalingFactorRed;
```

12.3. ScalingFactorBlue

Table 12.3. Parameter properties of ScalingFactorBlue

Property	Value
Name	**ScalingFactorBlue**
Display Name	**Scaling Factor Blue**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum** 0.0 **Maximum** 3.9990234375 **Stepsize** 9.765625E-4
Default value	**1.0**

Example 12.3. Usage of ScalingFactorBlue

```
/* Set */ ScalingFactorBlue = 1.0;
/* Get */ value_ = ScalingFactorBlue;
```

Chapter 13. ColorConverter

The color converter module is used to convert the input pixel format to an output pixel format. The conversion is performed post to the Bayer de-mosicing and just before the lookup table.

This applet can perform the following conversions.

Table 13.1. Color Conversion

Input Format	Mono	RGB	BiColor	YCbCr
Output Format				
Mono	yes	yes	yes	N/A
RGB	yes	yes	yes	N/A
BiColor	N/A	N/A	yes	N/A
YCbCr	N/A	N/A	N/A	yes

By setting the input and output format the conversion is automatically applied if a conversion is possible. Otherwise the applet will output unchanged values. See *PixelFormat* and *Format*.

Chapter 14. LookupTable

This Acquisition Applet includes a full resolution lookup table (LUT) for each of the three color components. Settings are applied to the acquired images just before transferring them to the host PC. Thus, it is the last pre-processing step on the frame grabber.

A lookup table includes one entry for every allowed input pixel value. The pixel value will be replaced by the value of the lookup table element. In other words, a new value is assigned to each pixel value. This can be used for image quality enhancements such as an added offset, a gain factor or gamma correction which can be performed by use of the processing module of this applet in a convenient way (see Module Chapter 15, '*Processing*'). The lookup table can also be loaded with custom values. Application areas are custom image enhancements or correct pixel classifications.

This applet is processing data with an internal resolution of 16 bits. But the lookup table has 14 input bits i.e. pixel values can be in the range [0, 16383]. For each of these 16383 elements, a table entry exists containing a new output value. The new values are in the range from 0 to 65536. All color components are treated separately. Since this applet uses 16 bit internally, consider that all values need to represent this value range. This LUT is applied to all pixel values before *Format* is applied. The input values for the LUT are aligned to the most significant bit (MSB).

In the following the parameters to use the lookup table are explained. Parameter *LutType* is important to be set correctly as it defines the lookup table operation mode.

14.1. LutEnable

It is possible to disable the functionality of this lookup table. The intenal processor enables a convenient way to improve the image quality using parameters such as offset, gain and gamma. By disabling the lookup table the processing functions are not available anymore. See category Chapter 15, '*Processing*' for a more detailed documentation concerning this. Set this parameter to **On** to use the look up table. By default it is set to **Off** disabling the lookup table functionality itself and the related processing functions.

Table 14.1. Parameter properties of LutEnable

Property	Value
Name	**`LutEnable`**
Display Name	**`Enabled`**
Interface	**`IEnumeration`**
Access policy	**`Read/Write/Change`**
Visibility	**`Beginner`**
Allowed values	**`On`** On **`Off`** Off
Default value	**`Off`**

Example 14.1. Usage of LutEnable

```
/* Set */ LutEnable = Off;
/* Get */ value_ = LutEnable;
```

14.2. LutType

There exist two basic possibilities to use and configure the lookup table. One possibility is to use the internal processor which allows a convenient way to improve the image quality using parameters such as offset, gain

and gamma. Check category Chapter 15, '*Processing*' for more detailed documentation. Set this parameter to **LutTypeProcessing** to use the processor.

The second possibility to use the lookup table is to load a file containing custom values to the lookup table. Set the parameter to **UserFile** to enable the possibility to load a custom file with lookup table entries.

Beside these two possibilities it is always possible to directly write to the lookup table entries using the field parameters *LutValueRed*, *LutValueGreen* and *LutValueBlue*. The use of these parameters will overwrite the settings made with the processor or the custom input file. Vice versa, changing a processing parameter or loading a custom lookup table file, will overwrite the settings made by the field parameters.

Table 14.2. Parameter properties of LutType

Property	Value	
Name	**LutType**	
Display Name	**Type**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**LutTypeProcessing** **UserFile**	Processor User File
Default value	**LutTypeProcessing**	

Example 14.2. Usage of LutType

```
/* Set */ LutType = LutTypeProcessing;
/* Get */ value_ = LutType;
```

14.3. LutValue

Table 14.3. Parameter properties of LutValue

Property	Value
Name	**LutValue**
Display Name	**LUT Values**
Interface	**IInteger (Field)**
Field Size	**16384**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Default value	**0**

Example 14.3. Usage of LutValue

```
/* Set */ for (i = 0; i < 16384; ++i)
{
    LutValueSelector = i;
    LutValue = 0;
}
/* Get */ for (i = 0; i < 16384; ++i)
{
    LutValueSelector = i;
    value_ = LutValue;
}
```

14.4. LutValueRed

Table 14.4. Parameter properties of LutValueRed

Property	Value
Name	**LutValueRed**
Display Name	**Red LUT Values**
Interface	**IInteger (Field)**
Field Size	**16384**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Default value	**0**

Example 14.4. Usage of LutValueRed

```
/* Set */ for (i = 0; i < 16384; ++i)
{
    LutValueRedSelector = i;
    LutValueRed = 0;
}
/* Get */ for (i = 0; i < 16384; ++i)
{
    LutValueRedSelector = i;
    value_ = LutValueRed;
}
```

14.5. LutValueGreen

Table 14.5. Parameter properties of LutValueGreen

Property	Value
Name	**LutValueGreen**
Display Name	**Green LUT Values**
Interface	**IInteger (Field)**
Field Size	**16384**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Default value	**0**

Example 14.5. Usage of LutValueGreen

```
/* Set */ for (i = 0; i < 16384; ++i)
{
    LutValueGreenSelector = i;
    LutValueGreen = 0;
}
/* Get */ for (i = 0; i < 16384; ++i)
{
    LutValueGreenSelector = i;
    value_ = LutValueGreen;
}
```

14.6. LutValueBlue

Table 14.6. Parameter properties of LutValueBlue

Property	Value
Name	**LutValueBlue**
Display Name	**Blue LUT Values**
Interface	**IInteger (Field)**
Field Size	**16384**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Default value	**0**

Example 14.6. Usage of LutValueBlue

```
/* Set */ for (i = 0; i < 16384; ++i)
{
    LutValueBlueSelector = i;
    LutValueBlue = 0;
}
/* Get */ for (i = 0; i < 16384; ++i)
{
    LutValueBlueSelector = i;
    value_ = LutValueBlue;
}
```

14.7. LutCustomFile

If parameter *LutType* is set to **UserFile**, the according path and filename to the file containing the custom lookup table entries can be set here. If the file is valid, the file values will be loaded to the lookup table. If the file is invalid, the call to this parameter will return an error.

A convenient way of getting a draft file, is to save the current lookup table settings to file using parameter *LutSaveFile*.

Please make sure to activate the Type of LUT *LutType* to "UserFile"/**UserFile** in order to make the changes and file names taking effect.

This section describes the file formats which are in use to fill the so called look-up tables (LUT). The purpose of a LUT is a transformation of pixel values from a input (source) image to the pixel values of an output image. This transformation is done by a kind of table, which contains the assignment between these pixel values (input pixel values - output pixel values). Basically the LUT is defined for gray format and color formats as well. When defining a LUT for color formats, the definition of tables has to be done for each color component. The LUT file format consists of 2 parts:

- Header section containing control and description information.
- Main section containing the assignment table for transforming pixel values form a source (input) image to a destination (output) image.

The following example shows how a grey scale lookup table description could look like:

```
# Lut data file v1.1
id=3;
nrOfElements=4096;
format=0;
number=0;
0,0;
1,1;
2,2;
3,3;
```

```
4,4;
5,5;
6,6;
…
4095,4095;
```

General Properties:

- File format extension should be ".lut"
- LUT file format is an ASCII file format consisting of multiple lines of data.
- Lines are defined by a line separator a <CR> <LF> line feed (0x3D 0x0D 0x0A).
- Lines consist of key / value pairs. Key and value are separated by "=". The value has to be followed by a semicolon ; (0x3B)
- Formats consist of header data, containing control information and the assignment table for a specific color component (gray / red, green, blue).
- Basically the LUT file color format follows the same rules as the gray image format. In addition, due to the fact, that each color component can has its own transformation, the definitions are repeated for each color component.

The following example shows how a color scale lookup table description could look like:

```
# Lut data file v1.1
[red]
id=0;
nrOfElements=256;
format=0;
number=0;
0,0;
1,1;
..
255,255;
[green]
id=1;
nrOfElements=256;
format=0;
number=0;
0,0;
1,1;
..
255,255;
[blue]
id=2;
nrOfElements=256;
format=0;
number=0;
0,0;
1,1;
..
255,255;
```

A more detailed explanation of the lookup table file format can be found in the Basler Framegrabber API manual.

Table 14.7. Parameter properties of LutCustomFile

Property	Value
Name	**LutCustomFile**
Display Name	**Load File**
Interface	**IString**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Default value	**""**

Example 14.7. Usage of LutCustomFile

14.8. LutSaveFile

To save the current lookup table configuration to a file, write the according output filename to this parameter. Keep in mind that you need to have full write access to the specified path.

Writing the current lookup table settings to a file is also a convenient way to exploit the settings made by the processor. Moreover, you will get a draft version of the lookup table file format. The values in the output file can directly be used to be loaded to the lookup table again using parameter *LutCustomFile*.

Table 14.8. Parameter properties of LutSaveFile

Property	Value
Name	**LutSaveFile**
Display Name	**Save File**
Interface	**IString**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Default value	**""**

Example 14.8. Usage of LutSaveFile

14.9. AppletProperties

In the following, some properties of the lookup table implementation are listed.

14.9.1. LutImplementationType

In this applet, a full lookup table is implemented and can be setup in a custom way. By default a linear representation is performed.

Table 14.9. Parameter properties of LutImplementationType

Property	Value
Name	**LutImplementationType**
Display Name	**LUT Implementation Type**
Interface	**IEnumeration**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**FullLUT** Full LUT **KneeLUT** Knee LUT

Example 14.9. Usage of LutImplementationType

```
/* Get */ value_ = LutImplementationType;
```

14.9.2. LutInputPixelBitDepth

This applet is using 14 lookup table input bits.

Table 14.10. Parameter properties of LutInputPixelBitDepth

Property	Value
Name	**LutInputPixelBitDepth**
Display Name	**LUT Input Pixel Bit Depth**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 16** **Stepsize 1**
Unit of measure	**bit**

Example 14.10. Usage of LutInputPixelBitDepth

```
/* Get */ value_ = LutInputPixelBitDepth;
```

14.9.3. LutOutputPixelBitDepth

This applet is using 16 lookup table output bits.

Table 14.11. Parameter properties of LutOutputPixelBitDepth

Property	Value
Name	**LutOutputPixelBitDepth**
Display Name	**LUT Output Pixel Bit Depth**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 16** **Stepsize 1**
Unit of measure	**bit**

Example 14.11. Usage of LutOutputPixelBitDepth

```
/* Get */ value_ = LutOutputPixelBitDepth;
```

Chapter 15. Processing

A convenient way to improve the image quality are the processing parameters. Using these parameters an offset, gain and gamma correction can be performed. Moreover, the image can be inverted.

Processor Activation

The processing parameters use the lookup table for determination of the correction values. For activation of the processing parameters, set *LutType* of category lookup table to **LutTypeProcessing** . Otherwise, parameter changes will have no effect.

All transformations apply in the following order:

1. Offset Correction, range [-1.0, +1.0], identity = 0
2. Gain Correction, range [0, 2^14[, identity = 1.0
3. Gamma Correction, range]0, inf], identity = 1.0
4. Invert, identity = 'off'

In this applet, a full lookup table with m = 14 input bits and n = 16 outputs bits is used to perform the corrections. Values are determined by

Equation 15.1. LUT Processor without Inversion

$$Output(x) = \left[\left[gain * \left(\frac{x}{2^{14}-1} + offset\right)\right]^{\frac{1}{gamma}}\right] * \left(2^{16}-1\right).$$

If the inversion is used, output values are determined by

Equation 15.2. LUT Processor with Inversion

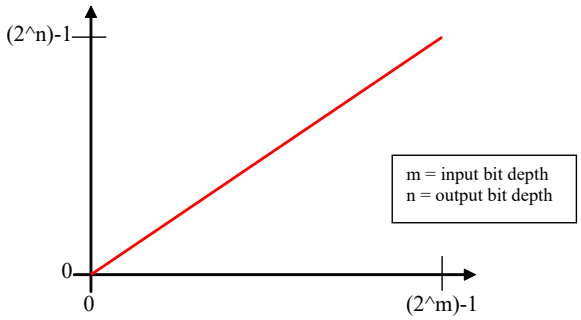
$$Output(x) = 2^{16} - 1 - \left[\left[gain * \left(\frac{x}{2^{14}-1} + offset\right)\right]^{\frac{1}{gamma}}\right] * \left(2^{16}-1\right),$$

where x represents the input pixel value i.e. is in the range from 0 to 2^14 - 1. If the determined output value is less than 0, it will be set to 0. If the determined output value is greater than 2^16 - 1 it is set to 2^16 - 1.

This applet processes each color component separately using the same processing parameters for each component.

If no parameters are changed, i.e. they are set to identity, the output values will be equal to the input values as shown in the figure below. In the following, you will find detailed explanations for all processing parameters.

Figure 15.1. Lookup Table Processing: Identity



15.1. ProcessingOffset

The offset is a relative value added to each pixel, which leads to a behavior similar to a brightness controller. A relative offset means, that e. g. 0.5 adds half of the total brightness to each pixel. In absolute numbers when using 8 bit/pixel, 128 is added to each pixel (0.5 x 255 = 127.5). If you rather want to add an absolute value to each pixel do the following calculation: e. g. add -51 to an 8 bit/pixel offset = -51 / 255 = -0.2.Figure 15.2 shows an example of an offset.

Figure 15.2. Lookup Table Processing: Offset

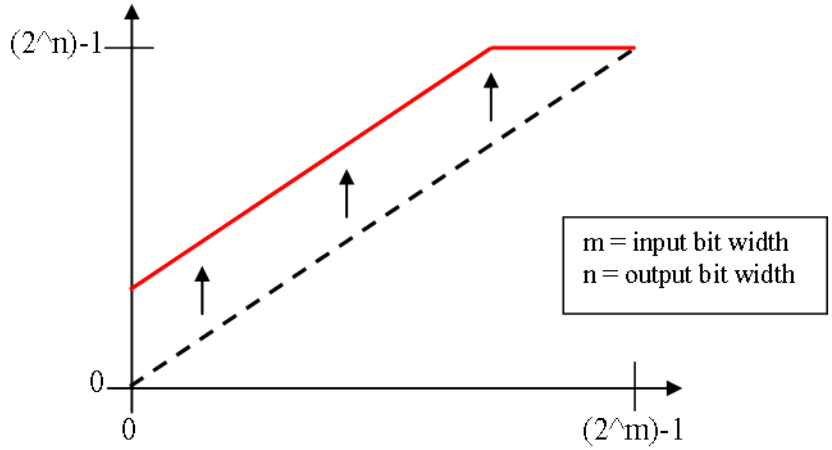


Table 15.1. Parameter properties of ProcessingOffset

Property	Value
Name	**ProcessingOffset**
Display Name	**Offset**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum -1.0** **Maximum 1.0** **Stepsize 2.220446049250313E-16**
Default value	**0.0**

Example 15.1. Usage of ProcessingOffset

```
/* Set */ ProcessingOffset = 0.0;
/* Get */ value_ = ProcessingOffset;
```

15.2. ProcessingGain

The gain is a multiplicative coefficient applied to each pixel, which leads to a behavior similar to a contrast controller. Each pixel value will be multiplied with the given value. For identity select value 1.0.

Figure 15.3. Lookup Table Processing: Gain

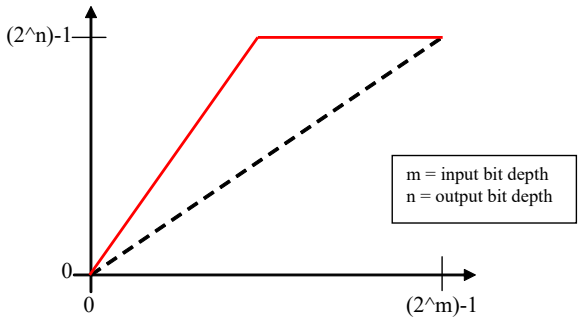


Table 15.2. Parameter properties of ProcessingGain

Property	Value
Name	**ProcessingGain**
Display Name	**Gain**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum** 0.0 **Maximum** 16384.0 **Stepsize** 2.220446049250313E-16
Default value	**1.0**

Example 15.2. Usage of ProcessingGain

```
/* Set */ ProcessingGain = 1.0;
/* Get */ value_ = ProcessingGain;
```

15.3. ProcessingGamma

The gamma correction is a power-law transformation applied to each pixel. Normalized pixel values p ranging [0, 1.0] transform like $p' = p^{1/gamma}$.

Figure 15.4. Lookup Table Processing: Gamma

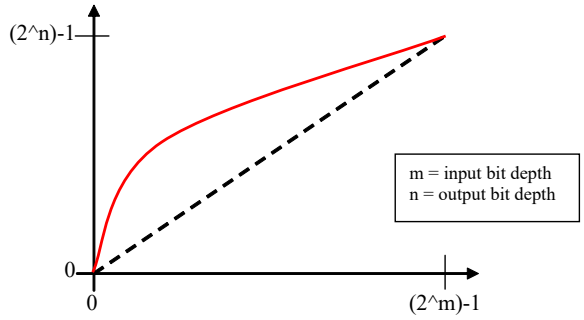


Table 15.3. Parameter properties of ProcessingGamma

Property	Value
Name	**ProcessingGamma**
Display Name	**Gamma**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum** -1000.0 **Maximum** 1000.0 **Stepsize** 2.220446049250313E-16
Default value	**1.0**

Example 15.3. Usage of ProcessingGamma

