

# microEnable 5 marathon ACX QP

AcquisitionApplets User Documentation for  
**Acq\_SingleCXP6x4LineRGB**

Functional Description  
For Framegrabber SDK Usage

Document Number: AW001767  
Part Number: 000 (English)  
Document Version: 01  
Release Date: 01 December 2022  
Applet Version 2.2.4.0

# 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>

**All material in this publication is subject to change without notice and is copyright Basler AG.**

---

# Table of Contents

1. Introduction .....	1
1.1. Features of Applet Acq_SingleCXP6x4LineRGB .....	1
1.1.1. Parameterization Order .....	2
1.2. Bandwidth .....	2
1.3. Requirements .....	3
1.3.1. Software Requirements .....	3
1.3.2. Hardware Requirements .....	3
1.3.3. License .....	4
1.4. Camera Interface .....	4
1.5. Image Transfer to PC Memory .....	4
2. Software Interface .....	5
3. CoaXPress .....	6
3.1. FG_PIXELFORMAT .....	6
3.2. FG_CXP_TRIGGER_PACKET_MODE .....	6
3.3. FG_TRIGGER_EVENT_COUNT .....	7
3.4. FG_TRIGGER_ACKNOWLEDGEMENT_COUNT .....	8
3.5. FG_TRIGGER_WAVE_VIOLATION .....	8
4. Camera .....	9
4.1. Events .....	9
4.1.1. FG_START_OF_FRAME_CAM_PORT_0 .....	9
4.1.2. FG_END_OF_FRAME_CAM_PORT_0 .....	9
4.1.3. FG_START_OF_LINE_CAM_PORT_0 .....	9
4.1.4. FG_END_OF_LINE_CAM_PORT_0 .....	9
5. Sensor Geometry .....	10
5.1. FG_VANTAGEPOINT .....	10
5.2. FG_SENSORWIDTH .....	10
5.3. FG_SENSORHEIGHT .....	11
6. ROI .....	13
6.1. FG_WIDTH .....	14
6.2. FG_HEIGHT .....	15
6.3. FG_XOFFSET .....	15
6.4. FG_YOFFSET .....	16
7. Digital I/O .....	17
7.1. Camera .....	17
7.1.1. FG_TRIGGERCAMERA_SOURCE .....	17
7.1.2. FG_TRIGGERCAMERA_POLARITY .....	18
7.2. GPO .....	19
7.2.1. FG_TRIGGEROUT_GPO_0_SOURCE et al. ....	19
7.2.2. FG_TRIGGEROUT_GPO_0_POLARITY et al. ....	20
7.2.3. FG_DIGIO_OUTPUT .....	21
7.3. GPI .....	22
7.3.1. FG_DIGIO_INPUT .....	22
7.4. Event Source .....	22
7.4.1. FG_CUSTOM_SIGNAL_EVENT_0_SOURCE .....	22
7.4.2. FG_CUSTOM_SIGNAL_EVENT_0_POLARITY .....	23
7.4.3. FG_CUSTOM_SIGNAL_EVENT_1_SOURCE .....	24
7.4.4. FG_CUSTOM_SIGNAL_EVENT_1_POLARITY .....	25
7.5. Events .....	26
7.5.1. FG_TRIGGER_INPUT0_RISING .....	26
7.5.2. FG_TRIGGER_INPUT0_FALLING .....	26
7.5.3. FG_CUSTOM_SIGNAL_EVENT_0 .....	26
7.5.4. FG_CUSTOM_SIGNAL_EVENT_1 .....	26
8. Line Trigger / ExSync .....	28
8.1. FG_LINETRIGGERMODE .....	28
8.2. FG_EXSYNCON .....	29
8.3. Line Trigger Input .....	30

8.3.1. FG_LINETRIGGERINSRC .....	31
8.3.2. FG_LINETRIGGERINPOLARITY .....	32
8.3.3. FG_LINETRIGGERDEBOUNCING .....	33
8.3.4. Downscale .....	33
8.3.4.1. FG_LINE_DOWNSCALE .....	33
8.3.4.2. FG_LINE_DOWNSCALEINIT .....	34
8.4. Shaft Encoder A/B Filter .....	35
8.4.1. FG_SHAFTENCODERON .....	35
8.4.2. FG_SHAFTENCODERMODE .....	36
8.4.3. FG_SHAFTENCODERINSRC .....	37
8.4.4. FG_SHAFTENCODERLEADING .....	38
8.4.5. FG_SHAFTENCODER_COMPENSATION_ENABLE .....	39
8.4.6. FG_SHAFTENCODER_COMPENSATION_COUNT .....	40
8.5. ExSync Output .....	46
8.5.1. FG_LINEPERIODE .....	46
8.5.2. FG_LINEEXPOSURE .....	47
8.5.3. FG_EXSYNCPOLARITY .....	48
8.5.4. FG_LINETRIGGERDELAY .....	49
9. Image Trigger / Flash .....	50
9.1. FG_IMGTRIGGERMODE .....	51
9.2. FG_IMGTRIGGERON .....	51
9.3. FG_FLASHON .....	52
9.4. FG_IMGTRIGGER_ASYNC_HEIGHT .....	52
9.5. FG_IMGTRIGGER_IS_BUSY .....	53
9.6. Image Trigger Input .....	53
9.6.1. FG_IMGTRIGGERINSRC .....	54
9.6.2. FG_IMGTRIGGERINPOLARITY .....	54
9.6.3. FG_IMGTRIGGERGATEDELAY .....	55
9.6.4. FG_IMGTRIGGERDEBOUNCING .....	55
9.6.5. FG_STROBEPULSEDELAY .....	56
9.6.6. Flash .....	57
9.6.6.1. FG_FLASH_POLARITY .....	57
9.6.7. Software Trigger .....	57
9.6.7.1. FG_SENDSOFTWARETRIGGER .....	57
9.6.7.2. FG_SETSOFTWARETRIGGER .....	58
10. Signal Analyzer .....	59
10.1. FG_SIGNAL_ANALYZER_0_SOURCE et al. ....	59
10.2. FG_SIGNAL_ANALYZER_0_POLARITY et al. ....	60
10.3. FG_SIGNAL_ANALYZER_0_PERIOD_CURRENT et al. ....	61
10.4. FG_SIGNAL_ANALYZER_0_PERIOD_MAX et al. ....	62
10.5. FG_SIGNAL_ANALYZER_0_PERIOD_MIN et al. ....	62
10.6. FG_SIGNAL_ANALYZER_0_PULSE_COUNT et al. ....	63
10.7. FG_SIGNAL_ANALYZER_PULSE_COUNT_DIFFERENCE .....	63
10.8. FG_SIGNAL_ANALYZER_CLEAR .....	64
11. Overflow .....	65
11.1. FG_FILLLEVEL .....	65
11.2. FG_OVERFLOW .....	66
11.3. Events .....	66
11.3.1. FG_OVERFLOW_CAM0 .....	66
12. Image Selector .....	69
12.1. FG_IMG_SELECT_PERIOD .....	69
12.2. FG_IMG_SELECT .....	70
13. White Balance .....	71
13.1. FG_SCALINGFACTOR_RED .....	71
13.2. FG_SCALINGFACTOR_BLUE .....	71
13.3. FG_SCALINGFACTOR_GREEN .....	72
14. Lookup Table .....	73
14.1. FG_LUT_ENABLE .....	73

---

14.2. FG_LUT_TYPE .....	74
14.3. FG_LUT_VALUE_RED .....	74
14.4. FG_LUT_VALUE_GREEN .....	75
14.5. FG_LUT_VALUE_BLUE .....	75
14.6. FG_LUT_CUSTOM_FILE .....	76
14.7. FG_LUT_SAVE_FILE .....	78
14.8. Applet Properties .....	78
14.8.1. FG_LUT_IMPLEMENTATION_TYPE .....	78
14.8.2. FG_LUT_IN_BITS .....	79
14.8.3. FG_LUT_OUT_BITS .....	79
15. Processing .....	81
15.1. FG_PROCESSING_OFFSET .....	82
15.2. FG_PROCESSING_GAIN .....	82
15.3. FG_PROCESSING_GAMMA .....	83
15.4. FG_PROCESSING_INVERT .....	84
16. Output Format .....	86
16.1. FG_FORMAT .....	86
16.2. FG_BITALIGNMENT .....	87
16.3. FG_PIXELDEPTH .....	88
16.4. FG_CUSTOM_BIT_SHIFT_RIGHT .....	88
17. Camera Simulator .....	90
17.1. FG_CAMERASIMULATOR_ENABLE .....	90
17.2. FG_CAMERASIMULATOR_WIDTH .....	91
17.3. FG_CAMERASIMULATOR_LINE_GAP .....	92
17.4. FG_CAMERASIMULATOR_HEIGHT .....	92
17.5. FG_CAMERASIMULATOR_FRAME_GAP .....	93
17.6. FG_CAMERASIMULATOR_PATTERN .....	94
17.7. FG_CAMERASIMULATOR_PATTERN_OFFSET .....	94
17.8. FG_CAMERASIMULATOR_ROLL .....	95
17.9. FG_CAMERASIMULATOR_SELECT_MODE .....	96
17.10. FG_CAMERASIMULATOR_PIXEL_FREQUENCY .....	96
17.11. FG_CAMERASIMULATOR_LINERATE .....	97
17.12. FG_CAMERASIMULATOR_FRAMERATE .....	97
17.13. FG_CAMERASIMULATOR_TRIGGER_MODE .....	98
17.14. FG_CAMERASIMULATOR_ACTIVE .....	99
17.15. FG_CAMERASIMULATOR_PASSIVE .....	99
18. Miscellaneous .....	101
18.1. FG_SYSTEMMONITOR_CHANNEL_CURRENT .....	101
18.2. FG_SYSTEMMONITOR_CHANNEL_VOLTAGE .....	101
18.3. FG_TIMEOUT .....	102
18.4. FG_APPLET_VERSION .....	102
18.5. FG_APPLET_REVISION .....	102
18.6. FG_APPLET_ID .....	103
18.7. FG_APPLET_BUILD_TIME .....	103
18.8. FG_HAP_FILE .....	104
18.9. FG_DMASTATUS .....	104
18.10. FG_CAMSTATUS .....	104
18.11. FG_CAMSTATUS_EXTENDED .....	105
18.12. FG_SYSTEMMONITOR_FPGA_TEMPERATURE .....	106
18.13. FG_SYSTEMMONITOR_FPGA_VCC_INT .....	106
18.14. FG_SYSTEMMONITOR_FPGA_VCC_AUX .....	107
18.15. FG_SYSTEMMONITOR_FPGA_VCC_BRAM .....	107
18.16. FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH .....	108
18.17. FG_SYSTEMMONITOR_CURRENT_LINK_SPEED .....	108
18.18. FG_SYSTEMMONITOR_PCIE_LINK_GEN2_CAPABLE .....	109
18.19. FG_SYSTEMMONITOR_PCIE_LINK_PARTNER_GEN2_CAPABLE .....	109
18.20. FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE .....	110
18.21. FG_SYSTEMMONITOR_EXTENSION_CONNECTOR_PRESENT .....	110

18.22. FG_ALTERNATIVE_BOARD_DETECTION .....	111
18.23. FG_SYSTEMMONITOR_FPGA_DNA .....	111
18.24. FG_SYSTEMMONITOR_CHANNEL_STATE .....	112
18.25. Debug .....	112
18.25.1. FG_DEBUGSOURCE .....	112
18.25.2. FG_DEBUGSOURCENAME .....	113
18.25.3. FG_DEBUGSAVECONFIG .....	113
18.25.4. FG_DEBUG_VERSION .....	114
18.25.5. Input .....	114
18.25.5.1. FG_DEBUGINENABLE .....	114
18.25.5.2. FG_DEBUGFILE .....	115
18.25.5.3. FG_DEBUGINSERT .....	115
18.25.5.4. FG_DEBUGWRITEPIXEL .....	116
18.25.5.5. FG_DEBUGWRITEFLAG .....	116
18.25.5.6. FG_DEBUGREADY .....	117
18.25.6. Output .....	117
18.25.6.1. FG_DEBUGOUTENABLE .....	117
18.25.6.2. FG_DEBUGOUTXPOS .....	118
18.25.6.3. FG_DEBUGOUTYPOS .....	118
18.25.6.4. FG_DEBUGOUTPIXEL .....	119
Glossary .....	120
Index .....	123

---

# Chapter 1. Introduction

This document provides you with detailed information on applet "Acq\_SingleCXP6x4LineRGB" for microEnable 5 marathon ACX QP frame grabber.



This document will outline the features and benefits of this applet. Furthermore, the output formats and the software interface is explained. The main part of this document includes the detailed description of all applet modules and their parameters which make the applet adaptable for numerous applications.


## 1.1. Features of Applet Acq\_SingleCXP6x4LineRGB

"Acq\_SingleCXP6x4LineRGB" is an applet for one camera (single-camera applet). You can configure the CoaXPress camera interface for CoaXPress cameras Version 1.1.1 transferring RGB pixels. Cameras with CoaXPress link aggregation of 4, 2 and 1 are allowed. The maximum link speed is CXP-6. 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 65536 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. 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 a high speed DMA channel. The output pixel format is 24 bits, 30 bit packed, 36 bit packed, 42 bit packed or 48 bits per pixel in BGR color component order.

You can easily include the applet into your own applications using the Silicon Software Framegrabber SDK.

Table 1.1. Feature Summary of Acq\_SingleCXP6x4LineRGB

Feature	Applet Property
Applet Name	 Acq_SingleCXP6x4LineRGB
Type of Applet	AcquisitionApplets
Board	microEnable 5 marathon ACX QP
Minimum Framegrabber SDK Version	5.6
No. of Cameras	1
Camera Type	CoaXPress, link aggregation max. 4, maximum speed CXP-6, Version 1.1.1
Sensor Type	Line Scan
Camera Format	RGB
Pixel Format	RGB8, RGB10, RGB12, RGB14, RGB16
Processing Bit Depth	16 Bit per color component
Sensor Correction / Tap Sorting	no
Maximum Images Dimensions	65536 * 8388607
ROI Stepsize	x: 8, y: 1
Mirroring	Yes, horizontal and vertical (use vantage point)
Image Selector	Yes
Noise Filter	No
Shading Correction	No
Dead Pixel Interpolation	No
Bayer Filter	No
Color White Balancing	Yes
Lookup Table	Full Resolution  Input bits = 14, Output bits = 16  Lookup table can be disabled.
DMA	Full Speed
DMA Image Output Format	BGR 24, 30 packed, 36 packed, 42 packed or 48 bit
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\_SingleCXP6x4LineRGB are listed in the following table.

Table 1.2. Bandwidth of Acq\_SingleCXP6x4LineRGB

Description	Bandwidth
Max. CXP Speed	CXP-6
Peak Bandwidth per Camera	800 MPixel/s
Mean Bandwidth per Camera	800 MPixel/s
DMA Bandwidth	1800 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.

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 1800 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 Silicon Software Framegrabber SDK installation is required. Ensure you use the applet with compatible versions only. You should also take care to use the board firmware and drivers included in the Basler Framegrabber SDK. The minimum Basler Framegrabber SDK version you need for using this applet is version 5.6.

For integration in 3rd party applications, check Chapter 2, '*Software Interface*'.

### 1.3.2. Hardware Requirements

To run applet "Acq\_SingleCXP6x4LineRGB", a Silicon Software microEnable 5 marathon ACX QP 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.

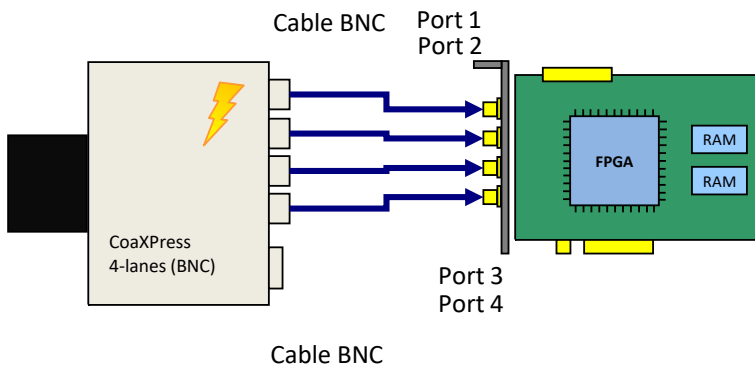
### 1.3.3. License

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\_SingleCXP6x4LineRGB" supports 1 CXP camera. The frame grabber has 4 connectors. Use four, two, or one CoaXPress cables to connect the camera with the frame grabber. The maximum link aggregation of this applet is four. 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. Image Transfer to PC Memory

The image transfer between frame grabber and PC is performed via DMA transfers. In this applet, only one DMA channel exists for transferring image data. The DMA channel has index 0. The applet output format can be set via the parameters of the output format module. See Chapter 16, '*Output Format*'. All outputs are little-endian coded.



### DMA Image Tag

The applet does not generate a valid DMA image tag (**FG\_IMAGE\_TAG**). You may check for lost or corrupted frames using the overflow module described in Chapter 11, '*Overflow*'.

---

# Chapter 2. Software Interface

The software interface of this applet is fully compatible to the Basler Framegrabber SDK. Please read the Basler Framegrabber API manual of the Basler Framegrabber SDK to understand how to include the frame grabbers and their applets into own applications. <https://docs.baslerweb.com/frame-grabbers/framegrabber-sdk-overview.html>

The Basler Framegrabber SDK includes functional SDK examples which use the features of the Framegrabber SDK. Most of these examples can be used with this AcquisitionApplets. These examples are very useful to learn on how to acquire images, set parameters and use events.

This document is focused on the explanation of the functionality and parameterization of the applet. The next chapters will list all parameters of this applet. Keep in mind that for multi-camera applets, parameters can be set for all cameras individually. The sample source codes parameterize the processing components of the first camera. The index in the source code examples has to be changed for the other cameras.

Amongst others, parameters of the applet are set and read using functions

- `int Fg_setParameter(Fg_Struct *Fg, const int Parameter, const void *Value, const unsigned int index)`
- `int Fg_setParameterWithType(Fg_Struct *Fg, const int Parameter, const void *Value, const unsigned int index, const enum FgParamTypes type)`
- `int Fg_getParameter(Fg_Struct *Fg, int Parameter, void *Value, const unsigned int index)`
- `int Fg_getParameterWithType(Fg_Struct *Fg, const int Parameter, void *Value, const unsigned int index, const enum FgParamTypes type)`

The index is used to address a DMA channel, a camera index or a processing logic index. It is important to understand the relations between cameras, processes, parameters and DMA channels. This AcquisitionApplets is a single camera applet and is using only one DMA channel. All parameterizations are made using index 0 only.

For applets having multiple DMA channels for each camera, the relation between the indices is more complex. Please check the respective documentation of these applets for more details.

---

# Chapter 3. CoaXPress

This applet can be used with one line scan camera. 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. Please, 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 (65536).

With the parameters the way trigger packets are sent from the frame grabber to the camera on the CXP link can be defined.

## 3.1. FG\_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.

Table 3.1. Parameter properties of FG\_PIXELFORMAT

Property	Value
Name	<b>FG_PIXELFORMAT</b>
Display Name	<b>Pixel Format</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>RGB8</b> RGB 8bit <b>RGB10</b> RGB 10bit <b>RGB12</b> RGB 12bit <b>RGB14</b> RGB 14bit <b>RGB16</b> RGB 16bit
Default value	<b>RGB8</b>

Example 3.1. Usage of FG\_PIXELFORMAT

```
int result = 0;
int value = RGB8;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_PIXELFORMAT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_PIXELFORMAT, &value, 0, type)) < 0) {
    /* error handling */
}
```

---

## 3.2. FG\_CXP\_TRIGGER\_PACKET\_MODE

Defines the trigger packet mode. For CXP, a packet for the trigger start i.e. rising edge and the trigger end i.e. falling edge is sent on the CXP links. The CXP standard defines a maximum trigger frequency based on the available uplink datarate. Since this can be limited to a certain value the maximum can be reduced. In order to overcome this limitation you can try to set this parameter to **FG\_RISING\_EDGE\_ONLY** so that only half the number of packets need to be transferred.

Table 3.2. Parameter properties of FG\_CXP\_TRIGGER\_PACKET\_MODE

Property	Value
Name	<b>FG_CXP_TRIGGER_PACKET_MODE</b>
Display Name	<b>CXP Trigger Packet Mode</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_STANDARD</b> Standard <b>FG_RISING_EDGE_ONLY</b> Rising Edge Only
Default value	<b>FG_STANDARD</b>

Example 3.2. Usage of FG\_CXP\_TRIGGER\_PACKET\_MODE

```
int result = 0;
int value = FG_STANDARD;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CXP_TRIGGER_PACKET_MODE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CXP_TRIGGER_PACKET_MODE, &value, 0, type)) < 0) {
    /* error handling */
}
```

### 3.3. FG\_TRIGGER\_EVENT\_COUNT

The parameter indicates how many trigger edge events have been sent to the camera.

Table 3.3. Parameter properties of FG\_TRIGGER\_EVENT\_COUNT

Property	Value
Name	<b>FG_TRIGGER_EVENT_COUNT</b>
Display Name	<b>Trigger Event Count</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 1048575</b> <b>Stepsize 1</b>

Example 3.3. Usage of FG\_TRIGGER\_EVENT\_COUNT

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_TRIGGER_EVENT_COUNT, &value, 0, type)) < 0) {
    /* error handling */
}
```

### 3.4. FG\_TRIGGER\_ACKNOWLEDGEMENT\_COUNT

The parameter indicates how many trigger acknowledgement packets sent by the camera (in answer to the trigger edge packets sent before) have been received by the frame grabber.

Table 3.4. Parameter properties of FG\_TRIGGER\_ACKNOWLEDGEMENT\_COUNT

Property	Value
Name	<b>FG_TRIGGER_ACKNOWLEDGEMENT_COUNT</b>
Display Name	<b>Trigger Acknowledgement Count</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 1048575</b> <b>Stepsize 1</b>

Example 3.4. Usage of FG\_TRIGGER\_ACKNOWLEDGEMENT\_COUNT

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_TRIGGER_ACKNOWLEDGEMENT_COUNT, &value, 0, type)) < 0) {
    /* error handling */
}
```

### 3.5. FG\_TRIGGER\_WAVE\_VIOLATION

The parameter is set to 1 if the applet detects a distance between two trigger edges which violates the minimal edge frequency. The parameter holds its value until it has been read. After being read, the parameter updates the value. Frequency control is running permanently and is not influenced by the read status of the parameter.

Table 3.5. Parameter properties of FG\_TRIGGER\_WAVE\_VIOLATION

Property	Value
Name	<b>FG_TRIGGER_WAVE_VIOLATION</b>
Display Name	<b>Trigger Wave Violation</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 1048575</b> <b>Stepsize 1</b>

Example 3.5. Usage of FG\_TRIGGER\_WAVE\_VIOLATION

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_TRIGGER_WAVE_VIOLATION, &value, 0, type)) < 0) {
    /* error handling */
}
```

---

# Chapter 4. Camera

This applet Acq\_SingleCXP6x4LineRGB for the microEnable 5 marathon ACX QP 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.

## 4.1. 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 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.

### 4.1.1. FG\_START\_OF\_FRAME\_CAM\_PORT\_0

This event is generated when the first pixel of one camera frame arrives at the applet. Except for the timestamp, the event has no additional data included. Keep in mind that a high framerate can cause high interrupt rates which might slow down the overall PC system. This event can only occur if the acquisition is running.

### 4.1.2. FG\_END\_OF\_FRAME\_CAM\_PORT\_0

This event is generated right after the last pixel of one camera frame arrives at the applet. Except for the timestamp, the event has no additional data included. Keep in mind that a high framerate can cause high interrupt rates which might slow down the overall PC system. This event can only occur if the acquisition is running.

### 4.1.3. FG\_START\_OF\_LINE\_CAM\_PORT\_0

This event is generated when the first pixel of camera line arrives at the framegrabber. Except for the timestamp, the event has no additional data included. 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 occur. This event can only occur if the acquisition is running.

### 4.1.4. FG\_END\_OF\_LINE\_CAM\_PORT\_0

This event is generated when the last pixel of camera line has arrives at the framegrabber. Except for the timestamp, the event has no additional data included. 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 5. Sensor Geometry

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.

## 5.1. FG\_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 *FG\_SENSORWIDTH* limits the maximum width.

If a vertical mirroring is active, the parameter *FG\_SENSORHEIGHT* limits the maximum height.

Table 5.1. Parameter properties of FG\_VANTAGEPOINT

Property	Value
Name	FG_VANTAGEPOINT
Display Name	Vantage Point
Type	Enumeration
Access policy	Read/Write
Storage policy	Persistent
Allowed values	FG_VANTAGEPOINT_TOP_LEFT Top-Left FG_VANTAGEPOINT_TOP_RIGHT Top-Right FG_VANTAGEPOINT_BOTTOM_LEFT Bottom-Left FG_VANTAGEPOINT_BOTTOM_RIGHT Bottom-Right
Default value	FG_VANTAGEPOINT_TOP_LEFT

Example 5.1. Usage of FG\_VANTAGEPOINT

```
int result = 0;
int value = FG_VANTAGEPOINT_TOP_LEFT;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_VANTAGEPOINT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_VANTAGEPOINT, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 5.2. FG\_SENSORWIDTH

To mirror the incoming data correctly, the parameter *FG\_SENSORWIDTH* is required. The value of *FG\_SENSORWIDTH* is ignored, if *FG\_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 *FG\_SENSORWIDTH* Is Not Used

If no mirroring is active, the value of the parameter *FG\_SENSORWIDTH* is not used. Instead, the sum of *FG\_XOFFSET* and *FG\_WIDTH* is used. This makes the use of the module easier as an extra configuration is avoided, if defaults are used.



