

# imaFlex CXP-12 Quad

AcquisitionApplets User Documentation for  
**Acq\_DualCXP12Line**

Functional Description  
For Framegrabber SDK Usage

Document Number: AW001872  
Part Number: 000 (English)  
Document Version: 03  
Release Date: 25 February 2025  
Applet Version 1.2.2.0

# Contacting Basler Support Worldwide

## **Europe, Middle East, Africa**

Tel. +49 4102 463 515

[support.europe@baslerweb.com](mailto:support.europe@baslerweb.com)

## **The Americas**

Tel. +1 610 280 0171

[support.usa@baslerweb.com](mailto:support.usa@baslerweb.com)

## **Asia-Pacific**

Tel. +65 6367 1355

[support.asia@baslerweb.com](mailto:support.asia@baslerweb.com)

## **Singapore**

Tel. +65 6367 1355

[support.asia@baslerweb.com](mailto:support.asia@baslerweb.com)

## **Taiwan**

Tel. +886 3 558 3955

[support.asia@baslerweb.com](mailto:support.asia@baslerweb.com)

## **China**

Tel. +86 10 6295 2828

[support.asia@baslerweb.com](mailto:support.asia@baslerweb.com)

## **Korea**

Tel. +82 31 714 3114

[support.asia@baslerweb.com](mailto:support.asia@baslerweb.com)

## **Japan**

Tel. +81 3 6672 2333

[support.asia@baslerweb.com](mailto: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_DualCXP12Line .....	1
1.1.1. Parameterization Order .....	3
1.2. Bandwidth .....	3
1.3. Requirements .....	3
1.3.1. Software Requirements .....	4
1.3.2. Hardware Requirements .....	4
1.3.3. License .....	4
1.4. Camera Interface .....	4
1.5. Frame ID .....	4
1.6. Image Transfer to PC Memory .....	5
1.7. DMA Image Tag .....	5
2. Software Interface .....	6
3. CoaXPress .....	7
3.1. FG_SYSTEMMONITOR_POWER_OVER_CXP_STATE .....	7
3.2. FG_SYSTEMMONITOR_POWER_OVER_CXP_CONTROLLER_ENABLED .....	7
3.3. FG_SYSTEMMONITOR_PORT_BIT_RATE .....	8
3.4. FG_SYSTEMMONITOR_STREAM_PACKET_SIZE .....	8
3.5. FG_SYSTEMMONITOR_CXP_STANDARD .....	9
3.6. FG_CXP_STREAM_PACKET_COUNT .....	10
3.7. FG_PIXELFORMAT .....	10
3.8. FG_SYSTEMMONITOR_USED_CXP_CONNECTIONS .....	11
3.9. FG_SYSTEMMONITOR_CXP_IMAGE_LINE_MODE .....	12
3.10. Test .....	12
3.10.1. FG_CXP_TRANSMITTED_PACKET_COUNT .....	12
3.10.2. FG_CXP_RECEIVED_PACKET_COUNT .....	13
3.10.3. FG_CXP_CORRUPTED_WORD_COUNT .....	13
3.10.4. FG_CXP_PACKET_LENGTH_ERROR_COUNT .....	14
3.10.5. FG_CXP_CLEAR_TEST_STATISTIC_PORT .....	14
4. Camera .....	16
4.1. Events .....	16
4.1.1. FG_CAMERA_STREAM_STATUS0 .....	16
4.1.2. FG_START_OF_FRAME_CAM_PORT_0 .....	18
4.1.3. FG_END_OF_FRAME_CAM_PORT_0 .....	18
4.1.4. FG_START_OF_LINE_CAM_PORT_0 .....	18
4.1.5. FG_END_OF_LINE_CAM_PORT_0 .....	18
5. Sensor Geometry .....	19
5.1. FG_VANTAGEPOINT .....	19
5.2. FG_SENSORWIDTH .....	19
5.3. FG_SENSORHEIGHT .....	20
6. ROI .....	22
6.1. FG_WIDTH .....	23
6.2. FG_HEIGHT .....	24
6.3. FG_XOFFSET .....	24
6.4. FG_YOFFSET .....	25
7. Digital I/O .....	26
7.1. Camera .....	26
7.1.1. FG_TRIGGERCAMERA_SOURCE_CXP0 .....	27
7.1.2. FG_TRIGGERCAMERA_SOURCE_EDGE_CXP0 .....	27
7.1.3. FG_TRIGGERCAMERA_SOURCE_CXP1 .....	28
7.1.4. FG_TRIGGERCAMERA_SOURCE_EDGE_CXP1 .....	29
7.1.5. FG_TRIGGERCAMERA_SOURCE_CXP2 .....	30
7.1.6. FG_TRIGGERCAMERA_SOURCE_EDGE_CXP2 .....	31
7.1.7. FG_TRIGGERCAMERA_SOURCE_CXP3 .....	32
7.1.8. FG_TRIGGERCAMERA_SOURCE_EDGE_CXP3 .....	33
7.2. GPO .....	34

7.2.1. FG_TRIGGEROUT_GPO_0_SOURCE et al. ....	34
7.2.2. FG_TRIGGEROUT_GPO_0_POLARITY et al. ....	35
7.2.3. FG_TRIGGEROUT_FRONT_GPO_0_SOURCE et al. ....	36
7.2.4. FG_TRIGGEROUT_FRONT_GPO_0_POLARITY et al. ....	37
7.3. GPI .....	38
7.3.1. FG_DIGIO_INPUT .....	38
7.4. Event Source .....	39
7.4.1. FG_CUSTOM_SIGNAL_EVENT_0_SOURCE .....	39
7.4.2. FG_CUSTOM_SIGNAL_EVENT_0_POLARITY .....	40
7.4.3. FG_CUSTOM_SIGNAL_EVENT_1_SOURCE .....	40
7.4.4. FG_CUSTOM_SIGNAL_EVENT_1_POLARITY .....	41
7.5. Events .....	42
7.5.1. FG_TRIGGER_INPUT0_RISING .....	42
7.5.2. FG_TRIGGER_INPUT0_FALLING .....	42
7.5.3. FG_CUSTOM_SIGNAL_EVENT_0 .....	42
7.5.4. FG_CUSTOM_SIGNAL_EVENT_1 .....	43
8. Line Trigger / ExSync .....	44
8.1. FG_LINETRIGGERMODE .....	44
8.2. FG_EXSYNCON .....	45
8.3. Line Trigger Input .....	46
8.3.1. FG_LINETRIGGERINSRC .....	47
8.3.2. FG_LINETRIGGERINPOLARITY .....	48
8.3.3. FG_LINETRIGGERDEBOUNCING .....	49
8.3.4. Downscale .....	49
8.3.4.1. FG_LINE_DOWNSCALE .....	49
8.3.4.2. FG_LINE_DOWNSCALEINIT .....	50
8.4. Shaft Encoder A/B Filter .....	51
8.4.1. FG_SHAFTENCODERON .....	51
8.4.2. FG_SHAFTENCODERMODE .....	52
8.4.3. FG_SHAFTENCODERINSRC .....	53
8.4.4. FG_SHAFTENCODERLEADING .....	54
8.4.5. FG_SHAFTENCODER_COMPENSATION_ENABLE .....	55
8.4.6. FG_SHAFTENCODER_COMPENSATION_COUNT .....	56
8.5. ExSync Output .....	62
8.5.1. FG_LINEPERIODE .....	62
8.5.2. FG_LINEEXPOSURE .....	63
8.5.3. FG_EXSYNCPOLARITY .....	64
8.5.4. FG_LINETRIGGERDELAY .....	65
9. Image Trigger / Flash .....	66
9.1. FG_IMGTRIGGERMODE .....	67
9.2. FG_IMGTRIGGERON .....	67
9.3. FG_FLASHON .....	68
9.4. FG_IMGTRIGGER_ASYNC_HEIGHT .....	68
9.5. FG_IMGTRIGGER_IS_BUSY .....	69
9.6. Image Trigger Input .....	69
9.6.1. FG_IMGTRIGGERINSRC .....	70
9.6.2. FG_IMGTRIGGERINPOLARITY .....	70
9.6.3. FG_IMGTRIGGERGATEDELAY .....	71
9.6.4. FG_IMGTRIGGERDEBOUNCING .....	71
9.6.5. FG_STROBEPULSEDELAY .....	72
9.6.6. Flash .....	73
9.6.6.1. FG_FLASH_POLARITY .....	73
9.6.7. Software Trigger .....	73
9.6.7.1. FG_SENDSOFTWARETRIGGER .....	73
9.6.7.2. FG_SETSOFTWARETRIGGER .....	74
10. Signal Analyzer .....	75
10.1. FG_SIGNAL_ANALYZER_0_SOURCE et al. ....	75
10.2. FG_SIGNAL_ANALYZER_0_POLARITY et al. ....	76

10.3. FG_SIGNAL_ANALYZER_0_PERIOD_CURRENT et al. ....	77
10.4. FG_SIGNAL_ANALYZER_0_PERIOD_MAX et al. ....	78
10.5. FG_SIGNAL_ANALYZER_0_PERIOD_MIN et al. ....	78
10.6. FG_SIGNAL_ANALYZER_0_PULSE_COUNT et al. ....	79
10.7. FG_SIGNAL_ANALYZER_PULSE_COUNT_DIFFERENCE ....	80
10.8. FG_SIGNAL_ANALYZER_CLEAR ....	80
11. Overflow ....	82
11.1. FG_FILLLEVEL ....	82
11.2. FG_OVERFLOW ....	83
11.3. FG_OVERFLOW_OFF_THRESHOLD ....	83
11.4. FG_OVERFLOW_ON_THRESHOLD ....	84
11.5. FG_OVERFLOW_ON_SYNC_THRESHOLD ....	85
11.6. FG_OVERFLOW_EVENT_SELECT ....	85
11.7. Events ....	86
11.7.1. FG_OVERFLOW_CAM0 ....	87
12. Image Selector ....	88
12.1. FG_IMG_SELECT_PERIOD ....	88
12.2. FG_IMG_SELECT ....	89
13. White Balance ....	90
13.1. FG_SCALINGFACTOR_GREEN ....	90
13.2. FG_SCALINGFACTOR_RED ....	90
13.3. FG_SCALINGFACTOR_BLUE ....	91
14. Color Converter ....	92
15. Lookup Table ....	93
15.1. FG_LUT_ENABLE ....	93
15.2. FG_LUT_TYPE ....	94
15.3. FG_LUT_VALUE ....	94
15.4. FG_LUT_VALUE_RED ....	95
15.5. FG_LUT_VALUE_GREEN ....	95
15.6. FG_LUT_VALUE_BLUE ....	96
15.7. FG_LUT_CUSTOM_FILE ....	97
15.8. FG_LUT_SAVE_FILE ....	98
15.9. Applet Properties ....	99
15.9.1. FG_LUT_IMPLEMENTATION_TYPE ....	99
15.9.2. FG_LUT_IN_BITS ....	99
15.9.3. FG_LUT_OUT_BITS ....	100
16. Processing ....	101
16.1. FG_PROCESSING_OFFSET ....	102
16.2. FG_PROCESSING_GAIN ....	102
16.3. FG_PROCESSING_GAMMA ....	103
16.4. FG_PROCESSING_INVERT ....	104
17. Output Format ....	106
17.1. FG_FORMAT ....	106
17.2. FG_BITALIGNMENT ....	109
17.3. FG_PIXELDEPTH ....	109
17.4. FG_CUSTOM_BIT_SHIFT_RIGHT ....	110
18. Camera Simulator ....	112
18.1. FG_CAMERASIMULATOR_ENABLE ....	112
18.2. FG_CAMERASIMULATOR_WIDTH ....	113
18.3. FG_CAMERASIMULATOR_LINE_GAP ....	114
18.4. FG_CAMERASIMULATOR_HEIGHT ....	114
18.5. FG_CAMERASIMULATOR_FRAME_GAP ....	115
18.6. FG_CAMERASIMULATOR_PATTERN ....	116
18.7. FG_CAMERASIMULATOR_PATTERN_OFFSET ....	116
18.8. FG_CAMERASIMULATOR_ROLL ....	117
18.9. FG_CAMERASIMULATOR_SELECT_MODE ....	118
18.10. FG_CAMERASIMULATOR_PIXEL_FREQUENCY ....	118
18.11. FG_CAMERASIMULATOR_LINERATE ....	119

18.12. FG_CAMERASIMULATOR_FRAMERATE .....	119
18.13. FG_CAMERASIMULATOR_TRIGGER_MODE .....	120
18.14. FG_CAMERASIMULATOR_ACTIVE .....	121
18.15. FG_CAMERASIMULATOR_PASSIVE .....	121
19. Miscellaneous .....	123
19.1. FG_TIMEOUT .....	123
19.2. FG_APPLET_ID .....	123
19.3. FG_APPLET_BUILD_TIME .....	124
19.4. FG_HAP_FILE .....	124
19.5. FG_CAMSTATUS .....	124
19.6. FG_CAMSTATUS_EXTENDED .....	125
19.7. FG_SYSTEMMONITOR_FPGA_DNA_LOW .....	126
19.8. FG_SYSTEMMONITOR_FPGA_DNA_HIGH .....	126
19.9. Version .....	126
19.9.1. FG_APPLET_VERSION .....	127
19.9.2. FG_APPLET_REVISION .....	127
19.9.3. FG_VISUALAPPLETS_BUILD_VERSION .....	127
19.10. Legacy .....	128
19.10.1. FG_CXP_TRIGGER_PACKET_MODE .....	128
19.10.2. FG_TRIGGERCAMERA_SOURCE .....	129
19.10.3. FG_TRIGGERCAMERA_POLARITY .....	130
19.11. Debug .....	130
19.11.1. FG_DEBUGSOURCE .....	131
19.11.2. FG_DEBUGSOURCENAME .....	131
19.11.3. FG_DEBUGSAVECONFIG .....	131
19.11.4. FG_DEBUG_SLOWMODE .....	132
19.11.5. FG_DEBUG_SOFTWRAE_SLOWGATE .....	132
19.11.6. FG_DEBUG_PWM_SLOWRATE .....	133
19.11.7. FG_DEBUG_VERSION .....	133
19.11.8. FG_DEBUG_FRAMEID_TO_FIRSTPIXEL .....	134
19.11.9. Input .....	134
19.11.9.1. FG_DEBUGINENABLE .....	134
19.11.9.2. FG_DEBUGFILE .....	135
19.11.9.3. FG_DEBUGINSERT .....	135
19.11.9.4. FG_DEBUGWRITEPIXEL .....	136
19.11.9.5. FG_DEBUGWRITEFLAG .....	136
19.11.9.6. FG_DEBUGREADY .....	137
19.11.9.7. FG_DEBUG_FORCE_FRAMEID .....	137
19.11.9.8. FG_DEBUG_FRAMEID .....	138
19.11.9.9. FG_DEBUG_ENABLE_SEQUENCEID .....	138
19.11.10. Output .....	139
19.11.10.1. FG_DEBUGOUTENABLE .....	139
19.11.10.2. FG_DEBUGOUTXPOS .....	140
19.11.10.3. FG_DEBUGOUTYPOS .....	140
19.11.10.4. FG_DEBUGOUTPIXEL .....	140
19.12. GenTL .....	141
19.12.1. FG_GENTL_INFO_VERSION .....	141
19.12.2. FG_GENTL_INFO_IGNOREFGFORMAT .....	141
19.12.3. FG_GENTL_INFO_OVERFLOWCAPABLE .....	142
20. Boardstatus .....	143
20.1. FG_SYSTEMMONITOR_CHANNEL_CURRENT .....	143
20.2. FG_SYSTEMMONITOR_CHANNEL_VOLTAGE .....	143
20.3. FG_SYSTEMMONITOR_MAPPED_TO_FG_PORT .....	144
20.4. FG_DMASTATUS .....	144
20.5. FG_SYSTEMMONITOR_FPGA_TEMPERATURE .....	145
20.6. FG_SYSTEMMONITOR_FPGA_VCC_INT .....	145
20.7. FG_SYSTEMMONITOR_FPGA_VCC_AUX .....	146
20.8. FG_SYSTEMMONITOR_FPGA_VCC_BRAM .....	146

20.9. FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH .....	147
20.10. FG_SYSTEMMONITOR_CURRENT_LINK_SPEED .....	147
20.11. FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE .....	148
20.12. FG_SYSTEMMONITOR_PCIE_TRAINED_REQUEST_SIZE .....	148
20.13. FG_SYSTEMMONITOR_EXTERNAL_POWER .....	149
20.14. FG_CXP_INPUT_MAPPED_FW_PORT_PORT .....	149
21. Errors .....	150
21.1. FG_SYSTEMMONITOR_DECODER_8B_10B_ERROR .....	150
21.2. FG_SYSTEMMONITOR_BYTE_ALIGNMENT_8B_10B_LOCKED .....	150
21.3. FG_SYSTEMMONITOR_RX_STREAM_INCOMPLETE_COUNT .....	151
21.4. FG_SYSTEMMONITOR_RX_UNKNOWN_DATA_RECEIVED_COUNT .....	151
21.5. FG_CXP_OVERTRIGGER_REQUEST_PULSECOUNT .....	152
21.6. FG_CXP_TRIGGER_ACK_MISSING_COUNT .....	152
21.7. FG_CXP_CONTROL_ACK_LOST_COUNT .....	153
21.8. FG_CXP_CONTROL_TAG_ERROR_COUNT .....	153
21.9. FG_CXP_CONTROL_ACK_INCOMPLETE_COUNT .....	154
21.10. FG_CXP_HEARTBEAT_INCOMPLETE_COUNT .....	154
21.11. FG_CXP_HEARTBEAT_MAX_PERIOD_VIOLATION_COUNT .....	155
21.12. FG_PACKET_TAG_ERROR_COUNT .....	155
21.13. FG_SYSTEMMONITOR_PACKETBUFFER_OVERFLOW_COUNT .....	156
21.14. FG_SYSTEMMONITOR_PACKETBUFFER_OVERFLOW_SOURCE .....	156
21.15. FG_CXP_IMAGETAG_ERROR_COUNT .....	157
21.16. FG_CXP_STREAMID_ERROR_COUNT .....	157
21.17. FG_CXP_CAMERA_MARKER_ERROR_COUNT .....	158
21.18. FG_CXP_CAMERA_UNEXPECTED_STARTUP_DATA .....	158
21.19. FG_CXP_CAMERA_FRAME_LOST_COUNT .....	159
21.20. FG_CXP_CAMERA_FRAME_CORRUPT_COUNT .....	159
21.21. CRC .....	160
21.21.1. FG_SYSTEMMONITOR_RX_PACKET_CRC_ERROR_COUNT .....	160
21.21.2. FG_CXP_STREAMPACKET_CRC_ERROR .....	160
21.21.3. FG_CXP_CONTROL_ACK_PACKET_CRC_ERROR .....	161
21.22. LengthErrors .....	161
21.22.1. FG_SYSTEMMONITOR_RX_LENGTH_ERROR_COUNT .....	161
21.22.2. FG_CXP_STREAMPACKET_LENGTH_ERROR .....	162
21.23. ReceivedPacketsCorrected .....	162
21.23.1. FG_CXP_ERROR_CORRECTED .....	162
21.23.2. FG_CXP_ERROR_CORRECTED_TRIGGER .....	163
21.23.3. FG_CXP_ERROR_CORRECTED_TRIGGER_ACK .....	163
21.23.4. FG_CXP_ERROR_CORRECTED_STREAM .....	164
21.23.5. FG_CXP_ERROR_CORRECTED_CONTROL_ACK .....	164
21.23.6. FG_CXP_ERROR_CORRECTED_LINKTEST .....	165
21.23.7. FG_CXP_ERROR_CORRECTED_HEARTBEAT .....	165
21.23.8. FG_CORRECTED_ERROR_COUNT .....	166
21.24. ReceivedPacketsUncorrected .....	166
21.24.1. FG_CXP_ERROR_UNCORRECTED .....	166
21.24.2. FG_CXP_ERROR_UNCORRECTED_TRIGGER .....	167
21.24.3. FG_CXP_ERROR_UNCORRECTED_TRIGGER_ACK .....	167
21.24.4. FG_CXP_ERROR_UNCORRECTED_STREAM .....	168
21.24.5. FG_CXP_ERROR_UNCORRECTED_CONTROL_ACK .....	168
21.24.6. FG_CXP_ERROR_UNCORRECTED_LINKTEST .....	169
21.24.7. FG_CXP_ERROR_UNCORRECTED_HEARTBEAT .....	169
21.24.8. FG_UNCORRECTED_ERROR_COUNT .....	170
21.25. UnsupportedPackets .....	170
21.25.1. FG_SYSTEMMONITOR_RX_UNSUPPORTED_PACKET_COUNT .....	170
21.25.2. FG_CXP_UNSUPPORTED_GPIO_RECEIVED .....	171
21.25.3. FG_CXP_UNSUPPORTED_EVENT_RECEIVED .....	171
21.25.4. FG_CXP_UNSUPPORTED_HEARTBEAT_RECEIVED .....	171
21.25.5. FG_CXP_UNSUPPORTED_GPIO_ACK_RECEIVED .....	172