```
/* Set */ ProcessingGamma = 1.0;
/* Get */ value_ = ProcessingGamma;
```

15.4. ProcessingInvert

When *ProcessingInvert* is set to **On**, the output is the negative of the input. Normalized pixel values p ranging [0, 1.0] transform to $p' = 1 - p$.

Figure 15.5. Lookup Table Processing: Invert

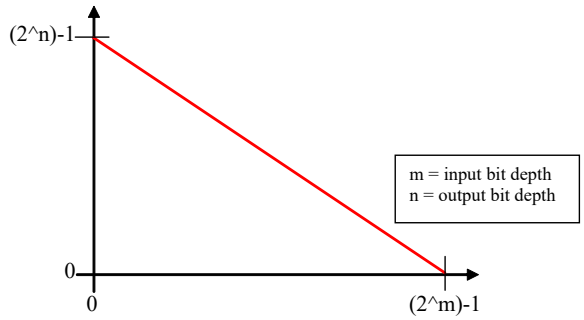


Table 15.4. Parameter properties of ProcessingInvert

Property	Value
Name	**ProcessingInvert**
Display Name	**Invert**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**On** On **Off** Off
Default value	**Off**

Example 15.4. Usage of ProcessingInvert

```
/* Set */ ProcessingInvert = Off;
/* Get */ value_ = ProcessingInvert;
```

Chapter 16. OutputFormat

The following parameter can be used to configure the applet's image output format i.e. the format and bit alignment.

Automatic Adaptation of the Output Format by the GenTL Adaptor

The GenTL adaptor can automatically set the output format based on the camera settings and a given mapping table. Changing the output format of the applet might get overwritten by the GenTL adaptor on acquisition start. You can only set the output format if this automatic adaptation is disabled. See the GenTL documentation parameter **`AutomaticFormatControl`** for more details.

The automatic adaptation applies for parameters *PixelFormat*, *Format*, *BitAlignment* and *CustomBitShiftRight*.

Depending on the setting of GenTL interface parameter **`OutputPackedFormats`** the automatic adaptation will either use the same pixel format as coming from the camera or an unpacked PC output format. Changing the output format of the applet might get overwritten by the GenTL on acquisition start. You can only set the output format if this automatic adaptation is disabled. See the GenTL documentation parameter **`AutomaticFormatControl`** for more details.

Output Format Setting Defines GenTL Buffer Info

The parameters define the DMA output format and therefore the GenTL buffer info values to inform the consumer about the used output pixel format of the interface.

16.1. Format

Parameter *Format* is used to set and determine the output formats of the DMA channels. An output format value specifies the number of bits and the color format of the output.

This applet has an internal processing bit width of 16 bits. Any selected camera pixel format is mapped to this internal bit width. Check the camera parameter section to learn about the mapping of the camera bits to the internal bit width. For a definition on how to map the internal bits to the output bits, check parameter *BitAlignment*.

Moreover, the color converter of this applet can convert between different color formats of the input and output. Check Chapter 13, '*ColorConverter*' for more information.

This applet supports the following output formats:

- **BGR8** and **RGB8**: 24 bit BGR/RGB color format with 8 bit/component.
- **BGRa8** and **RGBa8**: Color format with 8 bit/component. Component "a" has value zero.
- **BGR10p** and **RGB10p**: 30 bit BGR/RGB color format with 10 bit/component.

30 Bit Output Format

Note that in the 30 bit output format 1 pixel and its 3 color components are distributed over multiple bytes. Also, two successive pixel might share one byte. The pixel are directly aligned in memory. Thus 8 successive color components are stored in 10 byte. The DMA transfer might be filled with random content for the last bytes.

- **BGR12p** and **RGB12p**: 36 bit BGR/RGB color format with 12 bit/component.

36 Bit Output Format

Note that in the 36 bit output format 1 pixel and its 3 color components are distributed over multiple bytes. Also, two successive pixel might share one byte. The pixel are directly aligned in memory. Thus 2 successive color components are stored in 3 byte or two pixel in 9 Byte. The DMA transfer might be filled with random content for the last bytes.

- **BGR14p** and **RGB14p**: 42 bit BGR/RGB color format with 14 bit/component.

42 Bit Output Format

Note that in the 42 bit output format 1 pixel and its 3 color components are distributed over multiple bytes. Also, two successive pixel might share one byte. The pixel are directly aligned in memory. Thus 4 successive color components are stored in 7 byte or four pixel in 21 Byte. The DMA transfer might be filled with random content for the last bytes.

- **BGR16** and **RGB16**: 48 bit BGR/RGB color format with 16 bit/component.

BGR vs. RGB Memory Alignement

Note that the color components are either written to the PC buffer in the common blue, green, red (BGR) or red, green, blue order. So either the blue or red color component is at the lower memory address.

- **Mono8**: 8 bit grayscale format
- **Mono10p**: 10 bit grayscale format

10 Bit Output Format

Note that in the 10 bit output format 1 pixel is distributed over more than one byte. Also, two successive pixel share one byte. The pixel are directly aligned in memory. Thus 8 successive pixel are stored in 10 byte. The DMA transfer might be filled with random content for the last bytes.

- **Mono12p**: 12 bit grayscale format

12 Bit Output Format

Note that in the 12 bit output format 1 pixel is distributed over more than one byte. Also, two successive pixel share the same byte. The pixel are directly aligned in memory. Thus 2 successive pixel are stored in 3 byte. The DMA transfer might be filled with random content for the last bytes.

- **Mono14p**: 14 bit grayscale format

14 Bit Output Format

Note that in the 14 bit output format 1 pixel is distributed over more than one byte. Also, two successive pixel share the same byte. The pixel are directly aligned in memory. Thus 12 successive pixel are stored in 21 byte. The DMA transfer might be filled with random content for the last bytes.

- **Mono16**: 16 bit grayscale format

DMA Bandwidth

Keep in mind that for the 16 bit output mode, the DMA bandwidth might not be sufficient to process the camera input data. Check Section 1.2, 'Bandwidth' for more information.

- **BayerGR8**, **BayerRG8**, **BayerGB8** and **BayerBG8**: 8 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.

- **BayerGR10p**, **BayerRG10p**, **BayerGB10p** and **BayerBG10p**: 10 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.

10 Bit Output Format

Note that in the 10 bit output format 1 pixel is distributed over more than one byte. Also, two successive pixel share one byte. The pixel are directly aligned in memory. Thus 8 successive pixel are stored in 10 byte. The DMA transfer might be filled with random content for the last bytes.

- **BayerGR12p**, **BayerRG12p**, **BayerGB12p** and **BayerBG12p**: 12 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.

12 Bit Output Format

Note that in the 12 bit output format 1 pixel is distributed over more than one byte. Also, two successive pixel share the same byte. The pixel are directly aligned in memory. Thus 2 successive pixel are stored in 3 byte. The DMA transfer might be filled with random content for the last bytes.

- **BayerGR14p**, **BayerRG14p**, **BayerGB14p** and **BayerBG14p**: 14 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.

14 Bit Output Format

Note that in the 14 bit output format 1 pixel is distributed over more than one byte. Also, two successive pixel share the same byte. The pixel are directly aligned in memory. Thus 12 successive pixel are stored in 21 byte. The DMA transfer might be filled with random content for the last bytes.

- **BayerGR16**, **BayerRG16**, **BayerGB16** and **BayerBG16**: 16 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.

DMA Bandwidth

Keep in mind that for the 16 bit output mode, the DMA bandwidth might not be sufficient to process the camera input data. Check Section 1.2, 'Bandwidth' for more information.

- **YCbCr422_8**: YUV 422 output in 8 bit per component.

Table 16.1. Parameter properties of Format

Property	Value	
Name	**Format**	
Display Name	**Output Format**	
Interface	**IEnumeration**	
Access policy	**Read/Write**	
Visibility	**Beginner**	
Allowed values	**Mono8**	Mono 8
	Mono10p	Mono 10p
	Mono12p	Mono 12p
	Mono14p	Mono 14p
	Mono16	Mono 16
	BGR8	BGR 8bit
	BGR10p	BGR 10bit
	BGR12p	BGR 12bit
	BGR14p	BGR 14p
	BGR16	BGR 16bit
	RGB8	RGB 8
	RGB10p	RGB 10p
	RGB12p	RGB 12p
	RGB14p	RGB 14p
	RGB16	RGB 16
	BGRa8	BGRA 8
	RGBa8	RGBA 8
	BiColorRGBG8	BiColor RG BG 8
	BiColorRGBG10	BiColor RG BG 10
	BiColorRGBG12	BiColor RG BG 12
	BiColorBGRG8	BiColor BG RG 8
	BiColorBGRG10	BiColor BG RG 10
	BiColorBGRG12	BiColor BG RG 12
	YCbCr422_8	YCbCr422_8
Default value	**Mono8**	

Example 16.1. Usage of Format