Table 5.2. Parameter properties of FG\_SENSORWIDTH

Property	Value
Name	<b>FG_SENSORWIDTH</b>
Display Name	<b>Sensor Width</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 16</b> <b>Maximum 65536</b> <b>Stepsize 8</b>
Default value	<b>1024</b>
Unit of measure	<b>pixel</b>

Example 5.2. Usage of FG\_SENSORWIDTH

```

int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SENSORWIDTH, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SENSORWIDTH, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 5.3. FG\_SENSORHEIGHT

For mirroring, the applet needs to be parameterized with the exact sensor height transferred from the camera to the frame grabber. If you have set a region of interest in the camera, the parameter *FG\_SENSORHEIGHT* needs to be set to the ROI size.



**If No Vertical Mirroring Is Active, the Value of *FG\_SENSORHEIGHT* Is Not Used**

If no vertical mirroring is configured, the value of the parameter *FG\_SENSORHEIGHT* is not used. Instead, the sum of *FG\_YOFFSET* and *FG\_HEIGHT* is used. This makes the use of the module easier as an extra configuration is avoided, if defaults are used.

Table 5.3. Parameter properties of FG\_SENSORHEIGHT

Property	Value
Name	<b>FG_SENSORHEIGHT</b>
Display Name	<b>Sensor Height</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 1</b> <b>Maximum 8388607</b> <b>Stepsize 1</b>
Default value	<b>1024</b>
Unit of measure	<b>pixel</b>

### Example 5.3. Usage of FG\_SENSORHEIGHT

---

```
int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SENSORHEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

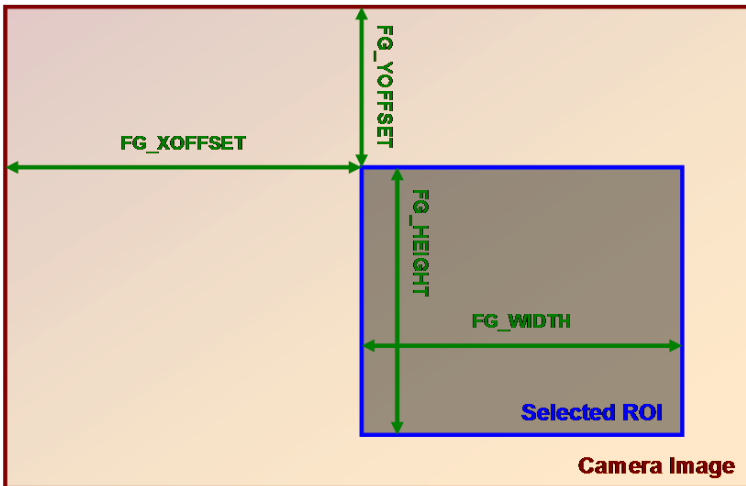
if ((result = Fg_getParameterWithType(fg, FG_SENSORHEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}
```

---

# Chapter 6. ROI

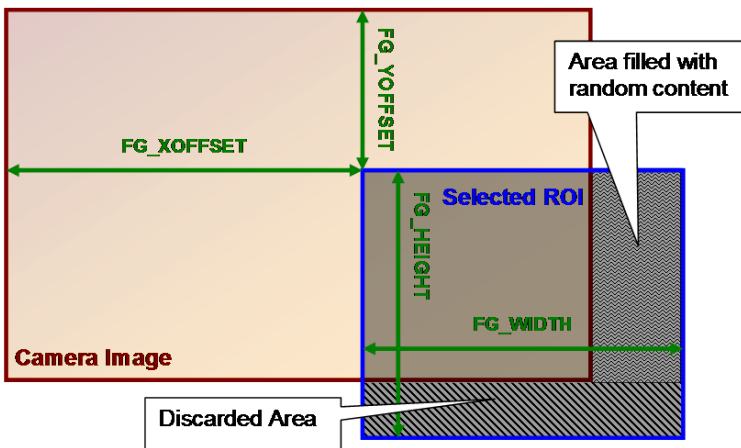
This module allows the definition of a region of interest (ROI), also called area of interest (AOI). A ROI allows the selection of a smaller subset pixel area from the input image. It is defined by using parameters *FG\_XOFFSET*, *FG\_WIDTH*, *FG\_YOFFSET* and *FG\_HEIGHT*. The following figure illustrates the parameters.

Figure 6.1. 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 6.2. 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 65536 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.



## Camera ROI

Most cameras allow the setting of a ROI inside the camera. The ROI settings described in this section are independent from the camera settings.



## 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.

## 6.1. FG\_WIDTH

The parameter specifies the width of the ROI. The values of parameters *FG\_WIDTH* + *FG\_XOFFSET* must not exceed the maximum image width of 65536 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 32768, the not mirrored image can have the full width of 65536.

Table 6.1. Parameter properties of FG\_WIDTH

Property	Value
Name	<b>FG_WIDTH</b>
Display Name	<b>Width</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 16</b> <b>Maximum 65536</b> <b>Stepsize 8</b>
Default value	<b>1024</b>
Unit of measure	<b>pixel</b>

Example 6.1. Usage of FG\_WIDTH

```
int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 6.2. FG\_HEIGHT

The parameter specifies the height of the ROI. The values of parameters *FG\_HEIGHT* + *FG\_YOFFSET* 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 the maximum height is limited by the DRAM and the sensor width (the sensor dimension needs to fit into the DRAM).

Table 6.2. Parameter properties of FG\_HEIGHT

Property	Value
Name	<b>FG_HEIGHT</b>
Display Name	<b>Height</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 1</b> <b>Maximum 8388607</b> <b>Stepsize 1</b>
Default value	<b>1024</b>
Unit of measure	<b>pixel</b>

Example 6.2. Usage of FG\_HEIGHT

```
int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 6.3. FG\_XOFFSET

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 6.3. Parameter properties of FG\_XOFFSET

Property	Value
Name	<b>FG_XOFFSET</b>
Display Name	<b>Xoffset</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 65520</b> <b>Stepsize 8</b>
Default value	<b>0</b>
Unit of measure	<b>pixel</b>

**Example 6.3. Usage of FG\_XOFFSET**

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_XOFFSET, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_XOFFSET, &value, 0, type)) < 0) {
    /* error handling */
}

```

**6.4. FG\_YOFFSET**

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 6.4. Parameter properties of FG\_YOFFSET**

Property	Value
Name	<b>FG_YOFFSET</b>
Display Name	<b>Yoffset</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 8388606</b> <b>Stepsize 1</b>
Default value	<b>0</b>
Unit of measure	<b>pixel</b>

**Example 6.4. Usage of FG\_YOFFSET**

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_YOFFSET, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_YOFFSET, &value, 0, type)) < 0) {
    /* error handling */
}

```

---

# Chapter 7. Digital I/O

The frame grabber provides digital inputs and digital outputs for triggering, light synchronization, hardware control etc. This microEnable 5 marathon ACX QP frame grabber has

- 8 general purpose digital inputs (GPIs) using the extension board connector of the frame grabber.
- 4 front general purpose digital inputs (Front GPIs) using the SUB-D connector on the frame grabber slot bracket or LightBridge front.
- 8 digital outputs on the GPO connector
- 2 digital outputs on the Front 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.
- **FG\_SIGNAL\_CAM0\_EXSYNC:** The Exsync signal. Usually the line trigger signal used to trigger the camera. Check Chapter 8, '*Line Trigger / ExSync*' for more information.
- **FG\_SIGNAL\_CAM0\_EXSYNC2:** The Exsync 2 signal a delayed exsync signal. Check *FG\_LINETRIGGERDELAY* for more information.
- **FG\_SIGNAL\_CAM0\_FLASH:** The flash signal. It is generated once at the start of each frame generated by the trigger module. Check Chapter 9, '*Image Trigger / Flash*' for more information.
- **FG\_SIGNAL\_CAM0\_LVAL:** The line valid signal of the received camera or simulator image data. The signal is high for the duration of the line data transfer.
- **FG\_SIGNAL\_CAM0\_FVAL:** 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 9, '*Image Trigger / Flash*' for more information.
- **FG\_SIGNAL\_GPI\_0 to FG\_SIGNAL\_GPI\_7 and FG\_SIGNAL\_FRONT\_GPI\_0 to FG\_SIGNAL\_FRONT\_GPI\_3:** Direct mapping of the digital input signal after debouncing.
- **FG\_SIGNAL\_CAM0\_LINE\_START:** Line start pulse. Use for events and signal analyzer.
- **FG\_SIGNAL\_CAM0\_LINE\_END:** Line end pulse. Use for events and signal analyzer.
- **FG\_SIGNAL\_CAM0\_FRAME\_START:** Frame start pulse. Use for events and signal analyzer.
- **FG\_SIGNAL\_CAM0\_FRAME\_END:** Frame end pulse. Use for events and signal analyzer.

## 7.1. Camera

### 7.1.1. FG\_TRIGGERCAMERA\_SOURCE

Select the signal source of the trigger signal output. For further explanation of the available sources see Chapter 7, 'Digital I/O'.

Note that you are using a CXP camera interface. The rising edge of the selected signal will start the exposure. A falling edge will end the exposure if the camera is set to an external controlled exposure time. Ensure a correct high active polarity of the source. For further information on CXP trigger packets see parameter `FG_CXP_TRIGGER_PACKET_MODE`.

Table 7.1. Parameter properties of `FG_TRIGGERCAMERA_SOURCE`

Property	Value																																						
Name	<code>FG_TRIGGERCAMERA_SOURCE</code>																																						
Display Name	Camera Trigger Source																																						
Type	Enumeration																																						
Access policy	Read/Write/Change																																						
Storage policy	Persistent																																						
Allowed values	<table border="0"> <tr><td><code>GND</code></td><td>Gnd</td></tr> <tr><td><code>VCC</code></td><td>Vcc</td></tr> <tr><td><code>FG_SIGNAL_CAM0_EXSYNC</code></td><td>Cam0 Exsync</td></tr> <tr><td><code>FG_SIGNAL_CAM0_EXSYNC2</code></td><td>Cam0 Exsync2</td></tr> <tr><td><code>FG_SIGNAL_CAM0_FLASH</code></td><td>Cam0 Flash</td></tr> <tr><td><code>FG_SIGNAL_CAM0_LVAL</code></td><td>Cam0 LVAL</td></tr> <tr><td><code>FG_SIGNAL_CAM0_FVAL</code></td><td>Cam0 FVAL</td></tr> <tr><td><code>FG_SIGNAL_GPI_0</code></td><td>GPI 0</td></tr> <tr><td><code>FG_SIGNAL_GPI_1</code></td><td>GPI 1</td></tr> <tr><td><code>FG_SIGNAL_GPI_2</code></td><td>GPI 2</td></tr> <tr><td><code>FG_SIGNAL_GPI_3</code></td><td>GPI 3</td></tr> <tr><td><code>FG_SIGNAL_GPI_4</code></td><td>GPI 4</td></tr> <tr><td><code>FG_SIGNAL_GPI_5</code></td><td>GPI 5</td></tr> <tr><td><code>FG_SIGNAL_GPI_6</code></td><td>GPI 6</td></tr> <tr><td><code>FG_SIGNAL_GPI_7</code></td><td>GPI 7</td></tr> <tr><td><code>FG_SIGNAL_FRONT_GPI_0</code></td><td>Front GPI 0</td></tr> <tr><td><code>FG_SIGNAL_FRONT_GPI_1</code></td><td>Front GPI 1</td></tr> <tr><td><code>FG_SIGNAL_FRONT_GPI_2</code></td><td>Front GPI 2</td></tr> <tr><td><code>FG_SIGNAL_FRONT_GPI_3</code></td><td>Front GPI 3</td></tr> </table>	<code>GND</code>	Gnd	<code>VCC</code>	Vcc	<code>FG_SIGNAL_CAM0_EXSYNC</code>	Cam0 Exsync	<code>FG_SIGNAL_CAM0_EXSYNC2</code>	Cam0 Exsync2	<code>FG_SIGNAL_CAM0_FLASH</code>	Cam0 Flash	<code>FG_SIGNAL_CAM0_LVAL</code>	Cam0 LVAL	<code>FG_SIGNAL_CAM0_FVAL</code>	Cam0 FVAL	<code>FG_SIGNAL_GPI_0</code>	GPI 0	<code>FG_SIGNAL_GPI_1</code>	GPI 1	<code>FG_SIGNAL_GPI_2</code>	GPI 2	<code>FG_SIGNAL_GPI_3</code>	GPI 3	<code>FG_SIGNAL_GPI_4</code>	GPI 4	<code>FG_SIGNAL_GPI_5</code>	GPI 5	<code>FG_SIGNAL_GPI_6</code>	GPI 6	<code>FG_SIGNAL_GPI_7</code>	GPI 7	<code>FG_SIGNAL_FRONT_GPI_0</code>	Front GPI 0	<code>FG_SIGNAL_FRONT_GPI_1</code>	Front GPI 1	<code>FG_SIGNAL_FRONT_GPI_2</code>	Front GPI 2	<code>FG_SIGNAL_FRONT_GPI_3</code>	Front GPI 3
<code>GND</code>	Gnd																																						
<code>VCC</code>	Vcc																																						
<code>FG_SIGNAL_CAM0_EXSYNC</code>	Cam0 Exsync																																						
<code>FG_SIGNAL_CAM0_EXSYNC2</code>	Cam0 Exsync2																																						
<code>FG_SIGNAL_CAM0_FLASH</code>	Cam0 Flash																																						
<code>FG_SIGNAL_CAM0_LVAL</code>	Cam0 LVAL																																						
<code>FG_SIGNAL_CAM0_FVAL</code>	Cam0 FVAL																																						
<code>FG_SIGNAL_GPI_0</code>	GPI 0																																						
<code>FG_SIGNAL_GPI_1</code>	GPI 1																																						
<code>FG_SIGNAL_GPI_2</code>	GPI 2																																						
<code>FG_SIGNAL_GPI_3</code>	GPI 3																																						
<code>FG_SIGNAL_GPI_4</code>	GPI 4																																						
<code>FG_SIGNAL_GPI_5</code>	GPI 5																																						
<code>FG_SIGNAL_GPI_6</code>	GPI 6																																						
<code>FG_SIGNAL_GPI_7</code>	GPI 7																																						
<code>FG_SIGNAL_FRONT_GPI_0</code>	Front GPI 0																																						
<code>FG_SIGNAL_FRONT_GPI_1</code>	Front GPI 1																																						
<code>FG_SIGNAL_FRONT_GPI_2</code>	Front GPI 2																																						
<code>FG_SIGNAL_FRONT_GPI_3</code>	Front GPI 3																																						
Default value	<code>FG_SIGNAL_CAM0_EXSYNC</code>																																						

Example 7.1. Usage of `FG_TRIGGERCAMERA_SOURCE`

```
int result = 0;
int value = FG_SIGNAL_CAM0_EXSYNC;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}
```

### 7.1.2. FG\_TRIGGERCAMERA\_POLARITY

Select the output polarity of the CC trigger signal. For further explanation of the available sources see Chapter 7, 'Digital I/O'.

Ensure that the signal has the correct polarity and pulse width. Check the camera user manual if the camera requires high active or low active signals.



Table 7.2. Parameter properties of FG\_TRIGGERCAMERA\_POLARITY

Property	Value
Name	<b>FG_TRIGGERCAMERA_POLARITY</b>
Display Name	<b>Camera Trigger Output Polarity</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_LOW</b> Low Active <b>FG_HIGH</b> High Active
Default value	<b>FG_HIGH</b>

Example 7.2. Usage of FG\_TRIGGERCAMERA\_POLARITY

```

int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_TRIGGERCAMERA_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGERCAMERA_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 7.2. GPO

### 7.2.1. FG\_TRIGGEROUT\_GPO\_0\_SOURCE et al.



#### Note

This description applies also to the following parameters: FG\_TRIGGEROUT\_GPO\_1\_SOURCE, FG\_TRIGGEROUT\_GPO\_2\_SOURCE, FG\_TRIGGEROUT\_GPO\_3\_SOURCE, FG\_TRIGGEROUT\_GPO\_4\_SOURCE, FG\_TRIGGEROUT\_GPO\_5\_SOURCE, FG\_TRIGGEROUT\_GPO\_6\_SOURCE, FG\_TRIGGEROUT\_GPO\_7\_SOURCE, FG\_TRIGGEROUT\_FRONT\_GPO\_0\_SOURCE, FG\_TRIGGEROUT\_FRONT\_GPO\_1\_SOURCE

Select the signal source of the General Purpose Output (GPO) or Front GPO. For further explanation of the available sources see Chapter 7, 'Digital I/O'.

You can change the polarity using parameter *FG\_TRIGGEROUT\_GPO\_X\_POLARITY*.

Table 7.3. Parameter properties of FG\_TRIGGEROUT\_GPO\_0\_SOURCE

Property	Value																																						
Name	<b>FG_TRIGGEROUT_GPO_0_SOURCE</b>																																						
Display Name	<b>GPO 0 Source</b>																																						
Type	<b>Enumeration</b>																																						
Access policy	<b>Read/Write/Change</b>																																						
Storage policy	<b>Persistent</b>																																						
Allowed values	<table border="0"> <tr><td><b>GND</b></td><td>Gnd</td></tr> <tr><td><b>VCC</b></td><td>Vcc</td></tr> <tr><td><b>FG_SIGNAL_CAM0_EXSYNC</b></td><td>Cam0 Exsync</td></tr> <tr><td><b>FG_SIGNAL_CAM0_EXSYNC2</b></td><td>Cam0 Exsync2</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FLASH</b></td><td>Cam0 Flash</td></tr> <tr><td><b>FG_SIGNAL_CAM0_LVAL</b></td><td>Cam0 LVAL</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FVAL</b></td><td>Cam0 FVAL</td></tr> <tr><td><b>FG_SIGNAL_GPI_0</b></td><td>GPI 0</td></tr> <tr><td><b>FG_SIGNAL_GPI_1</b></td><td>GPI 1</td></tr> <tr><td><b>FG_SIGNAL_GPI_2</b></td><td>GPI 2</td></tr> <tr><td><b>FG_SIGNAL_GPI_3</b></td><td>GPI 3</td></tr> <tr><td><b>FG_SIGNAL_GPI_4</b></td><td>GPI 4</td></tr> <tr><td><b>FG_SIGNAL_GPI_5</b></td><td>GPI 5</td></tr> <tr><td><b>FG_SIGNAL_GPI_6</b></td><td>GPI 6</td></tr> <tr><td><b>FG_SIGNAL_GPI_7</b></td><td>GPI 7</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_0</b></td><td>Front GPI 0</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_1</b></td><td>Front GPI 1</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_2</b></td><td>Front GPI 2</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_3</b></td><td>Front GPI 3</td></tr> </table>	<b>GND</b>	Gnd	<b>VCC</b>	Vcc	<b>FG_SIGNAL_CAM0_EXSYNC</b>	Cam0 Exsync	<b>FG_SIGNAL_CAM0_EXSYNC2</b>	Cam0 Exsync2	<b>FG_SIGNAL_CAM0_FLASH</b>	Cam0 Flash	<b>FG_SIGNAL_CAM0_LVAL</b>	Cam0 LVAL	<b>FG_SIGNAL_CAM0_FVAL</b>	Cam0 FVAL	<b>FG_SIGNAL_GPI_0</b>	GPI 0	<b>FG_SIGNAL_GPI_1</b>	GPI 1	<b>FG_SIGNAL_GPI_2</b>	GPI 2	<b>FG_SIGNAL_GPI_3</b>	GPI 3	<b>FG_SIGNAL_GPI_4</b>	GPI 4	<b>FG_SIGNAL_GPI_5</b>	GPI 5	<b>FG_SIGNAL_GPI_6</b>	GPI 6	<b>FG_SIGNAL_GPI_7</b>	GPI 7	<b>FG_SIGNAL_FRONT_GPI_0</b>	Front GPI 0	<b>FG_SIGNAL_FRONT_GPI_1</b>	Front GPI 1	<b>FG_SIGNAL_FRONT_GPI_2</b>	Front GPI 2	<b>FG_SIGNAL_FRONT_GPI_3</b>	Front GPI 3
<b>GND</b>	Gnd																																						
<b>VCC</b>	Vcc																																						
<b>FG_SIGNAL_CAM0_EXSYNC</b>	Cam0 Exsync																																						
<b>FG_SIGNAL_CAM0_EXSYNC2</b>	Cam0 Exsync2																																						
<b>FG_SIGNAL_CAM0_FLASH</b>	Cam0 Flash																																						
<b>FG_SIGNAL_CAM0_LVAL</b>	Cam0 LVAL																																						
<b>FG_SIGNAL_CAM0_FVAL</b>	Cam0 FVAL																																						
<b>FG_SIGNAL_GPI_0</b>	GPI 0																																						
<b>FG_SIGNAL_GPI_1</b>	GPI 1																																						
<b>FG_SIGNAL_GPI_2</b>	GPI 2																																						
<b>FG_SIGNAL_GPI_3</b>	GPI 3																																						
<b>FG_SIGNAL_GPI_4</b>	GPI 4																																						
<b>FG_SIGNAL_GPI_5</b>	GPI 5																																						
<b>FG_SIGNAL_GPI_6</b>	GPI 6																																						
<b>FG_SIGNAL_GPI_7</b>	GPI 7																																						
<b>FG_SIGNAL_FRONT_GPI_0</b>	Front GPI 0																																						
<b>FG_SIGNAL_FRONT_GPI_1</b>	Front GPI 1																																						
<b>FG_SIGNAL_FRONT_GPI_2</b>	Front GPI 2																																						
<b>FG_SIGNAL_FRONT_GPI_3</b>	Front GPI 3																																						
Default value	<b>FG_SIGNAL_CAM0_FLASH</b>																																						

Example 7.3. Usage of FG\_TRIGGEROUT\_GPO\_0\_SOURCE

```

int result = 0;
int value = FG_SIGNAL_CAM0_FLASH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_TRIGGEROUT_GPO_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGEROUT_GPO_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 7.2.2. FG\_TRIGGEROUT\_GPO\_0\_POLARITY et al.



### Note

This description applies also to the following parameters: FG\_TRIGGEROUT\_GPO\_1\_POLARITY, FG\_TRIGGEROUT\_GPO\_2\_POLARITY, FG\_TRIGGEROUT\_GPO\_3\_POLARITY, FG\_TRIGGEROUT\_GPO\_4\_POLARITY, FG\_TRIGGEROUT\_GPO\_5\_POLARITY, FG\_TRIGGEROUT\_GPO\_6\_POLARITY, FG\_TRIGGEROUT\_GPO\_7\_POLARITY, FG\_TRIGGEROUT\_FRONT\_GPO\_0\_POLARITY, FG\_TRIGGEROUT\_FRONT\_GPO\_1\_POLARITY

Select the output polarity the General Purpose Output (GPO) or Front GPO to invert the signal. For further explanation of the available sources see Chapter 7, 'Digital I/O'.

Table 7.4. Parameter properties of FG\_TRIGGEROUT\_GPO\_0\_POLARITY

Property	Value
Name	<b>FG_TRIGGEROUT_GPO_0_POLARITY</b>
Display Name	<b>GPO 0 Polarity</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_LOW</b> Low Active <b>FG_HIGH</b> High Active
Default value	<b>FG_HIGH</b>

Example 7.4. Usage of FG\_TRIGGEROUT\_GPO\_0\_POLARITY

```

int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_TRIGGEROUT_GPO_0_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGEROUT_GPO_0_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 7.2.3. FG\_DIGIO\_OUTPUT

Set the output value of outputs 3 and 7 using this parameter. Bit 0 of the parameter refers to the value at output 3, while bit 1 refers to output 7.



#### Legacy Parameter

This is a legacy parameter. The parameter will overwrite parameters *FG\_TRIGGEROUT\_GPO\_3\_SOURCE* and *FG\_TRIGGEROUT\_GPO\_7\_SOURCE*. If these parameters are not set to VCC or GND this parameter will return value -1 as it cannot represent the value as a bitmask. Writing value -1 to this parameter will have no effect on the hardware.

Table 7.5. Parameter properties of FG\_DIGIO\_OUTPUT

Property	Value
Name	<b>FG_DIGIO_OUTPUT</b>
Display Name	<b>Legacy Digital Output</b>
Type	<b>Signed Integer</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> -1 <b>Maximum</b> 3 <b>Stepsize</b> 1
Default value	<b>3</b>
Unit of measure	

Example 7.5. Usage of FG\_DIGIO\_OUTPUT

```

int result = 0;

```

```

int value = 3;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_DIGIO_OUTPUT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DIGIO_OUTPUT, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 7.3. GPI

### 7.3.1. FG\_DIGIO\_INPUT

Parameter *FG\_DIGIO\_INPUT* 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 *FG\_DIGIO\_INPUT*. 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 it's digital inputs 0, 2 and 5.

Table 7.6. Parameter properties of FG\_DIGIO\_INPUT

Property	Value
Name	<b>FG_DIGIO_INPUT</b>
Display Name	<b>Digital Input</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 4095</b> <b>Stepsize 1</b>

Unit of measure

Example 7.6. Usage of FG\_DIGIO\_INPUT

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_DIGIO_INPUT, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 7.4. Event Source

### 7.4.1. FG\_CUSTOM\_SIGNAL\_EVENT\_0\_SOURCE

FIXME\_ParameterDocumentation\_Missing\_FG\_CUSTOM\_SIGNAL\_EVENT\_0\_SOURCE

Table 7.7. Parameter properties of FG\_CUSTOM\_SIGNAL\_EVENT\_0\_SOURCE

Property	Value																																														
Name	<b>FG_CUSTOM_SIGNAL_EVENT_0_SOURCE</b>																																														
Display Name	<b>Custom Signal Event 0 Source</b>																																														
Type	<b>Enumeration</b>																																														
Access policy	<b>Read/Write/Change</b>																																														
Storage policy	<b>Persistent</b>																																														
Allowed values	<table border="0"> <tbody> <tr><td><b>GND</b></td><td>Gnd</td></tr> <tr><td><b>VCC</b></td><td>Vcc</td></tr> <tr><td><b>FG_SIGNAL_CAM0_EXSYNC</b></td><td>Cam0 Exsync</td></tr> <tr><td><b>FG_SIGNAL_CAM0_EXSYNC2</b></td><td>Cam0 Exsync2</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FLASH</b></td><td>Cam0 Flash</td></tr> <tr><td><b>FG_SIGNAL_CAM0_LVAL</b></td><td>Cam0 LVAL</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FVAL</b></td><td>Cam0 FVAL</td></tr> <tr><td><b>FG_SIGNAL_CAM0_LINE_START</b></td><td>Cam0 Line Transfer Start</td></tr> <tr><td><b>FG_SIGNAL_CAM0_LINE_END</b></td><td>Cam0 Line Transfer End</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FRAME_START</b></td><td>Cam0 Frame Transfer Start</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FRAME_END</b></td><td>Cam0 Frame Transfer End</td></tr> <tr><td><b>FG_SIGNAL_GPI_0</b></td><td>GPI 0</td></tr> <tr><td><b>FG_SIGNAL_GPI_1</b></td><td>GPI 1</td></tr> <tr><td><b>FG_SIGNAL_GPI_2</b></td><td>GPI 2</td></tr> <tr><td><b>FG_SIGNAL_GPI_3</b></td><td>GPI 3</td></tr> <tr><td><b>FG_SIGNAL_GPI_4</b></td><td>GPI 4</td></tr> <tr><td><b>FG_SIGNAL_GPI_5</b></td><td>GPI 5</td></tr> <tr><td><b>FG_SIGNAL_GPI_6</b></td><td>GPI 6</td></tr> <tr><td><b>FG_SIGNAL_GPI_7</b></td><td>GPI 7</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_0</b></td><td>Front GPI 0</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_1</b></td><td>Front GPI 1</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_2</b></td><td>Front GPI 2</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_3</b></td><td>Front GPI 3</td></tr> </tbody> </table>	<b>GND</b>	Gnd	<b>VCC</b>	Vcc	<b>FG_SIGNAL_CAM0_EXSYNC</b>	Cam0 Exsync	<b>FG_SIGNAL_CAM0_EXSYNC2</b>	Cam0 Exsync2	<b>FG_SIGNAL_CAM0_FLASH</b>	Cam0 Flash	<b>FG_SIGNAL_CAM0_LVAL</b>	Cam0 LVAL	<b>FG_SIGNAL_CAM0_FVAL</b>	Cam0 FVAL	<b>FG_SIGNAL_CAM0_LINE_START</b>	Cam0 Line Transfer Start	<b>FG_SIGNAL_CAM0_LINE_END</b>	Cam0 Line Transfer End	<b>FG_SIGNAL_CAM0_FRAME_START</b>	Cam0 Frame Transfer Start	<b>FG_SIGNAL_CAM0_FRAME_END</b>	Cam0 Frame Transfer End	<b>FG_SIGNAL_GPI_0</b>	GPI 0	<b>FG_SIGNAL_GPI_1</b>	GPI 1	<b>FG_SIGNAL_GPI_2</b>	GPI 2	<b>FG_SIGNAL_GPI_3</b>	GPI 3	<b>FG_SIGNAL_GPI_4</b>	GPI 4	<b>FG_SIGNAL_GPI_5</b>	GPI 5	<b>FG_SIGNAL_GPI_6</b>	GPI 6	<b>FG_SIGNAL_GPI_7</b>	GPI 7	<b>FG_SIGNAL_FRONT_GPI_0</b>	Front GPI 0	<b>FG_SIGNAL_FRONT_GPI_1</b>	Front GPI 1	<b>FG_SIGNAL_FRONT_GPI_2</b>	Front GPI 2	<b>FG_SIGNAL_FRONT_GPI_3</b>	Front GPI 3
<b>GND</b>	Gnd																																														
<b>VCC</b>	Vcc																																														
<b>FG_SIGNAL_CAM0_EXSYNC</b>	Cam0 Exsync																																														
<b>FG_SIGNAL_CAM0_EXSYNC2</b>	Cam0 Exsync2																																														
<b>FG_SIGNAL_CAM0_FLASH</b>	Cam0 Flash																																														
<b>FG_SIGNAL_CAM0_LVAL</b>	Cam0 LVAL																																														
<b>FG_SIGNAL_CAM0_FVAL</b>	Cam0 FVAL																																														
<b>FG_SIGNAL_CAM0_LINE_START</b>	Cam0 Line Transfer Start																																														
<b>FG_SIGNAL_CAM0_LINE_END</b>	Cam0 Line Transfer End																																														
<b>FG_SIGNAL_CAM0_FRAME_START</b>	Cam0 Frame Transfer Start																																														
<b>FG_SIGNAL_CAM0_FRAME_END</b>	Cam0 Frame Transfer End																																														
<b>FG_SIGNAL_GPI_0</b>	GPI 0																																														
<b>FG_SIGNAL_GPI_1</b>	GPI 1																																														
<b>FG_SIGNAL_GPI_2</b>	GPI 2																																														
<b>FG_SIGNAL_GPI_3</b>	GPI 3																																														
<b>FG_SIGNAL_GPI_4</b>	GPI 4																																														
<b>FG_SIGNAL_GPI_5</b>	GPI 5																																														
<b>FG_SIGNAL_GPI_6</b>	GPI 6																																														
<b>FG_SIGNAL_GPI_7</b>	GPI 7																																														
<b>FG_SIGNAL_FRONT_GPI_0</b>	Front GPI 0																																														
<b>FG_SIGNAL_FRONT_GPI_1</b>	Front GPI 1																																														
<b>FG_SIGNAL_FRONT_GPI_2</b>	Front GPI 2																																														
<b>FG_SIGNAL_FRONT_GPI_3</b>	Front GPI 3																																														
Default value	<b>FG_SIGNAL_CAM0_EXSYNC</b>																																														

**Example 7.7. Usage of FG\_CUSTOM\_SIGNAL\_EVENT\_0\_SOURCE**