21.25.6. FG_CXP_UNSUPPORTED_GPIO_REQUEST_RECEIVED .....	172
22. Revision History .....	174
22.1. Fixed Issues .....	174
22.1.1. Fixed in Version 1.2.2.0 .....	174
22.1.2. Fixed in Version 1.1.1.0 .....	174
22.2. Known Issues .....	175
Glossary .....	176
Index .....	179



---

# Chapter 1. Introduction

This document provides you with detailed information on applet "Acq\_DualCXP12Line" for imaFlex CXP-12 Quad 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\_DualCXP12Line

"Acq\_DualCXP12Line" is a dual-camera applet. Up to two individual cameras can be used in parallel. All features of this applet are available for both camera ports. You can configure the CoaXPress camera interface for CoaXPress cameras version 1.1.1 and 2.0, transferring grayscale (monochrome), BiColor pattern according to PFNC , or color pixels. Allowed pixel formats are Gray (Mono8, Mono10, Mono12, Mono14, Mono16), Color (RGB8, RGB10, RGB12, RGB14, RGB16), biColor (BiColorRGBG8, BiColorRGBG10, BiColorRGBG12, BiColorGRGB8, BiColorGRGB10, BiColorGRGB12, BiColorBGRG8, BiColorBGRG10, BiColorBGRG12, BiColorGBGR8, BiColorGBGR10, BiColorGBGR12) and YCbCr422\_8. You can use cameras with CoaXPress link aggregation of 2 and 1 with this applet. The maximum link speed is CXP-12. A multi-functional line trigger is included in the applet. This allows you to control the camera or external devices using frame grabber generated, external or software generated trigger pulses. Line scan cameras up to a width of 32768 pixels can be processed. The trigger system will generate images of a maximum height of 8388607 pixels. The applet is processing data at a bit depth of 16 bits. An image selector at the camera port facilitates the selection of one image out of a parameterizable sequence of images. This enables the distribution of the images to multiple frame grabber and PCs. For reverse operation, you can mirror the image in x-direction and y-direction before cutting the ROI. Acquired images are buffered in frame grabber memory. You can select a region of interest (ROI) for further processing. The stepsize of the ROI width is 16 pixel. The ROI stepsize for the image height is 1 line. This applets includes the special color interpolation filter for bilinear color linescan cameras. The first line is blue red, the second line is green only. A color converter automatically converts the input pixel formats to the output formats. In this applet conversions from monochrome, RGB or BiColor to monochrome and RGB can be performed. You can configure the 14 bit full resolution lookup table either by using a user defined table, or by using a processor. The processor gives you the opportunity to use pre-defined functions such as offset, gain, invert to enhance the image quality. The color components are processed individually. A gamma correction is possible.

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

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

Table 1.1. Feature Summary of Acq\_DualCXP12Line

Feature	Applet Property
Applet Name	 Acq_DualCXP12Line
Type of Applet	AcquisitionApplets
Board	imaFlex CXP-12 Quad
No. of Cameras	2 , asynchronous or synchronous
Camera Type	CoaXPress, link aggregation max. 2, maximum speed CXP-12, Version 1.1.1 and 2.0
Sensor Type	Line Scan
Camera Format	Monochrome, BiColor or RGB
Pixel Format	Gray (Mono8, Mono10, Mono12, Mono14, Mono16), Color (RGB8, RGB10, RGB12, RGB14, RGB16), biColor (BiColorRGBG8, BiColorRGBG10, BiColorRGBG12, BiColorGRGB8, BiColorGRGB10, BiColorGRGB12, BiColorBGRG8, BiColorBGRG10, BiColorBGRG12, BiColorGBGR8, BiColorGBGR10, BiColorGBGR12) and YCbCr422_8.
Processing Bit Depth	16 Bit per color component
Sensor Correction / Tap Sorting	no
Maximum Images Dimensions	32768 * 8388607
ROI Stepsize	x: 16, y: 1
Tap Geometry Sorting	1X-1Y only
Mirroring	Yes, horizontal and vertical (set the parameter <i>FG_VANTAGEPOINT</i> )
Image Selector	Yes
Noise Filter	No
Shading Correction	No
Dead Pixel Interpolation	No
Color Array Filter	Two lines. First Blue and Red, second Green. (or swapped)
Color White Balancing	Yes
Color Converter	yes, Mono, RGB or BiColor to Mono or RGB
Lookup Table	Full Resolution  Input bits = 14, Output bits = 16  Lookup table can be disabled.
DMA	Full Speed
DMA Image Output Format	All grayscale and color formats. See description above.
Event Generation	yes
Overflow Control	yes

### 1.1.1. Parameterization Order

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

## 1.2. Bandwidth

The maximum bandwidths of applet Acq\_DualCXP12Line are listed in the following table.

Table 1.2. Bandwidth of Acq\_DualCXP12Line

Description	Bandwidth
Max. CXP Speed	CXP-12
Peak Bandwidth per Camera	2400 MPixel/s
Mean Bandwidth per Camera	2400 MPixel/s
DMA Bandwidth	7200 MByte/s (depends on PC mainboard)

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

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

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

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



### Bandwidth Varies

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

## 1.3. Requirements

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

### 1.3.1. Software Requirements

To run this applet, a Basler 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.

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

### 1.3.2. Hardware Requirements

To run applet "Acq\_DualCXP12Line", a Basler imaFlex CXP-12 Quad frame grabber is required.

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

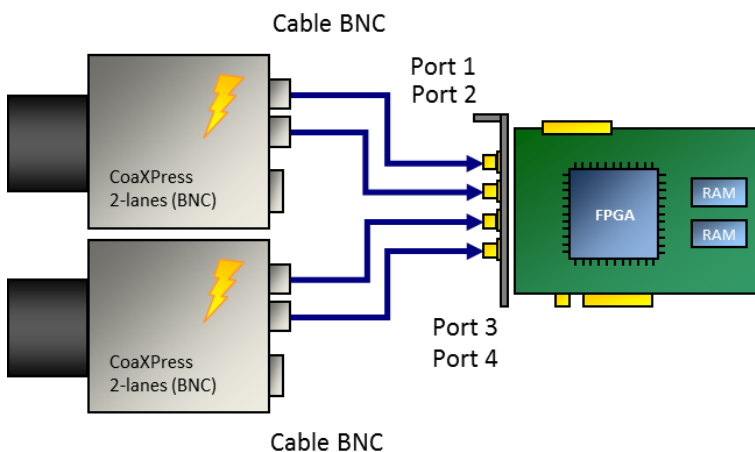
### 1.3.3. License

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

## 1.4. Camera Interface

Applet "Acq\_DualCXP12Line" supports 2 CXP cameras. The frame grabber has 4 connectors. Use two or one CoaXPress cables to connect each of the cameras with the frame grabber. The maximum link aggregation of this applet is two. The mapping of the ports between the camera and the frame grabber is not important. You can chose any order.

Figure 1.1. Camera Interface and Camera Cable Setup



## 1.5. Frame ID

For CoaXPress linescan cameras the CXP Source Tag is not used as it is constant throughout the acquisition. Instead an internal counter is used to represent frame IDs. This applet will output each frame to the host PC attached with this frame ID. Moreover, overflow events will also include this frame ID. By this, the exact mapping of a given frame in the host PC to the frame the frame grabber's image trigger is possible.

Check chapter Chapter 11, '*Overflow*' for more information about overflow conditions and the overflow event data structure including the frame ID.

Check chapter Section 1.7, 'DMA Image Tag' to get information on how to obtain the frame ID along with a given image in the host PC application.

## 1.6. Image Transfer to PC Memory

The image transfer between frame grabber and PC is performed via DMA transfers. In this applet, 2 DMA channels exist for transferring image data. One channel for each camera. The DMA channels have the same indices as the cameras, starting with 0. The applet output format can be set via the parameters of the output format module. See Chapter 17, '*Output Format*'. All outputs are little-endian coded.

## 1.7. DMA Image Tag

The applet generates a DMA image tag (**FG\_IMAGE\_TAG**) for every correct transmitted frame. The **FG\_IMAGE\_TAG** has the following structure:

Table 1.3. Structure of FG\_IMAGE\_TAG

Bits	Description
0..15	frameID transmitted by the Camera
16..29	reserved = 0
30	invalid image flag (the image was cut of due to overflow in the framegrabber)
31	Last Image of Multi Buffer Sequence (always 1 in case of an area applet)

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 dual camera applet and is using only one DMA channel for each camera. All parameterizations for the first camera are made using index 0 and all parameterizations for the second camera are made using index 1.

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 up to 2 line scan cameras. To receive correct image data from your camera, it is crucial that the camera output format matches the selected frame grabber input format. The following parameters configure the frame grabber's camera interface to match with the individual camera pixel format. Most cameras support different operation modes. Consult the manual of your camera to obtain the necessary information how to configure the camera to the desired pixel format.

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

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

## 3.1. FG\_SYSTEMMONITOR\_POWER\_OVER\_CXP\_STATE

Returns the power over CXP (PoCXP) state. Range: {BOOTING, POCXPOK, MAX\_CURR, LOW\_VOLT, OVER\_VOLT, ADC\_Chip\_Error}. The first 5 states are defined by the CXP standard for the PoCXP state machine. The last state ADC\_Chip\_Error represents an error when the communication between the FPGA and the ADC chip is disrupted. The communication between the FPGA and ADC chip measures the voltage and current of the channel.

Table 3.1. Parameter properties of FG\_SYSTEMMONITOR\_POWER\_OVER\_CXP\_STATE

Property	Value
Name	FG_SYSTEMMONITOR_POWER_OVER_CXP_STATE
Display Name	Systemmonitor Power Over CXP State
Type	Unsigned Integer Field
Field Size	4
Access policy	Read-Only
Storage policy	Transient

Example 3.1. Usage of FG\_SYSTEMMONITOR\_POWER\_OVER\_CXP\_STATE

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

## 3.2. FG\_SYSTEMMONITOR\_POWER\_OVER\_CXP\_CONTROLLER\_ENABLED

Indicates whether the power over CXP (PoCXP) controller is enabled. Range: {NO, YES}. YES: The camera is powered via the CXP cable when connected. NO: The camera is not powered via the CXP cable. This parameter doesn't indicate whether the camera is sourced or not, instead it indicates whether powering the camera via the CXP cable is enabled or not.

Table 3.2. Parameter properties of FG\_SYSTEMMONITOR\_POWER\_OVER\_CXP\_CONTROLLER\_ENABLED

Property	Value
Name	<b>FG_SYSTEMMONITOR_POWER_OVER_CXP_CONTROLLER_ENABLED</b>
Display Name	<b>Systemmonitor Power Over CXP Controller Enabled</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 3.2. Usage of FG\_SYSTEMMONITOR\_POWER\_OVER\_CXP\_CONTROLLER\_ENABLED

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

### 3.3. FG\_SYSTEMMONITOR\_PORT\_BIT\_RATE

Returns the port bit rate of the CXP channel.

Table 3.3. Parameter properties of FG\_SYSTEMMONITOR\_PORT\_BIT\_RATE

Property	Value
Name	<b>FG_SYSTEMMONITOR_PORT_BIT_RATE</b>
Display Name	<b>Systemmonitor Port Bit Rate</b>
Type	<b>Double Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Unit of measure	<b>Gb/s</b>

Example 3.3. Usage of FG\_SYSTEMMONITOR\_PORT\_BIT\_RATE

```

int result = 0;

FieldParameterDouble access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMDOUBLE;

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

```

### 3.4. FG\_SYSTEMMONITOR\_STREAM\_PACKET\_SIZE

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



Table 3.4. Parameter properties of FG\_SYSTEMMONITOR\_STREAM\_PACKET\_SIZE

Property	Value
Name	<b>FG_SYSTEMMONITOR_STREAM_PACKET_SIZE</b>
Display Name	<b>Systemmonitor Stream Packet Size</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 3.4. Usage of FG\_SYSTEMMONITOR\_STREAM\_PACKET\_SIZE

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

### 3.5. FG\_SYSTEMMONITOR\_CXP\_STANDARD

Returns the version of the used CXP standard.

Table 3.5. CXP Standard Version

CXP Standard Version		
CXP_1_0		
CXP_1_1_1		
CXP_2_0		
Unknown		

Table 3.6. Parameter properties of FG\_SYSTEMMONITOR\_CXP\_STANDARD

Property	Value
Name	<b>FG_SYSTEMMONITOR_CXP_STANDARD</b>
Display Name	<b>Systemmonitor CXP Standard</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 3.5. Usage of FG\_SYSTEMMONITOR\_CXP\_STANDARD

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

### 3.6. FG\_CXP\_STREAM\_PACKET\_COUNT

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

Table 3.7. Parameter properties of FG\_CXP\_STREAM\_PACKET\_COUNT

Property	Value
Name	<b>FG_CXP_STREAM_PACKET_COUNT</b>
Display Name	<b>CXP Stream Packet Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 3.6. Usage of FG\_CXP\_STREAM\_PACKET\_COUNT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

### 3.7. 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.

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

Table 3.8. 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>BiColorRGBG8</b> BiColor RG BG 8 <b>BiColorRGBG10</b> BiColor RG BG 10 <b>BiColorRGBG12</b> BiColor RG BG 12 <b>BiColorBGRG8</b> BiColor BG RG 8 <b>BiColorBGRG10</b> BiColor BG RG 10 <b>BiColorBGRG12</b> BiColor BG RG 12 <b>Mono8</b> Mono 8 <b>Mono10</b> Mono 10p <b>Mono12</b> Mono 12p <b>Mono14</b> Mono 14p <b>Mono16</b> Mono 16p <b>RGB8</b> RGB 8 <b>RGB10</b> RGB 10p <b>RGB12</b> RGB 12p <b>RGB14</b> RGB 14p <b>RGB16</b> RGB 16 <b>YUV422_8</b> YCbCr422_8
Default value	<b>Mono8</b>

Example 3.7. Usage of FG\_PIXELFORMAT

```

int result = 0;
int value = Mono8;
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 */
}

```

### 3.8. FG\_SYSTEMMONITOR\_USED\_CXP\_CONNECTIONS

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

Table 3.9. Parameter properties of FG\_SYSTEMMONITOR\_USED\_CXP\_CONNECTIONS

Property	Value
Name	<b>FG_SYSTEMMONITOR_USED_CXP_CONNECTIONS</b>
Display Name	<b>System Monitor Used Cxp Connections</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 1 <b>Maximum</b> 4 <b>Stepsize</b> 1

Example 3.8. Usage of FG\_SYSTEMMONITOR\_USED\_CXP\_CONNECTIONS

```

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

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

```

### 3.9. FG\_SYSTEMMONITOR\_CXP\_IMAGE\_LINE\_MODE

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

Table 3.10. Parameter properties of FG\_SYSTEMMONITOR\_CXP\_IMAGE\_LINE\_MODE

Property	Value
Name	<b>FG_SYSTEMMONITOR_CXP_IMAGE_LINE_MODE</b>
Display Name	<b>System Monitor Cxp Image Line Mode</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 1</b> <b>Stepsize 1</b>

Example 3.9. Usage of FG\_SYSTEMMONITOR\_CXP\_IMAGE\_LINE\_MODE

```

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

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

```

### 3.10. Test

This category gives information about CXP link test statistics. It shows counters for received and transmitted CXP packets as well as statistics on corrupted words and length errors. Additionally, it enables the user to reset the statistics counter for each channel separately.

#### 3.10.1. FG\_CXP\_TRANSMITTED\_PACKET\_COUNT

This parameter counts the amount of link test packets that were transmitted. This register is useful to see how many packets were sent since the start of the test. It is cleared with the clear bit from the test statistics clear register.

Table 3.11. Parameter properties of FG\_CXP\_TRANSMITTED\_PACKET\_COUNT

Property	Value
Name	<b>FG_CXP_TRANSMITTED_PACKET_COUNT</b>
Display Name	<b>CXP Transmitted Packets Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>

Example 3.10. Usage of FG\_CXP\_TRANSMITTED\_PACKET\_COUNT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

### 3.10.2. FG\_CXP\_RECEIVED\_PACKET\_COUNT

This parameter counts the amount of received link test packets. It is cleared with the clear bit from the test statistics clear register.