```
/* Set */ Format = Mono8;
/* Get */ value_ = Format;
```

16.2. BitAlignment

The bit alignment is used to map the pixel bits of the internal processing with a depth of 16 bit to the configured DMA output bit depth defined by parameter *Format*.

You can select three different modes: Left aligned, right aligned and a custom shift mode. If you select left aligned, the applet will map the upper bits of the internal processing bit width to the available output bits. If you select right aligned, the applet will map the lower bits of the internal processing bit width to the available output bits. If you want to define a custom bit shift, you'll need to set the parameter to CustomBitShift and use parameter *CustomBitShiftRight* to define the bit shift.

Keep in mind that the internal processing bit width has nothing to do with the camera pixel format. Check the camera parameter section to learn about the mapping of the camera bits to the internal bit width.

Table 16.2. Parameter properties of BitAlignment

Property	Value
Name	**BitAlignment**
Display Name	**Bit Alignment**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**LeftAligned** Left Aligned **RightAligned** Right Aligned **CustomBitShift** Custom Bit Shift
Default value	**LeftAligned**

Example 16.2. Usage of BitAlignment

```
/* Set */ BitAlignment = LeftAligned;
/* Get */ value_ = BitAlignment;
```

16.3. PixelDepth

The pixel depth read-only parameter is used to determine the number of bits used to process a pixel in the applet. It represents the internal bit width.

Table 16.3. Parameter properties of PixelDepth

Property	Value
Name	**PixelDepth**
Display Name	**Pixel Depth**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum** 0 **Maximum** 128 **Stepsize** 1
Unit of measure	**bit**

Example 16.3. Usage of PixelDepth

```
/* Get */ value_ = PixelDepth;
```

16.4. CustomBitShiftRight

This parameter can only be used if parameter *BitAlignment* is set to **CustomBitShift**. If it is enabled, you can define a custom right bit shift value for the DMA output of the frame grabber. A shift of 0 means that the most significant bits (MSB) of the internal processing bit width are mapped to the output MSB. For example, if the applet has an internal processing bit width of 12 bit and you select a 10 bit output, the upper 10 bits are mapped to the output. If you select however a bit width of two, the lower 10 bits are mapped to the output. Note that this applet has an internal bit width of 16 bits.

Table 16.4. Parameter properties of CustomBitShiftRight

Property	Value
Name	**CustomBitShiftRight**
Display Name	**Bit Shift Right**
Interface	**IInteger**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 15** **Stepsize 1**
Default value	**0**
Unit of measure	**bit**

Example 16.4. Usage of CustomBitShiftRight

```
/* Set */ CustomBitShiftRight = 0;
/* Get */ value_ = CustomBitShiftRight;
```

Chapter 17. Miscellaneous

This category summarizes other read and write parameters such as the camera status, buffer fill levels, DMA transfer lengths, and time stamps.

17.1. Version

The category provides version information.

17.1.1. AppletVersion

This parameter indicates the version number of the applet. Report this value when contacting the Basler support.

Table 17.1. Parameter properties of AppletVersion

Property	Value
Name	**AppletVersion**
Display Name	**Applet Version**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 256** **Stepsize 1**

Example 17.1. Usage of AppletVersion

```
/* Get */ value_ = AppletVersion;
```

17.1.2. AppletRevision

This parameter indicates the revision number of the applet. Report this value when contacting the Basler support.

Table 17.2. Parameter properties of AppletRevision

Property	Value
Name	**AppletRevision**
Display Name	**Applet Revision**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 256** **Stepsize 1**

Example 17.2. Usage of AppletRevision

```
/* Get */ value_ = AppletRevision;
```

17.1.3. VisualAppletsBuildVersion

Returns the VisualApplets version used to build the applets.

Table 17.3. Parameter properties of VisualAppletsBuildVersion

Property	Value
Name	**VisualAppletsBuildVersion**
Display Name	**Visual Applets Build Version**
Interface	**IString**
Access policy	**Read-Only**
Visibility	**Beginner**

Example 17.3. Usage of VisualAppletsBuildVersion

```
/* Get */ value_ = VisualAppletsBuildVersion;
```

Chapter 18. BoardStatus

This category gives information about the current framegrabber board status. For example, the number of used PCIe lanes, or the mapping of the physical and logical CXP ports. For imaWorx and imaFLex, it also shows if a trigger board is connected.

18.1. SystemmonitorMappedToFgPort

Indicates the frame grabber port mapping. Range: between 0 and 3.

Table 18.1. Parameter properties of SystemmonitorMappedToFgPort

Property	Value
Name	**SystemmonitorMappedToFgPort**
Display Name	**Systemmonitor Mapped to Fg Port**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 18.1. Usage of SystemmonitorMappedToFgPort

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorMappedToFgPortSelector = i;
    value_ = SystemmonitorMappedToFgPort;
}
```

18.2. SystemmonitorCurrentLinkSpeed

Returns the current link width of the frame grabber representing the number of PCIe lanes that are used for data transfer. This is a value that should correspond to the number of hardware lanes the frame grabber is requiring, otherwise the possible maximum of DMA bandwidth can be reduced drastically.

Table 18.2. Parameter properties of SystemmonitorCurrentLinkSpeed

Property	Value
Name	**SystemmonitorCurrentLinkSpeed**
Display Name	**Systemmonitor Current Link Speed**
Interface	**IFloat**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0.0** **Maximum 1000.0** **Stepsize 0.5**
Unit of measure	**Gb/s**

Example 18.2. Usage of SystemmonitorCurrentLinkSpeed

```
/* Get */ value_ = SystemmonitorCurrentLinkSpeed;
```

18.3. SystemmonitorPcieTrainedPayloadSize

Returns the PCIe packet size that was evaluated during the training period at boot-time.

Table 18.3. Parameter properties of SystemmonitorPcieTrainedPayloadSize

Property	Value
Name	**SystemmonitorPcieTrainedPayloadSize**
Display Name	**Systemmonitor PCIe Trained Payload Size**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 1024** **Stepsize 1**
Unit of measure	**byte**

Example 18.3. Usage of SystemmonitorPcieTrainedPayloadSize

```
/* Get */ value_ = SystemmonitorPcieTrainedPayloadSize;
```

18.4. SystemmonitorPcieTrainedRequestSize

Returns the size (in bytes) of the PCIe packets payload that are used for the data transmission between the frame grabber and the PCIe bridge.

Table 18.4. Parameter properties of SystemmonitorPcieTrainedRequestSize

Property	Value
Name	**SystemmonitorPcieTrainedRequestSize**
Display Name	**Systemmonitor PCIe Trained Request Size**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 4096** **Stepsize 1**
Unit of measure	**byte**

Example 18.4. Usage of SystemmonitorPcieTrainedRequestSize

```
/* Get */ value_ = SystemmonitorPcieTrainedRequestSize;
```

18.5. CxpInputMappedToFWPortPort

This parameter returns the firmware CXP channel, which is currently monitored by the module. There is not necessarily a one-by-one mapping between firmware port (i.e. the camera port resource) and frame grabber port (i.e. the physical connector). Instead, the mapping can be any permutation. The software discovery process reorders the channels and ports to achieve correct virtual interconnect. Range: 0 to 3 (2 bit).

Table 18.5. Parameter properties of CxpInputMappedToFWPortPort

Property	Value
Name	**CxpInputMappedToFWPortPort**
Display Name	**CXP Input Mapped to Firmware Port Port**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 18.5. Usage of CxpInputMappedToFWPortPort

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpInputMappedToFWPortPortSelector = i;
    value_ = CxpInputMappedToFWPortPort;
}
```

Chapter 19. Errors

This category gives information about the current error status. It shows error counters for different error types, such as packet errors, missing connection, undefined data or overtriggering. Additionally, it reports warning type errors, like the number of both corrected and uncorrected packets.

19.1. SystemmonitorDecoder8b10bError

Link stability counter. It is incremented when the number of measured symbols received by the channel transceiver are not in 8b10b encoding or/and have wrong disparity. Range: 0 to (2^48 - 1) (48 bit).

Table 19.1. Parameter properties of SystemmonitorDecoder8b10bError

Property	Value
Name	**SystemmonitorDecoder8b10bError**
Display Name	**Systemmonitor Decoder 8b10b Error**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.1. Usage of SystemmonitorDecoder8b10bError

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorDecoder8b10bErrorSelector = i;
    value_ = SystemmonitorDecoder8b10bError;
}
```

19.2. SystemmonitorByteAlignment8b10bLocked

Monitors whether the clock recovery has worked and valid 8b/10b signals are recognized.

Table 19.2. Parameter properties of SystemmonitorByteAlignment8b10bLocked

Property	Value
Name	**SystemmonitorByteAlignment8b10bLocked**
Display Name	**Systemmonitor Byte Alignment 8B 10 B Locked**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.2. Usage of SystemmonitorByteAlignment8b10bLocked

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorByteAlignment8b10bLockedSelector = i;
    value_ = SystemmonitorByteAlignment8b10bLocked;
}
```

19.3. SystemmonitorRxStreamIncompleteCount

Returns the number of received incomplete stream counts. Range: between 0 and 8191 in steps of 1.

Table 19.3. Parameter properties of SystemmonitorRxStreamIncompleteCount

Property	Value
Name	**SystemmonitorRxStreamIncompleteCount**
Display Name	**Systemmonitor Rx Stream Incomplete Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.3. Usage of SystemmonitorRxStreamIncompleteCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorRxStreamIncompleteCountSelector = i;
    value_ = SystemmonitorRxStreamIncompleteCount;
}
```

19.4. SystemmonitorRxUnknownDataReceivedCount

Returns the number of received unknown data, i.e. packets received that aren't defined in the CXP standard. Range: between 0 and 8191 in steps of 1.