```

int result = 0;
int value = FG_SIGNAL_CAM0_EXSYNC;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

```

**7.4.2. FG\_CUSTOM\_SIGNAL\_EVENT\_0\_POLARITY**

FIXME\_ParameterDocumentation\_Missing\_FG\_CUSTOM\_SIGNAL\_EVENT\_0\_POLARITY

Table 7.8. Parameter properties of FG\_CUSTOM\_SIGNAL\_EVENT\_0\_POLARITY

Property	Value
Name	<b>FG_CUSTOM_SIGNAL_EVENT_0_POLARITY</b>
Display Name	<b>Custom Signal Event 0 Polarity</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_LOW</b> Low Active <b>FG_HIGH</b> High Active
Default value	<b>FG_HIGH</b>

Example 7.8. Usage of FG\_CUSTOM\_SIGNAL\_EVENT\_0\_POLARITY

```

int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_0_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_0_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 7.4.3. FG\_CUSTOM\_SIGNAL\_EVENT\_1\_SOURCE

FIXME\_ParameterDocumentation\_Missing\_FG\_CUSTOM\_SIGNAL\_EVENT\_1\_SOURCE

Table 7.9. Parameter properties of FG\_CUSTOM\_SIGNAL\_EVENT\_1\_SOURCE

Property	Value																																														
Name	<b>FG_CUSTOM_SIGNAL_EVENT_1_SOURCE</b>																																														
Display Name	<b>Custom Signal Event 1 Source</b>																																														
Type	<b>Enumeration</b>																																														
Access policy	<b>Read/Write/Change</b>																																														
Storage policy	<b>Persistent</b>																																														
Allowed values	<table border="0"> <tbody> <tr><td><b>GND</b></td><td>Gnd</td></tr> <tr><td><b>VCC</b></td><td>Vcc</td></tr> <tr><td><b>FG_SIGNAL_CAM0_EXSYNC</b></td><td>Cam0 Exsync</td></tr> <tr><td><b>FG_SIGNAL_CAM0_EXSYNC2</b></td><td>Cam0 Exsync2</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FLASH</b></td><td>Cam0 Flash</td></tr> <tr><td><b>FG_SIGNAL_CAM0_LVAL</b></td><td>Cam0 LVAL</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FVAL</b></td><td>Cam0 FVAL</td></tr> <tr><td><b>FG_SIGNAL_CAM0_LINE_START</b></td><td>Cam0 Line Transfer Start</td></tr> <tr><td><b>FG_SIGNAL_CAM0_LINE_END</b></td><td>Cam0 Line Transfer End</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FRAME_START</b></td><td>Cam0 Frame Transfer Start</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FRAME_END</b></td><td>Cam0 Frame Transfer End</td></tr> <tr><td><b>FG_SIGNAL_GPI_0</b></td><td>GPI 0</td></tr> <tr><td><b>FG_SIGNAL_GPI_1</b></td><td>GPI 1</td></tr> <tr><td><b>FG_SIGNAL_GPI_2</b></td><td>GPI 2</td></tr> <tr><td><b>FG_SIGNAL_GPI_3</b></td><td>GPI 3</td></tr> <tr><td><b>FG_SIGNAL_GPI_4</b></td><td>GPI 4</td></tr> <tr><td><b>FG_SIGNAL_GPI_5</b></td><td>GPI 5</td></tr> <tr><td><b>FG_SIGNAL_GPI_6</b></td><td>GPI 6</td></tr> <tr><td><b>FG_SIGNAL_GPI_7</b></td><td>GPI 7</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_0</b></td><td>Front GPI 0</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_1</b></td><td>Front GPI 1</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_2</b></td><td>Front GPI 2</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_3</b></td><td>Front GPI 3</td></tr> </tbody> </table>	<b>GND</b>	Gnd	<b>VCC</b>	Vcc	<b>FG_SIGNAL_CAM0_EXSYNC</b>	Cam0 Exsync	<b>FG_SIGNAL_CAM0_EXSYNC2</b>	Cam0 Exsync2	<b>FG_SIGNAL_CAM0_FLASH</b>	Cam0 Flash	<b>FG_SIGNAL_CAM0_LVAL</b>	Cam0 LVAL	<b>FG_SIGNAL_CAM0_FVAL</b>	Cam0 FVAL	<b>FG_SIGNAL_CAM0_LINE_START</b>	Cam0 Line Transfer Start	<b>FG_SIGNAL_CAM0_LINE_END</b>	Cam0 Line Transfer End	<b>FG_SIGNAL_CAM0_FRAME_START</b>	Cam0 Frame Transfer Start	<b>FG_SIGNAL_CAM0_FRAME_END</b>	Cam0 Frame Transfer End	<b>FG_SIGNAL_GPI_0</b>	GPI 0	<b>FG_SIGNAL_GPI_1</b>	GPI 1	<b>FG_SIGNAL_GPI_2</b>	GPI 2	<b>FG_SIGNAL_GPI_3</b>	GPI 3	<b>FG_SIGNAL_GPI_4</b>	GPI 4	<b>FG_SIGNAL_GPI_5</b>	GPI 5	<b>FG_SIGNAL_GPI_6</b>	GPI 6	<b>FG_SIGNAL_GPI_7</b>	GPI 7	<b>FG_SIGNAL_FRONT_GPI_0</b>	Front GPI 0	<b>FG_SIGNAL_FRONT_GPI_1</b>	Front GPI 1	<b>FG_SIGNAL_FRONT_GPI_2</b>	Front GPI 2	<b>FG_SIGNAL_FRONT_GPI_3</b>	Front GPI 3
<b>GND</b>	Gnd																																														
<b>VCC</b>	Vcc																																														
<b>FG_SIGNAL_CAM0_EXSYNC</b>	Cam0 Exsync																																														
<b>FG_SIGNAL_CAM0_EXSYNC2</b>	Cam0 Exsync2																																														
<b>FG_SIGNAL_CAM0_FLASH</b>	Cam0 Flash																																														
<b>FG_SIGNAL_CAM0_LVAL</b>	Cam0 LVAL																																														
<b>FG_SIGNAL_CAM0_FVAL</b>	Cam0 FVAL																																														
<b>FG_SIGNAL_CAM0_LINE_START</b>	Cam0 Line Transfer Start																																														
<b>FG_SIGNAL_CAM0_LINE_END</b>	Cam0 Line Transfer End																																														
<b>FG_SIGNAL_CAM0_FRAME_START</b>	Cam0 Frame Transfer Start																																														
<b>FG_SIGNAL_CAM0_FRAME_END</b>	Cam0 Frame Transfer End																																														
<b>FG_SIGNAL_GPI_0</b>	GPI 0																																														
<b>FG_SIGNAL_GPI_1</b>	GPI 1																																														
<b>FG_SIGNAL_GPI_2</b>	GPI 2																																														
<b>FG_SIGNAL_GPI_3</b>	GPI 3																																														
<b>FG_SIGNAL_GPI_4</b>	GPI 4																																														
<b>FG_SIGNAL_GPI_5</b>	GPI 5																																														
<b>FG_SIGNAL_GPI_6</b>	GPI 6																																														
<b>FG_SIGNAL_GPI_7</b>	GPI 7																																														
<b>FG_SIGNAL_FRONT_GPI_0</b>	Front GPI 0																																														
<b>FG_SIGNAL_FRONT_GPI_1</b>	Front GPI 1																																														
<b>FG_SIGNAL_FRONT_GPI_2</b>	Front GPI 2																																														
<b>FG_SIGNAL_FRONT_GPI_3</b>	Front GPI 3																																														
Default value	<b>FG_SIGNAL_CAM0_FLASH</b>																																														

Example 7.9. Usage of FG\_CUSTOM\_SIGNAL\_EVENT\_1\_SOURCE

```

int result = 0;
int value = FG_SIGNAL_CAM0_FLASH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_1_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_1_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

```

#### 7.4.4. FG\_CUSTOM\_SIGNAL\_EVENT\_1\_POLARITY

FIXME\_ParameterDocumentation\_Missing\_FG\_CUSTOM\_SIGNAL\_EVENT\_1\_POLARITY

Table 7.10. Parameter properties of FG\_CUSTOM\_SIGNAL\_EVENT\_1\_POLARITY

Property	Value
Name	FG_CUSTOM_SIGNAL_EVENT_1_POLARITY
Display Name	Custom Signal Event 1 Polarity
Type	Enumeration
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	FG_LOW Low Active FG_HIGH High Active
Default value	FG_HIGH

Example 7.10. Usage of FG\_CUSTOM\_SIGNAL\_EVENT\_1\_POLARITY

```

int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_1_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_1_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 7.5. 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.

### 7.5.1. FG\_TRIGGER\_INPUT0\_RISING

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.

### 7.5.2. FG\_TRIGGER\_INPUT0\_FALLING

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.

### 7.5.3. FG\_CUSTOM\_SIGNAL\_EVENT\_0

doc\_node\_missing!

### 7.5.4. FG\_CUSTOM\_SIGNAL\_EVENT\_1



doc\_node\_missing!

---

# Chapter 8. Line Trigger / ExSync

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 *FG\_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 *FG\_LINETRIGGERMODE*.

For the mapping of the ExSync signals to the digital outputs check Chapter 7, 'Digital I/O'.

## 8.1. FG\_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  $90^\circ$  is achieved when setting phase to 4, which produces a camera trigger at times  $16 * n + 4$  of the input trigger signal.

Table 8.1. Parameter properties of FG\_LINETRIGGERMODE

Property	Value	
Name	<b>FG_LINETRIGGERMODE</b>	
Display Name	<b>Line Trigger Mode</b>	
Type	<b>Enumeration</b>	
Access policy	<b>Read/Write</b>	
Storage policy	<b>Persistent</b>	
Allowed values	<b>GRABBER_CONTROLLED</b>	Grabber controlled
	<b>ASYNC_TRIGGER</b>	Async External Trigger
	<b>GRABBER_CONTROLLED_GATED</b>	Gated, Grabber controlled
	<b>ASYNC_GATED</b>	Gated, External Triggered
Default value	<b>GRABBER_CONTROLLED</b>	

Example 8.1. Usage of FG\_LINETRIGGERMODE

```
int result = 0;
int value = GRABBER_CONTROLLED;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_LINETRIGGERMODE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINETRIGGERMODE, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 8.2. FG\_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 (**FG\_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 (**FG\_ON**) or Off (**FG\_OFF**). By switching this parameter On (**FG\_ON**) and Off (**FG\_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 8.2. Parameter properties of FG\_EXSYNCON

Property	Value
Name	FG_EXSYNCON
Display Name	Exsync Enable
Type	Enumeration
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	FG_ON On FG_OFF Off
Default value	FG_ON

Example 8.2. Usage of FG\_EXSYNCON

```

int result = 0;
int value = FG_ON;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_EXSYNCON, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_EXSYNCON, &value, 0, type)) < 0) {
    /* error handling */
}

```

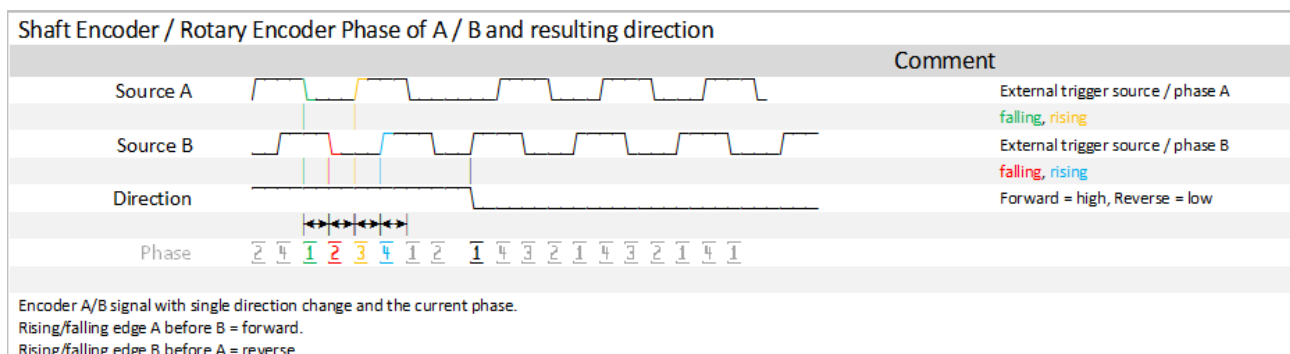
### 8.3. Line Trigger Input

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.

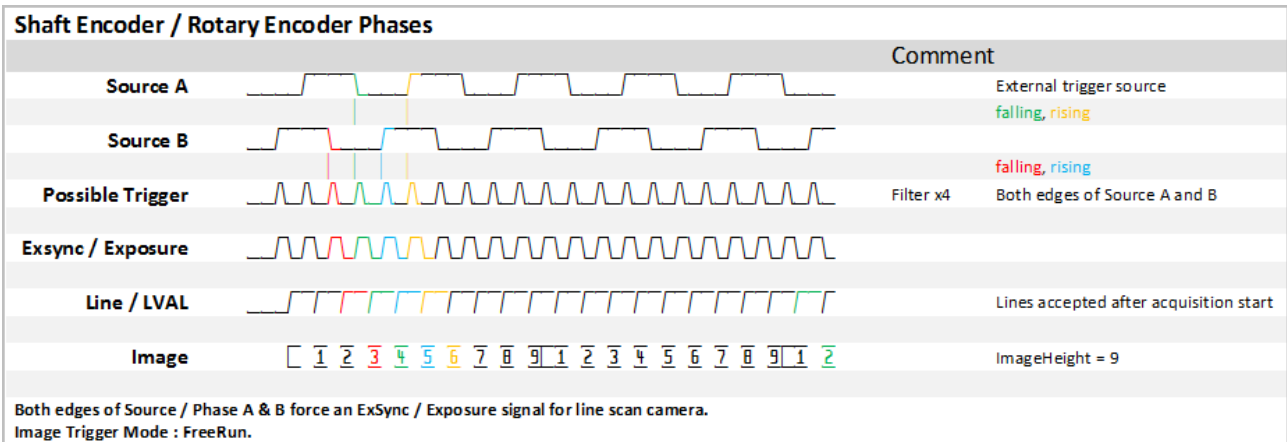
Most incremental shaft encoders provide 2 signals that are called A & B. By using these two signals the relative increments can be seen at the edges of these signals and a direction. In one direction the A-phase high state rises before the B-phase in the other direction, i.e. vice versa. If we do not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

Figure 8.1. Shaft Encoder, A &amp; 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 transferred. The time between exposure and transfer is for most line scan cameras very short.

Figure 8.2. 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 8.3. 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 9, 'Image Trigger / Flash'.

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.

### 8.3.1. FG\_LINETRIGGERINSRC

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 `FG_SHAFTENCODERINSRC`, 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 8.3, 'Line Trigger Input'.

Table 8.4. Parameter properties of FG\_LINETRIGGERINSRC

Property	Value																								
Name	<b>FG_LINETRIGGERINSRC</b>																								
Display Name	<b>Source A</b>																								
Type	<b>Enumeration</b>																								
Access policy	<b>Read/Write/Change</b>																								
Storage policy	<b>Persistent</b>																								
Allowed values	<table border="0"> <tr><td>TRGINSRC_GPI_0</td><td>GPI Trigger Source 0</td></tr> <tr><td>TRGINSRC_GPI_1</td><td>GPI Trigger Source 1</td></tr> <tr><td>TRGINSRC_GPI_2</td><td>GPI Trigger Source 2</td></tr> <tr><td>TRGINSRC_GPI_3</td><td>GPI Trigger Source 3</td></tr> <tr><td>TRGINSRC_GPI_4</td><td>GPI Trigger Source 4 (not available on LightBridge)</td></tr> <tr><td>TRGINSRC_GPI_5</td><td>GPI Trigger Source 5 (not available on LightBridge)</td></tr> <tr><td>TRGINSRC_GPI_6</td><td>GPI Trigger Source 6 (not available on LightBridge)</td></tr> <tr><td>TRGINSRC_GPI_7</td><td>GPI Trigger Source 7 (not available on LightBridge)</td></tr> <tr><td>TRGINSRC_FRONT_GPI_0</td><td>Front GPI Trigger Source 0</td></tr> <tr><td>TRGINSRC_FRONT_GPI_1</td><td>Front GPI Trigger Source 1</td></tr> <tr><td>TRGINSRC_FRONT_GPI_2</td><td>Front GPI Trigger Source 2</td></tr> <tr><td>TRGINSRC_FRONT_GPI_3</td><td>Front GPI Trigger Source 3</td></tr> </table>	TRGINSRC_GPI_0	GPI Trigger Source 0	TRGINSRC_GPI_1	GPI Trigger Source 1	TRGINSRC_GPI_2	GPI Trigger Source 2	TRGINSRC_GPI_3	GPI Trigger Source 3	TRGINSRC_GPI_4	GPI Trigger Source 4 (not available on LightBridge)	TRGINSRC_GPI_5	GPI Trigger Source 5 (not available on LightBridge)	TRGINSRC_GPI_6	GPI Trigger Source 6 (not available on LightBridge)	TRGINSRC_GPI_7	GPI Trigger Source 7 (not available on LightBridge)	TRGINSRC_FRONT_GPI_0	Front GPI Trigger Source 0	TRGINSRC_FRONT_GPI_1	Front GPI Trigger Source 1	TRGINSRC_FRONT_GPI_2	Front GPI Trigger Source 2	TRGINSRC_FRONT_GPI_3	Front GPI Trigger Source 3
TRGINSRC_GPI_0	GPI Trigger Source 0																								
TRGINSRC_GPI_1	GPI Trigger Source 1																								
TRGINSRC_GPI_2	GPI Trigger Source 2																								
TRGINSRC_GPI_3	GPI Trigger Source 3																								
TRGINSRC_GPI_4	GPI Trigger Source 4 (not available on LightBridge)																								
TRGINSRC_GPI_5	GPI Trigger Source 5 (not available on LightBridge)																								
TRGINSRC_GPI_6	GPI Trigger Source 6 (not available on LightBridge)																								
TRGINSRC_GPI_7	GPI Trigger Source 7 (not available on LightBridge)																								
TRGINSRC_FRONT_GPI_0	Front GPI Trigger Source 0																								
TRGINSRC_FRONT_GPI_1	Front GPI Trigger Source 1																								
TRGINSRC_FRONT_GPI_2	Front GPI Trigger Source 2																								
TRGINSRC_FRONT_GPI_3	Front GPI Trigger Source 3																								
Default value	<b>TRGINSRC_GPI_1</b>																								

Example 8.3. Usage of FG\_LINETRIGGERINSRC

```

int result = 0;
int value = TRGINSRC_GPI_1;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_LINETRIGGERINSRC, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINETRIGGERINSRC, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 8.3.2. FG\_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 *FG\_LINETRIGGERINSRC*. Otherwise, the ExSync generation starts on a rising edge. This is only relevant if the *FG\_LINETRIGGERMODE* is set to an external trigger.

Table 8.5. Parameter properties of FG\_LINETRIGGERINPOLARITY

Property	Value				
Name	<b>FG_LINETRIGGERINPOLARITY</b>				
Display Name	<b>Input Polarity</b>				
Type	<b>Enumeration</b>				
Access policy	<b>Read/Write/Change</b>				
Storage policy	<b>Persistent</b>				
Allowed values	<table border="0"> <tr><td>HIGH_ON_ZERO_LOW</td><td>Low Active</td></tr> <tr><td>HIGH_ON_ZERO_HIGH</td><td>High Active</td></tr> </table>	HIGH_ON_ZERO_LOW	Low Active	HIGH_ON_ZERO_HIGH	High Active
HIGH_ON_ZERO_LOW	Low Active				
HIGH_ON_ZERO_HIGH	High Active				
Default value	<b>HIGH_ACTIVE</b>				

**Example 8.4. Usage of FG\_LINETRIGGERINPOLARITY**

```

int result = 0;
int value = HIGH_ACTIVE;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_LINETRIGGERINPOLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINETRIGGERINPOLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 8.3.3. FG\_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 8.6. Parameter properties of FG\_LINETRIGGERDEBOUNCING**

Property	Value
Name	<b>FG_LINETRIGGERDEBOUNCING</b>
Display Name	<b>Debouncing Time</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0.008</b> <b>Maximum 65.0</b> <b>Stepsize 0.008</b>
Default value	<b>0.112</b>
Unit of measure	<b>µs</b>

**Example 8.5. Usage of FG\_LINETRIGGERDEBOUNCING**

```

int result = 0;
double value = 0.112;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_LINETRIGGERDEBOUNCING, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINETRIGGERDEBOUNCING, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 8.3.4. Downscale

#### 8.3.4.1. FG\_LINE\_DOWNSCALE

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 *FG\_LINETRIGGERMODE* is set to an external trigger mode. The parameter *FG\_LINE\_DOWNSCALEINIT* selects an initial delay of incoming pulses.

Figure 8.3. Downscale and Init phase behaviour

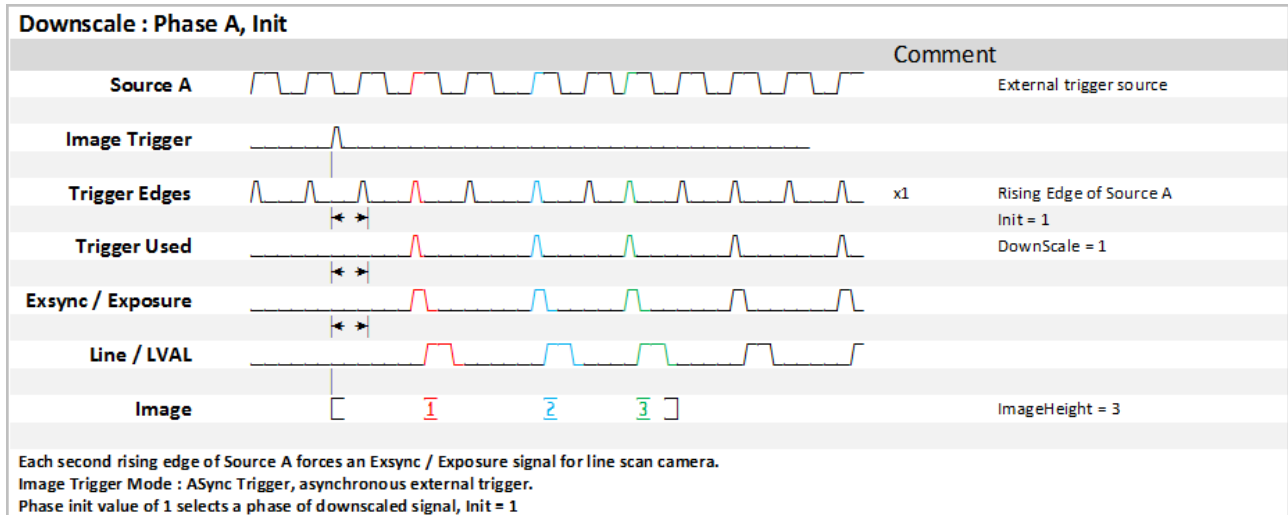


Table 8.7. Parameter properties of FG\_LINE\_DOWNSCALE

Property	Value
Name	FG_LINE_DOWNSCALE
Display Name	Downscale
Type	Unsigned Integer
Access policy	Read/Write
Storage policy	Persistent
Allowed values	Minimum 1 Maximum 255 Stepsize 1
Default value	1
Unit of measure	pulses

Example 8.6. Usage of FG\_LINE\_DOWNSCALE

```

int result = 0;
unsigned int value = 1;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_LINE_DOWNSCALE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINE_DOWNSCALE, &value, 0, type)) < 0) {
    /* error handling */
}
    
```

### 8.3.4.2. FG\_LINE\_DOWNSCALEINIT

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 *FG\_LINETRIGGERMODE* is set to an image gate dependent ExSync mode. This value is applied after the image start pulse. The parameter *FG\_LINE\_DOWNSCALE* represents the number of possible steps and an explaining figure is found in its description (Init=1).



Table 8.8. Parameter properties of FG\_LINE\_DOWNSCALEINIT

Property	Value
Name	<b>FG_LINE_DOWNSCALEINIT</b>
Display Name	<b>Phase</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 1</b> <b>Maximum 255</b> <b>Stepsize 1</b>
Default value	<b>1</b>
Unit of measure	<b>pulses</b>

Example 8.7. Usage of FG\_LINE\_DOWNSCALEINIT

```

int result = 0;
unsigned int value = 1;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_LINE_DOWNSCALEINIT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINE_DOWNSCALEINIT, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 8.4. Shaft Encoder A/B Filter

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.

### 8.4.1. FG\_SHAFTENCODERON

Switch the shaft encoder filter On or Off. This is only relevant if the `FG_LINETRIGGERMODE` is set to an external trigger mode. The functionalities of `FG_SHAFTENCODERMODE`, `FG_SHAFTENCODERINSRC`, `FG_SHAFTENCODERLEADING`, `FG_SHAFTENCODER_COMPENSATION_ENABLE`, `FG_SHAFTENCODER_COMPENSATION_COUNT` become relevant in the case this parameter is set to On = **FG\_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 `FG_SHAFTENCODERINSRC` is interpreted as phase B and the input signal at `FG_LINETRIGGERINSRC` as phase A. A sketch of the signal can be found in the description of parameter `FG_LINETRIGGERINSRC`.