Table 3.12. Parameter properties of FG\_CXP\_RECEIVED\_PACKET\_COUNT

Property	Value
Name	<b>FG_CXP_RECEIVED_PACKET_COUNT</b>
Display Name	<b>CXP Received Packet Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>

Example 3.11. Usage of FG\_CXP\_RECEIVED\_PACKET\_COUNT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

### 3.10.3. FG\_CXP\_CORRUPTED\_WORD\_COUNT

This parameter counts the amount of measured packet word errors. A packet word is a 32-bit CXP native word which carries 4 test characters. It is cleared with the clear bit from the test statistics clear register.

Table 3.13. Parameter properties of FG\_CXP\_CORRUPTED\_WORD\_COUNT

Property	Value
Name	<b>FG_CXP_CORRUPTED_WORD_COUNT</b>
Display Name	<b>CXP Corrupted Word Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>

Example 3.12. Usage of FG\_CXP\_CORRUPTED\_WORD\_COUNT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

### 3.10.4. FG\_CXP\_PACKET\_LENGTH\_ERROR\_COUNT

This parameter counts the amount of packets which didn't provide 1024 test words. CXP standard defines a test packet to contain 4096 test characters, i.e. 1024 x 32 bit words. This packet is repeated infinitely until the test is terminated. The count range is [0; 128]. The maximal value 128 means that there were at least 128 or more packets which violated the length requirements as defined in CXP 2.0 standard chapter 9.9.2.

Table 3.14. Parameter properties of FG\_CXP\_PACKET\_LENGTH\_ERROR\_COUNT

Property	Value
Name	<b>FG_CXP_PACKET_LENGTH_ERROR_COUNT</b>
Display Name	<b>CXP Packet Length Error Count</b>
Type	<b>Unsigned Integer Field (64 Bit)</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>

Example 3.13. Usage of FG\_CXP\_PACKET\_LENGTH\_ERROR\_COUNT

```

int result = 0;

FieldParameterAccess access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMACCESS;

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

```

### 3.10.5. FG\_CXP\_CLEAR\_TEST\_STATISTIC\_PORT

This parameter clears all test counters to zero for the channel selected by the parameter field index.

Table 3.15. Parameter properties of FG\_CXP\_CLEAR\_TEST\_STATISTIC\_PORT

Property	Value
Name	<b>FG_CXP_CLEAR_TEST_STATISTIC_PORT</b>
Display Name	<b>CXP Clear Test Statistic Port</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Default value	<b>0</b>

Example 3.14. Usage of FG\_CXP\_CLEAR\_TEST\_STATISTIC\_PORT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

for (unsigned int i = 0; i < 4; ++i)
{
    access.index = i;
    access.value = 0;

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

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

```

---

# Chapter 4. Camera

This applet Acq\_DualCXP12Line for the imaFlex CXP-12 Quad acquires the sensor data of a line scan camera. When this is performed some sensor dimension depending information can be used to register an event based callback function.

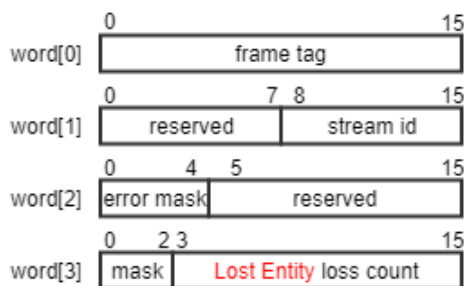
## 4.1. Events

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\_CAMERA\_STREAM\_STATUS0

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



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

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



- bit [2]: An error in the image header was detected which could not be corrected.
- bit [3]: A frame size error was detected, i.e. the image size defined in the CXP image header isn't matching the reconstructed frame size from the transmitted packets. This happens when the camera puts one info into the image header but transmits different amount of data as promoted in the header.
- bits [4:15]: Reserved, treat as don't care.
- word [3]
  - bit [0]: Event type, 0 = Corrupted Entity , 1 = Lost Entity.
    - **Corrupted Entity** means that the error happens within a frame and that this frame is already sourced into the VisualApplets pipeline.
    - **Lost Entity** means that the error occurred before the frame was forwarded to the following operators and the frame was discarded by the camera operator.
    - When a corrupted entity is observed, the operator will fill up the frame according to the CXP image header definition so that the following operators will not cause undefined behavior. During this fill-up, a new frame may arrive and will then get lost. The lost entity event will also be raised when the camera sends data with a gap according to the frame tag.
  - bit [1]: An event loss for type **Corrupted Entity** occurred. This means that preceding events of type **Corrupted Entity** got lost. This happens when the runtime software is not reacting to events and the internal event queues ran full.
  - bit [2]: An event loss for type **Lost Entity** occurred. This means that preceding events of type **Lost Entity** got lost. This happens when the runtime software (Framegrabber SDK) is not reacting to events and the internal event queues ran full.
  - bits [3:15]: amount of lost **Lost Entity** events.

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

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

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



## Differentiating Error Events Between Taps

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

#### **4.1.2. FG\_START\_OF\_FRAME\_CAM\_PORT\_0**

#### **4.1.3. FG\_END\_OF\_FRAME\_CAM\_PORT\_0**

#### **4.1.4. FG\_START\_OF\_LINE\_CAM\_PORT\_0**

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

#### **4.1.5. FG\_END\_OF\_LINE\_CAM\_PORT\_0**

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

# Chapter 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. The parameter dependency will then be *FG\_XOFFSET + FG\_WIDTH <= FG\_SENSORWIDTH*.

If a vertical mirroring is active, the parameter *FG\_SENSORHEIGHT* limits the maximum height. The parameter dependency will then be *FG\_YOFFSET + FG\_HEIGHT <= FG\_SENSORHEIGHT*.

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</b> 48 <b>Maximum</b> 32768 <b>Stepsize</b> 16
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 vertical mirroring or tap geometry sorting in vertical direction, the applet needs to be parameterized with the exact height transferred from the camera to the frame grabber. If you have set a region of interest in the camera, the parameter *FG\_SENSORHEIGHT* needs to be set to the ROI size, otherwise use the sensor height.



## If Only One Y-Zone Is Used and No Vertical Mirroring Is Active, the Value of *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 */
}

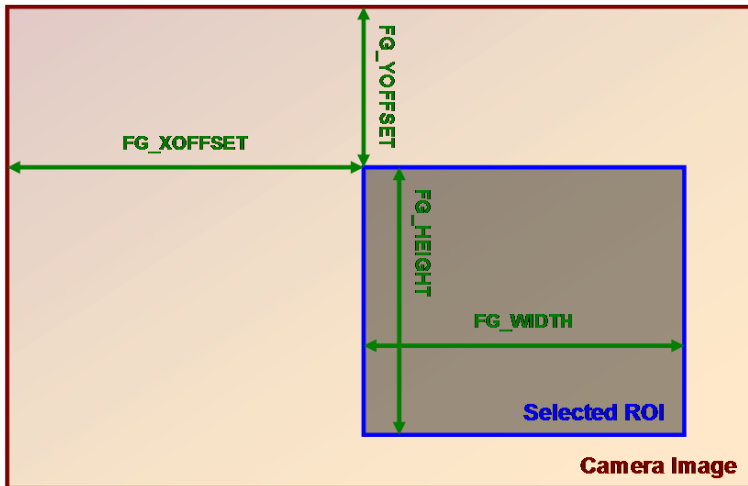
if ((result = Fg_getParameterWithType(fg, FG_SENSORHEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

```

# Chapter 6. ROI

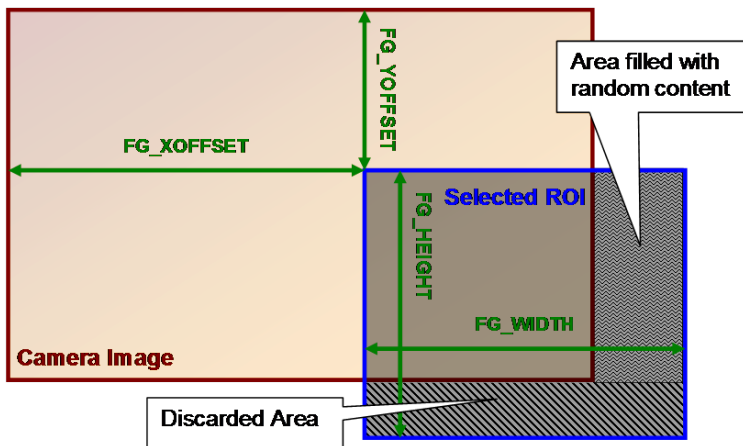
This module allows the definition of a region of interest (ROI), also called area of interest (AOI). A ROI allows the selection of a smaller subset pixel area from the input image. It is defined by using parameters *FG\_XOFFSET*, *FG\_WIDTH*, *FG\_YOFFSET* and *FG\_HEIGHT*. The following figure illustrates 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 32768 pixels and a maximum image height of 8388607 lines. The chosen ROI settings can have a direct influence on the maximum bandwidth of the applet as they define the image size and thus, define the amount of data.

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

Mind the minimum step size of the parameters. This applet has a minimum step size of 16 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 32768 pixels. If a horizontal mirroring is active the sensor width limits the maximum width (Width + XOffset). If furthermore vertical mirroring is active the maximum width is limited by the DRAM and sensor height (the sensor dimension needs to fit into the DRAM).



## Maximum image width is reduced for horizontal mirrored images

Limitations of the available BRAM in the FPGA allow only to store smaller lines and there for the images that can be mirrored needs to be smaller. A mirrored image can only have width of 21840, the not mirrored image can have the full width of 32768.

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</b> 48 <b>Maximum</b> 32768 <b>Stepsize</b> 16
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</b> 1 <b>Maximum</b> 8388607 <b>Stepsize</b> 1
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>Offset X</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 32720 <b>Stepsize</b> 16
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>Offset Y</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> <b>0</b> <b>Maximum</b> <b>8388606</b> <b>Stepsize</b> <b>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 imaFlex CXP-12 Quad frame grabber has

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

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

- **GND:** Value set to GND, zero. For digital outputs check for possibly inverted outputs.
- **VCC:** Value set to VCC, one. For digital outputs check for possibly inverted outputs.
- **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.
- Multi camera applet signal sources: The above signal source are available for all camera processes. Thus you can arbitrary select each signal. For example, you can use the same internal exsync signal to trigger all cameras. This allows a 100% synchronization of the cameras.
- **FG\_SIGNAL\_GPI\_0** to **FG\_SIGNAL\_GPI\_7:** 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

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

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

### 7.1.1. FG\_TRIGGERCAMERA\_SOURCE\_CXP0

Table 7.1. Parameter properties of FG\_TRIGGERCAMERA\_SOURCE\_CXP0

Property	Value
Name	FG_TRIGGERCAMERA_SOURCE_CXP0
Display Name	CXP Link Trigger 0 Source
Type	Enumeration
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	<div> <div>GND</div> <div>VCC</div> <div>FG_SIGNAL_CAM0_EXSYNC</div> <div>FG_SIGNAL_CAM0_EXSYNC2</div> <div>FG_SIGNAL_CAM0_FLASH</div> <div>FG_SIGNAL_CAM0_LVAL</div> <div>FG_SIGNAL_CAM0_FVAL</div> <div>FG_SIGNAL_CAM1_EXSYNC</div> <div>FG_SIGNAL_CAM1_EXSYNC2</div> <div>FG_SIGNAL_CAM1_FLASH</div> <div>FG_SIGNAL_CAM1_LVAL</div> <div>FG_SIGNAL_CAM1_FVAL</div> <div>FG_SIGNAL_GPI_0</div> <div>FG_SIGNAL_GPI_1</div> <div>FG_SIGNAL_GPI_2</div> <div>FG_SIGNAL_GPI_3</div> <div>FG_SIGNAL_GPI_4</div> <div>FG_SIGNAL_GPI_5</div> <div>FG_SIGNAL_GPI_6</div> <div>FG_SIGNAL_GPI_7</div> <div>FG_SIGNAL_FRONT_GPI_0</div> <div>FG_SIGNAL_FRONT_GPI_1</div> <div>FG_SIGNAL_FRONT_GPI_2</div> <div>FG_SIGNAL_FRONT_GPI_3</div> </div> <div> <div>GND</div> <div>VCC</div> <div>Signal Exsync</div> <div>Signal Exsync2</div> <div>Signal Flash</div> <div>Signal Line Valid</div> <div>Signal Frame Valid</div> <div>Signal Cam1 Exsync</div> <div>Signal Cam1 Exsync2</div> <div>Signal Cam1 Flash</div> <div>Signal Cam1 Line Valid</div> <div>Signal Cam1 Frame Valid</div> <div>Signal GPI 0</div> <div>Signal GPI 1</div> <div>Signal GPI 2</div> <div>Signal GPI 3</div> <div>Signal GPI 4</div> <div>Signal GPI 5</div> <div>Signal GPI 6</div> <div>Signal GPI 7</div> <div>Signal Front GPI 0</div> <div>Signal Front GPI 1</div> <div>Signal Front GPI 2</div> <div>Signal Front GPI 3</div> </div>
Default value	FG_SIGNAL_CAM0_EXSYNC

Example 7.1. Usage of FG\_TRIGGERCAMERA\_SOURCE\_CXP0

```

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_CXP0, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_CXP0, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 7.1.2. FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP0

Table 7.2. Parameter properties of FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP0

Property	Value
Name	<b>FG_TRIGGERCAMERA_SOURCE_EDGE_CXP0</b>
Display Name	<b>CXP Link Trigger 0 Source Edge</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_RISING_EDGE</b> Rising Edge <b>FG_FALLING_EDGE</b> Falling Edge
Default value	<b>FG_RISING_EDGE</b>

Example 7.2. Usage of FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP0

```

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

if ((result = Fg_setParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_EDGE_CXP0, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_EDGE_CXP0, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 7.1.3. FG\_TRIGGERCAMERA\_SOURCE\_CXP1

Table 7.3. Parameter properties of FG\_TRIGGERCAMERA\_SOURCE\_CXP1

Property	Value
Name	<b>FG_TRIGGERCAMERA_SOURCE_CXP1</b>
Display Name	<b>CXP Link Trigger 1 Source</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<div> <div> <b>GND</b>  <b>VCC</b>  <b>FG_SIGNAL_CAM0_EXSYNC</b>  <b>FG_SIGNAL_CAM0_EXSYNC2</b>  <b>FG_SIGNAL_CAM0_FLASH</b>  <b>FG_SIGNAL_CAM0_LVAL</b>  <b>FG_SIGNAL_CAM0_FVAL</b>  <b>FG_SIGNAL_CAM1_EXSYNC</b>  <b>FG_SIGNAL_CAM1_EXSYNC2</b>  <b>FG_SIGNAL_CAM1_FLASH</b>  <b>FG_SIGNAL_CAM1_LVAL</b>  <b>FG_SIGNAL_CAM1_FVAL</b>  <b>FG_SIGNAL_GPI_0</b>  <b>FG_SIGNAL_GPI_1</b>  <b>FG_SIGNAL_GPI_2</b>  <b>FG_SIGNAL_GPI_3</b>  <b>FG_SIGNAL_GPI_4</b>  <b>FG_SIGNAL_GPI_5</b>  <b>FG_SIGNAL_GPI_6</b>  <b>FG_SIGNAL_GPI_7</b>  <b>FG_SIGNAL_FRONT_GPI_0</b>  <b>FG_SIGNAL_FRONT_GPI_1</b>  <b>FG_SIGNAL_FRONT_GPI_2</b>  <b>FG_SIGNAL_FRONT_GPI_3</b> </div> <div> <b>GND</b>  <b>VCC</b>  Signal Exsync  Signal Exsync2  Signal Flash  Signal Line Valid  Signal Frame Valid  Signal Cam1 Exsync  Signal Cam1 Exsync2  Signal Cam1 Flash  Signal Cam1 Line Valid  Signal Cam1 Frame Valid  Signal GPI 0  Signal GPI 1  Signal GPI 2  Signal GPI 3  Signal GPI 4  Signal GPI 5  Signal GPI 6  Signal GPI 7  Signal Front GPI 0  Signal Front GPI 1  Signal Front GPI 2  Signal Front GPI 3 </div> </div>
Default value	<b>FG_SIGNAL_CAM0_EXSYNC</b>

Example 7.3. Usage of FG\_TRIGGERCAMERA\_SOURCE\_CXP1

```

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_CXP1, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_CXP1, &value, 0, type)) < 0) {
    /* error handling */
}

```

#### 7.1.4. FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP1

Table 7.4. Parameter properties of FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP1

Property	Value
Name	<b>FG_TRIGGERCAMERA_SOURCE_EDGE_CXP1</b>
Display Name	<b>CXP Link Trigger 1 Source Edge</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_RISING_EDGE</b> Rising Edge <b>FG_FALLING_EDGE</b> Falling Edge
Default value	<b>FG_FALLING_EDGE</b>

Example 7.4. Usage of FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP1

```

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

if ((result = Fg_setParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_EDGE_CXP1, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_EDGE_CXP1, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 7.1.5. FG\_TRIGGERCAMERA\_SOURCE\_CXP2

Table 7.5. Parameter properties of FG\_TRIGGERCAMERA\_SOURCE\_CXP2

Property	Value
Name	<b>FG_TRIGGERCAMERA_SOURCE_CXP2</b>
Display Name	<b>CXP Link Trigger 2 Source</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<div> <div> <b>GND</b>  <b>VCC</b>  <b>FG_SIGNAL_CAM0_EXSYNC</b>  <b>FG_SIGNAL_CAM0_EXSYNC2</b>  <b>FG_SIGNAL_CAM0_FLASH</b>  <b>FG_SIGNAL_CAM0_LVAL</b>  <b>FG_SIGNAL_CAM0_FVAL</b>  <b>FG_SIGNAL_CAM1_EXSYNC</b>  <b>FG_SIGNAL_CAM1_EXSYNC2</b>  <b>FG_SIGNAL_CAM1_FLASH</b>  <b>FG_SIGNAL_CAM1_LVAL</b>  <b>FG_SIGNAL_CAM1_FVAL</b>  <b>FG_SIGNAL_GPI_0</b>  <b>FG_SIGNAL_GPI_1</b>  <b>FG_SIGNAL_GPI_2</b>  <b>FG_SIGNAL_GPI_3</b>  <b>FG_SIGNAL_GPI_4</b>  <b>FG_SIGNAL_GPI_5</b>  <b>FG_SIGNAL_GPI_6</b>  <b>FG_SIGNAL_GPI_7</b>  <b>FG_SIGNAL_FRONT_GPI_0</b>  <b>FG_SIGNAL_FRONT_GPI_1</b>  <b>FG_SIGNAL_FRONT_GPI_2</b>  <b>FG_SIGNAL_FRONT_GPI_3</b> </div> <div> <b>GND</b>  <b>VCC</b>  Signal Exsync  Signal Exsync2  Signal Flash  Signal Line Valid  Signal Frame Valid  Signal Cam1 Exsync  Signal Cam1 Exsync2  Signal Cam1 Flash  Signal Cam1 Line Valid  Signal Cam1 Frame Valid  Signal GPI 0  Signal GPI 1  Signal GPI 2  Signal GPI 3  Signal GPI 4  Signal GPI 5  Signal GPI 6  Signal GPI 7  Signal Front GPI 0  Signal Front GPI 1  Signal Front GPI 2  Signal Front GPI 3 </div> </div>
Default value	<b>GND</b>

Example 7.5. Usage of FG\_TRIGGERCAMERA\_SOURCE\_CXP2