Table 19.4. Parameter properties of SystemmonitorRxUnknownDataReceivedCount

Property	Value
Name	**SystemmonitorRxUnknownDataReceivedCount**
Display Name	**Systemmonitor Rx Unknown Data Received Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.4. Usage of SystemmonitorRxUnknownDataReceivedCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorRxUnknownDataReceivedCountSelector = i;
    value_ = SystemmonitorRxUnknownDataReceivedCount;
}
```

19.5. CxpOvertriggerRequestPulseCount

This parameter counts the trigger requests that were skipped, because the transmitter was still busy by sending the previous trigger packet. See CXP 2.0 standard, chapter 9.3.2. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range: 0 to 4095 (12 bit). Bit 12 indicates an overflow.

Table 19.5. Parameter properties of CxpOvertriggerRequestPulseCount

Property	Value
Name	**CxpOvertriggerRequestPulseCount**
Display Name	**CXP Overtrigger Request Pulse Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.5. Usage of CxpOvertriggerRequestPulseCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpOvertriggerRequestPulseCountSelector = i;
    value_ = CxpOvertriggerRequestPulseCount;
}
```

19.6. CxpTriggerAckMissingCount

This parameter counts the situations in which a trigger packet was sent, but no acknowledgment packet was received for it yet, which then led to a timeout (480ns for 1-6Gb/s, 240ns for 10-12.5Gb/s). See CXP 2.0 standard, chapter 9.3.2. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range: 0 to 8191 (13 bit).

Table 19.6. Parameter properties of CxpTriggerAckMissingCount

Property	Value
Name	**CxpTriggerAckMissingCount**
Display Name	**CXP Lost Trigger ACK Missing Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.6. Usage of CxpTriggerAckMissingCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpTriggerAckMissingCountSelector = i;
    value_ = CxpTriggerAckMissingCount;
}
```

19.7. CxpControlAckLostCount

This parameter counts situations in which a control packet was sent but no acknowledgment packet was received for it yet and the timeout of 200 ms is reached. See CXP 2.0 standard, chapter 9.6.1.1. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 19.7. Parameter properties of CxpControlAckLostCount

Property	Value
Name	**CxpControlAckLostCount**
Display Name	**CXP Control ACK Lost Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.7. Usage of CxpControlAckLostCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpControlAckLostCountSelector = i;
    value_ = CxpControlAckLostCount;
}
```

19.8. CxpControlTagErrorCount

This parameter counts situations in which an acknowledgment for a control packet was received with a tag that doesn't match the expected tag sent in the corresponding request control packet. See CXP 2.0 standard, chapter 9.6.1.2. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 19.8. Parameter properties of CxpControlTagErrorCount

Property	Value
Name	**CxpControlTagErrorCount**
Display Name	**CXP Control Tag Error Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.8. Usage of CxpControlTagErrorCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpControlTagErrorCountSelector = i;
    value_ = CxpControlTagErrorCount;
}
```

19.9. CxpControlAckIncompleteCount

This parameter counts situations in which an incorrectly formatted acknowledgment for a control packet was received. Incorrectly formatted means that e.g. the end of packet indicator is missing etc. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 19.9. Parameter properties of CxpControlAckIncompleteCount

Property	Value
Name	**CxpControlAckIncompleteCount**
Display Name	**CXP Control ACK Incomplete Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.9. Usage of CxpControlAckIncompleteCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpControlAckIncompleteCountSelector = i;
    value_ = CxpControlAckIncompleteCount;
}
```

19.10. CxpHeartbeatIncompleteCount

This parameter counts situations in which the received heart beat packet is incomplete, e.g. it misses the end of the packet indicator. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 19.10. Parameter properties of CxpHeartbeatIncompleteCount

Property	Value
Name	**CxpHeartbeatIncompleteCount**
Display Name	**CXP Heartbeat Incomplete Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.10. Usage of CxpHeartbeatIncompleteCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpHeartbeatIncompleteCountSelector = i;
    value_ = CxpHeartbeatIncompleteCount;
}
```

19.11. CxpHeartbeatMaxPeriodViolationCount

The heartbeat period is defined in CXP 2.0 standard as 100ms maximum, i.e. within that time at least 1 heartbeat packet must be sent by the camera. This parameter counts the situations in which heartbeat packets exceeded this timeout (100ms). Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 19.11. Parameter properties of CxpHeartbeatMaxPeriodViolationCount

Property	Value
Name	**CxpHeartbeatMaxPeriodViolationCount**
Display Name	**CXP Hearbeat Max Period Violation Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.11. Usage of CxpHeartbeatMaxPeriodViolationCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpHeartbeatMaxPeriodViolationCountSelector = i;
    value_ = CxpHeartbeatMaxPeriodViolationCount;
}
```

19.12. PacketTagErrorCount

The parameter counts the number of lost CXP stream packets.

Table 19.12. Parameter properties of PacketTagErrorCount

Property	Value
Name	**PacketTagErrorCount**
Display Name	**Packet Tag Error Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 4095** **Stepsize 1**

Example 19.12. Usage of PacketTagErrorCount

```
/* Get */ value_ = PacketTagErrorCount;
```

19.13. SystemmonitorPacketbufferOverflowCount

This parameter counts the number of overflows that occur due to not correctly aligned package orders.

Table 19.13. Parameter properties of SystemmonitorPacketbufferOverflowCount

Property	Value
Name	**SystemmonitorPacketbufferOverflowCount**
Display Name	**Systemmonitor Packetbuffer Overflow Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 4095** **Stepsize 1**

Example 19.13. Usage of SystemmonitorPacketbufferOverflowCount

```
/* Get */ value_ = SystemmonitorPacketbufferOverflowCount;
```

19.14. SystemmonitorPacketbufferOverflowSource

This parameter returns the port that has overflows due to not correctly aligned package order.

Table 19.14. Parameter properties of SystemmonitorPacketbufferOverflowSource

Property	Value
Name	**SystemmonitorPacketbufferOverflowSource**
Display Name	**Systemmonitor Packetbuffer Overflow Source**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 15** **Stepsize 1**

Example 19.14. Usage of SystemmonitorPacketbufferOverflowSource

```
/* Get */ value_ = SystemmonitorPacketbufferOverflowSource;
```

19.15. CxpImageTagErrorCount

This parameter returns the number of image tag errors (jumps) in the CXP headers.

Table 19.15. Parameter properties of CxpImageTagErrorCount

Property	Value
Name	**CxpImageTagErrorCount**
Display Name	**CXP Image Tag Error Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 8191** **Stepsize 1**

Example 19.15. Usage of CxpImageTagErrorCount

```
/* Get */ value_ = CxpImageTagErrorCount;
```

19.16. CxpStreamIDErrorCount

The parameter counts how often the received stream ID value in the stream packets mismatches the stream ID value specified in the image header. The parameter is 13 bit wide, where the bits [11:0] represent the actual counter value and the bit [12] stands for the counter overflow. When the overflow bit is set, the counter value shall be treated as don't care. Range: 0 to 8191 (13 bit).

Table 19.16. Parameter properties of CxpStreamIDErrorCount

Property	Value
Name	**CxpStreamIDErrorCount**
Display Name	**CXP Stream ID Error Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 8191** **Stepsize 1**

Example 19.16. Usage of CxpStreamIDErrorCount

```
/* Get */ value_ = CxpStreamIDErrorCount;
```

19.17. CxpCameraMarkerErrorCount

This parameter counts how often the sequence of the CXP stream marker and the header or the line markers were incorrect. The parameter is 13 bit wide, where the bits [11:0] represent the actual counter value and the bit [12] stands for the counter overflow. When the overflow bit is set, the counter value shall be treated as don't care. Range: 0 to 8192 (13 bit).

Table 19.17. Parameter properties of CxpCameraMarkerErrorCount

Property	Value
Name	**CxpCameraMarkerErrorCount**
Display Name	**CXP Camera Marker Error Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 8191** **Stepsize 1**

Example 19.17. Usage of CxpCameraMarkerErrorCount

```
/* Get */ value_ = CxpCameraMarkerErrorCount;
```

19.18. CxpCameraUnexpectedStartupDataStatus

This parameter detects the error situation in which the first data value after the operator reset was unexpected, i.e. no image header has been received. This situation can happen due to a buggy implementation of the camera, frame grabber firmware or wrong software control of the discovery procedure. Also, a hardware defect of the camera could theoretically cause such a situation. Range: NO or YES.

Table 19.18. Parameter properties of CxpCameraUnexpectedStartupDataStatus

Property	Value
Name	**CxpCameraUnexpectedStartupDataStatus**
Display Name	**CXP Camera Unexpected Startup Data Status**
Interface	**IEnumeration**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Yes** Yes **No** No

Example 19.18. Usage of CxpCameraUnexpectedStartupDataStatus

```
/* Get */ value_ = CxpCameraUnexpectedStartupDataStatus;
```

19.19. CxpCameraFrameLostCount

This parameter counts the frames that were lost during acquisition and aren't sent into the applet image pipeline. Frames are lost when an error in the image header is detected or when a frame overlaps with another frame. The parameter is 25 bit wide where the bits [23:0] represent the actual counter value and bit [24] stands for the counter overflow. When the overflow bit is set, the counter value shall be treated as don't care. Range 0 to 33554431 (25 bit).