Table 8.9. Parameter properties of FG\_SHAFTENCODERON

Property	Value
Name	<b>FG_SHAFTENCODERON</b>
Display Name	<b>Enable</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_ON</b> On <b>FG_OFF</b> Off
Default value	<b>FG_OFF</b>

Example 8.8. Usage of FG\_SHAFTENCODERON

```

int result = 0;
int value = FG_OFF;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SHAFTENCODERON, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SHAFTENCODERON, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 8.4.2. FG\_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 *FG\_SHAFTENCODERON* is switched on. It enables you to adjust the number of increments per rotation of the shaft encoder. Together with the parameter *FG\_LINE\_DOWNSCALE* 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 8.4. Shaft Encoder Mode : Filter x4, x2, x1

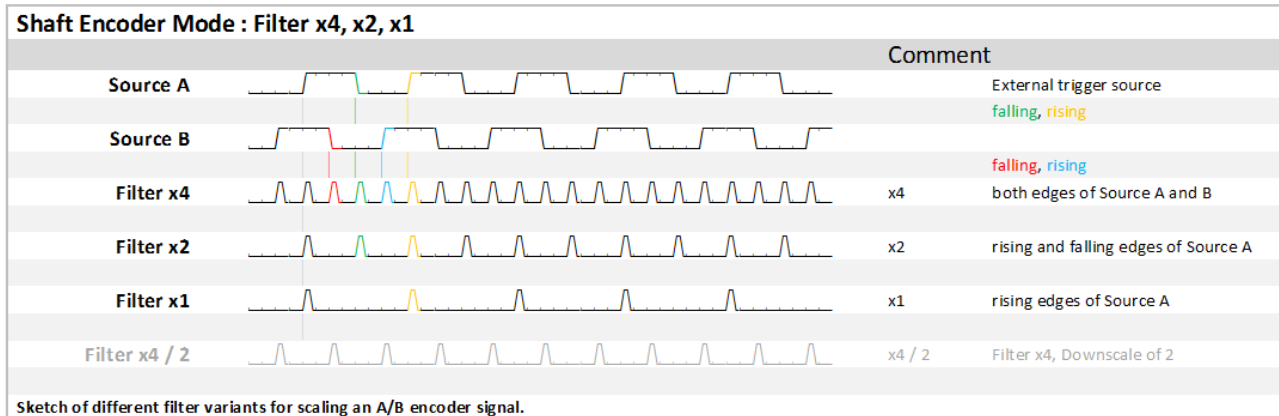


Table 8.10. Parameter properties of FG\_SHAFTENCODERMODE

Property	Value
Name	FG_SHAFTENCODERMODE
Display Name	Shaft Encoder Mode
Type	Enumeration
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	<b>FILTER_X1</b> Filter x1 <b>FILTER_X2</b> Filter x2 <b>FILTER_X4</b> Filter x4
Default value	<b>FILTER_X1</b>

Example 8.9. Usage of FG\_SHAFTENCODERMODE

```

int result = 0;
int value = FILTER_X1;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SHAFTENCODERMODE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SHAFTENCODERMODE, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 8.4.3. FG\_SHAFTENCODERINSRC

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 `FG_LINETRIGGERINSRC` and concerning the shaft encoder in the introduction of Section 8.3, 'Line Trigger Input'. Check the hardware documentation of the microEnable trigger board and the Framegrabber SDK manual for more details.

Table 8.11. Parameter properties of FG\_SHAFTENCODERINSRC

Property	Value
Name	<b>FG_SHAFTENCODERINSRC</b>
Display Name	<b>Source B</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>TRGINSRC_GPI_0</b> GPI Trigger Source 0 <b>TRGINSRC_GPI_1</b> GPI Trigger Source 1 <b>TRGINSRC_GPI_2</b> GPI Trigger Source 2 <b>TRGINSRC_GPI_3</b> GPI Trigger Source 3 <b>TRGINSRC_GPI_4</b> GPI Trigger Source 4 (not available on LightBridge) <b>TRGINSRC_GPI_5</b> GPI Trigger Source 5 (not available on LightBridge) <b>TRGINSRC_GPI_6</b> GPI Trigger Source 6 (not available on LightBridge) <b>TRGINSRC_GPI_7</b> GPI Trigger Source 7 (not available on LightBridge) <b>TRGINSRC_FRONT_GPI_0</b> Front GPI Trigger Source 0 <b>TRGINSRC_FRONT_GPI_1</b> Front GPI Trigger Source 1 <b>TRGINSRC_FRONT_GPI_2</b> Front GPI Trigger Source 2 <b>TRGINSRC_FRONT_GPI_3</b> Front GPI Trigger Source 3
Default value	<b>TRGINSRC_GPI_2</b>

Example 8.10. Usage of FG\_SHAFTENCODERINSRC

```

int result = 0;
int value = TRGINSRC_GPI_2;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SHAFTENCODERINSRC, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SHAFTENCODERINSRC, &value, 0, type)) < 0) {
    /* error handling */
}

```

#### 8.4.4. FG\_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 8.3, 'Line Trigger Input'.

Table 8.12. Parameter properties of FG\_SHAFTENCODERLEADING

Property	Value
Name	<b>FG_SHAFTENCODERLEADING</b>
Display Name	<b>Leading Signal</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>SOURCE_A</b> Source A <b>SOURCE_B</b> Source B
Default value	<b>SOURCE_A</b>

**Example 8.11. Usage of FG\_SHAFTENCODERLEADING**

```

int result = 0;
int value = SOURCE_A;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SHAFTENCODERLEADING, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SHAFTENCODERLEADING, &value, 0, type)) < 0) {
    /* error handling */
}

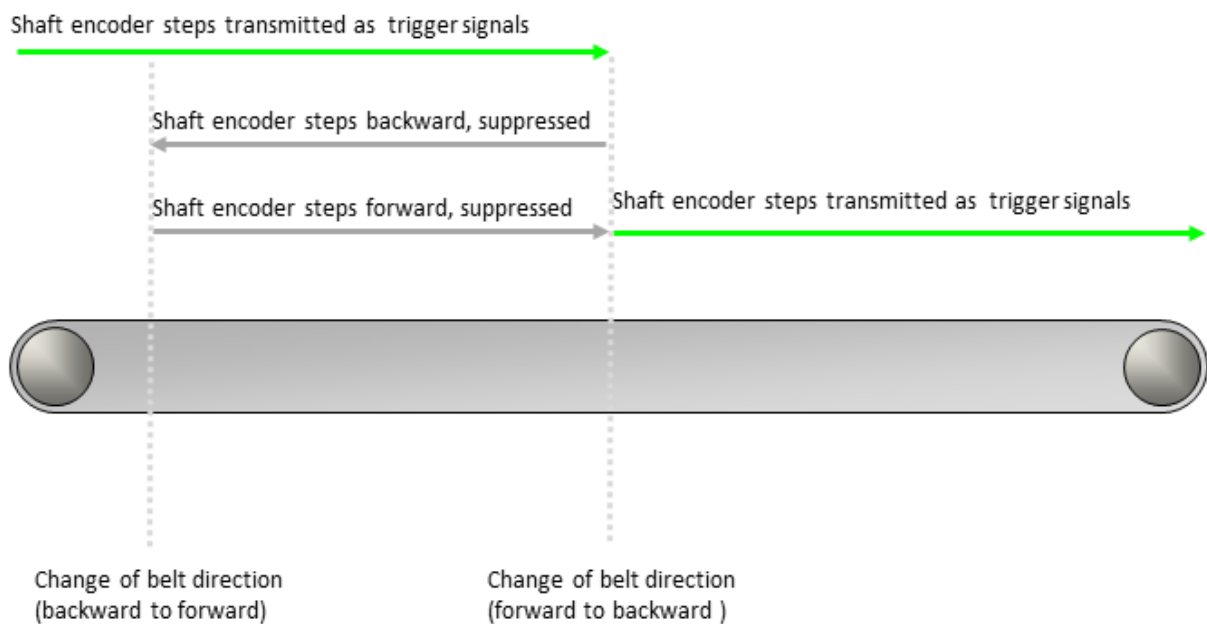
```

**8.4.5. FG\_SHAFTENCODER\_COMPENSATION\_ENABLE**

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 *FG\_SHAFTENCODER\_COMPENSATION\_ENABLE* switched ON:

Figure 8.5. 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 *FG\_SHAFTENCODER\_COMPENSATION\_ENABLE* switched OFF:

Figure 8.6. Shaft Encoder Compensation Enable = OFF

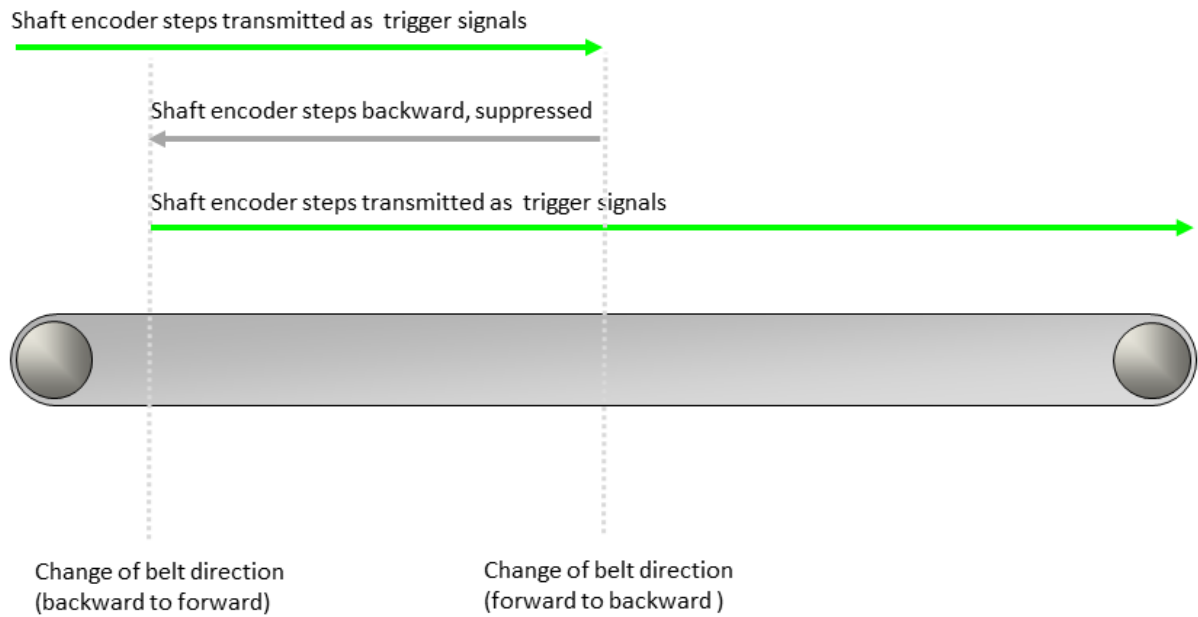


Table 8.13. Parameter properties of FG\_SHAFTENCODER\_COMPENSATION\_ENABLE

Property	Value
Name	<b>FG_SHAFTENCODER_COMPENSATION_ENABLE</b>
Display Name	<b>Compensation Enable</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_ON</b> On <b>FG_OFF</b> Off
Default value	<b>FG_ON</b>

Example 8.12. Usage of FG\_SHAFTENCODER\_COMPENSATION\_ENABLE

```

int result = 0;
int value = FG_ON;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SHAFTENCODER_COMPENSATION_ENABLE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SHAFTENCODER_COMPENSATION_ENABLE, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 8.4.6. FG\_SHAFTENCODER\_COMPENSATION\_COUNT

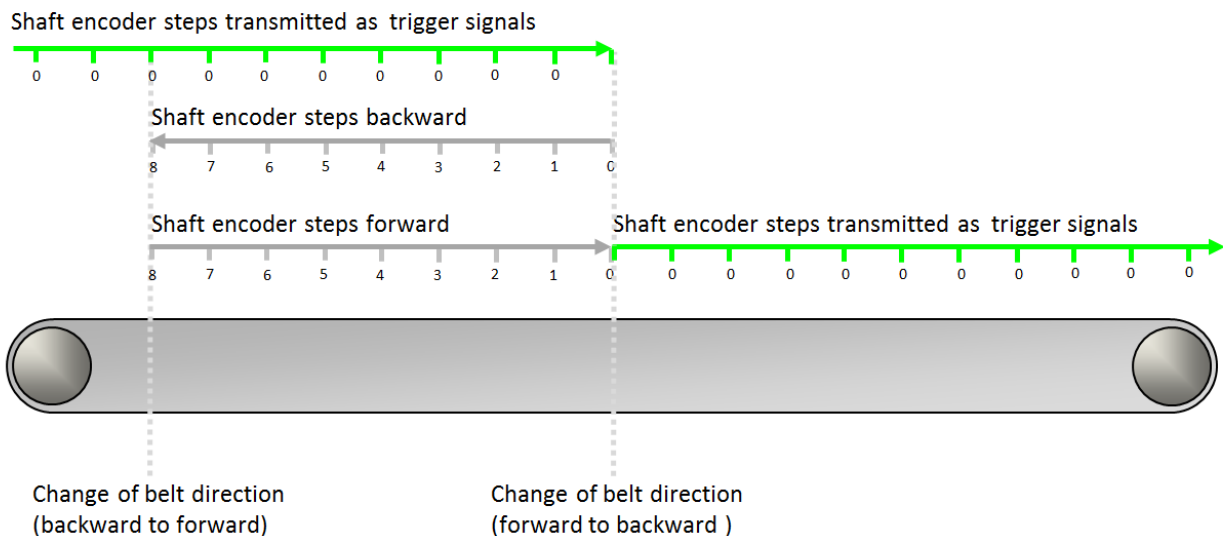
Using this parameter you can read and write the current shaft encoder rollback compensation counter. A compensation value zero indicates that currently no compensation is made. Therefore, you can reset the compensation by writing value zero to this parameter. Any other value will set a new compensation 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 8.3, 'Line Trigger Input'.

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 *FG\_SHAFTENCODER\_COMPENSATION\_ENABLE* 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 8.7. Shaft Encoder Compensation Enable = ON



The internal counter counts forwards as long as the transportation belt moves backwards. (In figure 8.7, from 0 to 8.)

The internal counter counts backwards while the transportation belt moves forwards. (In figure 8.7, 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 *FG\_SHAFTENCODER\_COMPENSATION\_COUNT*. Only if this value is 0, encoder steps are transmitted as trigger signals. If the value of parameter *FG\_SHAFTENCODER\_COMPENSATION\_COUNT* 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 *FG\_SHAFTENCODER\_COMPENSATION\_COUNT* 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.

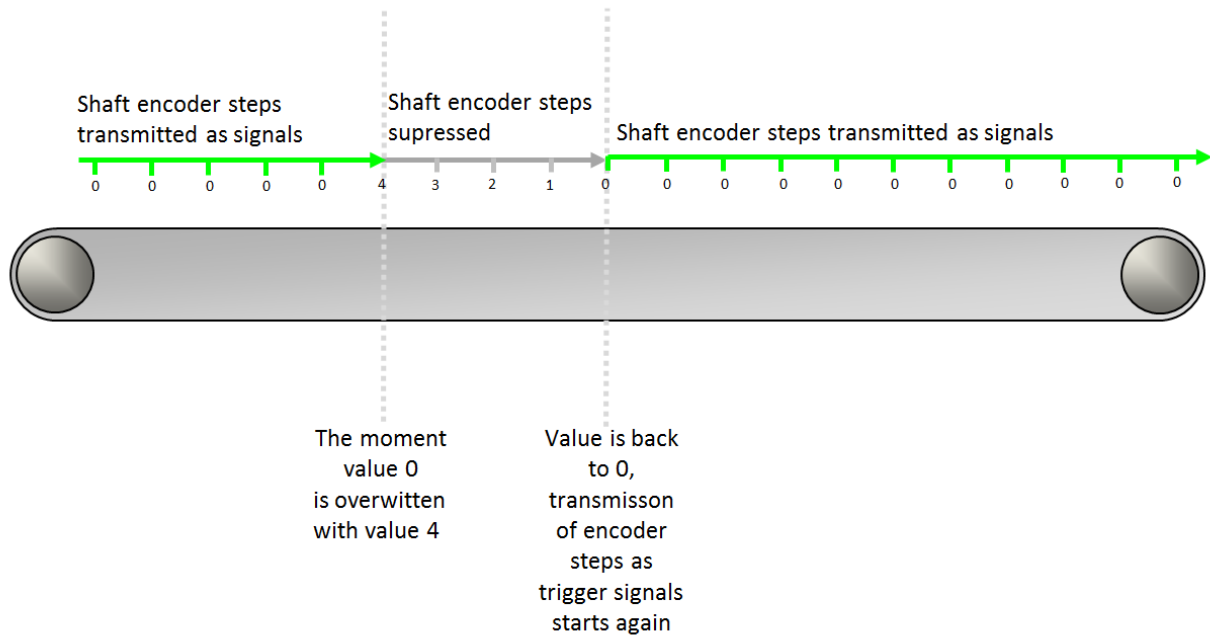
As soon as you enter a value for *FG\_SHAFTENCODER\_COMPENSATION\_COUNT*, this value overwrites the value the parameter holds before.

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

### Example 1:

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

Figure 8.8. Shaft Encoder Compensation Count Example 1

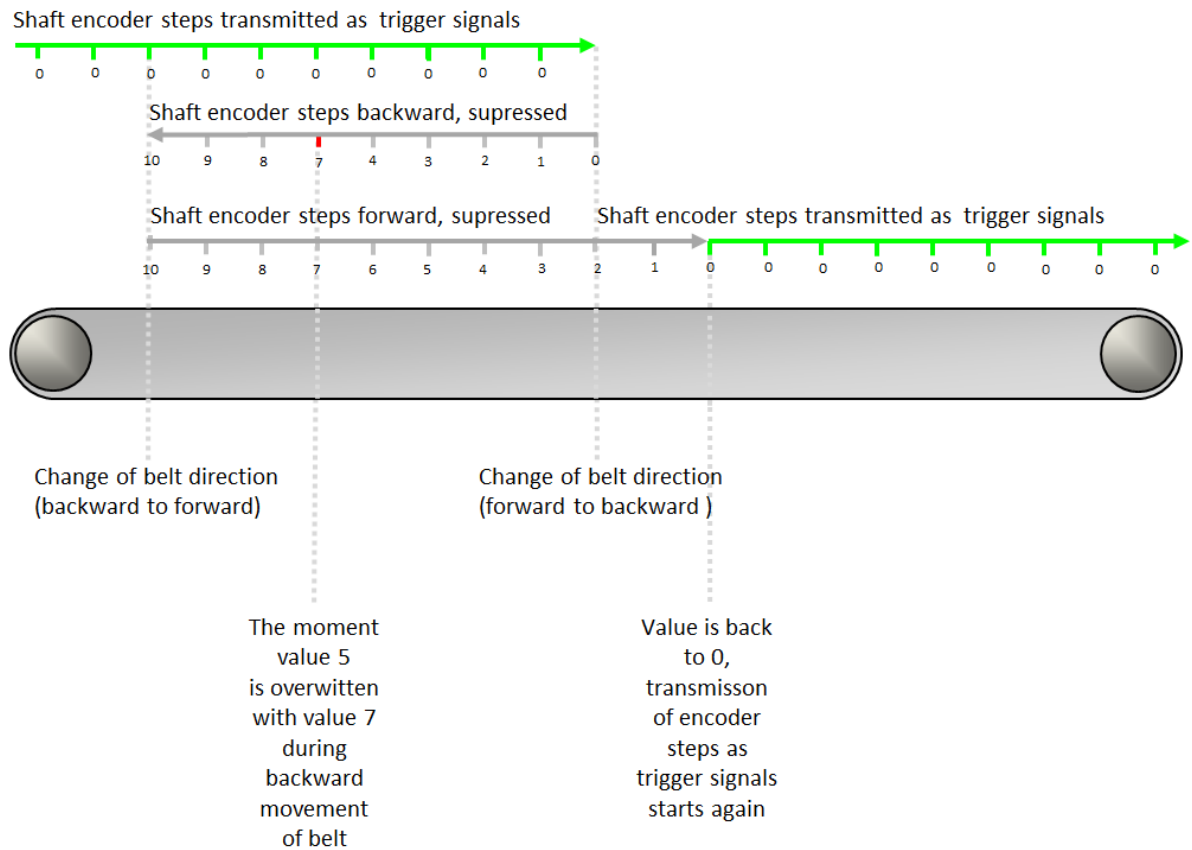


### Example 2:

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

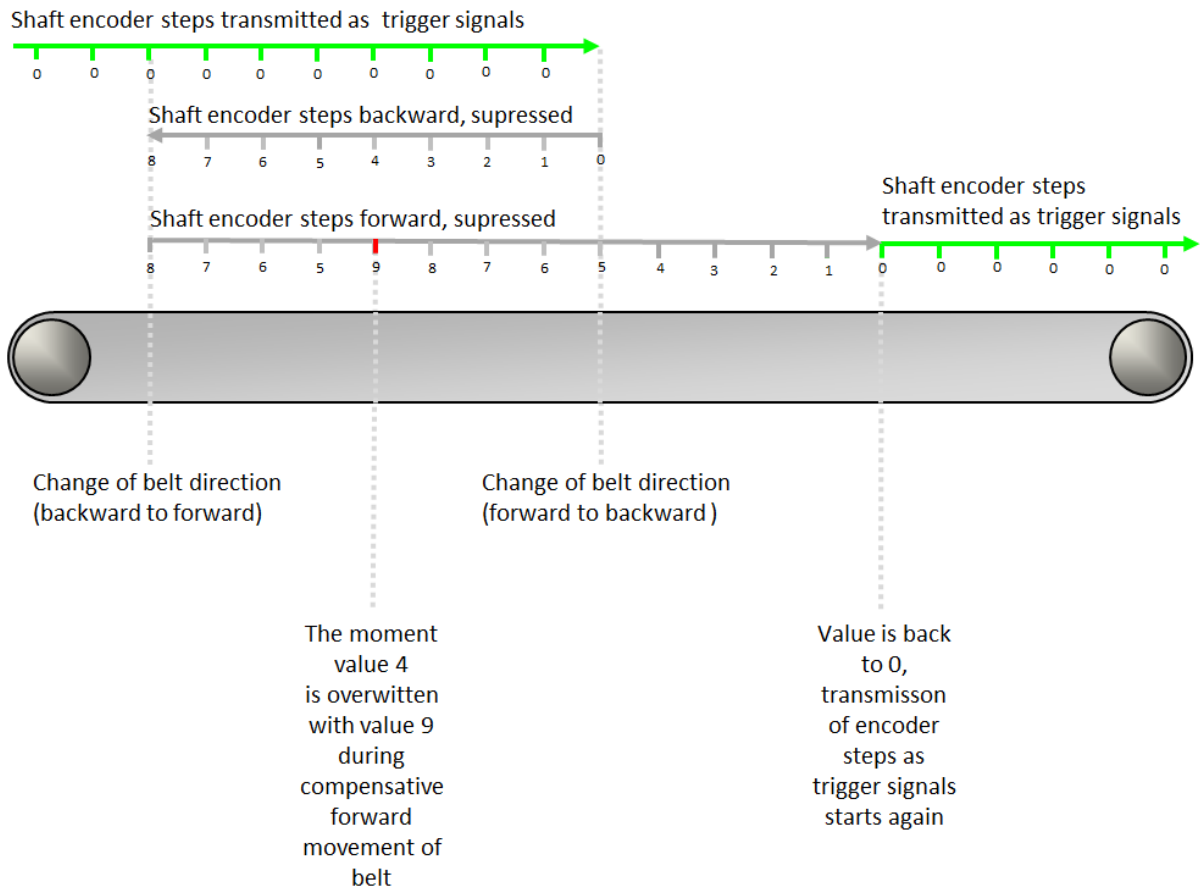


Figure 8.9. Shaft Encoder Compensation Count Example 2

**Example 3:**

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

Figure 8.10. 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 *FG\_SHAFTENCODER\_COMPENSATION\_COUNT* is not 0. Then, during compensative forward movement of the transportation belt, the value 4 of *FG\_SHAFTENCODER\_COMPENSATION\_COUNT* is overwritten with a smaller value, in our case with value 3. Result: Negative offset of -1 shaft encoder step.

Figure 8.11. Shaft Encoder Compensation Count Example 4

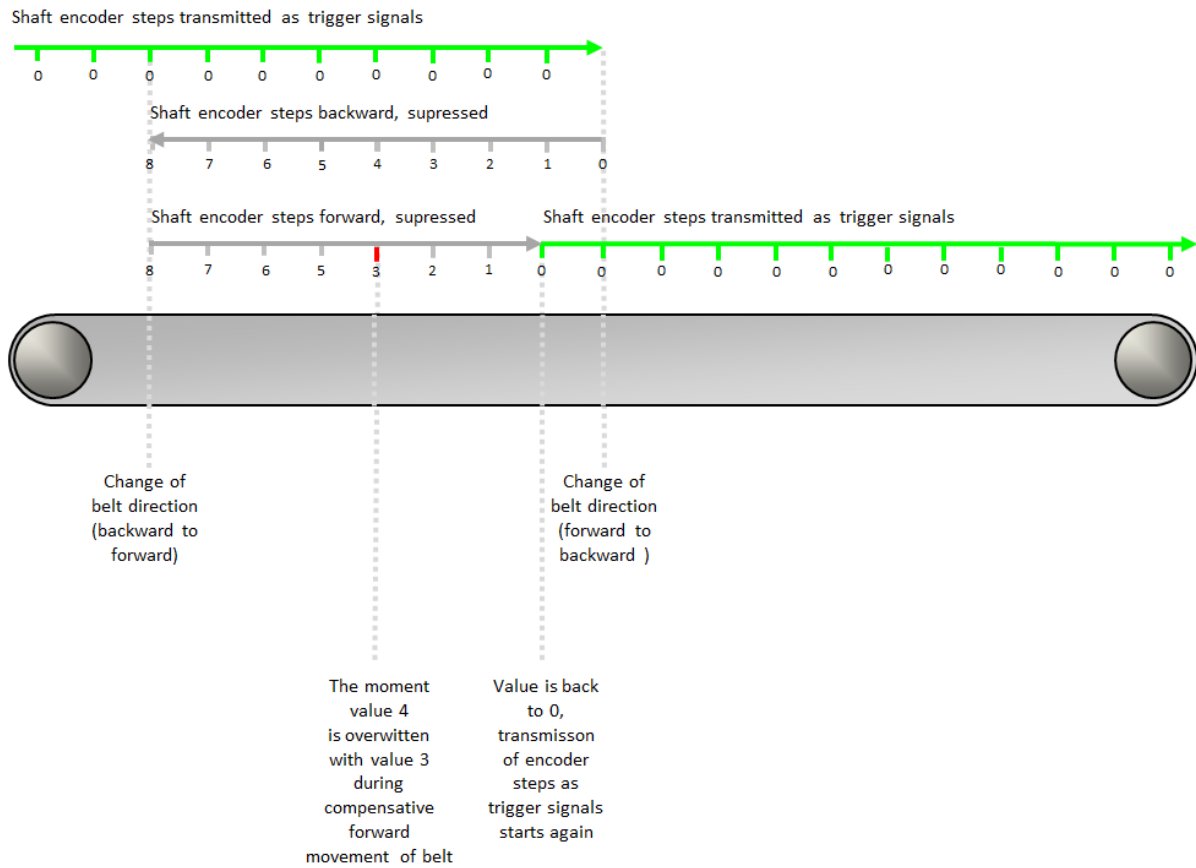


Table 8.14. Parameter properties of FG\_SHAFTENCODER\_COMPENSATION\_COUNT

Property	Value
Name	FG_SHAFTENCODER_COMPENSATION_COUNT
Display Name	Compensation Count
Type	Unsigned Integer
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	Minimum 0 Maximum 1048575 Stepsize 1
Default value	0
Unit of measure	pulses

Example 8.13. Usage of FG\_SHAFTENCODER\_COMPENSATION\_COUNT

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SHAFTENCODER_COMPENSATION_COUNT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SHAFTENCODER_COMPENSATION_COUNT, &value, 0, type)) < 0) {
    /* error handling */
}
    
```

## 8.5. ExSync Output

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

### 8.5.1. FG\_LINEPERIODE

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 *FG\_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 *FG\_LINEPERIODE* 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 8.1. Frequency to Period

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

Equation 8.2. 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 8.3. Velocity and Resolution to Period

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

Equation 8.4. Example:  $v = 53.4$  m/min,  $r = 4096$  pixels,  $w = 19.2$  cm Wide Web to Period

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

Table 8.15. Parameter properties of FG\_LINEPERIODE

Property	Value
Name	FG_LINEPERIODE
Display Name	Period
Type	Double
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	<b>Minimum 0.512</b> <b>Maximum 2097.144</b> <b>Stepsize 0.008</b>
Default value	200.0
Unit of measure	$\mu\text{s}$

Example 8.14. Usage of FG\_LINEPERIODE

```

int result = 0;
double value = 200.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_LINEPERIODE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINEPERIODE, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 8.5.2. FG\_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 *FG\_LINEPERIODE*. 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 *FG\_EXSYNCPOLARITY*.

Table 8.16. Parameter properties of FG\_LINEEXPOSURE

Property	Value
Name	<b>FG_LINEEXPOSURE</b>
Display Name	<b>Exposure Time</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0.512</b> <b>Maximum 1048.568</b> <b>Stepsize 0.008</b>
Default value	<b>19.0</b>
Unit of measure	<b>µs</b>

Example 8.15. Usage of FG\_LINEEXPOSURE

```

int result = 0;
double value = 19.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_LINEEXPOSURE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINEEXPOSURE, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 8.5.3. FG\_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 7, 'Digital I/O'.