```

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

if ((result = Fg_setParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_CXP2, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_CXP2, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 7.1.6. FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP2

Table 7.6. Parameter properties of FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP2

Property	Value
Name	<b>FG_TRIGGERCAMERA_SOURCE_EDGE_CXP2</b>
Display Name	<b>CXP Link Trigger 2 Source Edge</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_RISING_EDGE</b> Rising Edge <b>FG_FALLING_EDGE</b> Falling Edge
Default value	<b>FG_RISING_EDGE</b>

Example 7.6. Usage of FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP2

```

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

if ((result = Fg_setParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_EDGE_CXP2, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_EDGE_CXP2, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 7.1.7. FG\_TRIGGERCAMERA\_SOURCE\_CXP3



Table 7.7. Parameter properties of FG\_TRIGGERCAMERA\_SOURCE\_CXP3

Property	Value
Name	<b>FG_TRIGGERCAMERA_SOURCE_CXP3</b>
Display Name	<b>CXP Link Trigger 3 Source</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<div> <div> <b>GND</b>  <b>VCC</b>  <b>FG_SIGNAL_CAM0_EXSYNC</b>  <b>FG_SIGNAL_CAM0_EXSYNC2</b>  <b>FG_SIGNAL_CAM0_FLASH</b>  <b>FG_SIGNAL_CAM0_LVAL</b>  <b>FG_SIGNAL_CAM0_FVAL</b>  <b>FG_SIGNAL_CAM1_EXSYNC</b>  <b>FG_SIGNAL_CAM1_EXSYNC2</b>  <b>FG_SIGNAL_CAM1_FLASH</b>  <b>FG_SIGNAL_CAM1_LVAL</b>  <b>FG_SIGNAL_CAM1_FVAL</b>  <b>FG_SIGNAL_GPI_0</b>  <b>FG_SIGNAL_GPI_1</b>  <b>FG_SIGNAL_GPI_2</b>  <b>FG_SIGNAL_GPI_3</b>  <b>FG_SIGNAL_GPI_4</b>  <b>FG_SIGNAL_GPI_5</b>  <b>FG_SIGNAL_GPI_6</b>  <b>FG_SIGNAL_GPI_7</b>  <b>FG_SIGNAL_FRONT_GPI_0</b>  <b>FG_SIGNAL_FRONT_GPI_1</b>  <b>FG_SIGNAL_FRONT_GPI_2</b>  <b>FG_SIGNAL_FRONT_GPI_3</b> </div> <div> <b>GND</b>  <b>VCC</b>  Signal Exsync  Signal Exsync2  Signal Flash  Signal Line Valid  Signal Frame Valid  Signal Cam1 Exsync  Signal Cam1 Exsync2  Signal Cam1 Flash  Signal Cam1 Line Valid  Signal Cam1 Frame Valid  Signal GPI 0  Signal GPI 1  Signal GPI 2  Signal GPI 3  Signal GPI 4  Signal GPI 5  Signal GPI 6  Signal GPI 7  Signal Front GPI 0  Signal Front GPI 1  Signal Front GPI 2  Signal Front GPI 3 </div> </div>
Default value	<b>GND</b>

Example 7.7. Usage of FG\_TRIGGERCAMERA\_SOURCE\_CXP3

```

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

if ((result = Fg_setParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_CXP3, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_CXP3, &value, 0, type)) < 0) {
    /* error handling */
}

```

### 7.1.8. FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP3

Table 7.8. Parameter properties of FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP3

Property	Value
Name	<b>FG_TRIGGERCAMERA_SOURCE_EDGE_CXP3</b>
Display Name	<b>CXP Link Trigger 3 Source Edge</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_RISING_EDGE</b> Rising Edge <b>FG_FALLING_EDGE</b> Falling Edge
Default value	<b>FG_RISING_EDGE</b>

Example 7.8. Usage of FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP3

```

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

if ((result = Fg_setParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_EDGE_CXP3, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGERCAMERA_SOURCE_EDGE_CXP3, &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

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

You can change the polarity using parameter *FG\_TRIGGEROUT\_GPO\_0\_POLARITY*.

Table 7.9. Parameter properties of FG\_TRIGGEROUT\_GPO\_0\_SOURCE

Property	Value
Name	<b>FG_TRIGGEROUT_GPO_0_SOURCE</b>
Display Name	<b>Trigger Out GPO 0 Source</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<div> <div>GND</div> <div>VCC</div> <div>FG_SIGNAL_CAM0_EXSYNC</div> <div>FG_SIGNAL_CAM0_EXSYNC2</div> <div>FG_SIGNAL_CAM0_FLASH</div> <div>FG_SIGNAL_CAM0_LVAL</div> <div>FG_SIGNAL_CAM0_FVAL</div> <div>FG_SIGNAL_CAM1_EXSYNC</div> <div>FG_SIGNAL_CAM1_EXSYNC2</div> <div>FG_SIGNAL_CAM1_FLASH</div> <div>FG_SIGNAL_CAM1_LVAL</div> <div>FG_SIGNAL_CAM1_FVAL</div> <div>FG_SIGNAL_GPI_0</div> <div>FG_SIGNAL_GPI_1</div> <div>FG_SIGNAL_GPI_2</div> <div>FG_SIGNAL_GPI_3</div> <div>FG_SIGNAL_GPI_4</div> <div>FG_SIGNAL_GPI_5</div> <div>FG_SIGNAL_GPI_6</div> <div>FG_SIGNAL_GPI_7</div> <div>FG_SIGNAL_FRONT_GPI_0</div> <div>FG_SIGNAL_FRONT_GPI_1</div> <div>FG_SIGNAL_FRONT_GPI_2</div> <div>FG_SIGNAL_FRONT_GPI_3</div> </div> <div> <div>GND</div> <div>VCC</div> <div>Signal Exsync</div> <div>Signal Exsync2</div> <div>Signal Flash</div> <div>Signal Line Valid</div> <div>Signal Frame Valid</div> <div>Signal Cam1 Exsync</div> <div>Signal Cam1 Exsync2</div> <div>Signal Cam1 Flash</div> <div>Signal Cam1 Line Valid</div> <div>Signal Cam1 Frame Valid</div> <div>Signal GPI 0</div> <div>Signal GPI 1</div> <div>Signal GPI 2</div> <div>Signal GPI 3</div> <div>Signal GPI 4</div> <div>Signal GPI 5</div> <div>Signal GPI 6</div> <div>Signal GPI 7</div> <div>Signal Front GPI 0</div> <div>Signal Front GPI 1</div> <div>Signal Front GPI 2</div> <div>Signal Front GPI 3</div> </div>
Default value	<b>FG_SIGNAL_CAM0_FLASH</b>

Example 7.9. 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

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

Table 7.10. Parameter properties of FG\_TRIGGEROUT\_GPO\_0\_POLARITY

Property	Value
Name	<b>FG_TRIGGEROUT_GPO_0_POLARITY</b>
Display Name	<b>Trigger Out 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 7.10. 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\_TRIGGEROUT\_FRONT\_GPO\_0\_SOURCE et al.



#### Note

This description applies also to the following parameters:  
FG\_TRIGGEROUT\_FRONT\_GPO\_1\_SOURCE

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

You can change the polarity using parameter *FG\_TRIGGEROUT\_FRONT\_GPO\_0\_POLARITY*.

Table 7.11. Parameter properties of FG\_TRIGGEROUT\_FRONT\_GPO\_0\_SOURCE

Property	Value																																																
Name	FG_TRIGGEROUT_FRONT_GPO_0_SOURCE																																																
Display Name	Trigger Out Front GPO 0 Source																																																
Type	Enumeration																																																
Access policy	Read/Write/Change																																																
Storage policy	Persistent																																																
Allowed values	<table> <tr><td>GND</td><td>GND</td></tr> <tr><td>VCC</td><td>VCC</td></tr> <tr><td>FG_SIGNAL_CAM0_EXSYNC</td><td>Signal Exsync</td></tr> <tr><td>FG_SIGNAL_CAM0_EXSYNC2</td><td>Signal Exsync2</td></tr> <tr><td>FG_SIGNAL_CAM0_FLASH</td><td>Signal Flash</td></tr> <tr><td>FG_SIGNAL_CAM0_LVAL</td><td>Signal Line Valid</td></tr> <tr><td>FG_SIGNAL_CAM0_FVAL</td><td>Signal Frame Valid</td></tr> <tr><td>FG_SIGNAL_CAM1_EXSYNC</td><td>Signal Cam1 Exsync</td></tr> <tr><td>FG_SIGNAL_CAM1_EXSYNC2</td><td>Signal Cam1 Exsync2</td></tr> <tr><td>FG_SIGNAL_CAM1_FLASH</td><td>Signal Cam1 Flash</td></tr> <tr><td>FG_SIGNAL_CAM1_LVAL</td><td>Signal Cam1 Line Valid</td></tr> <tr><td>FG_SIGNAL_CAM1_FVAL</td><td>Signal Cam1 Frame Valid</td></tr> <tr><td>FG_SIGNAL_GPI_0</td><td>Signal GPI 0</td></tr> <tr><td>FG_SIGNAL_GPI_1</td><td>Signal GPI 1</td></tr> <tr><td>FG_SIGNAL_GPI_2</td><td>Signal GPI 2</td></tr> <tr><td>FG_SIGNAL_GPI_3</td><td>Signal GPI 3</td></tr> <tr><td>FG_SIGNAL_GPI_4</td><td>Signal GPI 4</td></tr> <tr><td>FG_SIGNAL_GPI_5</td><td>Signal GPI 5</td></tr> <tr><td>FG_SIGNAL_GPI_6</td><td>Signal GPI 6</td></tr> <tr><td>FG_SIGNAL_GPI_7</td><td>Signal GPI 7</td></tr> <tr><td>FG_SIGNAL_FRONT_GPI_0</td><td>Signal Front GPI 0</td></tr> <tr><td>FG_SIGNAL_FRONT_GPI_1</td><td>Signal Front GPI 1</td></tr> <tr><td>FG_SIGNAL_FRONT_GPI_2</td><td>Signal Front GPI 2</td></tr> <tr><td>FG_SIGNAL_FRONT_GPI_3</td><td>Signal Front GPI 3</td></tr> </table>	GND	GND	VCC	VCC	FG_SIGNAL_CAM0_EXSYNC	Signal Exsync	FG_SIGNAL_CAM0_EXSYNC2	Signal Exsync2	FG_SIGNAL_CAM0_FLASH	Signal Flash	FG_SIGNAL_CAM0_LVAL	Signal Line Valid	FG_SIGNAL_CAM0_FVAL	Signal Frame Valid	FG_SIGNAL_CAM1_EXSYNC	Signal Cam1 Exsync	FG_SIGNAL_CAM1_EXSYNC2	Signal Cam1 Exsync2	FG_SIGNAL_CAM1_FLASH	Signal Cam1 Flash	FG_SIGNAL_CAM1_LVAL	Signal Cam1 Line Valid	FG_SIGNAL_CAM1_FVAL	Signal Cam1 Frame Valid	FG_SIGNAL_GPI_0	Signal GPI 0	FG_SIGNAL_GPI_1	Signal GPI 1	FG_SIGNAL_GPI_2	Signal GPI 2	FG_SIGNAL_GPI_3	Signal GPI 3	FG_SIGNAL_GPI_4	Signal GPI 4	FG_SIGNAL_GPI_5	Signal GPI 5	FG_SIGNAL_GPI_6	Signal GPI 6	FG_SIGNAL_GPI_7	Signal GPI 7	FG_SIGNAL_FRONT_GPI_0	Signal Front GPI 0	FG_SIGNAL_FRONT_GPI_1	Signal Front GPI 1	FG_SIGNAL_FRONT_GPI_2	Signal Front GPI 2	FG_SIGNAL_FRONT_GPI_3	Signal Front GPI 3
GND	GND																																																
VCC	VCC																																																
FG_SIGNAL_CAM0_EXSYNC	Signal Exsync																																																
FG_SIGNAL_CAM0_EXSYNC2	Signal Exsync2																																																
FG_SIGNAL_CAM0_FLASH	Signal Flash																																																
FG_SIGNAL_CAM0_LVAL	Signal Line Valid																																																
FG_SIGNAL_CAM0_FVAL	Signal Frame Valid																																																
FG_SIGNAL_CAM1_EXSYNC	Signal Cam1 Exsync																																																
FG_SIGNAL_CAM1_EXSYNC2	Signal Cam1 Exsync2																																																
FG_SIGNAL_CAM1_FLASH	Signal Cam1 Flash																																																
FG_SIGNAL_CAM1_LVAL	Signal Cam1 Line Valid																																																
FG_SIGNAL_CAM1_FVAL	Signal Cam1 Frame Valid																																																
FG_SIGNAL_GPI_0	Signal GPI 0																																																
FG_SIGNAL_GPI_1	Signal GPI 1																																																
FG_SIGNAL_GPI_2	Signal GPI 2																																																
FG_SIGNAL_GPI_3	Signal GPI 3																																																
FG_SIGNAL_GPI_4	Signal GPI 4																																																
FG_SIGNAL_GPI_5	Signal GPI 5																																																
FG_SIGNAL_GPI_6	Signal GPI 6																																																
FG_SIGNAL_GPI_7	Signal GPI 7																																																
FG_SIGNAL_FRONT_GPI_0	Signal Front GPI 0																																																
FG_SIGNAL_FRONT_GPI_1	Signal Front GPI 1																																																
FG_SIGNAL_FRONT_GPI_2	Signal Front GPI 2																																																
FG_SIGNAL_FRONT_GPI_3	Signal Front GPI 3																																																
Default value	FG_SIGNAL_CAM0_FLASH																																																

Example 7.11. Usage of FG\_TRIGGEROUT\_FRONT\_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_FRONT_GPO_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGEROUT_FRONT_GPO_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

```

#### 7.2.4. FG\_TRIGGEROUT\_FRONT\_GPO\_0\_POLARITY et al.



#### Note

This description applies also to the following parameters:  
FG\_TRIGGEROUT\_FRONT\_GPO\_1\_POLARITY

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

Table 7.12. Parameter properties of FG\_TRIGGEROUT\_FRONT\_GPO\_0\_POLARITY

Property	Value
Name	<b>FG_TRIGGEROUT_FRONT_GPO_0_POLARITY</b>
Display Name	<b>Trigger Front Out 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 7.12. Usage of FG\_TRIGGEROUT\_FRONT\_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_FRONT_GPO_0_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGEROUT_FRONT_GPO_0_POLARITY, &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 its digital inputs 0, 2 and 5.

Table 7.13. 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</b> 0 <b>Maximum</b> 4095 <b>Stepsize</b> 1
Unit of measure	

Example 7.13. 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

Select the source for the custom signal event.

Table 7.14. Parameter properties of FG\_CUSTOM\_SIGNAL\_EVENT\_0\_SOURCE

Property	Value
Name	FG_CUSTOM_SIGNAL_EVENT_0_SOURCE
Display Name	Custom Signal Event 0 Source
Type	Enumeration
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	<div> <div>GND</div> <div>VCC</div> <div>FG_SIGNAL_CAM0_EXSYNC</div> <div>FG_SIGNAL_CAM0_EXSYNC2</div> <div>FG_SIGNAL_CAM0_FLASH</div> <div>FG_SIGNAL_CAM0_LVAL</div> <div>FG_SIGNAL_CAM0_FVAL</div> <div>FG_SIGNAL_CAM0_LINE_START</div> <div>FG_SIGNAL_CAM0_LINE_END</div> <div>FG_SIGNAL_CAM0_FRAME_START</div> <div>FG_SIGNAL_CAM0_FRAME_END</div> <div>FG_SIGNAL_CAM1_EXSYNC</div> <div>FG_SIGNAL_CAM1_EXSYNC2</div> <div>FG_SIGNAL_CAM1_FLASH</div> <div>FG_SIGNAL_CAM1_LVAL</div> <div>FG_SIGNAL_CAM1_FVAL</div> <div>FG_SIGNAL_CAM1_LINE_START</div> <div>FG_SIGNAL_CAM1_LINE_END</div> <div>FG_SIGNAL_CAM1_FRAME_START</div> <div>FG_SIGNAL_CAM1_FRAME_END</div> <div>FG_SIGNAL_GPI_0</div> <div>FG_SIGNAL_GPI_1</div> <div>FG_SIGNAL_GPI_2</div> <div>FG_SIGNAL_GPI_3</div> <div>FG_SIGNAL_GPI_4</div> <div>FG_SIGNAL_GPI_5</div> <div>FG_SIGNAL_GPI_6</div> <div>FG_SIGNAL_GPI_7</div> <div>FG_SIGNAL_FRONT_GPI_0</div> <div>FG_SIGNAL_FRONT_GPI_1</div> <div>FG_SIGNAL_FRONT_GPI_2</div> <div>FG_SIGNAL_FRONT_GPI_3</div> </div> <div> <div>GND</div> <div>VCC</div> <div>Signal Exsync</div> <div>Signal Exsync2</div> <div>Signal Flash</div> <div>Signal Line Valid</div> <div>Signal Frame Valid</div> <div>Signal Line Start</div> <div>Cam0 Line Transfer End</div> <div>Signal Frame Start</div> <div>Signal Frame End</div> <div>Signal Cam1 Exsync</div> <div>Signal Cam1 Exsync2</div> <div>Signal Cam1 Flash</div> <div>Signal Cam1 Line Valid</div> <div>Signal Cam1 Frame Valid</div> <div>Signal Cam1 Line Start</div> <div>Signal Cam1 Line End</div> <div>Signal Cam1 Frame Start</div> <div>Signal Cam1 Frame End</div> <div>Signal GPI 0</div> <div>Signal GPI 1</div> <div>Signal GPI 2</div> <div>Signal GPI 3</div> <div>Signal GPI 4</div> <div>Signal GPI 5</div> <div>Signal GPI 6</div> <div>Signal GPI 7</div> <div>Signal Front GPI 0</div> <div>Signal Front GPI 1</div> <div>Signal Front GPI 2</div> <div>Signal Front GPI 3</div> </div>
Default value	FG_SIGNAL_CAM0_EXSYNC

**Example 7.14. 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

Select the polarity for the custom signal event.

**Table 7.15. 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 7.15. 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

Select the source for the custom signal event.



Table 7.16. 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	<div> <div>GND</div> <div>VCC</div> <div>FG_SIGNAL_CAM0_EXSYNC</div> <div>FG_SIGNAL_CAM0_EXSYNC2</div> <div>FG_SIGNAL_CAM0_FLASH</div> <div>FG_SIGNAL_CAM0_LVAL</div> <div>FG_SIGNAL_CAM0_FVAL</div> <div>FG_SIGNAL_CAM0_LINE_START</div> <div>FG_SIGNAL_CAM0_LINE_END</div> <div>FG_SIGNAL_CAM0_FRAME_START</div> <div>FG_SIGNAL_CAM0_FRAME_END</div> <div>FG_SIGNAL_CAM1_EXSYNC</div> <div>FG_SIGNAL_CAM1_EXSYNC2</div> <div>FG_SIGNAL_CAM1_FLASH</div> <div>FG_SIGNAL_CAM1_LVAL</div> <div>FG_SIGNAL_CAM1_FVAL</div> <div>FG_SIGNAL_CAM1_LINE_START</div> <div>FG_SIGNAL_CAM1_LINE_END</div> <div>FG_SIGNAL_CAM1_FRAME_START</div> <div>FG_SIGNAL_CAM1_FRAME_END</div> <div>FG_SIGNAL_GPI_0</div> <div>FG_SIGNAL_GPI_1</div> <div>FG_SIGNAL_GPI_2</div> <div>FG_SIGNAL_GPI_3</div> <div>FG_SIGNAL_GPI_4</div> <div>FG_SIGNAL_GPI_5</div> <div>FG_SIGNAL_GPI_6</div> <div>FG_SIGNAL_GPI_7</div> <div>FG_SIGNAL_FRONT_GPI_0</div> <div>FG_SIGNAL_FRONT_GPI_1</div> <div>FG_SIGNAL_FRONT_GPI_2</div> <div>FG_SIGNAL_FRONT_GPI_3</div> </div> <div> <div>GND</div> <div>VCC</div> <div>Signal Exsync</div> <div>Signal Exsync2</div> <div>Signal Flash</div> <div>Signal Line Valid</div> <div>Signal Frame Valid</div> <div>Signal Line Start</div> <div>Cam0 Line Transfer End</div> <div>Signal Frame Start</div> <div>Signal Frame End</div> <div>Signal Cam1 Exsync</div> <div>Signal Cam1 Exsync2</div> <div>Signal Cam1 Flash</div> <div>Signal Cam1 Line Valid</div> <div>Signal Cam1 Frame Valid</div> <div>Signal Cam1 Line Start</div> <div>Signal Cam1 Line End</div> <div>Signal Cam1 Frame Start</div> <div>Signal Cam1 Frame End</div> <div>Signal GPI 0</div> <div>Signal GPI 1</div> <div>Signal GPI 2</div> <div>Signal GPI 3</div> <div>Signal GPI 4</div> <div>Signal GPI 5</div> <div>Signal GPI 6</div> <div>Signal GPI 7</div> <div>Signal Front GPI 0</div> <div>Signal Front GPI 1</div> <div>Signal Front GPI 2</div> <div>Signal Front GPI 3</div> </div>
Default value	<b>FG_SIGNAL_CAM0_FLASH</b>

Example 7.16. 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

Select the polarity for the custom signal event.

Table 7.17. Parameter properties of FG\_CUSTOM\_SIGNAL\_EVENT\_1\_POLARITY

Property	Value
Name	<b>FG_CUSTOM_SIGNAL_EVENT_1_POLARITY</b>
Display Name	<b>Custom Signal Event 1 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 7.17. 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

The event defined by *FG\_CUSTOM\_SIGNAL\_EVENT\_0\_SOURCE* and *FG\_CUSTOM\_SIGNAL\_EVENT\_0\_POLARITY*.

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

The event defined by *FG\_CUSTOM\_SIGNAL\_EVENT\_1\_SOURCE* and *FG\_CUSTOM\_SIGNAL\_EVENT\_1\_POLARITY*.

---

# 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	FG_LINETRIGGERMODE	
Display Name	Line Trigger Mode	
Type	Enumeration	
Access policy	Read/Write	
Storage policy	Persistent	
Allowed values	GRABBER_CONTROLLED	Grabber Controlled
	ASYNC_TRIGGER	Async External Trigger
	GRABBER_CONTROLLED_GATED	Grabber Controlled Gated
	ASYNC_GATED	Async Gated Trigger
Default value	GRABBER_CONTROLLED	

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	<b>FG_EXSYNCON</b>
Display Name	<b>ExSync On</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 8.2. 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 */
}