Table 19.19. Parameter properties of CxpCameraFrameLostCount

Property	Value
Name	**CxpCameraFrameLostCount**
Display Name	**CXP Camera Frame Lost Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum** 0 **Maximum** 33554431 **Stepsize** 1

Example 19.19. Usage of CxpCameraFrameLostCount

```
/* Get */ value_ = CxpCameraFrameLostCount;
```

19.20. CxpCameraFrameCorruptCount

This parameter counts the corrupted frames during acquisition. Corrupted frames are frames with error pixels which are sent to the applet image pipeline. The parameter is 25 bit wide where the bits [23:0] represent the actual counter value and bit [24] stands for the counter overflow. When the overflow bit is set, the counter value shall be treated as don't care. Range 0 to 33554431 (25 bit).

Table 19.20. Parameter properties of CxpCameraFrameCorruptCount

Property	Value
Name	**CxpCameraFrameCorruptCount**
Display Name	**CXP Camera Frame Corrupt Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 33554431** **Stepsize 1**

Example 19.20. Usage of CxpCameraFrameCorruptCount

```
/* Get */ value_ = CxpCameraFrameCorruptCount;
```

19.21. CrcErrors

This category gives information about packet CRC errors detected for stream packets and control packets.

19.21.1. SystemmonitorRxPacketCrcErrorCount

Returns the number of received packet CRC errors. Range: between 0 and 8191 in steps of 1.

Table 19.21. Parameter properties of SystemmonitorRxPacketCrcErrorCount

Property	Value
Name	**SystemmonitorRxPacketCrcErrorCount**
Display Name	**Systemmonitor Rx Packet CRC Error Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.21. Usage of SystemmonitorRxPacketCrcErrorCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorRxPacketCrcErrorCountSelector = i;
    value_ = SystemmonitorRxPacketCrcErrorCount;
}
```

19.21.2. CxpStreamPacketCrcError

This parameter returns information whether there were CRC errors in received stream packets. Range 0 (NO) to 1 (YES).

Table 19.22. Parameter properties of CxpStreamPacketCrcError

Property	Value
Name	**CxpStreamPacketCrcError**
Display Name	**CXP Stream Packet CRC Error**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.22. Usage of CxpStreamPacketCrcError

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpStreamPacketCrcErrorSelector = i;
    value_ = CxpStreamPacketCrcError;
}
```

19.21.3. CxpControlAckPacketCrcError

This parameter returns information whether there were CRC errors in received control acknowledgement packets. Range 0 (NO) to 1 (YES).

Table 19.23. Parameter properties of CxpControlAckPacketCrcError

Property	Value
Name	**CxpControlAckPacketCrcError**
Display Name	**CXP Control ACK Packet CRC Error**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.23. Usage of CxpControlAckPacketCrcError

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpControlAckPacketCrcErrorSelector = i;
    value_ = CxpControlAckPacketCrcError;
}
```

19.22. LengthErrors

This category gives information about packet length mismatches for different types of packets.

19.22.1. SystemmonitorRxLengthErrorCount

This parmeter counts how often the length of a CXP packet doesn't correspond to what is specified in the header and returns the number of length errors. Range: between 0 and 8191 in steps of 1.

Table 19.24. Parameter properties of SystemmonitorRxLengthErrorCount

Property	Value
Name	**SystemmonitorRxLengthErrorCount**
Display Name	**Systemmonitor Rx Length Error Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.24. Usage of SystemmonitorRxLengthErrorCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorRxLengthErrorCountSelector = i;
    value_ = SystemmonitorRxLengthErrorCount;
}
```

19.22.2. CxpStreamPacketLengthError

This parameter returns information whether a length error in the stream packets was detected. Range: 0 (NO) to 1 (YES).

Table 19.25. Parameter properties of CxpStreamPacketLengthError

Property	Value
Name	**CxpStreamPacketLengthError**
Display Name	**CXP Stream Packet Length Error**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.25. Usage of CxpStreamPacketLengthError

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpStreamPacketLengthErrorSelector = i;
    value_ = CxpStreamPacketLengthError;
}
```

19.23. ReceivedPacketsCorrected

This category gives information about errors which occurred in received packets which have been corrected.

19.23.1. CxpErrorCorrected

This parameter counts errors received in packet headers and trailers that were corrected. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 19.26. Parameter properties of CxpErrorCorrected

Property	Value
Name	**CxpErrorCorrected**
Display Name	**CXP Error Corrected**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.26. Usage of CxpErrorCorrected

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedSelector = i;
    value_ = CxpErrorCorrected;
}
```

19.23.2. CxpErrorCorrectedTrigger

This parameter returns the information whether errors were corrected in received trigger packets. Range 0 (NO) to 1 (YES).

Table 19.27. Parameter properties of CxpErrorCorrectedTrigger

Property	Value
Name	**CxpErrorCorrectedTrigger**
Display Name	**CXP Error Corrected Trigger**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.27. Usage of CxpErrorCorrectedTrigger

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedTriggerSelector = i;
    value_ = CxpErrorCorrectedTrigger;
}
```

19.23.3. CxpErrorCorrectedTriggerAck

This parameter returns the information whether errors were corrected in received trigger acknowledge packets. Range 0 (NO) to 1 (YES).

Table 19.28. Parameter properties of CxpErrorCorrectedTriggerAck

Property	Value
Name	**CxpErrorCorrectedTriggerAck**
Display Name	**CXP Error Corrected Trigger ACK**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.28. Usage of CxpErrorCorrectedTriggerAck

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedTriggerAckSelector = i;
    value_ = CxpErrorCorrectedTriggerAck;
}
```

19.23.4. CxpErrorCorrectedStream

This parameter returns the information whether errors were corrected in received stream packets. Range 0 (NO) to 1 (YES).

Table 19.29. Parameter properties of CxpErrorCorrectedStream

Property	Value
Name	**CxpErrorCorrectedStream**
Display Name	**CXP Error Corrected Stream**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.29. Usage of CxpErrorCorrectedStream

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedStreamSelector = i;
    value_ = CxpErrorCorrectedStream;
}
```

19.23.5. CxpErrorCorrectedControlAck

This parameter returns the information whether errors were corrected in received stream acknowledge packets. Range 0 (NO) to 1 (YES).

Table 19.30. Parameter properties of CxpErrorCorrectedControlAck

Property	Value
Name	**CxpErrorCorrectedControlAck**
Display Name	**CXP Error Corrected Control ACK**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.30. Usage of CxpErrorCorrectedControlAck

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedControlAckSelector = i;
    value_ = CxpErrorCorrectedControlAck;
}
```

19.23.6. CxpErrorCorrectedLinkTest

This parameter returns the information whether errors were corrected in received link test packets. Range 0 (NO) to 1 (YES).

Table 19.31. Parameter properties of CxpErrorCorrectedLinkTest

Property	Value
Name	**CxpErrorCorrectedLinkTest**
Display Name	**CXP Error Corrected Link Test**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.31. Usage of CxpErrorCorrectedLinkTest

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedLinkTestSelector = i;
    value_ = CxpErrorCorrectedLinkTest;
}
```

19.23.7. CxpErrorCorrectedHeartbeat

This parameter returns the information whether errors were corrected in received heartbeat packets. Range 0 (NO) to 1 (YES).

Table 19.32. Parameter properties of CxpErrorCorrectedHeartbeat

Property	Value
Name	**CxpErrorCorrectedHeartbeat**
Display Name	**CXP Error Corrected Heartbeat**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.32. Usage of CxpErrorCorrectedHeartbeat

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedHeartbeatSelector = i;
    value_ = CxpErrorCorrectedHeartbeat;
}
```

19.23.8. CameraCorrectedErrorCount

The parameter counts the number of single-byte error corrections in CXP stream packets.

Table 19.33. Parameter properties of CameraCorrectedErrorCount

Property	Value
Name	**CameraCorrectedErrorCount**
Display Name	**Corrected Error Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 4095** **Stepsize 1**

Example 19.33. Usage of CameraCorrectedErrorCount

```
/* Get */ value_ = CameraCorrectedErrorCount;
```

19.24. ReceivedPacketsUncorrected

This category gives information about errors which occurred in received packets and which could not be corrected.

19.24.1. CxpErrorUncorrected

This parameter counts errors received in packet headers and trailers that haven't been corrected. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 19.34. Parameter properties of CxpErrorUncorrected

Property	Value
Name	**CxpErrorUncorrected**
Display Name	**CXP Error Uncorrected**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.34. Usage of CxpErrorUncorrected

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedSelector = i;
    value_ = CxpErrorUncorrected;
}
```

19.24.2. CxpErrorUncorrectedTrigger

This parameter returns the information whether there were errors in received trigger packets that haven't been corrected. Range 0 (NO) to 1 (YES).

Table 19.35. Parameter properties of CxpErrorUncorrectedTrigger

Property	Value
Name	**CxpErrorUncorrectedTrigger**
Display Name	**CXP Error Uncorrected Trigger**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.35. Usage of CxpErrorUncorrectedTrigger

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedTriggerSelector = i;
```

```
    value_ = CxpErrorUncorrectedTrigger;
}
```

19.24.3. CxpErrorUncorrectedTriggerAck

This parameter returns the information whether there were errors in received trigger acknowledgement packets that haven't been corrected. Range 0 (NO) to 1 (YES).