Table 8.17. Parameter properties of FG\_EXSYNCPOLARITY

Property	Value
Name	<b>FG_EXSYNCPOLARITY</b>
Display Name	<b>Polarity</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_LOW</b> Low Active <b>FG_HIGH</b> High Active
Default value	<b>FG_HIGH</b>

Example 8.16. Usage of FG\_EXSYNCPOLARITY

```

int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_EXSYNCPOLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_EXSYNCPOLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 8.5.4. FG\_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 7, 'Digital I/O'.

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 8.18. Parameter properties of FG\_LINETRIGGERDELAY

Property	Value
Name	<b>FG_LINETRIGGERDELAY</b>
Display Name	<b>Delay-&gt;Exsync</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0.0</b> <b>Maximum 1048.568</b> <b>Stepsize 0.008</b>
Default value	<b>0.0</b>
Unit of measure	<b>µs</b>

Example 8.17. Usage of FG\_LINETRIGGERDELAY

```
int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_LINETRIGGERDELAY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINETRIGGERDELAY, &value, 0, type)) < 0) {
    /* error handling */
}
```

---

# Chapter 9. Image Trigger / Flash

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 *FG\_HEIGHT* of the display module. Also, offsets defined by *FG\_YOFFSET* 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 *FG\_HEIGHT* + *FG\_YOFFSET* 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 **ASYNC\_TRIGGER** mode, the gate is open for *FG\_IMGTRIGGER\_ASYNC\_HEIGHT* lines while this image is split into smaller chunks of *FG\_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 *FG\_YOFFSET* 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 *FG\_HEIGHT* + *FG\_YOFFSET* 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 *FG\_HEIGHT* + *FG\_YOFFSET* 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 *FG\_HEIGHT* lines. It will cut the image after *FG\_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 *FG\_HEIGHT*.

All images of a generated sequence will have a height of *FG\_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 `FG_YOFFSET` is not considered if the 'Gated Multi Buffer Trigger' mode is used. An y-offset cannot be set in the applet.

## 9.1. FG\_IMGTRIGGERMODE

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 9.1. Parameter properties of `FG_IMGTRIGGERMODE`

Property	Value										
Name	<b>FG_IMGTRIGGERMODE</b>										
Display Name	<b>Mode</b>										
Type	<b>Enumeration</b>										
Access policy	<b>Read/Write</b>										
Storage policy	<b>Persistent</b>										
Allowed values	<table border="0"> <tr> <td><b>FREE_RUN</b></td> <td>Free Run</td> </tr> <tr> <td><b>ASYNC_TRIGGER</b></td> <td>Async External Trigger</td> </tr> <tr> <td><b>ASYNC_TRIGGER_MULTIFRAME</b></td> <td>Async Multi Buffer External Trigger</td> </tr> <tr> <td><b>ASYNC_GATED</b></td> <td>Gated, External Triggered</td> </tr> <tr> <td><b>ASYNC_GATED_MULTIFRAME</b></td> <td>Gated Multi Buffer, External Triggered</td> </tr> </table>	<b>FREE_RUN</b>	Free Run	<b>ASYNC_TRIGGER</b>	Async External Trigger	<b>ASYNC_TRIGGER_MULTIFRAME</b>	Async Multi Buffer External Trigger	<b>ASYNC_GATED</b>	Gated, External Triggered	<b>ASYNC_GATED_MULTIFRAME</b>	Gated Multi Buffer, External Triggered
<b>FREE_RUN</b>	Free Run										
<b>ASYNC_TRIGGER</b>	Async External Trigger										
<b>ASYNC_TRIGGER_MULTIFRAME</b>	Async Multi Buffer External Trigger										
<b>ASYNC_GATED</b>	Gated, External Triggered										
<b>ASYNC_GATED_MULTIFRAME</b>	Gated Multi Buffer, External Triggered										
Default value	<b>FREE_RUN</b>										

Example 9.1. Usage of `FG_IMGTRIGGERMODE`

```
int result = 0;
int value = FREE_RUN;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGERMODE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGERMODE, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 9.2. FG\_IMGTRIGGERON

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 9.2. Parameter properties of `FG_IMGTRIGGERON`

Property	Value				
Name	<b>FG_IMGTRIGGERON</b>				
Display Name	<b>Image Trigger Enable</b>				
Type	<b>Enumeration</b>				
Access policy	<b>Read/Write/Change</b>				
Storage policy	<b>Persistent</b>				
Allowed values	<table border="0"> <tr> <td><b>FG_ON</b></td> <td>On</td> </tr> <tr> <td><b>FG_OFF</b></td> <td>Off</td> </tr> </table>	<b>FG_ON</b>	On	<b>FG_OFF</b>	Off
<b>FG_ON</b>	On				
<b>FG_OFF</b>	Off				
Default value	<b>FG_ON</b>				

**Example 9.2. Usage of FG\_IMGTRIGGERON**

```

int result = 0;
int value = FG_ON;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGERON, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGERON, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 9.3. FG\_FLASHON

To enable the flash output use this parameter.

For the mapping of the flash signal to the digital IO check Chapter 7, 'Digital I/O'.

Table 9.3. Parameter properties of FG\_FLASHON

Property	Value
Name	FG_FLASHON
Display Name	Flash Enable
Type	Enumeration
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	FG_ON On FG_OFF Off
Default value	FG_ON

**Example 9.3. Usage of FG\_FLASHON**

```

int result = 0;
int value = FG_ON;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_FLASHON, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_FLASHON, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 9.4. FG\_IMGTRIGGER\_ASYNC\_HEIGHT

This parameter only has influence in the image trigger mode *FG\_IMGTRIGGERMODE Async Trigger Multi Frame ASYNC\_TRIGGER\_MULTIFRAME*. The value is used to define the image height of the frame after the trigger pulse. Whereas parameter *FG\_HEIGHT* defines the chunk height.

If the value of *FG\_IMGTRIGGER\_ASYNC\_HEIGHT* is less than *FG\_HEIGHT*, the frame is not split into multiple frames and will result in a smaller output frame.

Table 9.4. Parameter properties of FG\_IMGTRIGGER\_ASYNC\_HEIGHT

Property	Value
Name	<b>FG_IMGTRIGGER_ASYNC_HEIGHT</b>
Display Name	<b>Async Image Trigger Height</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 1</b> <b>Maximum 16777216</b> <b>Stepsize 1</b>
Default value	<b>1024</b>
Unit of measure	<b>lines</b>

Example 9.4. Usage of FG\_IMGTRIGGER\_ASYNC\_HEIGHT

```

int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGER_ASYNC_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGER_ASYNC_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 9.5. FG\_IMGTRIGGER\_IS\_BUSY

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 9.5. Parameter properties of FG\_IMGTRIGGER\_IS\_BUSY

Property	Value
Name	<b>FG_IMGTRIGGER_IS_BUSY</b>
Display Name	<b>Image Trigger is Busy</b>
Type	<b>Enumeration</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>IS_BUSY</b> Busy Flag is set <b>IS_NOT_BUSY</b> Busy Flag is not set

Example 9.5. Usage of FG\_IMGTRIGGER\_IS\_BUSY

```

int result = 0;
int value = IS_NOT_BUSY;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGER_IS_BUSY, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 9.6. Image Trigger Input

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.

### 9.6.1. FG\_IMGTRIGGERINSRC

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 **TRGINSOFTWARE**.

Table 9.6. Parameter properties of FG\_IMGTRIGGERINSRC

Property	Value																										
Name	<b>FG_IMGTRIGGERINSRC</b>																										
Display Name	<b>Source</b>																										
Type	<b>Enumeration</b>																										
Access policy	<b>Read/Write/Change</b>																										
Storage policy	<b>Persistent</b>																										
Allowed values	<table border="0"> <tr><td><b>TRGINSRC_GPI_0</b></td><td>GPI Trigger Source 0</td></tr> <tr><td><b>TRGINSRC_GPI_1</b></td><td>GPI Trigger Source 1</td></tr> <tr><td><b>TRGINSRC_GPI_2</b></td><td>GPI Trigger Source 2</td></tr> <tr><td><b>TRGINSRC_GPI_3</b></td><td>GPI Trigger Source 3</td></tr> <tr><td><b>TRGINSRC_GPI_4</b></td><td>GPI Trigger Source 4 (not available on LightBridge)</td></tr> <tr><td><b>TRGINSRC_GPI_5</b></td><td>GPI Trigger Source 5 (not available on LightBridge)</td></tr> <tr><td><b>TRGINSRC_GPI_6</b></td><td>GPI Trigger Source 6 (not available on LightBridge)</td></tr> <tr><td><b>TRGINSRC_GPI_7</b></td><td>GPI Trigger Source 7 (not available on LightBridge)</td></tr> <tr><td><b>TRGINSRC_FRONT_GPI_0</b></td><td>Front GPI Trigger Source 0</td></tr> <tr><td><b>TRGINSRC_FRONT_GPI_1</b></td><td>Front GPI Trigger Source 1</td></tr> <tr><td><b>TRGINSRC_FRONT_GPI_2</b></td><td>Front GPI Trigger Source 2</td></tr> <tr><td><b>TRGINSRC_FRONT_GPI_3</b></td><td>Front GPI Trigger Source 3</td></tr> <tr><td><b>TRGINSOFTWARE</b></td><td>Software Trigger</td></tr> </table>	<b>TRGINSRC_GPI_0</b>	GPI Trigger Source 0	<b>TRGINSRC_GPI_1</b>	GPI Trigger Source 1	<b>TRGINSRC_GPI_2</b>	GPI Trigger Source 2	<b>TRGINSRC_GPI_3</b>	GPI Trigger Source 3	<b>TRGINSRC_GPI_4</b>	GPI Trigger Source 4 (not available on LightBridge)	<b>TRGINSRC_GPI_5</b>	GPI Trigger Source 5 (not available on LightBridge)	<b>TRGINSRC_GPI_6</b>	GPI Trigger Source 6 (not available on LightBridge)	<b>TRGINSRC_GPI_7</b>	GPI Trigger Source 7 (not available on LightBridge)	<b>TRGINSRC_FRONT_GPI_0</b>	Front GPI Trigger Source 0	<b>TRGINSRC_FRONT_GPI_1</b>	Front GPI Trigger Source 1	<b>TRGINSRC_FRONT_GPI_2</b>	Front GPI Trigger Source 2	<b>TRGINSRC_FRONT_GPI_3</b>	Front GPI Trigger Source 3	<b>TRGINSOFTWARE</b>	Software Trigger
<b>TRGINSRC_GPI_0</b>	GPI Trigger Source 0																										
<b>TRGINSRC_GPI_1</b>	GPI Trigger Source 1																										
<b>TRGINSRC_GPI_2</b>	GPI Trigger Source 2																										
<b>TRGINSRC_GPI_3</b>	GPI Trigger Source 3																										
<b>TRGINSRC_GPI_4</b>	GPI Trigger Source 4 (not available on LightBridge)																										
<b>TRGINSRC_GPI_5</b>	GPI Trigger Source 5 (not available on LightBridge)																										
<b>TRGINSRC_GPI_6</b>	GPI Trigger Source 6 (not available on LightBridge)																										
<b>TRGINSRC_GPI_7</b>	GPI Trigger Source 7 (not available on LightBridge)																										
<b>TRGINSRC_FRONT_GPI_0</b>	Front GPI Trigger Source 0																										
<b>TRGINSRC_FRONT_GPI_1</b>	Front GPI Trigger Source 1																										
<b>TRGINSRC_FRONT_GPI_2</b>	Front GPI Trigger Source 2																										
<b>TRGINSRC_FRONT_GPI_3</b>	Front GPI Trigger Source 3																										
<b>TRGINSOFTWARE</b>	Software Trigger																										
Default value	<b>TRGINSRC_GPI_0</b>																										

Example 9.6. Usage of FG\_IMGTRIGGERINSRC

```

int result = 0;
int value = TRGINSRC_GPI_0;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGERINSRC, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGERINSRC, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 9.6.2. FG\_IMGTRIGGERINPOLARITY

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

Table 9.7. Parameter properties of FG\_IMGTRIGGERINPOLARITY

Property	Value				
Name	<b>FG_IMGTRIGGERINPOLARITY</b>				
Display Name	<b>Polarity</b>				
Type	<b>Enumeration</b>				
Access policy	<b>Read/Write/Change</b>				
Storage policy	<b>Persistent</b>				
Allowed values	<table border="0"> <tr><td><b>HIGH_ON_ZERO_LOW</b></td><td>Low Active</td></tr> <tr><td><b>HIGH_ON_ZERO_HIGH</b></td><td>High Active</td></tr> </table>	<b>HIGH_ON_ZERO_LOW</b>	Low Active	<b>HIGH_ON_ZERO_HIGH</b>	High Active
<b>HIGH_ON_ZERO_LOW</b>	Low Active				
<b>HIGH_ON_ZERO_HIGH</b>	High Active				
Default value	<b>HIGH_ACTIVE</b>				

**Example 9.7. Usage of FG\_IMGTRIGGERINPOLARITY**

```

int result = 0;
int value = HIGH_ACTIVE;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGERINPOLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGERINPOLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

```

**9.6.3. FG\_IMGTRIGGERGATEDELAY**

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 9.8. Parameter properties of FG\_IMGTRIGGERGATEDELAY

Property	Value
Name	<b>FG_IMGTRIGGERGATEDELAY</b>
Display Name	<b>Gate Delay</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 65535</b> <b>Stepsize 1</b>
Default value	<b>0</b>
Unit of measure	<b>lines</b>

**Example 9.8. Usage of FG\_IMGTRIGGERGATEDELAY**

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGERGATEDELAY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGERGATEDELAY, &value, 0, type)) < 0) {
    /* error handling */
}

```

**9.6.4. FG\_IMGTRIGGERDEBOUNCING**

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 9.9. Parameter properties of FG\_IMGTRIGGERDEBOUNCING

Property	Value
Name	<b>FG_IMGTRIGGERDEBOUNCING</b>
Display Name	<b>Debouncing Time</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0.008</b> <b>Maximum 65.0</b> <b>Stepsize 0.008</b>
Default value	<b>0.112</b>
Unit of measure	<b>µs</b>

Example 9.9. Usage of FG\_IMGTRIGGERDEBOUNCING

```

int result = 0;
double value = 0.112;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGERDEBOUNCING, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGERDEBOUNCING, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 9.6.5. FG\_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 9.10. Parameter properties of FG\_STROBEPULSEDELAY

Property	Value
Name	<b>FG_STROBEPULSEDELAY</b>
Display Name	<b>Flash Delay-&gt;Image Trigger</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 65535</b> <b>Stepsize 1</b>
Default value	<b>0</b>
Unit of measure	<b>lines</b>

Example 9.10. Usage of FG\_STROBEPULSEDELAY

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_STROBEPULSEDELAY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_STROBEPULSEDELAY, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 9.6.6. Flash

### 9.6.6.1. FG\_FLASH\_POLARITY

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 7, 'Digital I/O'.

Table 9.11. Parameter properties of FG\_FLASH\_POLARITY

Property	Value
Name	<b>FG_FLASH_POLARITY</b>
Display Name	<b>Flash Polarity</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_LOW</b> Low Active <b>FG_HIGH</b> High Active
Default value	<b>FG_HIGH</b>

Example 9.11. Usage of FG\_FLASH\_POLARITY

```
int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_FLASH_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_FLASH_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 9.6.7. Software Trigger

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 *FG\_IMGTRIGGERINSRC* to software trigger.

### 9.6.7.1. FG\_SENDSOFTWARETRIGGER

A software trigger pulse can be sent by use of this parameter.

Table 9.12. Parameter properties of FG\_SENDSOFTWARETRIGGER

Property	Value
Name	<b>FG_SENDSOFTWARETRIGGER</b>
Display Name	<b>Software Trigger Pulse</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Transient</b>
Allowed values	<b>FG_APPLY</b> Apply
Default value	<b>FG_APPLY</b>

**Example 9.12. Usage of FG\_SENDSOFTWARETRIGGER**

```

int result = 0;
int value = FG_APPLY;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SENDSOFTWARETRIGGER, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SENDSOFTWARETRIGGER, &value, 0, type)) < 0) {
    /* error handling */
}

```

**9.6.7.2. FG\_SETSOFTWARETRIGGER**

The software trigger state can be set to zero = low or one = high.

Table 9.13. Parameter properties of FG\_SETSOFTWARETRIGGER

Property	Value
Name	<b>FG_SETSOFTWARETRIGGER</b>
Display Name	<b>Software Trigger State</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_LOW</b> Low Active <b>FG_HIGH</b> High Active

Default value

**Example 9.13. Usage of FG\_SETSOFTWARETRIGGER**

```

int result = 0;
int value = FG_ZERO;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SETSOFTWARETRIGGER, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SETSOFTWARETRIGGER, &value, 0, type)) < 0) {
    /* error handling */
}

```



---

# Chapter 10. Signal Analyzer

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.

## 10.1. FG\_SIGNAL\_ANALYZER\_0\_SOURCE et al.



### Note

This description applies also to the following parameters: FG\_SIGNAL\_ANALYZER\_1\_SOURCE

Select the source signal for the trigger analyzer. For further explanation of the available sources see Chapter 7, 'Digital I/O'. In addition, the line/frame start/end pulses can be used as signal sources, too.

Table 10.1. Parameter properties of FG\_SIGNAL\_ANALYZER\_0\_SOURCE

Property	Value																																														
Name	<b>FG_SIGNAL_ANALYZER_0_SOURCE</b>																																														
Display Name	<b>Signal Analyzer 0 Source</b>																																														
Type	<b>Enumeration</b>																																														
Access policy	<b>Read/Write/Change</b>																																														
Storage policy	<b>Persistent</b>																																														
Allowed values	<table border="0"> <tbody> <tr><td><b>GND</b></td><td>Gnd</td></tr> <tr><td><b>VCC</b></td><td>Vcc</td></tr> <tr><td><b>FG_SIGNAL_CAM0_EXSYNC</b></td><td>Cam0 Exsync</td></tr> <tr><td><b>FG_SIGNAL_CAM0_EXSYNC2</b></td><td>Cam0 Exsync2</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FLASH</b></td><td>Cam0 Flash</td></tr> <tr><td><b>FG_SIGNAL_CAM0_LVAL</b></td><td>Cam0 LVAL</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FVAL</b></td><td>Cam0 FVAL</td></tr> <tr><td><b>FG_SIGNAL_CAM0_LINE_START</b></td><td>Cam0 Line Transfer Start</td></tr> <tr><td><b>FG_SIGNAL_CAM0_LINE_END</b></td><td>Cam0 Line Transfer End</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FRAME_START</b></td><td>Cam0 Frame Transfer Start</td></tr> <tr><td><b>FG_SIGNAL_CAM0_FRAME_END</b></td><td>Cam0 Frame Transfer End</td></tr> <tr><td><b>FG_SIGNAL_GPI_0</b></td><td>GPI 0</td></tr> <tr><td><b>FG_SIGNAL_GPI_1</b></td><td>GPI 1</td></tr> <tr><td><b>FG_SIGNAL_GPI_2</b></td><td>GPI 2</td></tr> <tr><td><b>FG_SIGNAL_GPI_3</b></td><td>GPI 3</td></tr> <tr><td><b>FG_SIGNAL_GPI_4</b></td><td>GPI 4</td></tr> <tr><td><b>FG_SIGNAL_GPI_5</b></td><td>GPI 5</td></tr> <tr><td><b>FG_SIGNAL_GPI_6</b></td><td>GPI 6</td></tr> <tr><td><b>FG_SIGNAL_GPI_7</b></td><td>GPI 7</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_0</b></td><td>Front GPI 0</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_1</b></td><td>Front GPI 1</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_2</b></td><td>Front GPI 2</td></tr> <tr><td><b>FG_SIGNAL_FRONT_GPI_3</b></td><td>Front GPI 3</td></tr> </tbody> </table>	<b>GND</b>	Gnd	<b>VCC</b>	Vcc	<b>FG_SIGNAL_CAM0_EXSYNC</b>	Cam0 Exsync	<b>FG_SIGNAL_CAM0_EXSYNC2</b>	Cam0 Exsync2	<b>FG_SIGNAL_CAM0_FLASH</b>	Cam0 Flash	<b>FG_SIGNAL_CAM0_LVAL</b>	Cam0 LVAL	<b>FG_SIGNAL_CAM0_FVAL</b>	Cam0 FVAL	<b>FG_SIGNAL_CAM0_LINE_START</b>	Cam0 Line Transfer Start	<b>FG_SIGNAL_CAM0_LINE_END</b>	Cam0 Line Transfer End	<b>FG_SIGNAL_CAM0_FRAME_START</b>	Cam0 Frame Transfer Start	<b>FG_SIGNAL_CAM0_FRAME_END</b>	Cam0 Frame Transfer End	<b>FG_SIGNAL_GPI_0</b>	GPI 0	<b>FG_SIGNAL_GPI_1</b>	GPI 1	<b>FG_SIGNAL_GPI_2</b>	GPI 2	<b>FG_SIGNAL_GPI_3</b>	GPI 3	<b>FG_SIGNAL_GPI_4</b>	GPI 4	<b>FG_SIGNAL_GPI_5</b>	GPI 5	<b>FG_SIGNAL_GPI_6</b>	GPI 6	<b>FG_SIGNAL_GPI_7</b>	GPI 7	<b>FG_SIGNAL_FRONT_GPI_0</b>	Front GPI 0	<b>FG_SIGNAL_FRONT_GPI_1</b>	Front GPI 1	<b>FG_SIGNAL_FRONT_GPI_2</b>	Front GPI 2	<b>FG_SIGNAL_FRONT_GPI_3</b>	Front GPI 3
<b>GND</b>	Gnd																																														
<b>VCC</b>	Vcc																																														
<b>FG_SIGNAL_CAM0_EXSYNC</b>	Cam0 Exsync																																														
<b>FG_SIGNAL_CAM0_EXSYNC2</b>	Cam0 Exsync2																																														
<b>FG_SIGNAL_CAM0_FLASH</b>	Cam0 Flash																																														
<b>FG_SIGNAL_CAM0_LVAL</b>	Cam0 LVAL																																														
<b>FG_SIGNAL_CAM0_FVAL</b>	Cam0 FVAL																																														
<b>FG_SIGNAL_CAM0_LINE_START</b>	Cam0 Line Transfer Start																																														
<b>FG_SIGNAL_CAM0_LINE_END</b>	Cam0 Line Transfer End																																														
<b>FG_SIGNAL_CAM0_FRAME_START</b>	Cam0 Frame Transfer Start																																														
<b>FG_SIGNAL_CAM0_FRAME_END</b>	Cam0 Frame Transfer End																																														
<b>FG_SIGNAL_GPI_0</b>	GPI 0																																														
<b>FG_SIGNAL_GPI_1</b>	GPI 1																																														
<b>FG_SIGNAL_GPI_2</b>	GPI 2																																														
<b>FG_SIGNAL_GPI_3</b>	GPI 3																																														
<b>FG_SIGNAL_GPI_4</b>	GPI 4																																														
<b>FG_SIGNAL_GPI_5</b>	GPI 5																																														
<b>FG_SIGNAL_GPI_6</b>	GPI 6																																														
<b>FG_SIGNAL_GPI_7</b>	GPI 7																																														
<b>FG_SIGNAL_FRONT_GPI_0</b>	Front GPI 0																																														
<b>FG_SIGNAL_FRONT_GPI_1</b>	Front GPI 1																																														
<b>FG_SIGNAL_FRONT_GPI_2</b>	Front GPI 2																																														
<b>FG_SIGNAL_FRONT_GPI_3</b>	Front GPI 3																																														
Default value	<b>FG_SIGNAL_CAM0_EXSYNC</b>																																														

Example 10.1. Usage of FG\_SIGNAL\_ANALYZER\_0\_SOURCE

```

int result = 0;
int value = FG_SIGNAL_CAM0_EXSYNC;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SIGNAL_ANALYZER_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 10.2. FG\_SIGNAL\_ANALYZER\_0\_POLARITY et al.



### Note

This description applies also to the following parameters: FG\_SIGNAL\_ANALYZER\_1\_POLARITY

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 10.2. Parameter properties of FG\_SIGNAL\_ANALYZER\_0\_POLARITY

Property	Value
Name	FG_SIGNAL_ANALYZER_0_POLARITY
Display Name	Signal Analyzer 0 Polarity
Type	Enumeration
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	FG_LOW Low Active FG_HIGH High Active
Default value	FG_HIGH

Example 10.2. Usage of FG\_SIGNAL\_ANALYZER\_0\_POLARITY

```

int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SIGNAL_ANALYZER_0_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_0_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 10.3. FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_CURRENT et al.



#### Note

This description applies also to the following parameters:  
FG\_SIGNAL\_ANALYZER\_1\_PERIOD\_CURRENT

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 10.3. Parameter properties of FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_CURRENT

Property	Value
Name	FG_SIGNAL_ANALYZER_0_PERIOD_CURRENT
Display Name	Signal Analyzer 0 Current Period
Type	Double
Access policy	Read-Only
Storage policy	Persistent
Allowed values	Minimum 0.008 Maximum 3.435973836E7 Stepsize 0.008
Unit of measure	µs

Example 10.3. Usage of FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_CURRENT

```

int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_0_PERIOD_CURRENT, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 10.4. FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_MAX et al.



### Note

This description applies also to the following parameters:  
FG\_SIGNAL\_ANALYZER\_1\_PERIOD\_MAX

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 10.4. Parameter properties of FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_MAX

Property	Value
Name	FG_SIGNAL_ANALYZER_0_PERIOD_MAX
Display Name	Signal Analyzer 0 Max Period
Type	Double
Access policy	Read-Only
Storage policy	Persistent
Allowed values	Minimum 0.008 Maximum 3.435973836E7 Stepsize 0.008
Unit of measure	µs

Example 10.4. Usage of FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_MAX

```
int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_0_PERIOD_MAX, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 10.5. FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_MIN et al.