```

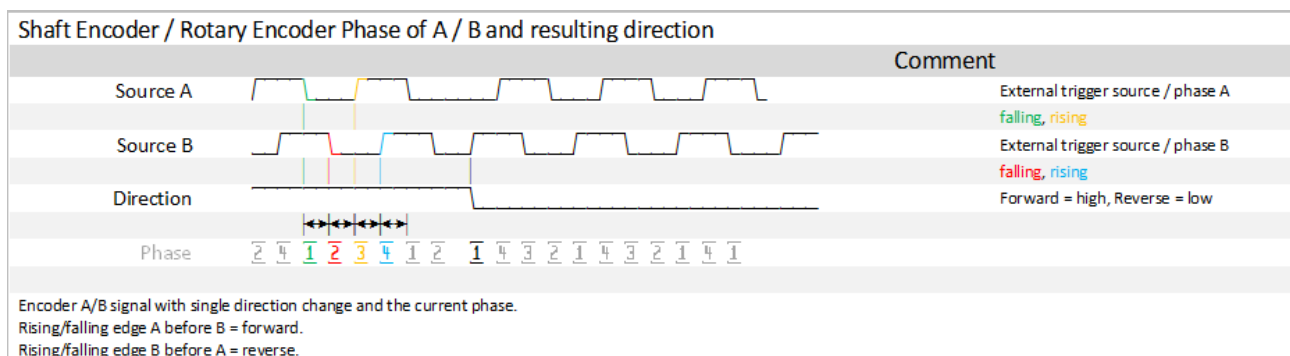
### 8.3. Line Trigger Input

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.

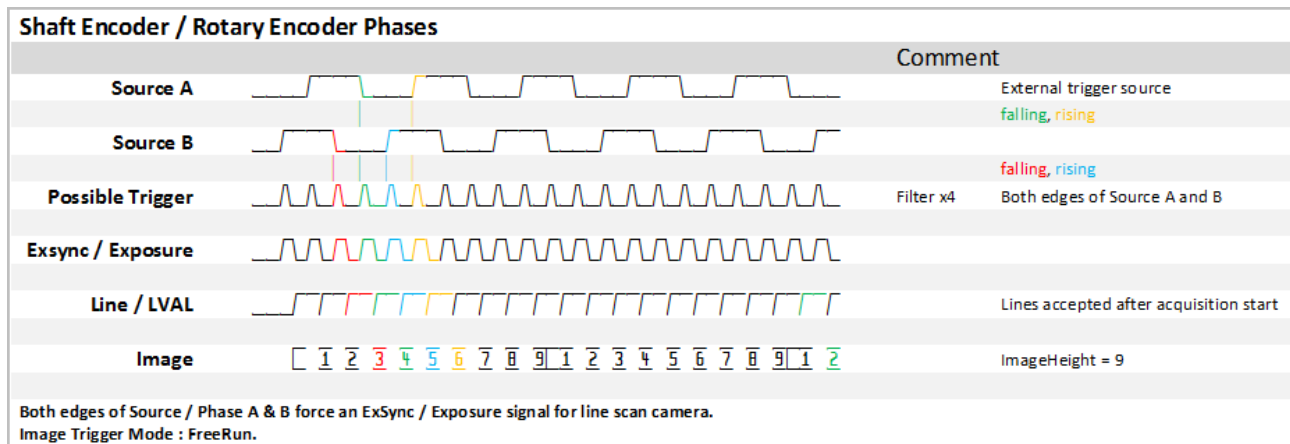
Most incremental shaft encoders provide 2 signals that are called A & B. By using these two signals the relative increments can be seen at the edges of these signals and a direction. In one direction the A-phase high state rises before the B-phase in the other direction, i.e. vice versa. If we do not 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>Line Trigger In Source</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<div> <div>TRGINSRC_GPI_0</div> <div>GPI Trigger Source 0</div> </div> <div> <div>TRGINSRC_GPI_1</div> <div>GPI Trigger Source 1</div> </div> <div> <div>TRGINSRC_GPI_2</div> <div>GPI Trigger Source 2</div> </div> <div> <div>TRGINSRC_GPI_3</div> <div>GPI Trigger Source 3</div> </div> <div> <div>TRGINSRC_GPI_4</div> <div>GPI Trigger Source 4</div> </div> <div> <div>TRGINSRC_GPI_5</div> <div>GPI Trigger Source 5</div> </div> <div> <div>TRGINSRC_GPI_6</div> <div>GPI Trigger Source 6</div> </div> <div> <div>TRGINSRC_GPI_7</div> <div>GPI Trigger Source 7</div> </div> <div> <div>TRGINSRC_FRONT_GPI_0</div> <div>Trigger In Source Front GPI 0</div> </div> <div> <div>TRGINSRC_FRONT_GPI_1</div> <div>Trigger In Source Front GPI 1</div> </div> <div> <div>TRGINSRC_FRONT_GPI_2</div> <div>Trigger In Source Front GPI 2</div> </div> <div> <div>TRGINSRC_FRONT_GPI_3</div> <div>Trigger In Source Front GPI 3</div> </div>
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>Line Trigger In Polarity</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<div> <div>HIGH_ON_ZERO_LOW</div> <div>Low Active</div> </div> <div> <div>HIGH_ON_ZERO_HIGH</div> <div>High Active</div> </div>
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>Line Trigger Debouncing</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0.0032 <b>Maximum</b> 26.0 <b>Stepsize</b> 0.0032
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

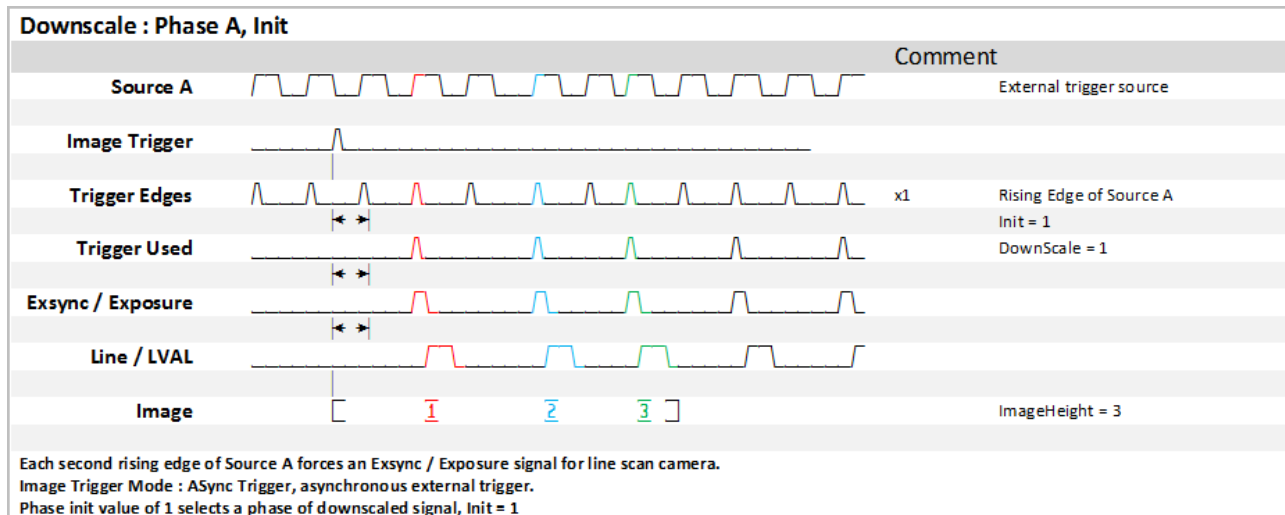


Table 8.7. Parameter properties of FG\_LINE\_DOWNSCALE

Property	Value
Name	FG_LINE_DOWNSCALE
Display Name	Line 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>Line Downscale Init</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>Shaft Encoder On</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

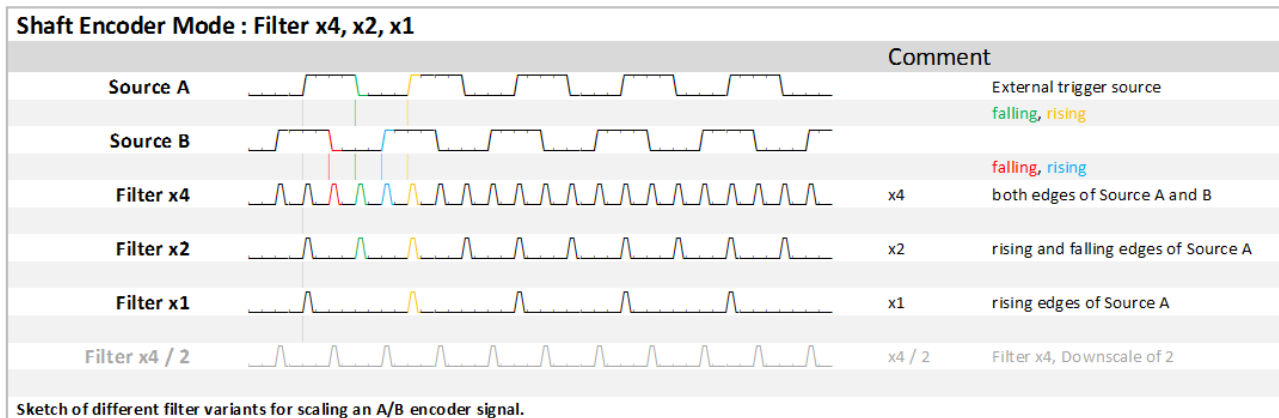


Table 8.10. Parameter properties of FG\_SHAFTENCODERMODE

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\_LINE\_TRIGGER\_IN\_SRC* 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>Shaft Encoder Input Source</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 <b>TRGINSRC_GPI_5</b> GPI Trigger Source 5 <b>TRGINSRC_GPI_6</b> GPI Trigger Source 6 <b>TRGINSRC_GPI_7</b> GPI Trigger Source 7 <b>TRGINSRC_FRONT_GPI_0</b> Trigger In Source Front GPI 0 <b>TRGINSRC_FRONT_GPI_1</b> Trigger In Source Front GPI 1 <b>TRGINSRC_FRONT_GPI_2</b> Trigger In Source Front GPI 2 <b>TRGINSRC_FRONT_GPI_3</b> Trigger In Source Front GPI 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>Shaft Encoder Leading</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 */
}