Table 19.36. Parameter properties of CxpErrorUncorrectedTriggerAck

Property	Value
Name	**CxpErrorUncorrectedTriggerAck**
Display Name	**CXP Error Uncorrected Trigger ACK**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.36. Usage of CxpErrorUncorrectedTriggerAck

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedTriggerAckSelector = i;
    value_ = CxpErrorUncorrectedTriggerAck;
}
```

19.24.4. CxpErrorUncorrectedStream

This parameter returns the information whether there were errors in received stream packets that haven't been corrected. Range 0 (NO) to 1 (YES).

Table 19.37. Parameter properties of CxpErrorUncorrectedStream

Property	Value
Name	**CxpErrorUncorrectedStream**
Display Name	**CXP Error Uncorrected Stream**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.37. Usage of CxpErrorUncorrectedStream

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedStreamSelector = i;
    value_ = CxpErrorUncorrectedStream;
}
```

19.24.5. CxpErrorUncorrectedControlAck

This parameter returns information whether there were errors in received control acknowledgement packets that haven't been corrected. Range 0 (NO) to 1 (YES).

Table 19.38. Parameter properties of CxpErrorUncorrectedControlAck

Property	Value
Name	**CxpErrorUncorrectedControlAck**
Display Name	**CXP Error Uncorrected Control ACK**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.38. Usage of CxpErrorUncorrectedControlAck

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedControlAckSelector = i;
    value_ = CxpErrorUncorrectedControlAck;
}
```

19.24.6. CxpErrorUncorrectedLinkTest

This parameter returns information whether there were errors in received link test packets that haven't been corrected. Range 0 (NO) to 1 (YES).

Table 19.39. Parameter properties of CxpErrorUncorrectedLinkTest

Property	Value
Name	**CxpErrorUncorrectedLinkTest**
Display Name	**CXP Error Uncorrected Link Test**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.39. Usage of CxpErrorUncorrectedLinkTest

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedLinkTestSelector = i;
    value_ = CxpErrorUncorrectedLinkTest;
}
```

19.24.7. CxpErrorUncorrectedHeartbeat

This parameter returns information whether there were errors in received heartbeat packets that haven't been corrected. Range 0 (NO) to 1 (YES).

Table 19.40. Parameter properties of CxpErrorUncorrectedHeartbeat

Property	Value
Name	**CxpErrorUncorrectedHeartbeat**
Display Name	**CXP Error Uncorrected Heartbeat**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.40. Usage of CxpErrorUncorrectedHeartbeat

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedHeartbeatSelector = i;
    value_ = CxpErrorUncorrectedHeartbeat;
}
```

19.24.8. CameraUncorrectedErrorCount

This parameter counts the number of uncorrected errors. Bit[2] indicates multiple byte errors in CXP stream packets.

Table 19.41. Parameter properties of CameraUncorrectedErrorCount

Property	Value
Name	**CameraUncorrectedErrorCount**
Display Name	**Uncorrected Error Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 4095** **Stepsize 1**

Example 19.41. Usage of CameraUncorrectedErrorCount

```
/* Get */ value_ = CameraUncorrectedErrorCount;
```

19.25. UnsupportedPackets

This category gives information about unsupported packets that have been received.

19.25.1. SystemmonitorRxUnsupportedPacketUnit

This parameter returns the number of received unsupported packets. Range: between 0 and 8191 in steps of 1.

Table 19.42. Parameter properties of SystemmonitorRxUnsupportedPacketUnit

Property	Value
Name	**SystemmonitorRxUnsupportedPacketUnit**
Display Name	**Systemmonitor Rx Unsupported Packet Unit**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.42. Usage of SystemmonitorRxUnsupportedPacketUnit

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorRxUnsupportedPacketUnitSelector = i;
```

```
    value_ = SystemmonitorRxUnsupportedPacketUnit;
}
```

19.25.2. CxpUnsupportedGpioReceived

This parameter returns information whether a GPIO packet was received while using a CXP standard higher than 1.0. Range: 0 (NO) to 1 (YES).

Table 19.43. Parameter properties of CxpUnsupportedGpioReceived

Property	Value
Name	**CxpUnsupportedGpioReceived**
Display Name	**CXP Unsupported GPIO Received**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.43. Usage of CxpUnsupportedGpioReceived

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpUnsupportedGpioReceivedSelector = i;
    value_ = CxpUnsupportedGpioReceived;
}
```

19.25.3. CxpUnsupportedEventReceived

This parameter returns information whether an event packet was received while using a CXP standard less than 2.0. Range: 0 (NO) to 1 (YES).

Table 19.44. Parameter properties of CxpUnsupportedEventReceived

Property	Value
Name	**CxpUnsupportedEventReceived**
Display Name	**CXP Unsupported Event Received**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.44. Usage of CxpUnsupportedEventReceived

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpUnsupportedEventReceivedSelector = i;
    value_ = CxpUnsupportedEventReceived;
}
```

19.25.4. CxpUnsupportedHeartbeatReceived

This parameter returns information whether a heartbeat packet was received while using a CXP standard less than 2.0. Range: 0 (NO) to 1 (YES).

Table 19.45. Parameter properties of CxpUnsupportedHeartbeatReceived

Property	Value
Name	**CxpUnsupportedHeartbeatReceived**
Display Name	**CXP Unsupported Hearbeat Received**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.45. Usage of CxpUnsupportedHeartbeatReceived

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpUnsupportedHeartbeatReceivedSelector = i;
    value_ = CxpUnsupportedHeartbeatReceived;
}
```

19.25.5. CxpUnsupportedGpioAckReceived

This parameter returns information whether a GPIO acknowledgment was received while using a CXP standard higher than 1.0. Range: 0 (NO) to 1 (YES).

Table 19.46. Parameter properties of CxpUnsupportedGpioAckReceived

Property	Value
Name	**CxpUnsupportedGpioAckReceived**
Display Name	**CXP Unsupported GPIO ACK Received**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.46. Usage of CxpUnsupportedGpioAckReceived

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpUnsupportedGpioAckReceivedSelector = i;
    value_ = CxpUnsupportedGpioAckReceived;
}
```

19.25.6. CxpUnsupportedGpioRequestReceived

This parameter returns information whether a GPIO request from VisualApplets was received while using a CXP standard higher than 1.0. Range: 0 (NO) to 1 (YES).

Table 19.47. Parameter properties of CxpUnsupportedGpioRequestReceived

Property	Value
Name	**CxpUnsupportedGpioRequestReceived**
Display Name	**CXP Unsupported GPIO Request Received**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 19.47. Usage of CxpUnsupportedGpioRequestReceived