### Note

This description applies also to the following parameters:  
FG\_SIGNAL\_ANALYZER\_1\_PERIOD\_MIN

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

Table 10.5. Parameter properties of FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_MIN

Property	Value
Name	FG_SIGNAL_ANALYZER_0_PERIOD_MIN
Display Name	Signal Analyzer 0 Min Period
Type	Double
Access policy	Read-Only
Storage policy	Persistent
Allowed values	Minimum 0.008 Maximum 3.435973836E7 Stepsize 0.008
Unit of measure	µs

**Example 10.5. Usage of FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_MIN**

```

int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_0_PERIOD_MIN, &value, 0, type)) < 0) {
    /* error handling */
}

```

**10.6. FG\_SIGNAL\_ANALYZER\_0\_PULSE\_COUNT et al.****Note**

This description applies also to the following parameters:  
FG\_SIGNAL\_ANALYZER\_1\_PULSE\_COUNT

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 10.6. Parameter properties of FG\_SIGNAL\_ANALYZER\_0\_PULSE\_COUNT

Property	Value
Name	FG_SIGNAL_ANALYZER_0_PULSE_COUNT
Display Name	Signal Analyzer 0 Pulse Count
Type	Unsigned Integer
Access policy	Read-Only
Storage policy	Persistent
Allowed values	Minimum 0 Maximum 4294967295 Stepsize 1
Unit of measure	µs

**Example 10.6. Usage of FG\_SIGNAL\_ANALYZER\_0\_PULSE\_COUNT**

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_0_PULSE_COUNT, &value, 0, type)) < 0) {
    /* error handling */
}

```

**10.7. FG\_SIGNAL\_ANALYZER\_PULSE\_COUNT\_DIFFERENCE**

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 exsync 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 10.7. Parameter properties of FG\_SIGNAL\_ANALYZER\_PULSE\_COUNT\_DIFFERENCE

Property	Value
Name	<b>FG_SIGNAL_ANALYZER_PULSE_COUNT_DIFFERENCE</b>
Display Name	<b>Signal Analyzer Pulse Count Difference</b>
Type	<b>Signed Integer (64 Bit)</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum -4294967296</b> <b>Maximum 4294967295</b> <b>Stepsize 1</b>
Unit of measure	<b>µs</b>

Example 10.7. Usage of FG\_SIGNAL\_ANALYZER\_PULSE\_COUNT\_DIFFERENCE

```

int result = 0;
int64_t value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_INT64_T;

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_PULSE_COUNT_DIFFERENCE, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 10.8. FG\_SIGNAL\_ANALYZER\_CLEAR

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 10.8. Parameter properties of FG\_SIGNAL\_ANALYZER\_CLEAR

Property	Value
Name	<b>FG_SIGNAL_ANALYZER_CLEAR</b>
Display Name	<b>Signal Analyzer Clear</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_APPLY</b> Apply
Default value	<b>FG_APPLY</b>

Example 10.8. Usage of FG\_SIGNAL\_ANALYZER\_CLEAR

```

int result = 0;
int value = FG_APPLY;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SIGNAL_ANALYZER_CLEAR, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_CLEAR, &value, 0, type)) < 0) {
    /* error handling */
}

```

# Chapter 11. Overflow

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 *FG\_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 transferred 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.

- 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.

A way to detect the overflows is to read parameter *FG\_OVERFLOW* or check for event *FG\_OVERFLOW\_CAM0*. 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.

## 11.1. FG\_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 11.1. Parameter properties of FG\_FILLLEVEL

Property	Value
Name	<b>FG_FILLLEVEL</b>
Display Name	<b>Buffer fill level</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 100</b> <b>Stepsize 1</b>
Unit of measure	<b>%</b>

Example 11.1. Usage of FG\_FILLLEVEL

```
int result = 0;
```

```

unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_FILLLEVEL, &value, 0, type)) < 0) {
    /* error handling */
}
    
```

## 11.2. FG\_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 11.2. Parameter properties of FG\_OVERFLOW

Property	Value
Name	<b>FG_OVERFLOW</b>
Display Name	<b>Buffer overflow</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 1</b> <b>Stepsize 1</b>

Example 11.2. Usage of FG\_OVERFLOW

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_OVERFLOW, &value, 0, type)) < 0) {
    /* error handling */
}
    
```

## 11.3. 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 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.

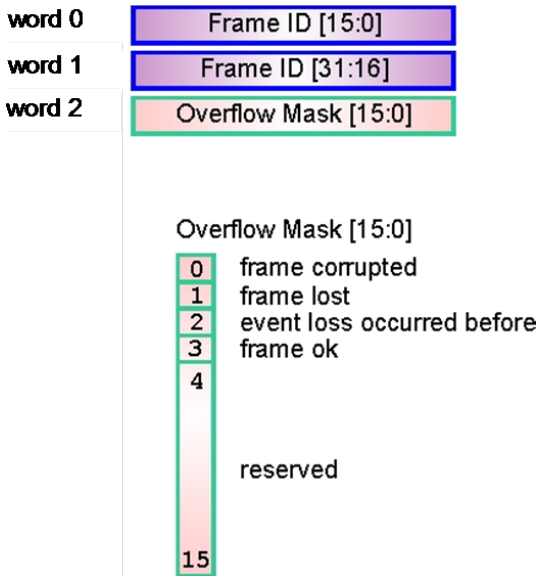
### 11.3.1. FG\_OVERFLOW\_CAM0

Overflow events are generated for each corrupted or lost frame. In contrast to the other events presented in this document, the overflow event transports data, namely the type of overflow, the image number and the



timestamp. The following figure illustrates the event data. Data is included in a 64 Bit data packet. The first 32 Bit include the frame number. Bits 32 to 47 include an overflow mask.

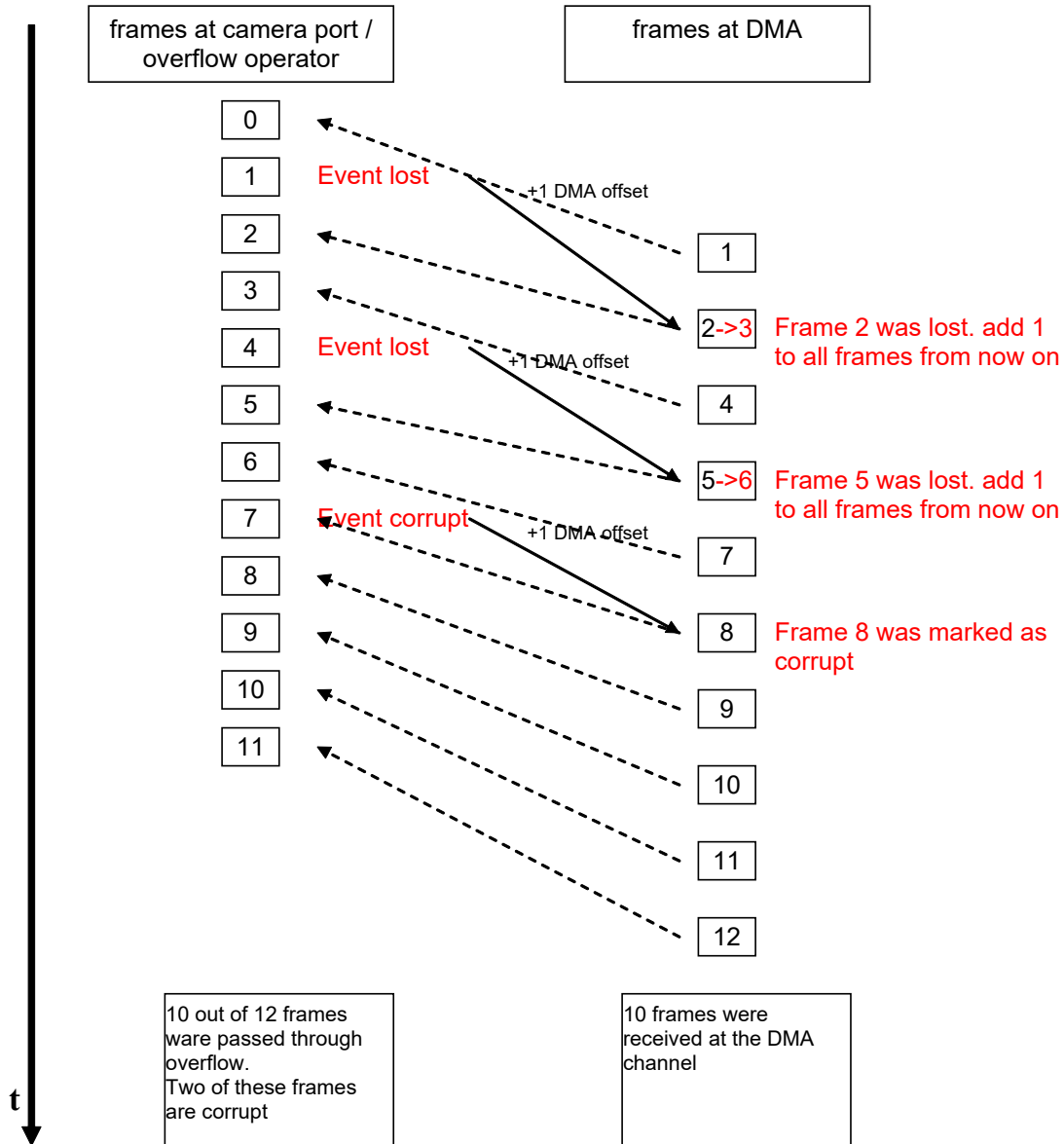
Figure 11.1. Illustration of Overflow Data Packet



Note that the frame number is reset on acquisition start. Also note that the first frame will have frame number zero, while a DMA transfer starts with frame number one. The frame number is a 32 Bit value. If it's maximum is reached, it will start from zero again. Keep in mind that on a 64 Bit runtime, the DMA transfer number will be a 64 Bit value. If the frame corrupted flag is set, the frame with the frame number in the event is corrupted i.e. it will not have it's full length but is still transfered via DMA channel. If the frame lost flag is set, the frame with the frame number in the event was fully discarded. No DMA transfer will exist for this frame. The corrupted frame flag and the frame lost flag will never occur for the same event. The flag "event loss occurred before" is an additional security mechanism. It means that an event has been lost. This can only happen at very high event rates and should not happen under normal conditions.

The analysis of the overflow events depends on the user requirements. In the following, an example is shown on how to ensure the integrity if the DMA data by analyzing the events and DMA transfers.

Figure 11.2. Analysis of Overflow Data



In the example, two frames got lost and one is marked as corrupted. As the events are not synchronous with the DMA transfers, for analysis a software queue (push and pull) is required to allocate the events to the DMA transfers.

# Chapter 12. Image Selector

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 `FG_IMG_SELECT_PERIOD` and `FG_IMG_SELECT`. 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:
  - `FG_IMG_SELECT_PERIOD = 2`  
`FG_IMG_SELECT = 0`
2. Grabber 1:
  - `FG_IMG_SELECT_PERIOD = 2`  
`FG_IMG_SELECT = 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.

## 12.1. FG\_IMG\_SELECT\_PERIOD

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 `FG_IMG_SELECT`.

Table 12.1. Parameter properties of `FG_IMG_SELECT_PERIOD`

Property	Value
Name	<code>FG_IMG_SELECT_PERIOD</code>
Display Name	<b>Image select period</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 1</b> <b>Maximum 256</b> <b>Stepsize 1</b>
Default value	<b>1</b>
Unit of measure	<b>image</b>

Example 12.1. Usage of `FG_IMG_SELECT_PERIOD`

```
int result = 0;
unsigned int value = 1;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMG_SELECT_PERIOD, &value, 0, type)) < 0) {
    /* error handling */
}
```

---

```

if ((result = Fg_getParameterWithType(fg, FG_IMG_SELECT_PERIOD, &value, 0, type)) < 0) {
    /* error handling */
}

```

---

## 12.2. FG\_IMG\_SELECT

The parameter *FG\_IMG\_SELECT* specifies a particular image from the image set defined by *FG\_IMG\_SELECT\_PERIOD*. 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 *FG\_IMG\_SELECT\_PERIOD*.

Table 12.2. Parameter properties of FG\_IMG\_SELECT

Property	Value
Name	<b>FG_IMG_SELECT</b>
Display Name	<b>Image select</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 255</b> <b>Stepsize 1</b>
Default value	<b>0</b>
Unit of measure	<b>image</b>

Example 12.2. Usage of FG\_IMG\_SELECT

---

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMG_SELECT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMG_SELECT, &value, 0, type)) < 0) {
    /* error handling */
}

```

---

# Chapter 13. White Balance

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.

## 13.1. FG\_SCALINGFACTOR\_RED

Table 13.1. Parameter properties of FG\_SCALINGFACTOR\_RED

Property	Value
Name	FG_SCALINGFACTOR_RED
Display Name	Red Gain
Type	Double
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	Minimum 0.0 Maximum 3.9990234375 Stepsize 9.765625E-4
Default value	1.0

Example 13.1. Usage of FG\_SCALINGFACTOR\_RED

```
int result = 0;
double value = 1.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_SCALINGFACTOR_RED, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SCALINGFACTOR_RED, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 13.2. FG\_SCALINGFACTOR\_BLUE

Table 13.2. Parameter properties of FG\_SCALINGFACTOR\_BLUE

Property	Value
Name	FG_SCALINGFACTOR_BLUE
Display Name	Blue Gain
Type	Double
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	Minimum 0.0 Maximum 3.9990234375 Stepsize 9.765625E-4
Default value	1.0

Example 13.2. Usage of FG\_SCALINGFACTOR\_BLUE

```
int result = 0;
double value = 1.0;
```

---

```

const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_SCALINGFACTOR_BLUE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SCALINGFACTOR_BLUE, &value, 0, type)) < 0) {
    /* error handling */
}

```

---

### 13.3. FG\_SCALINGFACTOR\_GREEN

Table 13.3. Parameter properties of FG\_SCALINGFACTOR\_GREEN

Property	Value
Name	<b>FG_SCALINGFACTOR_GREEN</b>
Display Name	<b>Green Gain</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0.0</b> <b>Maximum 3.9990234375</b> <b>Stepsize 9.765625E-4</b>
Default value	<b>1.0</b>

Example 13.3. Usage of FG\_SCALINGFACTOR\_GREEN

---

```

int result = 0;
double value = 1.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_SCALINGFACTOR_GREEN, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SCALINGFACTOR_GREEN, &value, 0, type)) < 0) {
    /* error handling */
}

```

---

# Chapter 14. Lookup Table

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 *FG\_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 *FG\_LUT\_TYPE* is important to be set correctly as it defines the lookup table operation mode.

## 14.1. FG\_LUT\_ENABLE

It is possible to disable the functionality of this lookup table. The internal 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 **FG\_ON** to use the look up table. By default it is set to **FG\_OFF** disabling the lookup table functionality itself and the related processing functions.

Table 14.1. Parameter properties of FG\_LUT\_ENABLE

Property	Value
Name	<b>FG_LUT_ENABLE</b>
Display Name	<b>Enabled</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_ON</b> On <b>FG_OFF</b> Off
Default value	<b>FG_OFF</b>

Example 14.1. Usage of FG\_LUT\_ENABLE

```
int result = 0;
int value = FG_OFF;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_LUT_ENABLE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LUT_ENABLE, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 14.2. FG\_LUT\_TYPE

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 **LUT\_TYPE\_PROCESSING** 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 **LUT\_TYPE\_CUSTOM** 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 **FG\_LUT\_VALUE\_RED**, **FG\_LUT\_VALUE\_GREEN** and **FG\_LUT\_VALUE\_BLUE**. 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 FG\_LUT\_TYPE

Property	Value
Name	<b>FG_LUT_TYPE</b>
Display Name	<b>Type</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>LUT_TYPE_PROCESSING</b> Processor <b>LUT_TYPE_CUSTOM</b> User file
Default value	<b>LUT_TYPE_PROCESSING</b>

Example 14.2. Usage of FG\_LUT\_TYPE

```
int result = 0;
int value = LUT_TYPE_PROCESSING;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_LUT_TYPE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LUT_TYPE, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 14.3. FG\_LUT\_VALUE\_RED

Table 14.3. Parameter properties of FG\_LUT\_VALUE\_RED

Property	Value
Name	<b>FG_LUT_VALUE_RED</b>
Display Name	<b>Red LUT Values</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>16384</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Transient</b>
Default value	<b>0</b>



**Example 14.3. Usage of FG\_LUT\_VALUE\_RED**

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

for (unsigned int i = 0; i < 16384; ++i)
{
    access.index = i;
    access.value = 0;

    if ((result = Fg_setParameterWithType(fg, FG_LUT_VALUE_RED, &access, 0, type)) < 0) {
        /* error handling */
    }

    if ((result = Fg_getParameterWithType(fg, FG_LUT_VALUE_RED, &access, 0, type)) < 0) {
        /* error handling */
    }
}

```

## 14.4. FG\_LUT\_VALUE\_GREEN

Table 14.4. Parameter properties of FG\_LUT\_VALUE\_GREEN

Property	Value
Name	FG_LUT_VALUE_GREEN
Display Name	Green LUT Values
Type	Unsigned Integer Field
Field Size	16384
Access policy	Read/Write/Change
Storage policy	Transient
Default value	0

**Example 14.4. Usage of FG\_LUT\_VALUE\_GREEN**

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

for (unsigned int i = 0; i < 16384; ++i)
{
    access.index = i;
    access.value = 0;

    if ((result = Fg_setParameterWithType(fg, FG_LUT_VALUE_GREEN, &access, 0, type)) < 0) {
        /* error handling */
    }

    if ((result = Fg_getParameterWithType(fg, FG_LUT_VALUE_GREEN, &access, 0, type)) < 0) {
        /* error handling */
    }
}

```

## 14.5. FG\_LUT\_VALUE\_BLUE

Table 14.5. Parameter properties of FG\_LUT\_VALUE\_BLUE

Property	Value
Name	FG_LUT_VALUE_BLUE
Display Name	Blue LUT Values
Type	Unsigned Integer Field
Field Size	16384
Access policy	Read/Write/Change
Storage policy	Transient
Default value	0

Example 14.5. Usage of FG\_LUT\_VALUE\_BLUE

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

for (unsigned int i = 0; i < 16384; ++i)
{
    access.index = i;
    access.value = 0;

    if ((result = Fg_setParameterWithType(fg, FG_LUT_VALUE_BLUE, &access, 0, type)) < 0) {
        /* error handling */
    }

    if ((result = Fg_getParameterWithType(fg, FG_LUT_VALUE_BLUE, &access, 0, type)) < 0) {
        /* error handling */
    }
}
```

## 14.6. FG\_LUT\_CUSTOM\_FILE

If parameter *FG\_LUT\_TYPE* is set to **LUT\_TYPE\_CUSTOM**, 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 *FG\_LUT\_SAVE\_FILE*.

Please make sure to activate the Type of LUT *FG\_LUT\_TYPE* to "UserFile"/**LUT\_TYPE\_CUSTOM** 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 14.6. Parameter properties of FG\_LUT\_CUSTOM\_FILE

Property	Value
Name	FG_LUT_CUSTOM_FILE
Display Name	Load File
Type	String
Access policy	Read/Write/Change
Storage policy	Persistent
Default value	""

**Example 14.6. Usage of FG\_LUT\_CUSTOM\_FILE**

```
int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_setParameterWithType(fg, FG_LUT_CUSTOM_FILE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LUT_CUSTOM_FILE, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 14.7. FG\_LUT\_SAVE\_FILE

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 *FG\_LUT\_CUSTOM\_FILE*.

Table 14.7. Parameter properties of FG\_LUT\_SAVE\_FILE

Property	Value
Name	<b>FG_LUT_SAVE_FILE</b>
Display Name	<b>Save File</b>
Type	<b>String</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Transient</b>
Default value	<b>""</b>

**Example 14.7. Usage of FG\_LUT\_SAVE\_FILE**

```
int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_setParameterWithType(fg, FG_LUT_SAVE_FILE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LUT_SAVE_FILE, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 14.8. Applet Properties

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

### 14.8.1. FG\_LUT\_IMPLEMENTATION\_TYPE

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 14.8. Parameter properties of FG\_LUT\_IMPLEMENTATION\_TYPE

Property	Value
Name	FG_LUT_IMPLEMENTATION_TYPE
Display Name	LUT Implementation Type
Type	Enumeration
Access policy	Read-Only
Storage policy	Transient
Allowed values	LUT_IMPLEMENTATION_FULL_LUT LUT_IMPLEMENTATION_KNEELUT

Example 14.8. Usage of FG\_LUT\_IMPLEMENTATION\_TYPE

```
int result = 0;
int value = LUT_IMPLEMENTATION_FULL_LUT;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_LUT_IMPLEMENTATION_TYPE, &value, 0, type)) < 0) {
    /* error handling */
}
```

### 14.8.2. FG\_LUT\_IN\_BITS

This applet is using 14 lookup table input bits.

Table 14.9. Parameter properties of FG\_LUT\_IN\_BITS

Property	Value
Name	FG_LUT_IN_BITS
Display Name	LUT In Bits
Type	Unsigned Integer
Access policy	Read-Only
Storage policy	Transient
Unit of measure	bit

Example 14.9. Usage of FG\_LUT\_IN\_BITS

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_LUT_IN_BITS, &value, 0, type)) < 0) {
    /* error handling */
}
```

### 14.8.3. FG\_LUT\_OUT\_BITS

This applet is using 16 lookup table output bits.

Table 14.10. Parameter properties of FG\_LUT\_OUT\_BITS

Property	Value
Name	<b>FG_LUT_OUT_BITS</b>
Display Name	<b>LUT Out Bits</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Unit of measure	<b>bit</b>

Example 14.10. Usage of FG\_LUT\_OUT\_BITS

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_LUT_OUT_BITS, &value, 0, type)) < 0) {
    /* error handling */
}

```

# 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 *FG\_LUT\_TYPE* of category lookup table to **LUT\_TYPE\_PROCESSING**. 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<sup>14</sup>], 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

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

If the inversion is used, output values are determined by

Equation 15.2. LUT Processor with Inversion

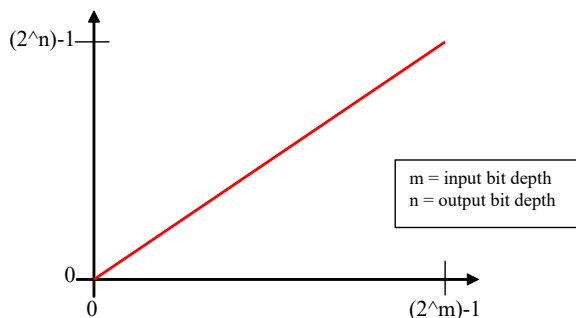
$$Output(x) = 2^{16} - 1 - \left[ \left[ gain * \left( \frac{x}{2^{14} - 1} + offset \right) \right]^{\frac{1}{gamma}} \right] * (2^{16} - 1),$$

where x represents the input pixel value i.e. is in the range from 0 to 2<sup>14</sup> - 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<sup>16</sup> - 1 it is set to 2<sup>16</sup> - 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. FG\_PROCESSING\_OFFSET

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 ( $0.5 \times 255 = 127.5$ ). 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

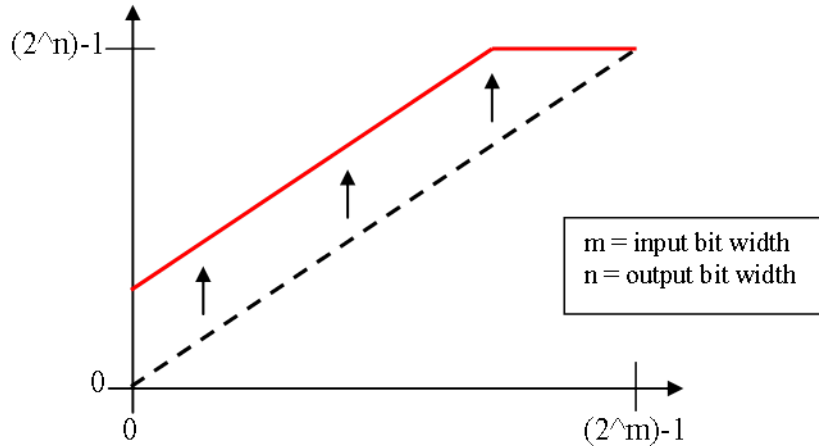


Table 15.1. Parameter properties of FG\_PROCESSING\_OFFSET

Property	Value
Name	FG_PROCESSING_OFFSET
Display Name	Offset
Type	Double
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	<b>Minimum</b> -1.0 <b>Maximum</b> 1.0 <b>Stepsize</b> 2.220446049250313E-16
Default value	0.0

Example 15.1. Usage of FG\_PROCESSING\_OFFSET

```

int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_PROCESSING_OFFSET, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_PROCESSING_OFFSET, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 15.2. FG\_PROCESSING\_GAIN

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

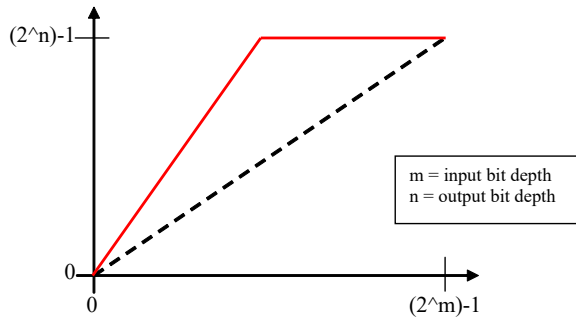


Table 15.2. Parameter properties of FG\_PROCESSING\_GAIN

Property	Value
Name	<b>FG_PROCESSING_GAIN</b>
Display Name	<b>Gain</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0.0</b> <b>Maximum 16384.0</b> <b>Stepsize 2.220446049250313E-16</b>
Default value	<b>1.0</b>

Example 15.2. Usage of FG\_PROCESSING\_GAIN

```

int result = 0;
double value = 1.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_PROCESSING_GAIN, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_PROCESSING_GAIN, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 15.3. FG\_PROCESSING\_GAMMA

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

Figure 15.4. Lookup Table Processing: Gamma

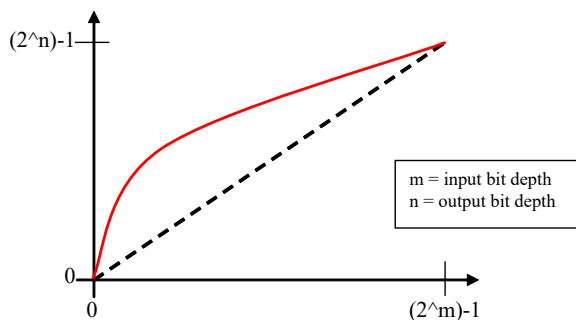


Table 15.3. Parameter properties of FG\_PROCESSING\_GAMMA

Property	Value
Name	<b>FG_PROCESSING_GAMMA</b>
Display Name	<b>Gamma</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum -1000.0</b> <b>Maximum 1000.0</b> <b>Stepsize 2.220446049250313E-16</b>
Default value	<b>1.0</b>

Example 15.3. Usage of FG\_PROCESSING\_GAMMA

```

int result = 0;
double value = 1.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_PROCESSING_GAMMA, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_PROCESSING_GAMMA, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 15.4. FG\_PROCESSING\_INVERT

When *FG\_PROCESSING\_INVERT* is set to **FG\_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

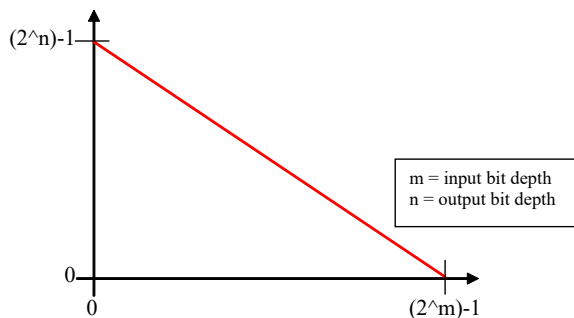


Table 15.4. Parameter properties of FG\_PROCESSING\_INVERT

Property	Value
Name	<b>FG_PROCESSING_INVERT</b>
Display Name	<b>Invert</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_ON On</b> <b>FG_OFF Off</b>
Default value	<b>FG_OFF</b>

### Example 15.4. Usage of FG\_PROCESSING\_INVERT

---