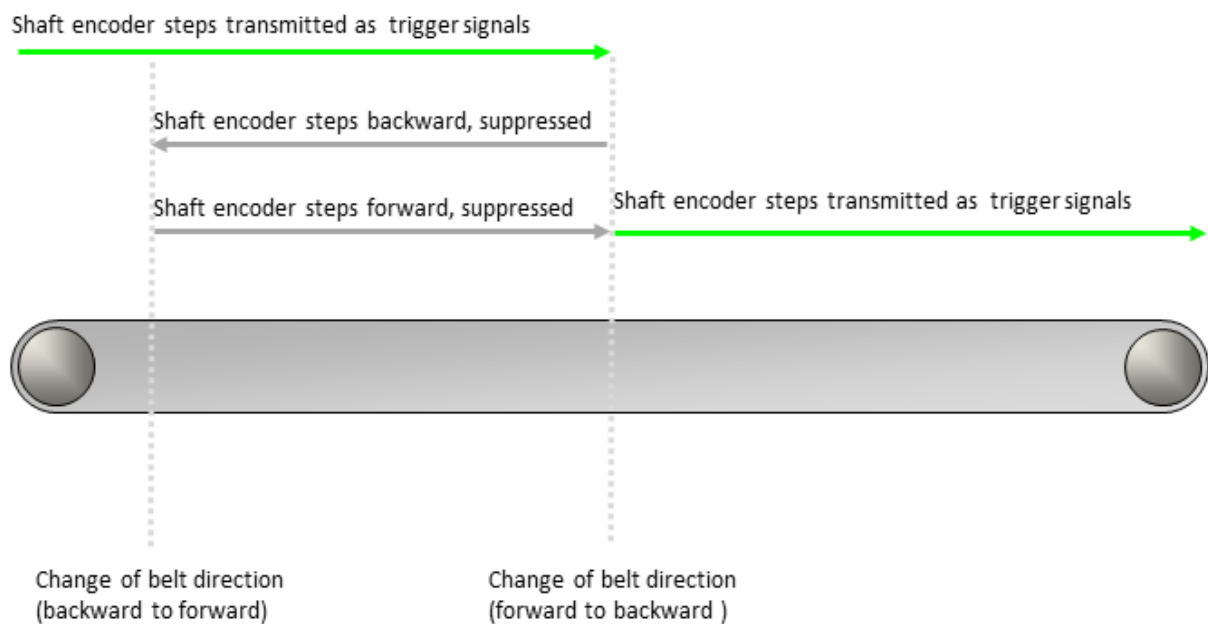
```

**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

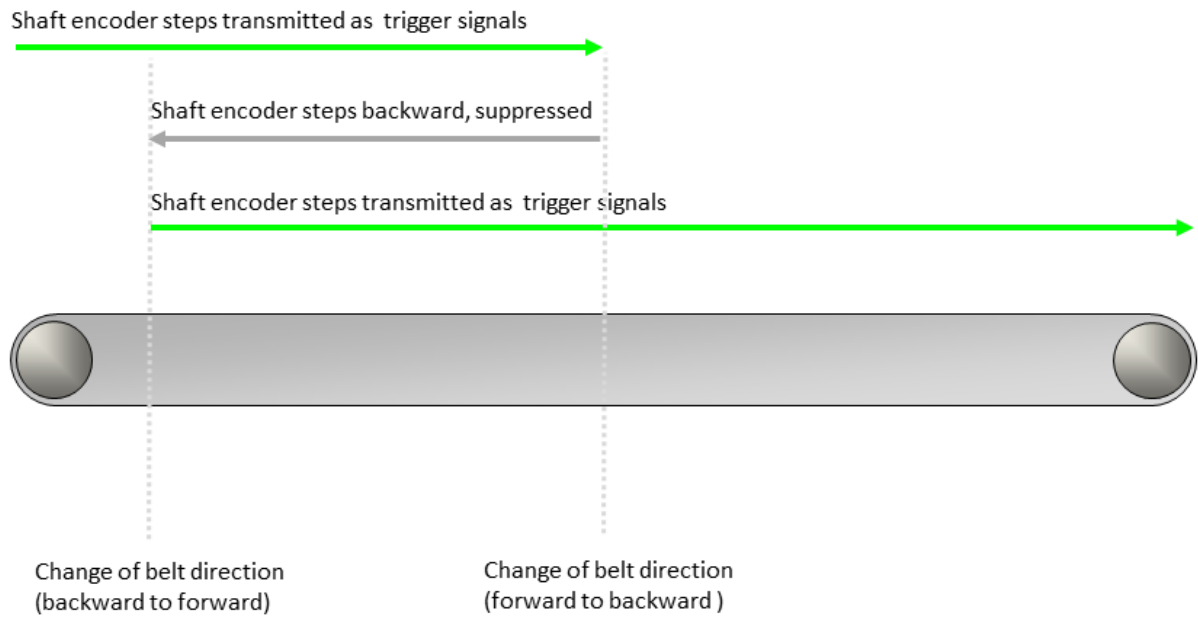


Table 8.13. Parameter properties of FG\_SHAFTENCODER\_COMPENSATION\_ENABLE

Property	Value
Name	<b>FG_SHAFTENCODER_COMPENSATION_ENABLE</b>
Display Name	<b>Shaft Encoder 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'.

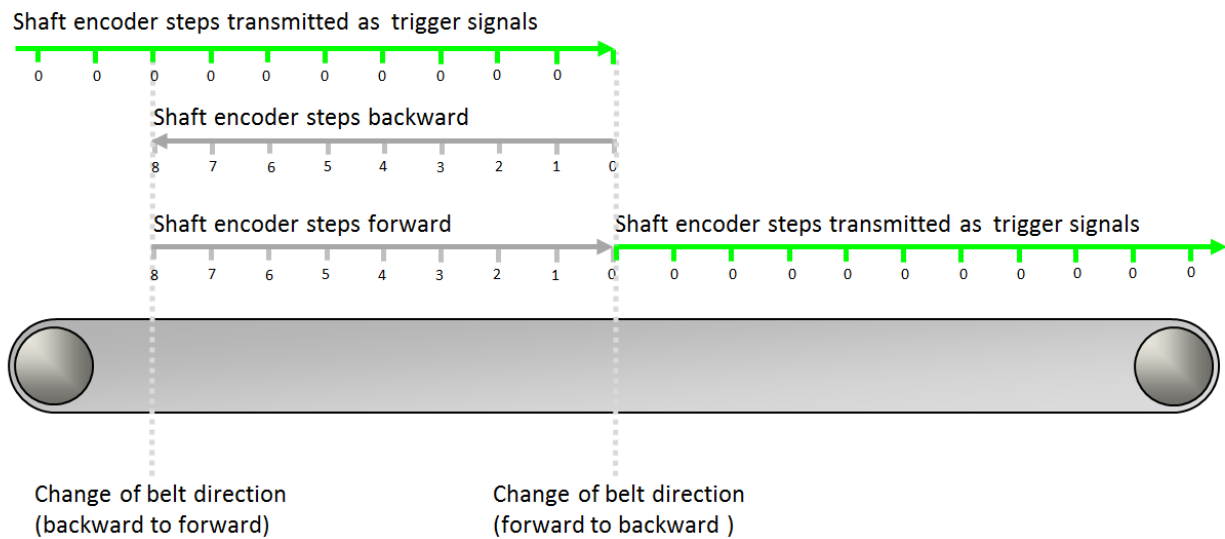


It is based on a 20bit counter enabling a backward movement of up to 1048575 encoder pulses. An overflow of this value will not occur since it will skip all additional pulses for a compensation state of more than 1048575. By this the count of the rollback compensation is limited by 2 to the power of 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.

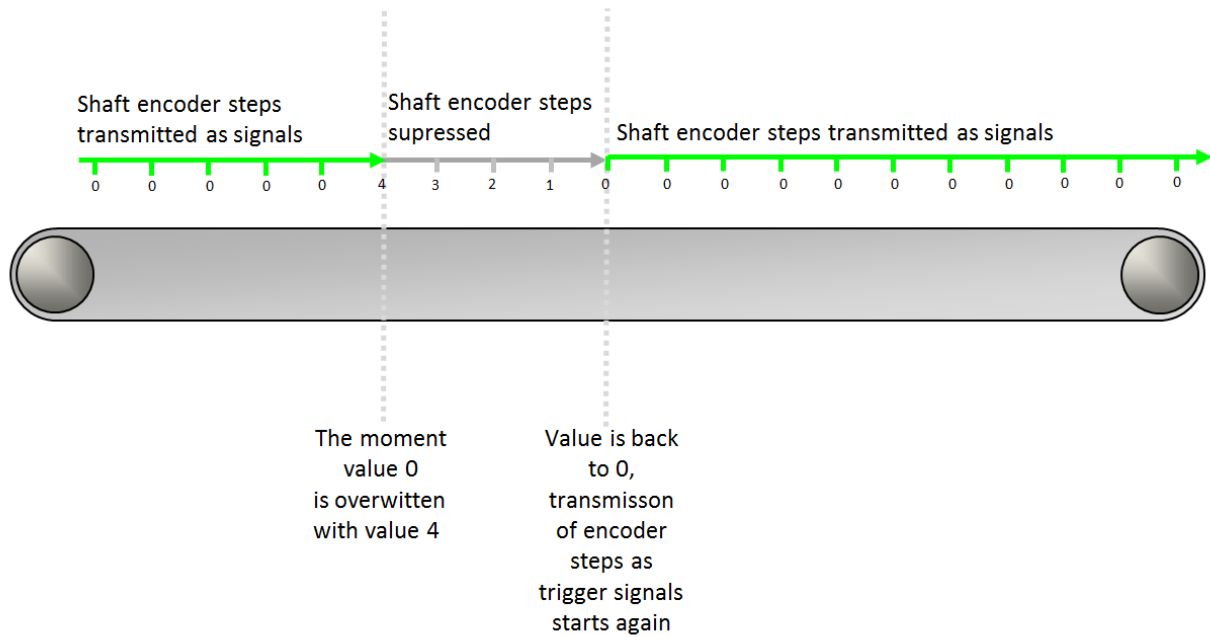
As soon as you enter a value for *FG\_SHAFTENCODER\_COMPENSATION\_COUNT*, this value overwrites the value the parameter holds before.

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.

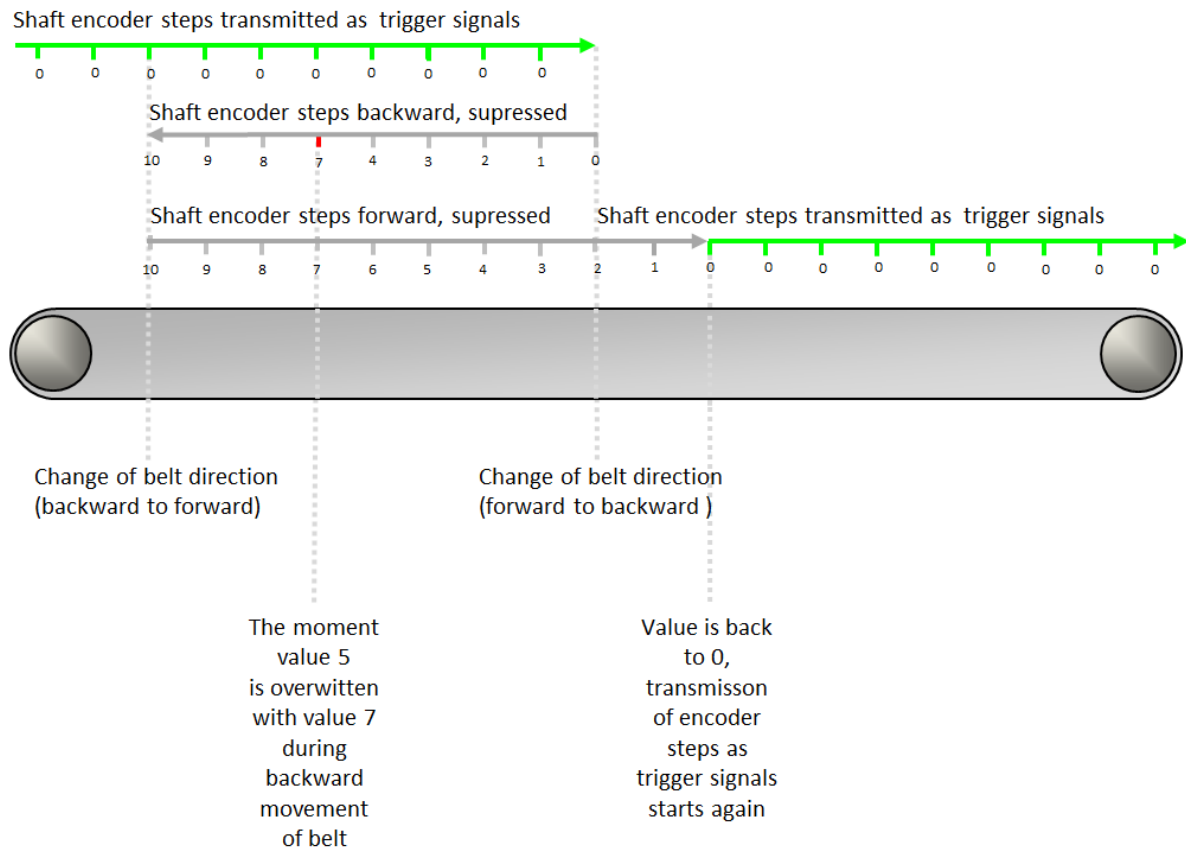
Figure 8.8. Shaft Encoder Compensation Count Example 1



### Example 2:

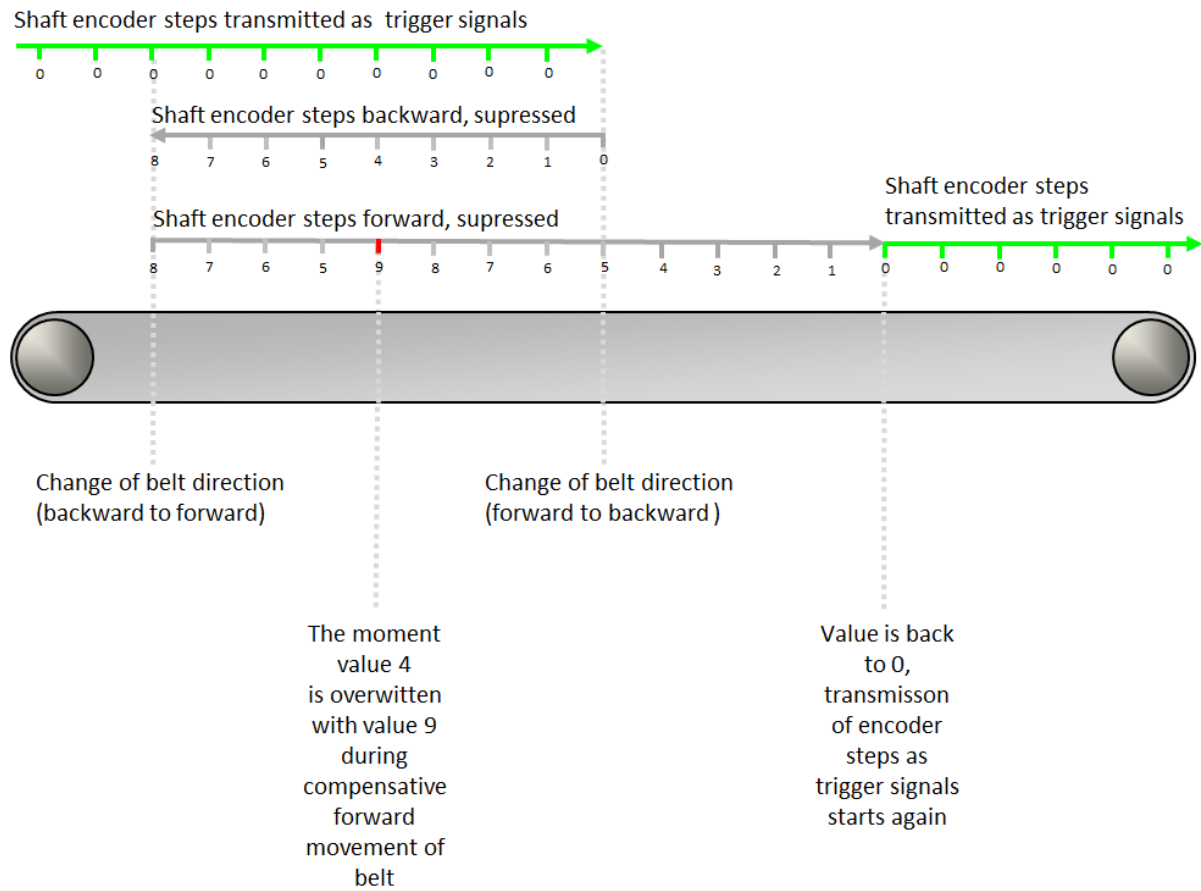
The transportation belt is moving backward, the (backward) shaft encoder steps are suppressed, and the value of *FG\_SHAFTENCODER\_COMPENSATION\_COUNT* is not 0. Then, during backward movement of the transportation belt, the value 5 of *FG\_SHAFTENCODER\_COMPENSATION\_COUNT* is overwritten by value 7. 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

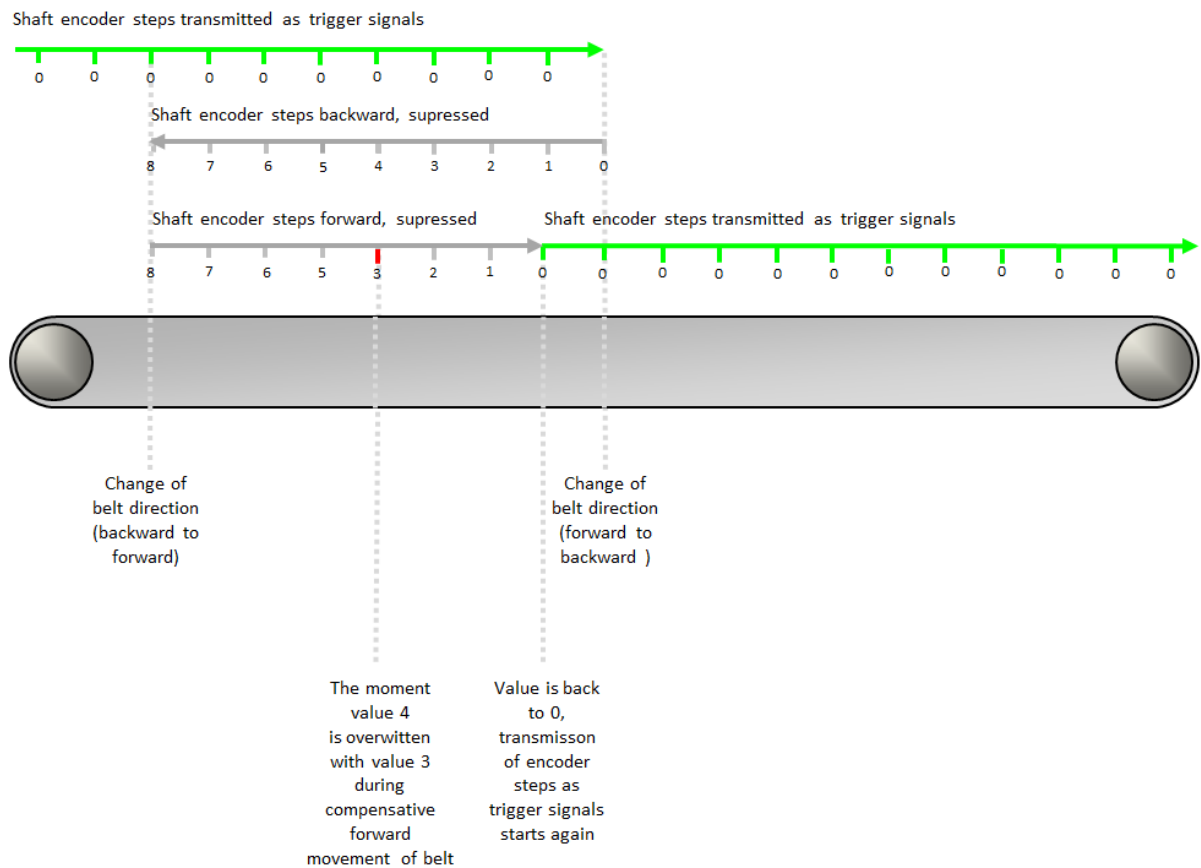


Table 8.14. Parameter properties of FG\_SHAFTENCODER\_COMPENSATION\_COUNT

Property	Value
Name	FG_SHAFTENCODER_COMPENSATION_COUNT
Display Name	Shaft Encoder 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

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

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

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

Table 8.15. Parameter properties of FG\_LINEPERIODE

Property	Value
Name	FG_LINEPERIODE
Display Name	Line Period
Type	Double
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	<b>Minimum</b> 0.2048 <b>Maximum</b> 838.8576 <b>Stepsize</b> 0.0032
Default value	200.0
Unit of measure	$\mu s$

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>Line Exposure</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> <b>0.2048</b> <b>Maximum</b> <b>419.4272</b> <b>Stepsize</b> <b>0.0032</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>ExSync 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>Line Trigger Delay</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 419.4272</b> <b>Stepsize 0.0032</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>Image Trigger Mode</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FREE_RUN</b> Free Run <b>ASYNC_TRIGGER</b> Async External Trigger <b>ASYNC_TRIGGER_MULTIFRAME</b> Async External Trigger Multiframe <b>ASYNC_GATED</b> Async Gated Trigger <b>ASYNC_GATED_MULTIFRAME</b> Async Gated Trigger Multiframe
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 On</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 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	<b>FG_FLASHON</b>
Display Name	<b>Flash On</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 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>Image Trigger Async 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 <b>IS_NOT_BUSY</b> Not Busy

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>Image Trigger Input Source</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<div> <div>TRGINSRC_GPI_0</div> <div>TRGINSRC_GPI_1</div> <div>TRGINSRC_GPI_2</div> <div>TRGINSRC_GPI_3</div> <div>TRGINSRC_GPI_4</div> <div>TRGINSRC_GPI_5</div> <div>TRGINSRC_GPI_6</div> <div>TRGINSRC_GPI_7</div> <div>TRGINSRC_FRONT_GPI_0</div> <div>TRGINSRC_FRONT_GPI_1</div> <div>TRGINSRC_FRONT_GPI_2</div> <div>TRGINSRC_FRONT_GPI_3</div> <div>TRGINSOFTWARE</div> </div> <div> <div>GPI Trigger Source 0</div> <div>GPI Trigger Source 1</div> <div>GPI Trigger Source 2</div> <div>GPI Trigger Source 3</div> <div>GPI Trigger Source 4</div> <div>GPI Trigger Source 5</div> <div>GPI Trigger Source 6</div> <div>GPI Trigger Source 7</div> <div>Trigger In Source Front GPI 0</div> <div>Trigger In Source Front GPI 1</div> <div>Trigger In Source Front GPI 2</div> <div>Trigger In Source Front GPI 3</div> <div>Software Trigger</div> </div>
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>Image Trigger Input Polarity</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<div> <div>HIGH_ON_ZERO_LOW</div> <div>HIGH_ON_ZERO_HIGH</div> </div> <div> <div>Low Active</div> <div>High Active</div> </div>
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>Image Trigger Gate Delay</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 65535 <b>Stepsize</b> 1
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>Image Trigger Debouncing</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 0.0032</b> <b>Maximum 26.0</b> <b>Stepsize 0.0032</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>Strobe Pulse 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 9.10. 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. Ensure to enable the software trigger by *FG\_IMGTRIGGERINSRC*.

Table 9.12. Parameter properties of FG\_SENDSOFTWARETRIGGER

Property	Value
Name	<b>FG_SENDSOFTWARETRIGGER</b>
Display Name	<b>Send Software Trigger</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum</b> 1 <b>Maximum</b> 1 <b>Stepsize</b> 1
Default value	<b>1</b>

Example 9.12. Usage of FG\_SENDSOFTWARETRIGGER

```

int result = 0;
unsigned int value = 1;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_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 = inactive = low or one = active = high. Ensure to enable the software trigger by *FG\_IMGTRIGGERINSRC*.