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpUnsupportedGpioRequestReceivedSelector = i;
    value_ = CxpUnsupportedGpioRequestReceived;
}
```

Chapter 20. Revision History

Revision history of Acquisition Applets releases.

Applet Version	Release Date	Change Log	Delivered with
1.0.3.0	06 Sep 2021	Initial version of this applet.	Framegrabber SDK 5.9
2.0.4.0	01 Mar 2022	Applets built anew for release without functional changes of the applets.	Framegrabber SDK 5.10
2.3.8.0	30 Jun 2023	• Frame-IDs are now generated for DMA transfer and for GenTL buffer info. • Updated the overflow module: Changed the behavior in overflow conditions and added frame-ID to overflow event. • Extended the overflow event by the sequence end flag to indicate the end of an image trigger multi buffer sequence. • The CXP-12 frame grabber now supports stream packet sizes of maximum 16 KB. • Bug fixes.	Framegrabber SDK 5.11
2.4.8.0	22 Dec 2023	• CoaXPress trigger packets updated to CoaXPress 2.1: The applet now supports sending the CXP **LinkTrigger0**, **LinkTrigger1**, **LinkTrigger2** and **LinkTrigger3**. For each link trigger, an individual source can be selected. By default, **LinkTrigger0** is the rising edge of the trigger source and **LinkTrigger1** is the falling edge of the trigger, which usually represents the exposure time. The default setting is equal to previous versions. Thus, the applet is fully compatible to CXP 1.1 and 2.0. **LinkTrigger2** and **LinkTrigger3** are not used by default. The parameters *CxpTriggerPacketMode* (CxpTriggerPacketMode), *TriggerCameraSource* (TriggerCameraSource) and *TriggerCameraPolarity* (TriggerCameraPolarity) became legacy. • Bug fixes.	pylon 7.5.0 & Framegrabber SDK 5.11.2
2.5.8.0	31 Jul 2024	• This applet has been extended with additional parameters and events for problem analysis, e.g. to find errors in the CXP stream. The new parameters and events have been added to the parameter tree. • Bug fixes. See Section 20.1, 'Fixed Issues' for a detailed list of fixed issues.	pylon 8.0.0 & Framegrabber SDK 5.11.3
2.6.9.0	25 Feb 2025	Bug fixes. See Section 20.1, 'Fixed Issues' for a detailed list of fixed issues.	pylon 8.1.0 & Framegrabber SDK 5.11.4

20.1. Fixed Issues

20.1.1. Fixed in Version 2.6.9.0

- To have less artifacts at the edges, the pixels are mirrored. Before fixing this issue, the mirroring on the right side copied the last pixel before starting to mirror. This led to a shift by 1 pixel and therefore moved a red/blue pixel to the position of a green one and vise versa. This has been fixed. (Ticket ID: 318735)
- The filter coefficients have been adjusted to match the ones used in the camera. It used to be -4 8 -4 and has been changed to -0.5 1 -0.5. (Ticket ID: 318737)
- The modes using a leading green component are not defined in PFNC and thus aren't defined in CXP. To reduce confusion, these modes have been removed. (Ticket ID: 319377)
- Before fixing this issue, the applet stated that the signal analyzer measures the period in micro seconds (µs), although it does measure it in nano seconds (ns). The unit has been corrected.
- Before fixing this issue, the event **CameraStreamStatus** returned information that was hard to understand as it required in-depth knowledge of the underlying implementation. This has been fixed. Now, the event provides additional fields so you can understand the decoding by GenICam more easily. (Ticket ID: 315913)
- Before fixing this issues, the ranges of some GenICam parameters were incorrect. The affected parameters were static information such as PCIe speed and parameters that were read only. This has been fixed.

20.1.2. Fixed in Version 2.5.8.0

- Solved communication issues with Opto-Coupled Trigger 5 board. (Ticket ID: 306774)
- Before fixing this issue, in some cases CXP status parameters showed wrong and misleading values. This has been fixed. (Ticket ID: 282288)
- Before fixing this issue, the front GPI 2 and front GPI 3 couldn't be used as a source for GPI events **CustomSignalEvent0** and **CustomSignalEvent1** of line acquisition applets. (Ticket ID: 303278)

20.1.3. Fixed in Version 2.4.8.0

- Before fixing this issue, when image mirroring was activated in line applets, (i.e. *VantagePoint* = TopRight or BottomRight), the actual maximum image width could have been less than documented and depended on the applet and the used pixel format. This has been fixed. (Ticket-ID: 280289)

20.1.4. Fixed in Version 2.3.8.0

- The following CXP status parameters did not work correctly and have been fixed:
  - *SystemmonitorUsedCxpConnections*
  - *PacketTagErrorCount*
  - *CameraCorrectedErrorCount*
  - *CameraUncorrectedErrorCount*
  - *SystemmonitorPacketbufferOverflowCount*
  - *SystemmonitorPacketbufferOverflowSource*
  - *SystemmonitorCxpImageLineMode*

  - **FG_TRIGGER_EVENT_COUNT**
  - **FG_TRIGGER_ACKNOWLEDGEMENT_COUNT**
  - **FG_TRIGGER_WAVE_VIOLATION**
- The GenICam naming was adjusted in line scan applets to remove names that were not adhering to the GenICam standard. The **SendSoftwareTrigger** as well as **SignalAnalyzerClear** was changed to an **iCommand** (used to be an enumeration). Some content of tooltips and description has been corrected accordingly. (Ticket-IDs: 256285 & 271274)
- Fixed period calculation in module Signal Analyzer. The value was wrong by a factor of 2.5. (Ticket-ID: 274220)
- The bandwidth was limited, which means that CXP cameras could not be used with full bandwidth. To solve this issue, the addressing scheme of the DRAM was adjusted to use the bandwidth more efficiently. Now all applets support at least the full CXP-12 speed on all links. In case of debayering, higher bandwidth can be achieved, but a debayering at full speed is in some cases impossible due to PCIe limitations.

20.1.5. Fixed in Version 2.0.4.0

Bitrate switching has been fixed. Prior to fixing this issue, switching the bitrate of a CXP link could put your frame grabber into a failure state. This has been fixed.

20.2. Known Issues

- In rare cases, loading an applet can fail with the error message “(...) : -2050 (Design is invalid)”. In this case, re-load the applet and contact the Basler Support [https://www.baslerweb.com/en/sales-support/support-contact/]. (Ticket-ID: 259458)

Glossary

Area of Interest (AOI)	See Region of Interest.
Board	A Basler hardware. Usually, a board is represented by a frame grabber. Boards might comprise multiple devices.
Board ID Number	An identification number of a Basler board in a PC system. The number is not fixed to a specific hardware but has to be unique in a PC system.
Camera Index	The index of a camera connected to a frame grabber. The first camera will have index zero. Mind the difference between the camera index and the frame grabber camera port. See also Camera Port.
Camera Port	The Basler frame grabber connectors for cameras are called camera ports. They are numbered {0, 1, 2, ...} or enumerated {A, B, C, ... }. Depending on the interface one camera could be connected to multiple camera ports. Also, multiple cameras could be connected to one camera port.
Camera Tap	See Tap.
Device	A board can consist of multiple devices. Devices are numbered. The first device usually has number one.
Direct Memory Access (DMA)	A DMA transfer allows hardware subsystems within the computer to access the system memory independently of the central processing unit (CPU). Basler uses DMAs for data transfer such as image data between a board e.g. a frame grabber and a PC. Data transfers can be established in multiple directions i.e. from a frame grabber to the PC (download) and from the PC to a frame grabber (upload). Multiple DMA channels may exist for one board. Control and configuration data usually do not use DMA channels.
DMA Channel	See DMA Index.
DMA Index	The index of a DMA transfer channel. See also Direct Memory Access.
Event	In programming or runtime environments, a callback function is a piece of executable code that is passed as an argument, which is expected to call back (execute) exactly that time an event is triggered. These events are not related to a special camera functionality and based on frame grabber internal functionality. Basler uses hardware interrupts for the event transfer and processing is absolutely optimized for low latency. These interrupts are only produced by the frame grabber if an event is registered and activated by software. If an event is fired at a very high frequency this may influence the system performance. For example these events can be used to check the reliability between a frame trigger input and the resulting and expected camera frame. The Basler Framegrabber SDK enables an application to get these event notifications about certain state changes at the data flow from camera to RAM and the image and trigger processing as well. Please consult the Basler Framegrabber SDK documentation for more details concerning the implementation of this functionality. Some events are enabled to produce additional data, which is described for the event itself.

Frame Grabber	Usually a PC hardware using PCI express to interface the camera and grab camera images. The frame grabber will grab, buffer, pre-process and forward the images to the PC memory. Moreover, the frame grabber performs the trigger signal processing to trigger the camera, external lights and controllers. On V-series frame grabber custom processing can be implemented using VisualApplets. See also Direct Memory Access, Interface Card, VisualApplets.
GenICam	Generic Interface for Cameras is a generic programming interface for machine vision (industrial) cameras.
GenTL	GenICam Transport Layer. This is the transport layer interface for enumerating cameras, grabbing images from the camera, and moving them to the user application.
Interface Card	Usually a PC hardware using PCI express to interface the camera and grab camera images. The interface card will grab, buffer and forward the images to the PC memory. Moreover, the interface card performs the trigger signal processing to trigger the camera, external lights and controllers. See also Direct Memory Access, Frame Grabber.
Port	See Camera Port.
Process	An image or signal data processing block. A process can include one or more cameras, one or more DMA channels and modules.
Region of Interest (ROI)	Represents a part of a frame. Mostly rectangular and within the original image boundaries. Defined by source coordinates and its dimension. The frame grabber cuts the region of interest from the camera image. A region of interest might reduce or increase the required bandwidth and the corresponding image dimension.
Sensor Tap	See Tap.
Software Callback	See Event.
Tap	Some cameras have multiple taps. This means, they can acquire or transfer more than one pixel at a time which increses the camera's acquisition speed. The camera sensor tap readout order varies. Some cameras read the pixels interlaced using multiple taps, while some cameras read the pixel simultaneously from different locations on the sensor. The reconstruction of the frame is called sensor readout correction. The Camera Link interface is also using multiple taps for image transfer to increase the bandwidth. These taps are independent from the sensor taps.
Trigger	In machine vision and image processing, a trigger is an event which causes an action. This can be for example the initiation of a new line or frame acquisition, the control of external hardware such as flash lights or actions by a software applications. Trigger events can be initiated by external sources, an internal frequency generator (timer) or software applications. The event itself is mostly based on a rising or falling edge of a electrical signal.
Trigger Input	A logic input of a trigger IO. The first input has index 0. Check mapping of input pins to logic inputs in the hardware documentation.
Trigger Output	A logic output of a trigger IO. The first output has index 1. Please check the mapping of output pins to logic outputs in the hardware documentation. The electrical characteristics and specification can be found related to the selected or used trigger board/connector.
Trigger Reliability	See Event.

User Interrupt	See Event.
VisualApplets	Simple programming of FPGA-based image processing devices. VisualApplets enables access to the FPGA processors in the image processing hardware, such as frame grabbers, industrial cameras and image processing devices, to implement individual image processing applications.

Index

D

E

F

H

I

L

M

O

P

R

S

T

V

W