```
int result = 0;
int value = FG_OFF;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_PROCESSING_INVERT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_PROCESSING_INVERT, &value, 0, type)) < 0) {
    /* error handling */
}
```

---

---

# Chapter 16. Output Format

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

## 16.1. FG\_FORMAT

Parameter *FG\_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 *FG\_BITALIGNMENT*.

This applet has no integrated color converter. If you select a different color pixel format between the input and output no valid output data can be generated.

This applet supports the following output formats:

- **FG\_COL24**: 24 bit BGR color format with 8 bit/component.
- **FG\_COL30**: 30 bit BGR 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.

- **FG\_COL36**: 36 bit BGR 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.

- **FG\_COL42**: 42 bit BGR 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. The DMA transfer might be filled with random content for the last bytes.

- **FG\_COL48**: 48 bit BGR color format with 16 bit/component.



### BGR Memory Alignment

Note that the color components are written to the PC buffer in the common blue, green, red (BGR) order. This means, that the blue component is at the lower memory address, while red is at the highest memory address of the components triple.

Table 16.1. Parameter properties of FG\_FORMAT

Property	Value
Name	<b>FG_FORMAT</b>
Display Name	<b>Output Format</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_COL24</b> Color 24bit <b>FG_COL30</b> Color 30bit <b>FG_COL36</b> Color 36bit <b>FG_COL42</b> Color 42bit <b>FG_COL48</b> Color 48bit
Default value	<b>FG_COL24</b>

Example 16.1. Usage of FG\_FORMAT

```

int result = 0;
int value = FG_COL24;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_FORMAT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_FORMAT, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 16.2. FG\_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 *FG\_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 *FG\_CUSTOM\_BIT\_SHIFT\_RIGHT* 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 FG\_BITALIGNMENT

Property	Value
Name	<b>FG_BITALIGNMENT</b>
Display Name	<b>Alignment</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_LEFT_ALIGNED</b> Left Aligned <b>FG_RIGHT_ALIGNED</b> Right Aligned <b>FG_CUSTOM_BIT_SHIFT_MODE</b> Custom Bit Shift
Default value	<b>FG_LEFT_ALIGNED</b>

**Example 16.2. Usage of FG\_BITALIGNMENT**

```

int result = 0;
int value = FG_LEFT_ALIGNED;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_BITALIGNMENT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_BITALIGNMENT, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 16.3. FG\_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 FG\_PIXELDEPTH**

Property	Value
Name	<b>FG_PIXELDEPTH</b>
Display Name	<b>Pixel Depth</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 128</b> <b>Stepsize 1</b>
Unit of measure	<b>bit</b>

**Example 16.3. Usage of FG\_PIXELDEPTH**

```

int result = 0;
unsigned int value = 8;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_PIXELDEPTH, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 16.4. FG\_CUSTOM\_BIT\_SHIFT\_RIGHT

This parameter can only be used if parameter *FG\_BITALIGNMENT* is set to **FG\_CUSTOM\_BIT\_SHIFT\_MODE**. 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 FG\_CUSTOM\_BIT\_SHIFT\_RIGHT

Property	Value
Name	<b>FG_CUSTOM_BIT_SHIFT_RIGHT</b>
Display Name	<b>Bit Shift Right</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 15</b> <b>Stepsize 1</b>
Default value	<b>0</b>
Unit of measure	<b>bit</b>

Example 16.4. Usage of FG\_CUSTOM\_BIT\_SHIFT\_RIGHT

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CUSTOM_BIT_SHIFT_RIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CUSTOM_BIT_SHIFT_RIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

```

---

# Chapter 17. Camera Simulator

The camera simulator is a convenient way to simulate cameras for first time applet tests. If the simulator is enabled it generates pattern frames of specified size and speed. The image data is replaced at the position of the camera i.e. all applet processing functionalities are applied to the generated images. Note that camera specific settings of the applet will not have any functionality.

The generated images are horizontal, diagonal or vertical grayscale patterns, such as the one shown in the following figure.

Figure 17.1. Generator Pattern



## No Sub-Sensor sorting in Generated Images

The camera simulator will generate a simple grayscale pattern. If the camera or this applet uses sub sensor pixel sorting (sensor correction), the simulator will not generate images which represent the camera sensor.

## 17.1. FG\_CAMERASIMULATOR\_ENABLE

The camera simulator is enabled with this parameter. When you switch between camera mode and simulator, the applet will finalize the current frame before switching to the other input. Note that an activated simulator will have effect on parameter *FG\_CAMSTATUS*.



## Only 8bit support

The camera simulator will produce valid 8bit values only for 8bit pixel format. All other pixel formats will consist of packed 8bit data inside the packed format.

This will cause strange images in the simulation for higher bit depth than 8bit. Since this function is not related to productive usage this should be acceptable.



Table 17.1. Parameter properties of FG\_CAMERASIMULATOR\_ENABLE

Property	Value
Name	<b>FG_CAMERASIMULATOR_ENABLE</b>
Display Name	<b>Image Source</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_CAMPOR</b> Camera <b>FG_CAMERASIMULATOR</b> Simulator
Default value	<b>FG_CAMPOR</b>

Example 17.1. Usage of FG\_CAMERASIMULATOR\_ENABLE

```

int result = 0;
int value = FG_CAMPOR;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_ENABLE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_ENABLE, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 17.2. FG\_CAMERASIMULATOR\_WIDTH

The width of the generated frame is set with this parameter. You can enter any value. The applet will automatically round up to the next valid value limited due to internal processing granularity.

The range of the width depends on other parameters and is automatically determined from the applet. Decrease the speed for extending the range of the width value.

Table 17.2. Parameter properties of FG\_CAMERASIMULATOR\_WIDTH

Property	Value
Name	<b>FG_CAMERASIMULATOR_WIDTH</b>
Display Name	<b>Width</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 1</b> <b>Maximum 131088</b> <b>Stepsize 1</b>
Default value	<b>1024</b>
Unit of measure	<b>pixel</b>

Example 17.2. Usage of FG\_CAMERASIMULATOR\_WIDTH

```

int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

```

---

```

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}

```

---

### 17.3. FG\_CAMERASIMULATOR\_LINE\_GAP

The simulator will generate a gap between the lines. The length of the gap is defined by this parameter. So the time of the gap depends on the pixel clock and the value.

You can enter any value. The applet will automatically round up to the next valid value.

The range of the line gap depends on other parameters and is automatically determined from the applet. Decrease the speed for extending the range of the line gap value.

The parameter can only be changed if `FG_CAMERASIMULATOR_SELECT_MODE` is set to **FG\_PIXEL\_FREQUENCY**.

Table 17.3. Parameter properties of FG\_CAMERASIMULATOR\_LINE\_GAP

Property	Value
Name	<b>FG_CAMERASIMULATOR_LINE_GAP</b>
Display Name	<b>Line Gap</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 131088</b> <b>Stepsize 1</b>
Default value	<b>0</b>
Unit of measure	<b>pixel</b>

Example 17.3. Usage of FG\_CAMERASIMULATOR\_LINE\_GAP

---

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_LINE_GAP, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_LINE_GAP, &value, 0, type)) < 0) {
    /* error handling */
}

```

---

### 17.4. FG\_CAMERASIMULATOR\_HEIGHT

The height of the generated frame is set with this parameter.

The range of the height depends on other parameters and is automatically determined from the applet. Decrease the speed for extending the range of the height value.

Table 17.4. Parameter properties of FG\_CAMERASIMULATOR\_HEIGHT

Property	Value
Name	FG_CAMERASIMULATOR_HEIGHT
Display Name	Height
Type	Unsigned Integer
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	Minimum 1 Maximum 65536 Stepsize 1
Default value	1024
Unit of measure	pixel

Example 17.4. Usage of FG\_CAMERASIMULATOR\_HEIGHT

```

int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 17.5. FG\_CAMERASIMULATOR\_FRAME\_GAP

The simulator will generate a gap between the frames. The length of the gap is defined by this parameter. So the time of the gap depends on the line rate and the value.

The range of the frame gap depends on other parameters and is automatically determined from the applet. Decrease the speed for extending the range of the frame gap value.

The parameter can not be changed if parameter *FG\_CAMERASIMULATOR\_SELECT\_MODE* is set to **FG\_FRAMERATE**.

Table 17.5. Parameter properties of FG\_CAMERASIMULATOR\_FRAME\_GAP

Property	Value
Name	FG_CAMERASIMULATOR_FRAME_GAP
Display Name	Frame Gap
Type	Unsigned Integer
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	Minimum 0 Maximum 65536 Stepsize 1
Default value	0
Unit of measure	pixel

Example 17.5. Usage of FG\_CAMERASIMULATOR\_FRAME\_GAP

```

int result = 0;
unsigned int value = 0;

```

```

const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_FRAME_GAP, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_FRAME_GAP, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 17.6. FG\_CAMERASIMULATOR\_PATTERN

The simulator will generate pixel value ramps from 0 to 255. For this RGB camera input, the same values are mapped to all color components. As this applet has an internal bit width of 16 bits, the values are mapped to the upper 8 bits of each color component.

The following three types of patterns can be generated and selected by this parameter.

- **FG\_HORIZONTAL**

A horizontal pattern. Values are increased by 1 in x-direction.

- **FG\_VERTICAL**

A vertical pattern. Values are increased by 1 in y-direction.

- **FG\_DIAGONAL**

A diagonal pattern. Values are increased by 1 in x and y-direction.

Table 17.6. Parameter properties of FG\_CAMERASIMULATOR\_PATTERN

Property	Value
Name	<b>FG_CAMERASIMULATOR_PATTERN</b>
Display Name	<b>Pattern</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_HORIZONTAL</b> Horizontal <b>FG_VERTICAL</b> Vertical <b>FG_DIAGONAL</b> Diagonal
Default value	<b>FG_DIAGONAL</b>

Example 17.6. Usage of FG\_CAMERASIMULATOR\_PATTERN

```

int result = 0;
int value = FG_DIAGONAL;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_PATTERN, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_PATTERN, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 17.7. FG\_CAMERASIMULATOR\_PATTERN\_OFFSET

Using this parameter, an offset value can be added to the generated patterns. After acquisition start, the offset is added. For example, the very first pixel of an image will start with the offset value instead of 0.

Table 17.7. Parameter properties of FG\_CAMERASIMULATOR\_PATTERN\_OFFSET

Property	Value
Name	<b>FG_CAMERASIMULATOR_PATTERN_OFFSET</b>
Display Name	<b>Pattern Offset</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 255</b> <b>Stepsize 1</b>
Default value	<b>0</b>
Unit of measure	<b>pixel value</b>

Example 17.7. Usage of FG\_CAMERASIMULATOR\_PATTERN\_OFFSET

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_PATTERN_OFFSET, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_PATTERN_OFFSET, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 17.8. FG\_CAMERASIMULATOR\_ROLL

The generated pattern can be 'rolled'. With every new frame, all pattern pixels are increased by value one. At the wrap-around value 256, the pixel will get value 0. The generated images look like a moving (rolling) image.

Table 17.8. Parameter properties of FG\_CAMERASIMULATOR\_ROLL

Property	Value
Name	<b>FG_CAMERASIMULATOR_ROLL</b>
Display Name	<b>Roll</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_ON</b> On <b>FG_OFF</b> Off
Default value	<b>FG_ON</b>

Example 17.8. Usage of FG\_CAMERASIMULATOR\_ROLL

```

int result = 0;
int value = FG_ON;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_ROLL, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_ROLL, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 17.9. FG\_CAMERASIMULATOR\_SELECT\_MODE

The simulator will generate the images with a certain speed. Users are allowed to select whether they want to set the pixel frequency, line rate or frame rate to control the speed. This parameter selects the mode.

Table 17.9. Parameter properties of FG\_CAMERASIMULATOR\_SELECT\_MODE

Property	Value
Name	FG_CAMERASIMULATOR_SELECT_MODE
Display Name	Speed Mode
Type	Enumeration
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	FG_PIXEL_FREQUENCY    Pixel Frequency FG_LINERATE            Line Rate FG_FRAMERATE          Frame Rate
Default value	FG_LINERATE

Example 17.9. Usage of FG\_CAMERASIMULATOR\_SELECT\_MODE

```

int result = 0;
int value = FG_LINERATE;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_SELECT_MODE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_SELECT_MODE, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 17.10. FG\_CAMERASIMULATOR\_PIXEL\_FREQUENCY

This parameter sets the pixel frequency. Note that the generator only simulates cameras. It is made for a first time use of the applet and user SDK verification. The camera simulator cannot reflect the exact timings and frequencies of cameras.

To set the pixel frequency, you will need to set parameter *FG\_CAMERASIMULATOR\_SELECT\_MODE* to **FG\_PIXEL\_FREQUENCY**.

Any floating point value can be inserted to the parameter. However, the applet will round the value to the next valid value. Read the parameter value to find out the new rounded value.

Table 17.10. Parameter properties of FG\_CAMERASIMULATOR\_PIXEL\_FREQUENCY

Property	Value
Name	FG_CAMERASIMULATOR_PIXEL_FREQUENCY
Display Name	Pixel Frequency
Type	Double
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	Minimum    0.24999999999999994 Maximum    2400.0 Stepsize    0.5
Default value	39.375
Unit of measure	MHz

Example 17.10. Usage of FG\_CAMERASIMULATOR\_PIXEL\_FREQUENCY

```

int result = 0;
double value = 39.375;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_PIXEL_FREQUENCY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_PIXEL_FREQUENCY, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 17.11. FG\_CAMERASIMULATOR\_LINERATE

This parameter sets the line rate of the generated images.

To set the line rate, you will need to set parameter *FG\_CAMERASIMULATOR\_SELECT\_MODE* to **FG\_LINERATE**.

In line rate mode, the pixel frequency is set to the maximum.

Any floating point value can be inserted to the parameter. However, the applet will round the value to the next valid value. Read the parameter value to find out the new rounded value.

Table 17.11. Parameter properties of FG\_CAMERASIMULATOR\_LINERATE

Property	Value
Name	<b>FG_CAMERASIMULATOR_LINERATE</b>
Display Name	<b>Line Rate</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0.15</b> <b>Maximum 7.5E7</b> <b>Stepsize 7.0E-11</b>
Default value	<b>10240.0</b>
Unit of measure	<b>Hz</b>

Example 17.11. Usage of FG\_CAMERASIMULATOR\_LINERATE

```

int result = 0;
double value = 10240.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_LINERATE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_LINERATE, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 17.12. FG\_CAMERASIMULATOR\_FRAMERATE

This parameter sets the frame rate of the generated images. For parameter *FG\_TRIGGER\_FRAMESPERSECOND* only frame values up to the upper value limit of *FG\_CAMERASIMULATOR\_FRAMERATE* are valid.

To set the frame rate, you will need to set parameter `FG_CAMERASIMULATOR_SELECT_MODE` to `FG_FRAMERATE`.

In frame rate mode, the pixel frequency is set to the maximum and the line gap is set to zero.

Any floating point value can be inserted to the parameter. However, the applet will round the value to the next valid value. Read the parameter value to find out the new rounded value.

Table 17.12. Parameter properties of `FG_CAMERASIMULATOR_FRAMERATE`

Property	Value
Name	<code>FG_CAMERASIMULATOR_FRAMERATE</code>
Display Name	Framerate
Type	Double
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	Minimum 0.15 Maximum 7.5E7 Stepsize 7.0E-11
Default value	10.0
Unit of measure	Hz

Example 17.12. Usage of `FG_CAMERASIMULATOR_FRAMERATE`

```
int result = 0;
double value = 10.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_FRAMERATE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_FRAMERATE, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 17.13. FG\_CAMERASIMULATOR\_TRIGGER\_MODE

You can either use the camera simulator in free run mode or the simulator can be triggered by the output of the trigger module of this applet. As this applet uses a CoaxPress camera interface, the CXP trigger output of the respective camera port is used as camera simulator trigger input. The rising edge of the trigger will be used.

You can choose between line trigger and frame trigger mode. In line trigger mode, a rising edge at the input will output a line from the camera simulator. For frame trigger mode, the input will trigger the output of a frame.



### Trigger frequency must not exceed the speed of the camera simulator

Same as for real cameras, it is very important that the frequency of the trigger pulses do not exceed the maximum speed of the camera simulator. Set the camera simulator to a sufficiently large speed to avoid line or frame lost.



Table 17.13. Parameter properties of FG\_CAMERASIMULATOR\_TRIGGER\_MODE

Property	Value
Name	<b>FG_CAMERASIMULATOR_TRIGGER_MODE</b>
Display Name	<b>Trigger Mode</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>SIMULATION_FREE_RUN</b> Free Run <b>RISING_EDGE_TRIGGERS_LINE</b> Rising Edge Triggers Line <b>RISING_EDGE_TRIGGERS_FRAME</b> Rising Edge Triggers Frame
Default value	<b>SIMULATION_FREE_RUN</b>

Example 17.13. Usage of FG\_CAMERASIMULATOR\_TRIGGER\_MODE

```

int result = 0;
int value = SIMULATION_FREE_RUN;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_TRIGGER_MODE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_TRIGGER_MODE, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 17.14. FG\_CAMERASIMULATOR\_ACTIVE

Table 17.14. Parameter properties of FG\_CAMERASIMULATOR\_ACTIVE

Property	Value
Name	<b>FG_CAMERASIMULATOR_ACTIVE</b>
Display Name	<b>Active Parts</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 1</b> <b>Maximum 2000</b> <b>Stepsize 1</b>
Unit of measure	<b>pixel</b>

Example 17.14. Usage of FG\_CAMERASIMULATOR\_ACTIVE

```

int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_ACTIVE, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 17.15. FG\_CAMERASIMULATOR\_PASSIVE

Table 17.15. Parameter properties of FG\_CAMERASIMULATOR\_PASSIVE

Property	Value
Name	<b>FG_CAMERASIMULATOR_PASSIVE</b>
Display Name	<b>Passive Parts</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 1</b> <b>Maximum 2000</b> <b>Stepsize 1</b>
Unit of measure	<b>pixel</b>

Example 17.15. Usage of FG\_CAMERASIMULATOR\_PASSIVE

```
int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_PASSIVE, &value, 0, type)) < 0) {
    /* error handling */
}
```

---

# Chapter 18. Miscellaneous

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

## 18.1. FG\_SYSTEMMONITOR\_CHANNEL\_CURRENT

Returns the channel current in Ampere.

Table 18.1. Parameter properties of FG\_SYSTEMMONITOR\_CHANNEL\_CURRENT

Property	Value
Name	FG_SYSTEMMONITOR_CHANNEL_CURRENT
Display Name	Channel Current
Type	Double Field
Field Size	4
Access policy	Read-Only
Storage policy	Transient
Unit of measure	A

Example 18.1. Usage of FG\_SYSTEMMONITOR\_CHANNEL\_CURRENT

```
int result = 0;

FieldParameterDouble access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMDOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_CHANNEL_CURRENT, &access, 0, type)) < 0) {
    /* error handling */
}
}
```

## 18.2. FG\_SYSTEMMONITOR\_CHANNEL\_VOLTAGE

Returns the channel voltage.

Table 18.2. Parameter properties of FG\_SYSTEMMONITOR\_CHANNEL\_VOLTAGE

Property	Value
Name	FG_SYSTEMMONITOR_CHANNEL_VOLTAGE
Display Name	Channel Voltage
Type	Double Field
Field Size	4
Access policy	Read-Only
Storage policy	Transient
Unit of measure	V

Example 18.2. Usage of FG\_SYSTEMMONITOR\_CHANNEL\_VOLTAGE

```
int result = 0;

FieldParameterDouble access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMDOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_CHANNEL_VOLTAGE, &access, 0, type)) < 0) {
    /* error handling */
}
}
```

## 18.3. FG\_TIMEOUT

This parameter is used to set a timeout for DMA transfers. After a timeout the acquisition is stopped. But it is only an internal value that should not be used directly. Use the timeout value described in the Framegrabber API or microDisplay for acquisition in order to handle the functionality correctly.

Table 18.3. Parameter properties of FG\_TIMEOUT

Property	Value
Name	<b>FG_TIMEOUT</b>
Display Name	<b>Timeout</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 2</b> <b>Maximum 2147483646</b> <b>Stepsize 1</b>
Default value	<b>1000000</b>
Unit of measure	<b>seconds</b>

Example 18.3. Usage of FG\_TIMEOUT

```
int result = 0;
unsigned int value = 1000000;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_TIMEOUT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TIMEOUT, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 18.4. FG\_APPLET\_VERSION

This parameter represents the version number of the applet. Please report this value for any support of the applet.

Table 18.4. Parameter properties of FG\_APPLET\_VERSION

Property	Value
Name	<b>FG_APPLET_VERSION</b>
Display Name	<b>Applet version</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 18.4. Usage of FG\_APPLET\_VERSION

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_APPLET_VERSION, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 18.5. FG\_APPLET\_REVISION

This parameter represents the revision number of the applet. Please report this value for any support case with the applet.

Table 18.5. Parameter properties of FG\_APPLET\_REVISION

Property	Value
Name	<b>FG_APPLET_REVISION</b>
Display Name	<b>Applet revision</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 18.5. Usage of FG\_APPLET\_REVISION

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_APPLET_REVISION, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 18.6. FG\_APPLET\_ID

This parameter returns the unique applet id of the applet as a string parameter.

Table 18.6. Parameter properties of FG\_APPLET\_ID

Property	Value
Name	<b>FG_APPLET_ID</b>
Display Name	<b>Applet Id</b>
Type	<b>String</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 18.6. Usage of FG\_APPLET\_ID

```
int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_getParameterWithType(fg, FG_APPLET_ID, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 18.7. FG\_APPLET\_BUILD\_TIME

This string parameter returns the hardware applet (HAP) build timestamp. To obtain the build time of the applet, check the DLL / SO file details. Mainly, this parameter is required for internal usage only.

Table 18.7. Parameter properties of FG\_APPLET\_BUILD\_TIME

Property	Value
Name	<b>FG_APPLET_BUILD_TIME</b>
Display Name	<b>Build Time</b>
Type	<b>String</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

**Example 18.7. Usage of FG\_APPLET\_BUILD\_TIME**

```

int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_getParameterWithType(fg, FG_APPLET_BUILD_TIME, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.8. FG\_HAP\_FILE

The name of the Hardware-Applet (HAP) file on which this applet is based. Please report this read-only string parameter for any support case of the applet.

**Table 18.8. Parameter properties of FG\_HAP\_FILE**

Property	Value
Name	<b>FG_HAP_FILE</b>
Display Name	<b>HAP file</b>
Type	<b>String</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

**Example 18.8. Usage of FG\_HAP\_FILE**

```

int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_getParameterWithType(fg, FG_HAP_FILE, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.9. FG\_DMASTATUS

Using this parameter the status of a DMA channel can be obtained. Value "1" represents a started DMA i.e. a started acquisition. Value "0" represents a stopped acquisition.

**Table 18.9. Parameter properties of FG\_DMASTATUS**

Property	Value
Name	<b>FG_DMASTATUS</b>
Display Name	<b>DMA Status</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

**Example 18.9. Usage of FG\_DMASTATUS**

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_DMASTATUS, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.10. FG\_CAMSTATUS

For CoaXPress, this parameter is not used. Please use the SDK's GenICam functions to identify camera detection state. More details on this can be found in the Basler Framegrabber SDK documentation.

Table 18.10. Parameter properties of FG\_CAMSTATUS

Property	Value
Name	<b>FG_CAMSTATUS</b>
Display Name	<b>Camera Status</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 1</b> <b>Stepsize 1</b>

Example 18.10. Usage of FG\_CAMSTATUS

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_CAMSTATUS, &value, 0, type)) < 0) {
    /* error handling */
}
```

## 18.11. FG\_CAMSTATUS\_EXTENDED

This parameter provides extended information on the pixel clock from the camera, LVAL and FVAL, as well as the camera trigger signals, external trigger signals, buffer overflow status and buffer status. Each bit of the eight bit output word represents one parameter listed in the following:

- 0 = CameraClk, currently NOT supported by CoaXPress interface.
- 1 = CameraLval, currently NOT supported by CoaXPress interface.
- 2 = CameraFval, currently NOT supported by CoaXPress interface.
- 3 = Camera CC1 Signal, currently NOT supported by CoaXPress interface.
- 4 = ExTrg / external trigger, currently NOT supported by CoaXPress interface.
- 5 = BufferOverflow
- 6 = BufStatus, LSB
- 7 = BufStatus, MSB

Table 18.11. Parameter properties of FG\_CAMSTATUS\_EXTENDED

Property	Value
Name	<b>FG_CAMSTATUS_EXTENDED</b>
Display Name	<b>Camera Status Extended</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 255</b> <b>Stepsize 1</b>

**Example 18.11. Usage of FG\_CAMSTATUS\_EXTENDED**

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_CAMSTATUS_EXTENDED, &value, 0, type)) < 0) {
    /* error handling */
}

```

**18.12. FG\_SYSTEMMONITOR\_FPGA\_TEMPERATURE**

Returns the current FPGA die temperature.

Table 18.12. Parameter properties of FG\_SYSTEMMONITOR\_FPGA\_TEMPERATURE

Property	Value
Name	<b>FG_SYSTEMMONITOR_FPGA_TEMPERATURE</b>
Display Name	<b>FGPA Temperature</b>
Type	<b>Double</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum 0.0</b> <b>Maximum 1000.0</b> <b>Stepsize 0.0</b>
Unit of measure	<b>Celsius</b>

**Example 18.12. Usage of FG\_SYSTEMMONITOR\_FPGA\_TEMPERATURE**

```

int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_FPGA_TEMPERATURE, &value, 0, type)) < 0) {
    /* error handling */
}

```

**18.13. FG\_SYSTEMMONITOR\_FPGA\_VCC\_INT**

Returns the current FPGA internal voltage.

Table 18.13. Parameter properties of FG\_SYSTEMMONITOR\_FPGA\_VCC\_INT

Property	Value
Name	<b>FG_SYSTEMMONITOR_FPGA_VCC_INT</b>
Display Name	<b>FGPA Vcc Int</b>
Type	<b>Double</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum -1000.0</b> <b>Maximum 1000.0</b> <b>Stepsize 0.0</b>
Unit of measure	<b>V</b>

**Example 18.13. Usage of FG\_SYSTEMMONITOR\_FPGA\_VCC\_INT**

```

int result = 0;

```



```

double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_FPGA_VCC_INT, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.14. FG\_SYSTEMMONITOR\_FPGA\_VCC\_AUX

Returns the current FPGA Vcc auxiliary voltage.

Table 18.14. Parameter properties of FG\_SYSTEMMONITOR\_FPGA\_VCC\_AUX

Property	Value
Name	<b>FG_SYSTEMMONITOR_FPGA_VCC_AUX</b>
Display Name	<b>FGPA Vcc Aux</b>
Type	<b>Double</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum -1000.0</b> <b>Maximum 1000.0</b> <b>Stepsize 0.0</b>
Unit of measure	<b>V</b>

Example 18.14. Usage of FG\_SYSTEMMONITOR\_FPGA\_VCC\_AUX

```

int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_FPGA_VCC_AUX, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.15. FG\_SYSTEMMONITOR\_FPGA\_VCC\_BRAM

Returns the current FPGA Vcc of the BlockRAM voltage.

Table 18.15. Parameter properties of FG\_SYSTEMMONITOR\_FPGA\_VCC\_BRAM

Property	Value
Name	<b>FG_SYSTEMMONITOR_FPGA_VCC_BRAM</b>
Display Name	<b>FGPA Vcc BRAM</b>
Type	<b>Double</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum -1000.0</b> <b>Maximum 1000.0</b> <b>Stepsize 0.0</b>
Unit of measure	<b>V</b>

Example 18.15. Usage of FG\_SYSTEMMONITOR\_FPGA\_VCC\_BRAM

```

int result = 0;

```

```

double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_FPGA_VCC_BRAM, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.16. FG\_SYSTEMMONITOR\_CURRENT\_LINK\_WIDTH

Returns the current link width of the frame grabber representing the number of PCIe lanes being 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 18.16. Parameter properties of FG\_SYSTEMMONITOR\_CURRENT\_LINK\_WIDTH

Property	Value
Name	<b>FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH</b>
Display Name	<b>Current Link Width</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 15</b> <b>Stepsize 0</b>
Unit of measure	<b>lanes</b>

Example 18.16. Usage of FG\_SYSTEMMONITOR\_CURRENT\_LINK\_WIDTH

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.17. FG\_SYSTEMMONITOR\_CURRENT\_LINK\_SPEED

Returns the current link width of the frame grabber representing the number of PCIe lanes being 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 18.17. Parameter properties of FG\_SYSTEMMONITOR\_CURRENT\_LINK\_SPEED

Property	Value
Name	<b>FG_SYSTEMMONITOR_CURRENT_LINK_SPEED</b>
Display Name	<b>Current Link Speed</b>
Type	<b>Double</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum 0.0</b> <b>Maximum 1000.0</b> <b>Stepsize 0.0</b>
Unit of measure	<b>GB/s</b>

**Example 18.17. Usage of FG\_SYSTEMMONITOR\_CURRENT\_LINK\_SPEED**

```

int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_CURRENT_LINK_SPEED, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.18. FG\_SYSTEMMONITOR\_PCIE\_LINK\_GEN2\_CAPABLE

Returns if PCIe generation 2 is supported by current applet of the frame grabber.

**Table 18.18. Parameter properties of FG\_SYSTEMMONITOR\_PCIE\_LINK\_GEN2\_CAPABLE**

Property	Value
Name	<b>FG_SYSTEMMONITOR_PCIE_LINK_GEN2_CAPABLE</b>
Display Name	<b>PCIe Link Gen 2 Capable</b>
Type	<b>Enumeration</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>FG_YES</b> Yes <b>FG_NO</b> No

**Example 18.18. Usage of FG\_SYSTEMMONITOR\_PCIE\_LINK\_GEN2\_CAPABLE**

```

int result = 0;
int value = FG_YES;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_PCIE_LINK_GEN2_CAPABLE, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.19. FG\_SYSTEMMONITOR\_PCIE\_LINK\_PARTNER\_GEN2\_CAPABLE

Returns if the expected PCIe generation 2 is supported by the partner. The partner would be the mainboard or in detail the corresponding PCIe interface on the host side.