Table 9.13. Parameter properties of FG\_SETSOFTWARETRIGGER

Property	Value
Name	<b>FG_SETSOFTWARETRIGGER</b>
Display Name	<b>Set Software Trigger</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	FG_SIGNAL_ANALYZER_0_SOURCE
Display Name	Signal Analyzer 0 Source
Type	Enumeration
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	<div> <div>GND</div> <div>VCC</div> <div>FG_SIGNAL_CAM0_EXSYNC</div> <div>FG_SIGNAL_CAM0_EXSYNC2</div> <div>FG_SIGNAL_CAM0_FLASH</div> <div>FG_SIGNAL_CAM0_LVAL</div> <div>FG_SIGNAL_CAM0_FVAL</div> <div>FG_SIGNAL_CAM0_LINE_START</div> <div>FG_SIGNAL_CAM0_LINE_END</div> <div>FG_SIGNAL_CAM0_FRAME_START</div> <div>FG_SIGNAL_CAM0_FRAME_END</div> <div>FG_SIGNAL_CAM1_EXSYNC</div> <div>FG_SIGNAL_CAM1_EXSYNC2</div> <div>FG_SIGNAL_CAM1_FLASH</div> <div>FG_SIGNAL_CAM1_LVAL</div> <div>FG_SIGNAL_CAM1_FVAL</div> <div>FG_SIGNAL_CAM1_LINE_START</div> <div>FG_SIGNAL_CAM1_LINE_END</div> <div>FG_SIGNAL_CAM1_FRAME_START</div> <div>FG_SIGNAL_CAM1_FRAME_END</div> <div>FG_SIGNAL_GPI_0</div> <div>FG_SIGNAL_GPI_1</div> <div>FG_SIGNAL_GPI_2</div> <div>FG_SIGNAL_GPI_3</div> <div>FG_SIGNAL_GPI_4</div> <div>FG_SIGNAL_GPI_5</div> <div>FG_SIGNAL_GPI_6</div> <div>FG_SIGNAL_GPI_7</div> <div>FG_SIGNAL_FRONT_GPI_0</div> <div>FG_SIGNAL_FRONT_GPI_1</div> <div>FG_SIGNAL_FRONT_GPI_2</div> <div>FG_SIGNAL_FRONT_GPI_3</div> </div> <div> <div>GND</div> <div>VCC</div> <div>Signal Exsync</div> <div>Signal Exsync2</div> <div>Signal Flash</div> <div>Signal Line Valid</div> <div>Signal Frame Valid</div> <div>Signal Line Start</div> <div>Cam0 Line Transfer End</div> <div>Signal Frame Start</div> <div>Signal Frame End</div> <div>Signal Cam1 Exsync</div> <div>Signal Cam1 Exsync2</div> <div>Signal Cam1 Flash</div> <div>Signal Cam1 Line Valid</div> <div>Signal Cam1 Frame Valid</div> <div>Signal Cam1 Line Start</div> <div>Signal Cam1 Line End</div> <div>Signal Cam1 Frame Start</div> <div>Signal Cam1 Frame End</div> <div>Signal GPI 0</div> <div>Signal GPI 1</div> <div>Signal GPI 2</div> <div>Signal GPI 3</div> <div>Signal GPI 4</div> <div>Signal GPI 5</div> <div>Signal GPI 6</div> <div>Signal GPI 7</div> <div>Signal Front GPI 0</div> <div>Signal Front GPI 1</div> <div>Signal Front GPI 2</div> <div>Signal Front GPI 3</div> </div>
Default value	FG_SIGNAL_CAM0_EXSYNC

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 0.0 Maximum 1.3743895344E7 Stepsize 0.0032
Unit of measure	ns

**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 0.0 Maximum 1.3743895344E7 Stepsize 0.0032
Unit of measure	ns

**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	<b>FG_SIGNAL_ANALYZER_0_PERIOD_MIN</b>
Display Name	<b>Signal Analyzer 0 Min Period</b>
Type	<b>Double</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> <b>0.0032</b> <b>Maximum</b> <b>1.3743895344E7</b> <b>Stepsize</b> <b>0.0032</b>
Unit of measure	<b>ns</b>

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	<b>FG_SIGNAL_ANALYZER_0_PULSE_COUNT</b>
Display Name	<b>Signal Analyzer 0 Pulse Count</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> <b>0</b> <b>Maximum</b> <b>4294967295</b> <b>Stepsize</b> <b>1</b>
Unit of measure	<b>pulses</b>

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>pulses</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>Unsigned Integer</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum 1</b> <b>Maximum 1</b> <b>Stepsize 1</b>
Default value	<b>1</b>

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

```
int result = 0;
unsigned int value = 1;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_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. These images will be marked incomplete in the **FG\_IMAGE\_TAG** (bit 30 is set to '1').

- Lost Frames:

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

The buffer overflow threshold *FG\_OVERFLOW\_ON\_THRESHOLD* and *FG\_OVERFLOW\_ON\_SYNC\_THRESHOLD* default ensures that under normal conditions frames can be completed or will be fully dropped so that corrupted frames are avoided.

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 detected, 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 too 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>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. FG\_OVERFLOW\_OFF\_THRESHOLD

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

Table 11.3. Parameter properties of FG\_OVERFLOW\_OFF\_THRESHOLD

Property	Value
Name	<b>FG_OVERFLOW_OFF_THRESHOLD</b>
Display Name	<b>Overflow Off Threshold</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0.0 <b>Maximum</b> 100.0 <b>Stepsize</b> 0.5
Default value	<b>50.0</b>

Example 11.3. Usage of FG\_OVERFLOW\_OFF\_THRESHOLD

```

int result = 0;
double value = 50.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

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

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

```

## 11.4. FG\_OVERFLOW\_ON\_THRESHOLD

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

Table 11.4. Parameter properties of FG\_OVERFLOW\_ON\_THRESHOLD

Property	Value
Name	<b>FG_OVERFLOW_ON_THRESHOLD</b>
Display Name	<b>Overflow On Threshold</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0.0 <b>Maximum</b> 100.0 <b>Stepsize</b> 0.5
Default value	<b>99.5</b>

Example 11.4. Usage of FG\_OVERFLOW\_ON\_THRESHOLD

```

int result = 0;
double value = 99.5;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

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

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

```

## 11.5. FG\_OVERFLOW\_ON\_SYNC\_THRESHOLD

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

Table 11.5. Parameter properties of FG\_OVERFLOW\_ON\_SYNC\_THRESHOLD

Property	Value
Name	<b>FG_OVERFLOW_ON_SYNC_THRESHOLD</b>
Display Name	<b>Overflow Sync On Threshold</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> <b>0.0</b> <b>Maximum</b> <b>100.0</b> <b>Stepsize</b> <b>0.5</b>
Default value	<b>80.0</b>

Example 11.5. Usage of FG\_OVERFLOW\_ON\_SYNC\_THRESHOLD

```

int result = 0;
double value = 80.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

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

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

```

## 11.6. FG\_OVERFLOW\_EVENT\_SELECT

The *FG\_OVERFLOW\_CAM0* Event. Allows to generate events if one of the following conditions is meet.

Table 11.6. Event select for *FG\_OVERFLOW\_CAM0*

Value	Description
<b>FG_OVERFLOW_EVENT_INCOMPLETE</b>	Each incomplete frame will generate an Event containing the information that the frame is incomplete and the frameID
<b>FG_OVERFLOW_EVENT_LOST</b>	Each lost frame will generate an Event containing the information that the frame is lost and the frameID
<b>FG_OVERFLOW_EVENT_INCOMPLETE_LOST</b>	Each lost or incomplete frame will generate an Event containing the information that the frame is lost/incomplete and the frameID
<b>FG_OVERFLOW_EVENT_OK</b>	Each correct frame will generate an Event containing the information that the frame is transfered correct and the frameID of the frame
<b>FG_OVERFLOW_EVENT_OK_INCOMPLETE</b>	Each incomplete or correct frame will generate an Event containing the information that the frame is correct or incomplete and the frameID
<b>FG_OVERFLOW_EVENT_OK_LOST</b>	Each lost or correct frame will generate an Event containing the information that the frame is correct or lost and the frameID
<b>FG_OVERFLOW_EVENT_ALL</b>	Each frame will generate an Event containing the status(lost, incomplete or correct) of the frame and the frameID

Table 11.7. Parameter properties of *FG\_OVERFLOW\_EVENT\_SELECT*

Property	Value
Name	<b>FG_OVERFLOW_EVENT_SELECT</b>
Display Name	<b>Overflow Event Select</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_OVERFLOW_EVENT_INCOMPLETE</b> Incomplete <b>FG_OVERFLOW_EVENT_LOST</b> Lost <b>FG_OVERFLOW_EVENT_INCOMPLETE_LOST</b> Incomplete Lost <b>FG_OVERFLOW_EVENT_OK</b> OK <b>FG_OVERFLOW_EVENT_OK_INCOMPLETE</b> Incomplete OK <b>FG_OVERFLOW_EVENT_OK_LOST</b> Lost OK <b>FG_OVERFLOW_EVENT_ALL</b> All
Default value	<b>FG_OVERFLOW_EVENT_INCOMPLETE_LOST</b>

Example 11.6. Usage of *FG\_OVERFLOW\_EVENT\_SELECT*

```

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

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

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

```

## 11.7. Events

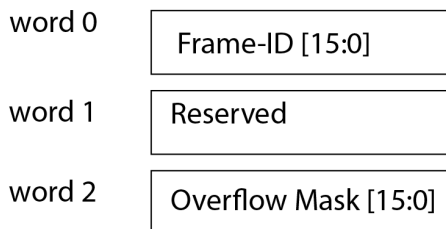
In programming or runtime environments, a callback function is a piece of executable code that is passed as an argument, which is expected to call back (execute) exactly that time an event is triggered. This applet can generate some software callback events based on the memory overflow condition as explained 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.

### 11.7.1. FG\_OVERFLOW\_CAM0

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

Figure 11.1. Illustration of Overflow Data Packet



#### Overflow Mask [15:0]

0	Frame is truncated
1	Frame is lost
2	Reserved
3	Frame is complete
4	End of sequence
5	Reserved
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

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

# 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	<b><math>FG\_IMG\_SELECT\_PERIOD</math></b>
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.

The applet Acq\_DualCXP12Line performs a Bayer de-mosaicing. The white balancing is performed prior to the Bayer de-mosaicing, to ensure the correction of the raw data and avoid subsequent faults during processing.

## 13.1. FG\_SCALINGFACTOR\_GREEN

Table 13.1. Parameter properties of FG\_SCALINGFACTOR\_GREEN

Property	Value
Name	FG_SCALINGFACTOR_GREEN
Display Name	Scaling Factor Green
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\_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 */
}
```

## 13.2. FG\_SCALINGFACTOR\_RED

Table 13.2. Parameter properties of FG\_SCALINGFACTOR\_RED

Property	Value
Name	FG_SCALINGFACTOR_RED
Display Name	Scaling Factor Red
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\_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 */
}

```

## 13.3. FG\_SCALINGFACTOR\_BLUE

**Table 13.3. Parameter properties of FG\_SCALINGFACTOR\_BLUE**

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

**Example 13.3. 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 */
}

```

---

# Chapter 14. Color Converter

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

This applet can perform the following conversions.

Table 14.1. Color Conversion

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

By setting the input and output format the conversion is automatically applied if a conversion is possible. Otherwise the applet will output unchanged values. See *FG\_PIXELFORMAT* and *FG\_FORMAT*.

# Chapter 15. 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 16, '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.

## 15.1. 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 16, '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 15.1. 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 15.1. 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 */
}
```

## 15.2. 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 16, '*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 15.2. 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 15.2. 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 */
}
```

## 15.3. FG\_LUT\_VALUE

Table 15.3. Parameter properties of FG\_LUT\_VALUE

Property	Value
Name	<b>FG_LUT_VALUE</b>
Display Name	<b>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 15.3. Usage of FG\_LUT\_VALUE**

```
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, &access, 0, type)) < 0) {
        /* error handling */
    }

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

## 15.4. FG\_LUT\_VALUE\_RED

**Table 15.4. Parameter properties of FG\_LUT\_VALUE\_RED**

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

**Example 15.4. 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 */
    }
}
```

## 15.5. FG\_LUT\_VALUE\_GREEN

Table 15.5. 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 15.5. 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 */
    }
}

```

## 15.6. FG\_LUT\_VALUE\_BLUE

Table 15.6. 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 15.6. 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 */
}
}

```

---

## 15.7. 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 15.7. 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 15.7. 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 */
}
```

## 15.8. 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 15.8. 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 15.8. 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 */
}

```

## 15.9. Applet Properties

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

### 15.9.1. 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 15.9. Parameter properties of FG\_LUT\_IMPLEMENTATION\_TYPE

Property	Value
Name	<b>FG_LUT_IMPLEMENTATION_TYPE</b>
Display Name	<b>LUT Implementation Type</b>
Type	<b>Enumeration</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>LUT_IMPLEMENTATION_FULL_LUT</b> Full LUT <b>LUT_IMPLEMENTATION_KNEELUT</b> Knee LUT

Example 15.9. 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 */
}

```

### 15.9.2. FG\_LUT\_IN\_BITS

This applet is using 14 lookup table input bits.

Table 15.10. Parameter properties of FG\_LUT\_IN\_BITS

Property	Value
Name	<b>FG_LUT_IN_BITS</b>
Display Name	<b>LUT Input Pixel Bit 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 16</b> <b>Stepsize 1</b>
Unit of measure	<b>bit</b>

Example 15.10. 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 */
}

```

### 15.9.3. FG\_LUT\_OUT\_BITS

This applet is using 16 lookup table output bits.

Table 15.11. Parameter properties of FG\_LUT\_OUT\_BITS

Property	Value
Name	<b>FG_LUT_OUT_BITS</b>
Display Name	<b>LUT Output Pixel Bit 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 16</b> <b>Stepsize 1</b>
Unit of measure	<b>bit</b>

Example 15.11. 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 16. 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 16.1. 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 16.2. LUT Processor with Inversion

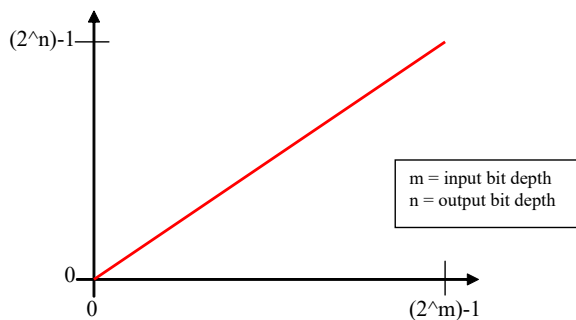
$$Output(x) = 2^{16} - 1 - \left[ \left[ gain * \left( \frac{x}{2^{14} - 1} + offset \right) \right]^{\frac{1}{gamma}} \right] * (2^{16} - 1),$$

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 16.1. Lookup Table Processing: Identity



## 16.1. 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 16.2 shows an example of an offset.

Figure 16.2. Lookup Table Processing: Offset

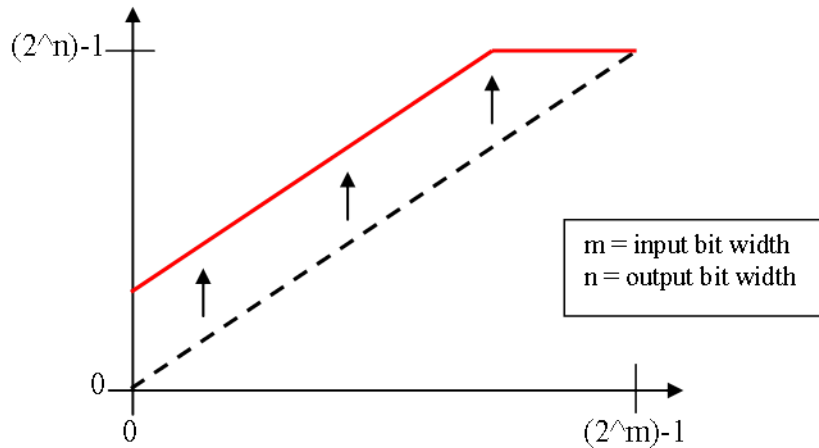


Table 16.1. Parameter properties of FG\_PROCESSING\_OFFSET

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

Example 16.1. 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 */
}
```

## 16.2. 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 16.3. Lookup Table Processing: Gain

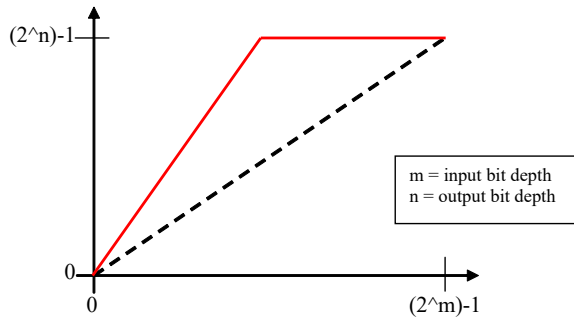


Table 16.2. Parameter properties of FG\_PROCESSING\_GAIN

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</b> 0.0 <b>Maximum</b> 16384.0 <b>Stepsize</b> 2.220446049250313E-16
Default value	<b>1.0</b>

Example 16.2. 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 */
}