**Table 18.19. Parameter properties of FG\_SYSTEMMONITOR\_PCIE\_LINK\_PARTNER\_GEN2\_CAPABLE**

Property	Value
Name	<b>FG_SYSTEMMONITOR_PCIE_LINK_PARTNER_GEN2_CAPABLE</b>
Display Name	<b>PCIe Link Partner Gen 2 Capable</b>
Type	<b>Enumeration</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>FG_YES</b> Yes <b>FG_NO</b> No

**Example 18.19. Usage of FG\_SYSTEMMONITOR\_PCIE\_LINK\_PARTNER\_GEN2\_CAPABLE**

```

int result = 0;
int value = FG_YES;

```

```

const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_PCIE_LINK_PARTNER_GEN2_CAPABLE, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.20. FG\_SYSTEMMONITOR\_PCIE\_TRAINED\_PAYLOAD\_SIZE

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

Table 18.20. Parameter properties of FG\_SYSTEMMONITOR\_PCIE\_TRAINED\_PAYLOAD\_SIZE

Property	Value
Name	FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE
Display Name	PCIe Trained Payload Size
Type	Unsigned Integer
Access policy	Read-Only
Storage policy	Transient
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 1024 <b>Stepsize</b> 0
Unit of measure	byte

Example 18.20. Usage of FG\_SYSTEMMONITOR\_PCIE\_TRAINED\_PAYLOAD\_SIZE

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.21. FG\_SYSTEMMONITOR\_EXTENSION\_CONNECTOR\_PRESENT

Returns if a extension connector is present on the frame grabber board.

Table 18.21. Parameter properties of FG\_SYSTEMMONITOR\_EXTENSION\_CONNECTOR\_PRESENT

Property	Value
Name	FG_SYSTEMMONITOR_EXTENSION_CONNECTOR_PRESENT
Display Name	Extension Connector Present
Type	Enumeration
Access policy	Read-Only
Storage policy	Transient
Allowed values	<b>FG_YES</b> Yes <b>FG_NO</b> No

Example 18.21. Usage of FG\_SYSTEMMONITOR\_EXTENSION\_CONNECTOR\_PRESENT

```

int result = 0;
int value = FG_YES;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

```

```

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_EXTENSION_CONNECTOR_PRESENT, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.22. FG\_ALTERNATIVE\_BOARD\_DETECTION

Returns the current state of the alternative frame grabber PCIe board detection algorithm. If value = FG\_OFF, the Silicon Software default algorithm is used. If value = FG\_ON, an alternative board detection algorithm is used.

This parameter is used for support purposes only.

Table 18.22. Parameter properties of FG\_ALTERNATIVE\_BOARD\_DETECTION

Property	Value
Name	FG_ALTERNATIVE_BOARD_DETECTION
Display Name	Alternative Board Detection
Type	Enumeration
Access policy	Read-Only
Storage policy	Transient
Allowed values	FG_ON On FG_OFF Off

Example 18.22. Usage of FG\_ALTERNATIVE\_BOARD\_DETECTION

```

int result = 0;
int value = FG_OFF;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_ALTERNATIVE_BOARD_DETECTION, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.23. FG\_SYSTEMMONITOR\_FPGA\_DNA

The parameter *FG\_SYSTEMMONITOR\_FPGA\_DNA* provides the 57 bit unique FPGA DNA as an integer value.

Table 18.23. Parameter properties of FG\_SYSTEMMONITOR\_FPGA\_DNA

Property	Value
Name	FG_SYSTEMMONITOR_FPGA_DNA
Display Name	FPGA DNA
Type	Unsigned Integer (64 Bit)
Access policy	Read-Only
Storage policy	Transient
Allowed values	Minimum 0 Maximum 144115188075855872 Stepsize 0

Example 18.23. Usage of FG\_SYSTEMMONITOR\_FPGA\_DNA

```

int result = 0;

```

```

uint64_t value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT64_T;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_FPGA_DNA, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.24. FG\_SYSTEMMONITOR\_CHANNEL\_STATE

Returns the Power over CXP link status. Value range is based on enumerator Fg\_PoCXPState.

Table 18.24. Power over CXP state enumerator

Fg_PoCXPState	Description	Value
BOOTING	booting, not initialized	0x001
NOCABLE	no cable connected	0x002
NOPOCXP	no Power over CXP	0x004
POCXPOK	Power over CXP OK	0x008
MIN_CURR	minimum current	0x010
MAX_CURR	maximum current	0x020
LOW_VOLT	low voltage	0x040
OVER_VOLT	over voltage	0x080
ADC_Chip_Error	ADC Chip Error	0x100

The defines for the single values are named FG\_POCXSTATE\_\* corresponding to the enumerator names.

Table 18.25. Parameter properties of FG\_SYSTEMMONITOR\_CHANNEL\_STATE

Property	Value
Name	FG_SYSTEMMONITOR_CHANNEL_STATE
Display Name	Channel State
Type	Unsigned Integer Field
Field Size	4
Access policy	Read-Only
Storage policy	Transient

Example 18.24. Usage of FG\_SYSTEMMONITOR\_CHANNEL\_STATE

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_CHANNEL_STATE, &access, 0, type)) < 0) {
    /* error handling */
}

```

## 18.25. Debug

### 18.25.1. FG\_DEBUGSOURCE

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.26. Parameter properties of FG\_DEBUGSOURCE

Property	Value
Name	<b>FG_DEBUGSOURCE</b>
Display Name	<b>Debug Source</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 10</b> <b>Stepsize 1</b>
Default value	<b>0</b>

Example 18.25. Usage of FG\_DEBUGSOURCE

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGSOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGSOURCE, &value, 0, type)) < 0) {
    /* error handling */
}
```

### 18.25.2. FG\_DEBUGSOURCECENAME

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.27. Parameter properties of FG\_DEBUGSOURCECENAME

Property	Value
Name	<b>FG_DEBUGSOURCECENAME</b>
Display Name	<b>Debug Source Name</b>
Type	<b>String</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 18.26. Usage of FG\_DEBUGSOURCECENAME

```
int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_getParameterWithType(fg, FG_DEBUGSOURCECENAME, &value, 0, type)) < 0) {
    /* error handling */
}
```

### 18.25.3. FG\_DEBUGSAVECONFIG

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.28. Parameter properties of FG\_DEBUGSAVECONFIG

Property	Value
Name	<b>FG_DEBUGSAVECONFIG</b>
Display Name	<b>Debug Save Config</b>
Type	<b>String</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Transient</b>
Default value	<b>""</b>

Example 18.27. Usage of FG\_DEBUGSAVECONFIG

```

int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGSAVECONFIG, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGSAVECONFIG, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.25.4. FG\_DEBUG\_VERSION

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.29. Parameter properties of FG\_DEBUG\_VERSION

Property	Value
Name	<b>FG_DEBUG_VERSION</b>
Display Name	<b>Version</b>
Type	<b>String</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 18.28. Usage of FG\_DEBUG\_VERSION

```

int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_getParameterWithType(fg, FG_DEBUG_VERSION, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.25.5. Input

### 18.25.5.1. FG\_DEBUGINENABLE

This parameter is for internal testing. Please DON'T use this parameter.



Table 18.30. Parameter properties of FG\_DEBUGINENABLE

Property	Value
Name	<b>FG_DEBUGINENABLE</b>
Display Name	<b>Debug Input Mode</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_ON</b> On <b>FG_OFF</b> Off
Default value	<b>FG_OFF</b>

Example 18.29. Usage of FG\_DEBUGINENABLE

```

int result = 0;
int value = FG_OFF;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGINENABLE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGINENABLE, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 18.25.5.2. FG\_DEBUGFILE

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.31. Parameter properties of FG\_DEBUGFILE

Property	Value
Name	<b>FG_DEBUGFILE</b>
Display Name	<b>Debug File</b>
Type	<b>String</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Default value	<b>""</b>

Example 18.30. Usage of FG\_DEBUGFILE

```

int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGFILE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGFILE, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 18.25.5.3. FG\_DEBUGINSERT

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.32. Parameter properties of FG\_DEBUGINSERT

Property	Value
Name	<b>FG_DEBUGINSERT</b>
Display Name	<b>Debug Insert Image</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Transient</b>
Allowed values	<b>FG_APPLY</b> Apply
Default value	<b>FG_APPLY</b>

Example 18.31. Usage of FG\_DEBUGINSERT

```

int result = 0;
int value = FG_APPLY;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGINSERT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGINSERT, &value, 0, type)) < 0) {
    /* error handling */
}

```

#### 18.25.5.4. FG\_DEBUGWRITEPIXEL

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.33. Parameter properties of FG\_DEBUGWRITEPIXEL

Property	Value
Name	<b>FG_DEBUGWRITEPIXEL</b>
Display Name	<b>Debug Write Pixel</b>
Type	<b>Unsigned Integer (64 Bit)</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 144115188075855872</b> <b>Stepsize 1</b>
Default value	<b>0</b>

Example 18.32. Usage of FG\_DEBUGWRITEPIXEL

```

int result = 0;
uint64_t value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT64_T;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGWRITEPIXEL, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGWRITEPIXEL, &value, 0, type)) < 0) {
    /* error handling */
}

```

#### 18.25.5.5. FG\_DEBUGWRITEFLAG

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.34. Parameter properties of FG\_DEBUGWRITEFLAG

Property	Value
Name	<b>FG_DEBUGWRITEFLAG</b>
Display Name	<b>Debug Write Flag</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Transient</b>
Allowed values	<b>FG_ENDOFLINE</b> EndOfLine <b>FG_ENDOFFRAME</b> EndOfFrame
Default value	<b>FG_ENDOFLINE</b>

Example 18.33. Usage of FG\_DEBUGWRITEFLAG

```

int result = 0;
int value = FG_ENDOFLINE;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGWRITEFLAG, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGWRITEFLAG, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 18.25.5.6. FG\_DEBUGREADY

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.35. Parameter properties of FG\_DEBUGREADY

Property	Value
Name	<b>FG_DEBUGREADY</b>
Display Name	<b>Debug Write Ready</b>
Type	<b>Enumeration</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>FG_YES</b> Yes <b>FG_NO</b> No

Example 18.34. Usage of FG\_DEBUGREADY

```

int result = 0;
int value = FG_NOE;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_DEBUGREADY, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 18.25.6. Output

### 18.25.6.1. FG\_DEBUGOUTENABLE

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.36. Parameter properties of FG\_DEBUGOUTENABLE

Property	Value
Name	<b>FG_DEBUGOUTENABLE</b>
Display Name	<b>Debug Output Mode</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_ON</b> On <b>FG_OFF</b> Off
Default value	<b>FG_OFF</b>

Example 18.35. Usage of FG\_DEBUGOUTENABLE

```

int result = 0;
int value = FG_OFF;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGOUTENABLE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGOUTENABLE, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 18.25.6.2. FG\_DEBUGOUTXPOS

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.37. Parameter properties of FG\_DEBUGOUTXPOS

Property	Value
Name	<b>FG_DEBUGOUTXPOS</b>
Display Name	<b>Debug Output XPosition</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum 16</b> <b>Maximum 65536</b> <b>Stepsize 1</b>
Unit of measure	<b>pixel</b>

Example 18.36. Usage of FG\_DEBUGOUTXPOS

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_DEBUGOUTXPOS, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 18.25.6.3. FG\_DEBUGOUTYPOS

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.38. Parameter properties of FG\_DEBUGOUTYPOS

Property	Value
Name	<b>FG_DEBUGOUTYPOS</b>
Display Name	<b>Debug Output YPosition</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum 1</b> <b>Maximum 8388607</b> <b>Stepsize 1</b>
Unit of measure	<b>pixel</b>

Example 18.37. Usage of FG\_DEBUGOUTYPOS

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_DEBUGOUTYPOS, &value, 0, type)) < 0) {
    /* error handling */
}

```

#### 18.25.6.4. FG\_DEBUGOUTPIXEL

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.39. Parameter properties of FG\_DEBUGOUTPIXEL

Property	Value
Name	<b>FG_DEBUGOUTPIXEL</b>
Display Name	<b>Debug Output Pixel</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum 0</b> <b>Maximum -1</b> <b>Stepsize 1</b>
Unit of measure	<b>pixel</b>

Example 18.38. Usage of FG\_DEBUGOUTPIXEL

```

int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_DEBUGOUTPIXEL, &value, 0, type)) < 0) {
    /* error handling */
}

```

---

# Glossary

Area of Interest (AOI)	See Region of Interest.
Board	A Silicon Software hardware. Usually, a board is represented by a frame grabber. Boards might comprise multiple devices.
Board ID Number	An identification number of a Silicon Software 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 Silicon Software 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)	<p>A DMA transfer allows hardware subsystems within the computer to access the system memory independently of the central processing unit (CPU).</p> <p>Silicon Software 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.</p>
DMA Channel	See DMA Index.
DMA Index	The index of a DMA transfer channel. See also Direct Memory Access.
Event	<p>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.</p> <p>Silicon Software 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.</p> <p>For example these events can be used to check the reliability between a frame trigger input and the resulting and expected camera frame.</p> <p>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.</p>

---

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 increases 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

## A

Area of Interest, 13

## B

Bandwidth, 2

## C

Camera, 9

Events, 9

Format, 6

Interface, 4, 9

Camera Simulator, 90, 90

Camera Trigger Source, 17, 19, 22, 22

Camera::Events, 9

CoaXPress, 6

## D

Debugging, 59

Digital I/O, 17, 17

Digital I/O::Camera, 17

Digital I/O::Event Source, 22

Digital I/O::Events, 26

Digital I/O::GPI, 22

Digital I/O::GPO, 19

## E

Events

Camera, 9

Overflow, 66

Trigger, 26

## F

Features, 1

FG\_ALTERNATIVE\_BOARD\_DETECTION, 111

FG\_APPLET\_BUILD\_TIME, 103

FG\_APPLET\_ID, 103

FG\_APPLET\_REVISION, 102

FG\_APPLET\_VERSION, 102

FG\_BITALIGNMENT, 87

FG\_CAMERASIMULATOR\_ACTIVE, 99

FG\_CAMERASIMULATOR\_ENABLE, 90

FG\_CAMERASIMULATOR\_FRAMERATE, 97

FG\_CAMERASIMULATOR\_FRAME\_GAP, 93

FG\_CAMERASIMULATOR\_HEIGHT, 92

FG\_CAMERASIMULATOR\_LINERATE, 97

FG\_CAMERASIMULATOR\_LINE\_GAP, 92

FG\_CAMERASIMULATOR\_PASSIVE, 99

FG\_CAMERASIMULATOR\_PATTERN, 94

FG\_CAMERASIMULATOR\_PATTERN\_OFFSET, 94

FG\_CAMERASIMULATOR\_PIXEL\_FREQUENCY, 96

FG\_CAMERASIMULATOR\_ROLL, 95

FG\_CAMERASIMULATOR\_SELECT\_MODE, 96

FG\_CAMERASIMULATOR\_TRIGGER\_MODE, 98

FG\_CAMERASIMULATOR\_WIDTH, 91

FG\_CAMSTATUS, 104  
FG\_CAMSTATUS\_EXTENDED, 105  
FG\_CUSTOM\_BIT\_SHIFT\_RIGHT, 88  
FG\_CUSTOM\_SIGNAL\_EVENT\_0, 26  
FG\_CUSTOM\_SIGNAL\_EVENT\_0\_POLARITY, 23  
FG\_CUSTOM\_SIGNAL\_EVENT\_0\_SOURCE, 22  
FG\_CUSTOM\_SIGNAL\_EVENT\_1, 26  
FG\_CUSTOM\_SIGNAL\_EVENT\_1\_POLARITY, 25  
FG\_CUSTOM\_SIGNAL\_EVENT\_1\_SOURCE, 24  
FG\_CXP\_TRIGGER\_PACKET\_MODE, 6  
FG\_DEBUGFILE, 115  
FG\_DEBUGINENABLE, 114  
FG\_DEBUGINSERT, 115  
FG\_DEBUGOUTENABLE, 117  
FG\_DEBUGOUTPIXEL, 119  
FG\_DEBUGOUTXPOS, 118  
FG\_DEBUGOUTYPOS, 118  
FG\_DEBUGREADY, 117  
FG\_DEBUGSAVECONFIG, 113  
FG\_DEBUGSOURCE, 112  
FG\_DEBUGSOURCENAME, 113  
FG\_DEBUGWRITEFLAG, 116  
FG\_DEBUGWRITEPIXEL, 116  
FG\_DEBUG\_VERSION, 114  
FG\_DIGIO\_INPUT, 22  
FG\_DIGIO\_OUTPUT, 21  
FG\_DMASTATUS, 104  
FG\_END\_OF\_FRAME\_CAM\_PORT\_0, 9  
FG\_END\_OF\_LINE\_CAM\_PORT\_0, 9  
FG\_EXSYNCON, 29  
FG\_EXSYNCPOLARITY, 48  
FG\_FILLLEVEL, 65  
FG\_FLASHON, 52  
FG\_FLASH\_POLARITY, 57  
FG\_FORMAT, 86  
FG\_HAP\_FILE, 104  
FG\_HEIGHT, 15  
FG\_IMGTRIGGERDEBOUNCING, 55  
FG\_IMGTRIGGERGATEDELAY, 55  
FG\_IMGTRIGGERINPOLARITY, 54  
FG\_IMGTRIGGERINSRC, 54  
FG\_IMGTRIGGERMODE, 51  
FG\_IMGTRIGGERON, 51  
FG\_IMGTRIGGER\_ASYNC\_HEIGHT, 52  
FG\_IMGTRIGGER\_IS\_BUSY, 53  
FG\_IMG\_SELECT, 70  
FG\_IMG\_SELECT\_PERIOD, 69  
FG\_LINEEXPOSURE, 47  
FG\_LINEPERIODE, 46  
FG\_LINETRIGGERDEBOUNCING, 33  
FG\_LINETRIGGERDELAY, 49  
FG\_LINETRIGGERINPOLARITY, 32  
FG\_LINETRIGGERINSRC, 31  
FG\_LINETRIGGERMODE, 28  
FG\_LINE\_DOWNSCALE, 33  
FG\_LINE\_DOWNSCALEINIT, 34  
FG\_LUT\_CUSTOM\_FILE, 76  
FG\_LUT\_ENABLE, 73

FG\_LUT\_IMPLEMENTATION\_TYPE, 78  
FG\_LUT\_IN\_BITS, 79  
FG\_LUT\_OUT\_BITS, 79  
FG\_LUT\_SAVE\_FILE, 78  
FG\_LUT\_TYPE, 74  
FG\_LUT\_VALUE\_BLUE, 75  
FG\_LUT\_VALUE\_GREEN, 75  
FG\_LUT\_VALUE\_RED, 74  
FG\_OVERFLOW, 66  
FG\_OVERFLOW\_CAM0, 66  
FG\_PIXELDEPTH, 88  
FG\_PIXELFORMAT, 6  
FG\_PROCESSING\_GAIN, 82  
FG\_PROCESSING\_GAMMA, 83  
FG\_PROCESSING\_INVERT, 84  
FG\_PROCESSING\_OFFSET, 82  
FG\_SCALINGFACTOR\_BLUE, 71  
FG\_SCALINGFACTOR\_GREEN, 72  
FG\_SCALINGFACTOR\_RED, 71  
FG\_SENDSOFTWARETRIGGER, 57  
FG\_SENSORHEIGHT, 11  
FG\_SENSORWIDTH, 10  
FG\_SETSOFTWARETRIGGER, 58  
FG\_SHAFTENCODERINSRC, 37  
FG\_SHAFTENCODERLEADING, 38  
FG\_SHAFTENCODERMODE, 36  
FG\_SHAFTENCODERON, 35  
FG\_SHAFTENCODER\_COMPENSATION\_COUNT, 40  
FG\_SHAFTENCODER\_COMPENSATION\_ENABLE, 39  
FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_CURRENT, 61  
FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_MAX, 62  
FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_MIN, 62  
FG\_SIGNAL\_ANALYZER\_0\_POLARITY, 60  
FG\_SIGNAL\_ANALYZER\_0\_PULSE\_COUNT, 63  
FG\_SIGNAL\_ANALYZER\_0\_SOURCE, 59  
FG\_SIGNAL\_ANALYZER\_1\_PERIOD\_CURRENT, 61  
FG\_SIGNAL\_ANALYZER\_1\_PERIOD\_MAX, 62  
FG\_SIGNAL\_ANALYZER\_1\_PERIOD\_MIN, 62  
FG\_SIGNAL\_ANALYZER\_1\_POLARITY, 60  
FG\_SIGNAL\_ANALYZER\_1\_PULSE\_COUNT, 63  
FG\_SIGNAL\_ANALYZER\_1\_SOURCE, 59  
FG\_SIGNAL\_ANALYZER\_CLEAR, 64  
FG\_SIGNAL\_ANALYZER\_PULSE\_COUNT\_DIFFERENCE, 63  
FG\_START\_OF\_FRAME\_CAM\_PORT\_0, 9  
FG\_START\_OF\_LINE\_CAM\_PORT\_0, 9  
FG\_STROBEPULSEDELAY, 56  
FG\_SYSTEMMONITOR\_CHANNEL\_CURRENT, 101  
FG\_SYSTEMMONITOR\_CHANNEL\_STATE, 112  
FG\_SYSTEMMONITOR\_CHANNEL\_VOLTAGE, 101  
FG\_SYSTEMMONITOR\_CURRENT\_LINK\_SPEED, 108  
FG\_SYSTEMMONITOR\_CURRENT\_LINK\_WIDTH, 108  
FG\_SYSTEMMONITOR\_EXTENSION\_CONNECTOR\_PRESENT, 110  
FG\_SYSTEMMONITOR\_FPGA\_DNA, 111  
FG\_SYSTEMMONITOR\_FPGA\_TEMPERATURE, 106  
FG\_SYSTEMMONITOR\_FPGA\_VCC\_AUX, 107  
FG\_SYSTEMMONITOR\_FPGA\_VCC\_BRAM, 107  
FG\_SYSTEMMONITOR\_FPGA\_VCC\_INT, 106  
FG\_SYSTEMMONITOR\_PCIE\_LINK\_GEN2\_CAPABLE, 109

FG\_SYSTEMMONITOR\_PCIE\_LINK\_PARTNER\_GEN2\_CAPABLE, 109  
FG\_SYSTEMMONITOR\_PCIE\_TRAINED\_PAYLOAD\_SIZE, 110  
FG\_TIMEOUT, 102  
FG\_TRIGGERCAMERA\_POLARITY, 18  
FG\_TRIGGERCAMERA\_SOURCE, 17  
FG\_TRIGGEROUT\_FRONT\_GPO\_0\_POLARITY, 20  
FG\_TRIGGEROUT\_FRONT\_GPO\_0\_SOURCE, 19  
FG\_TRIGGEROUT\_FRONT\_GPO\_1\_POLARITY, 20  
FG\_TRIGGEROUT\_FRONT\_GPO\_1\_SOURCE, 19  
FG\_TRIGGEROUT\_GPO\_0\_POLARITY, 20  
FG\_TRIGGEROUT\_GPO\_0\_SOURCE, 19  
FG\_TRIGGEROUT\_GPO\_1\_POLARITY, 20  
FG\_TRIGGEROUT\_GPO\_1\_SOURCE, 19  
FG\_TRIGGEROUT\_GPO\_2\_POLARITY, 20  
FG\_TRIGGEROUT\_GPO\_2\_SOURCE, 19  
FG\_TRIGGEROUT\_GPO\_3\_POLARITY, 20  
FG\_TRIGGEROUT\_GPO\_3\_SOURCE, 19  
FG\_TRIGGEROUT\_GPO\_4\_POLARITY, 20  
FG\_TRIGGEROUT\_GPO\_4\_SOURCE, 19  
FG\_TRIGGEROUT\_GPO\_5\_POLARITY, 20  
FG\_TRIGGEROUT\_GPO\_5\_SOURCE, 19  
FG\_TRIGGEROUT\_GPO\_6\_POLARITY, 20  
FG\_TRIGGEROUT\_GPO\_6\_SOURCE, 19  
FG\_TRIGGEROUT\_GPO\_7\_POLARITY, 20  
FG\_TRIGGEROUT\_GPO\_7\_SOURCE, 19  
FG\_TRIGGER\_ACKNOWLEDGEMENT\_COUNT, 8  
FG\_TRIGGER\_EVENT\_COUNT, 7  
FG\_TRIGGER\_INPUT0\_FALLING, 26  
FG\_TRIGGER\_INPUT0\_RISING, 26  
FG\_TRIGGER\_WAVE\_VIOLATION, 8  
FG\_VANTAGEPOINT, 10  
FG\_WIDTH, 14  
FG\_XOFFSET, 15  
FG\_YOFFSET, 16  
Format, 86

## G

Generator, 90

## I

Image Select, 69  
Image Selector, 69  
Image Transfer, 4  
Image Trigger / Flash, 50  
Image Trigger / Flash::Image Trigger Input, 53  
Image Trigger / Flash::Image Trigger Input::Flash, 57  
Image Trigger / Flash::Image Trigger Input::Software Trigger, 57

## L

Line Trigger / ExSync, 28  
Line Trigger / ExSync::ExSync Output, 46  
Line Trigger / ExSync::Line Trigger Input, 30  
Line Trigger / ExSync::Line Trigger Input::Downscale, 33  
Line Trigger / ExSync::Shaft Encoder A/B Filter, 35  
Lookup Table, 73, 73  
Lookup Table::Applet Properties, 78

## **M**

Miscellaneous, 101  
Miscellaneous::Debug, 112  
Miscellaneous::Debug::Input, 114  
Miscellaneous::Debug::Output, 117

## **O**

Output Format, 86  
Overflow, 65, 65  
    Events, 66  
Overflow::Events, 66

## **P**

PC Interface, 4  
Pixel Format, 6  
Processing, 81  
Processor, 81

## **R**

Region of Interest, 13  
ROI, 13

## **S**

Sensor Geometry, 10, 10  
Signal Analyzer, 59, 59  
Software Interface, 5  
Specifications, 1

## **T**

Trigger  
    Digital Input, 22  
    Events, 26  
    Input, 22

## **W**

White Balance, 71, 71