```

## 16.3. 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 16.4. Lookup Table Processing: Gamma

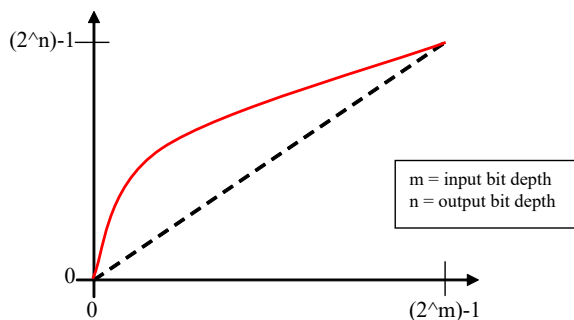


Table 16.3. Parameter properties of FG\_PROCESSING\_GAMMA

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</b> -1000.0 <b>Maximum</b> 1000.0 <b>Stepsize</b> 2.220446049250313E-16
Default value	<b>1.0</b>

Example 16.3. 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 */
}

```

## 16.4. 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 16.5. Lookup Table Processing: Invert

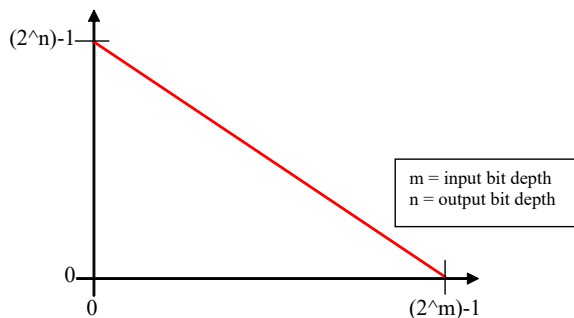


Table 16.4. Parameter properties of FG\_PROCESSING\_INVERT

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</b> On <b>FG_OFF</b> Off
Default value	<b>FG_OFF</b>



**Example 16.4. 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 17. Output Format

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

## 17.1. 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*.

Moreover, the color converter of this applet can convert between different color formats of the input and output. Check Chapter 14, '*Color Converter*' for more information.

This applet supports the following output formats:

- **FG\_BGR8** and **FG\_RGB8**: 24 bit BGR/RGB color format with 8 bit/component.
- **FG\_BGRA8** and **FG\_RGBA8**: Color format with 8 bit/component. Component "a" has value zero.
- **FG\_BGR10** and **FG\_RGB10**: 30 bit BGR/RGB color format with 10 bit/component.



### 30 Bit Output Format

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

- **FG\_BGR12** and **FG\_RGB12**: 36 bit BGR/RGB color format with 12 bit/component.



### 36 Bit Output Format

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

- **FG\_BGR14** and **FG\_RGB14**: 42 bit BGR/RGB color format with 14 bit/component.



### 42 Bit Output Format

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

- **FG\_BGR16** and **FG\_RGB16**: 48 bit BGR/RGB color format with 16 bit/component.



### BGR vs. RGB Memory Alignement

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

- **FG\_MONO8**: 8 bit grayscale format
- **FG\_MONO10**: 10 bit grayscale format



## 10 Bit Output Format

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

- **FG\_MONO12**: 12 bit grayscale format



## 12 Bit Output Format

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

- **FG\_MONO14**: 14 bit grayscale format



## 14 Bit Output Format

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

- **FG\_MONO16**: 16 bit grayscale format



## DMA Bandwidth

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

- **FG\_BAYERGR8**, **FG\_BAYERRG8**, **FG\_BAYERGB8** and **FG\_BAYERBG8**: 8 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.
- **FG\_BAYERGR10**, **FG\_BAYERRG10**, **FG\_BAYERGB10** and **FG\_BAYERBG10**: 10 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.



## 10 Bit Output Format

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

- **FG\_BAYERGR12**, **FG\_BAYERRG12**, **FG\_BAYERGB12** and **FG\_BAYERBG12**: 12 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.



## 12 Bit Output Format

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

- **FG\_BAYERGR14**, **FG\_BAYERRG14**, **FG\_BAYERGB14** and **FG\_BAYERBG14**: 14 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.



## 14 Bit Output Format

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

- **FG\_BAYERGR16, FG\_BAYERRG16, FG\_BAYERGB16 and FG\_BAYERBG16:** 16 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.



## DMA Bandwidth

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

- **FG\_YUV422\_8:** YUV 422 output in 8 bit per component.

Table 17.1. Parameter properties of FG\_FORMAT

Property	Value	
Name	FG_FORMAT	
Display Name	Output Format	
Type	Enumeration	
Access policy	Read/Write	
Storage policy	Persistent	
Allowed values	<div> <div>FG_MON08</div> <div>Mono 8</div> </div> <div> <div>FG_MON010</div> <div>Mono 10p</div> </div> <div> <div>FG_MON012</div> <div>Mono 12p</div> </div> <div> <div>FG_MON014</div> <div>Mono 14p</div> </div> <div> <div>FG_MON016</div> <div>Mono 16</div> </div> <div> <div>FG_BGR8</div> <div>BGR 8bit</div> </div> <div> <div>FG_BGR10</div> <div>BGR 10bit</div> </div> <div> <div>FG_BGR12</div> <div>BGR 12bit</div> </div> <div> <div>FG_BGR14</div> <div>BGR 14p</div> </div> <div> <div>FG_BGR16</div> <div>BGR 16bit</div> </div> <div> <div>FG_RGB8</div> <div>RGB 8</div> </div> <div> <div>FG_RGB10</div> <div>RGB 10p</div> </div> <div> <div>FG_RGB12</div> <div>RGB 12p</div> </div> <div> <div>FG_RGB14</div> <div>RGB 14p</div> </div> <div> <div>FG_RGB16</div> <div>RGB 16</div> </div> <div> <div>FG_BGRA8</div> <div>BGRA 8</div> </div> <div> <div>FG_RGBA8</div> <div>RGBA 8</div> </div> <div> <div>FG_BICOLOR_RGBG8</div> <div>BiColor RG BG 8</div> </div> <div> <div>FG_BICOLOR_RGBG10</div> <div>BiColor RG BG 10</div> </div> <div> <div>FG_BICOLOR_RGBG12</div> <div>BiColor RG BG 12</div> </div> <div> <div>FG_BICOLOR_BGRG8</div> <div>BiColor BG RG 8</div> </div> <div> <div>FG_BICOLOR_BGRG10</div> <div>BiColor BG RG 10</div> </div> <div> <div>FG_BICOLOR_BGRG12</div> <div>BiColor BG RG 12</div> </div> <div> <div>FG_YUV422_8</div> <div>YCbCr422_8</div> </div>	
Default value	FG_MON08	

Example 17.1. Usage of FG\_FORMAT

```
int result = 0;
```

```

int value = FG_MONO8;
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 */
}

```

## 17.2. 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 17.2. Parameter properties of FG\_BITALIGNMENT

Property	Value
Name	<b>FG_BITALIGNMENT</b>
Display Name	<b>Bit 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 17.2. 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 */
}

```

## 17.3. 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 17.3. 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 17.3. 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 */
}

```

## 17.4. 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 17.4. 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 17.4. 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 */  
}
```

---

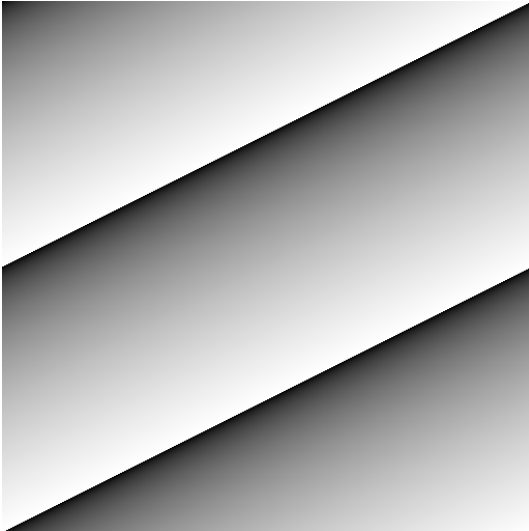
---

# Chapter 18. 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 18.1. 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.

### 18.1. 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 18.1. 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 18.1. 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 */
}

```

## 18.2. 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 18.2. 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 65544</b> <b>Stepsize 1</b>
Default value	<b>1024</b>
Unit of measure	<b>pixel</b>

Example 18.2. 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 */
}

```

### 18.3. 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 18.3. 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 65544</b> <b>Stepsize 1</b>
Default value	<b>0</b>
Unit of measure	<b>pixel</b>

Example 18.3. 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 */
}

```

### 18.4. 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 18.4. Parameter properties of FG\_CAMERASIMULATOR\_HEIGHT

Property	Value
Name	<b>FG_CAMERASIMULATOR_HEIGHT</b>
Display Name	<b>Height</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 65536</b> <b>Stepsize 1</b>
Default value	<b>1024</b>
Unit of measure	<b>pixel</b>

Example 18.4. 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 */
}

```

## 18.5. 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 18.5. Parameter properties of FG\_CAMERASIMULATOR\_FRAME\_GAP

Property	Value
Name	<b>FG_CAMERASIMULATOR_FRAME_GAP</b>
Display Name	<b>Frame 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 65536</b> <b>Stepsize 1</b>
Default value	<b>0</b>
Unit of measure	<b>pixel</b>

Example 18.5. 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 */
}

```

## 18.6. FG\_CAMERASIMULATOR\_PATTERN

The simulator will generate pixel value ramps from 0 to 255. As this applet is capable of using monochrome bayer or RGB inputs.

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 18.6. 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 18.6. 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 */
}

```

## 18.7. 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 18.7. 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 18.7. 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 */
}

```

## 18.8. 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 18.8. 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 18.8. 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 */
}

```

## 18.9. 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 18.9. Parameter properties of FG\_CAMERASIMULATOR\_SELECT\_MODE

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

Example 18.9. 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 */
}

```

## 18.10. 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 18.10. Parameter properties of FG\_CAMERASIMULATOR\_PIXEL\_FREQUENCY

Property	Value
Name	<b>FG_CAMERASIMULATOR_PIXEL_FREQUENCY</b>
Display Name	<b>Pixel Frequency</b>
Type	<b>Double</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0.9375 <b>Maximum</b> 3750.0 <b>Stepsize</b> 1.8749999999999996
Default value	<b>39.375</b>
Unit of measure	<b>MHz</b>

**Example 18.10. 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 */
}

```

## 18.11. 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 18.11. 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 1.0416666666666667E8</b> <b>Stepsize 7.0E-11</b>
Default value	<b>10240.0</b>
Unit of measure	<b>Hz</b>

**Example 18.11. 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 */
}

```

## 18.12. FG\_CAMERASIMULATOR\_FRAMERATE

This parameter sets the frame rate of the generated images.

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 18.12. Parameter properties of FG\_CAMERASIMULATOR\_FRAMERATE

Property	Value
Name	FG_CAMERASIMULATOR_FRAMERATE
Display Name	Framerate
Type	Double
Access policy	Read/Write/Change
Storage policy	Persistent
Allowed values	<b>Minimum</b> 0.15 <b>Maximum</b> 1.0416666666666667E8 <b>Stepsize</b> 7.0E-11
Default value	10.0
Unit of measure	Hz

Example 18.12. 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 */
}

```

## 18.13. 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 18.13. 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 18.13. 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 */
}

```

## 18.14. FG\_CAMERASIMULATOR\_ACTIVE

Table 18.14. 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</b> 1 <b>Maximum</b> 2000 <b>Stepsize</b> 1
Unit of measure	<b>pixel</b>

Example 18.14. 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 */
}

```

## 18.15. FG\_CAMERASIMULATOR\_PASSIVE

Table 18.15. 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 18.15. 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 19. Miscellaneous

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

## 19.1. FG\_TIMEOUT

This parameter is used to set a timeout for DMA transfers. After a timeout the acquisition is stopped. But it is only a 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 19.1. Parameter properties of FG\_TIMEOUT

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

Example 19.1. 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 */
}
```

## 19.2. FG\_APPLET\_ID

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

Table 19.2. Parameter properties of FG\_APPLET\_ID

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

Example 19.2. 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 */
}

```

### 19.3. 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 19.3. 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 19.3. 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 */
}

```

### 19.4. 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 19.4. 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 19.4. 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 */
}

```

### 19.5. 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 19.5. 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</b> 0 <b>Maximum</b> 1 <b>Stepsize</b> 1

Example 19.5. 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 */
}

```

## 19.6. 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 19.6. 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</b> 0 <b>Maximum</b> 255 <b>Stepsize</b> 1

Example 19.6. 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 */
}

```

## 19.7. FG\_SYSTEMMONITOR\_FPGA\_DNA\_LOW

The parameter *FG\_SYSTEMMONITOR\_FPGA\_DNA\_LOW* provides the lower 57 bit unique FPGA DNA.

Table 19.7. Parameter properties of FG\_SYSTEMMONITOR\_FPGA\_DNA\_LOW

Property	Value
Name	<b>FG_SYSTEMMONITOR_FPGA_DNA_LOW</b>
Display Name	<b>FPGA DNA Low</b>
Type	<b>Unsigned Integer (64 Bit)</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 144115188075855872 <b>Stepsize</b> 1

Example 19.7. Usage of FG\_SYSTEMMONITOR\_FPGA\_DNA\_LOW

```

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_LOW, &value, 0, type)) < 0) {
    /* error handling */
}

```

## 19.8. FG\_SYSTEMMONITOR\_FPGA\_DNA\_HIGH

The parameter *FG\_SYSTEMMONITOR\_FPGA\_DNA\_HIGH* provides the upper 32s bit unique FPGA DNA.

Table 19.8. Parameter properties of FG\_SYSTEMMONITOR\_FPGA\_DNA\_HIGH

Property	Value
Name	<b>FG_SYSTEMMONITOR_FPGA_DNA_HIGH</b>
Display Name	<b>FPGA DNA High</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 4294967295 <b>Stepsize</b> 1

Example 19.8. Usage of FG\_SYSTEMMONITOR\_FPGA\_DNA\_HIGH

```

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

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

```

## 19.9. Version

The category provides version information.

### 19.9.1. FG\_APPLET\_VERSION

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

Table 19.9. 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>
Allowed values	<b>Minimum</b> <b>0</b> <b>Maximum</b> <b>256</b> <b>Stepsize</b> <b>1</b>

Example 19.9. 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 */
}
```

### 19.9.2. FG\_APPLET\_REVISION

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

Table 19.10. 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>
Allowed values	<b>Minimum</b> <b>0</b> <b>Maximum</b> <b>256</b> <b>Stepsize</b> <b>1</b>

Example 19.10. 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 */
}
```

### 19.9.3. FG\_VISUALAPPLETS\_BUILD\_VERSION

Returns the VisualApplets version used to build the applets.

Table 19.11. Parameter properties of FG\_VISUALAPPLETS\_BUILD\_VERSION

Property	Value
Name	<b>FG_VISUALAPPLETS_BUILD_VERSION</b>
Display Name	<b>VisualApplets Build Version</b>
Type	<b>String</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 19.11. Usage of FG\_VISUALAPPLETS\_BUILD\_VERSION

```

int result = 0;
char* value = n/a;
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

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

```

## 19.10. Legacy

This category includes the legacy parameter for user software compatibility. The parameter of this category shouldn't be used anymore.

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

This parameter is for legacy use only and shouldn't be used anymore.

Setting the parameter to the CXP standard triggers re-writing the *FG\_TRIGGERCAMERA\_SOURCE* legacy parameter.

However, if you set the parameter to rising edge only, the legacy compatibility mode applies. In this case, *FG\_TRIGGERCAMERA\_SOURCE\_CXP1*, *FG\_TRIGGERCAMERA\_SOURCE\_CXP2* and *FG\_TRIGGERCAMERA\_SOURCE\_CXP3* are set to **GND**.

Be careful with this parameter as it overwrites the *FG\_TRIGGERCAMERA\_SOURCE\_CXP0* and *FG\_TRIGGERCAMERA\_SOURCE\_CXP0* parameters.

This legacy parameter can't be used in configuration files anymore.

Table 19.12. 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> CXP Trigger Standard <b>FG_RISING_EDGE_ONLY</b> CXP Trigger Rising
Default value	<b>FG_STANDARD</b>

Example 19.12. 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 */
}

```

### 19.10.2. FG\_TRIGGERCAMERA\_SOURCE

This is a legacy parameter. It is replaced by the *FG\_TRIGGERCAMERA\_SOURCE\_CXP0* and *FG\_TRIGGERCAMERA\_SOURCE\_CXP1* parameters. Before, the legacy parameter controlled the generation for CXP LinkTrigger0 associated with the start of a pulse and CXP LinkTrigger1 associated with the end of a pulse depending on the polarity settings. To keep the compatibility, when writing to this parameter, the value is copied to *FG\_TRIGGERCAMERA\_SOURCE\_CXP0* and sets the same value to *FG\_TRIGGERCAMERA\_SOURCE\_CXP1*.

Furthermore, *FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP1* is set to the inverse value of *FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP0*.

Reading the value only represents the status of *FG\_TRIGGERCAMERA\_SOURCE\_CXP0* and can be ambiguous as the new parameters offer more possibilities which can't be represented with the legacy parameter.

This legacy parameter can't be used in configuration files anymore.

Table 19.13. Parameter properties of FG\_TRIGGERCAMERA\_SOURCE

Property	Value
Name	<b>FG_TRIGGERCAMERA_SOURCE</b>
Display Name	<b>Legacy Trigger Camera Source</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Transient</b>
Allowed values	<div> <div> GND VCC FG_SIGNAL_CAM0_EXSYNC FG_SIGNAL_CAM0_EXSYNC2 FG_SIGNAL_CAM0_FLASH FG_SIGNAL_CAM0_LVAL FG_SIGNAL_CAM0_FVAL FG_SIGNAL_CAM1_EXSYNC FG_SIGNAL_CAM1_EXSYNC2 FG_SIGNAL_CAM1_FLASH FG_SIGNAL_CAM1_LVAL FG_SIGNAL_CAM1_FVAL FG_SIGNAL_GPI_0 FG_SIGNAL_GPI_1 FG_SIGNAL_GPI_2 FG_SIGNAL_GPI_3 FG_SIGNAL_GPI_4 FG_SIGNAL_GPI_5 FG_SIGNAL_GPI_6 FG_SIGNAL_GPI_7 FG_SIGNAL_FRONT_GPI_0 FG_SIGNAL_FRONT_GPI_1 FG_SIGNAL_FRONT_GPI_2 FG_SIGNAL_FRONT_GPI_3 </div> <div> GND VCC Signal Exsync Signal Exsync2 Signal Flash Signal Line Valid Signal Frame Valid Signal Cam1 Exsync Signal Cam1 Exsync2 Signal Cam1 Flash Signal Cam1 Line Valid Signal Cam1 Frame Valid Signal GPI 0 Signal GPI 1 Signal GPI 2 Signal GPI 3 Signal GPI 4 Signal GPI 5 Signal GPI 6 Signal GPI 7 Signal Front GPI 0 Signal Front GPI 1 Signal Front GPI 2 Signal Front GPI 3 </div> </div>
Default value	<b>FG_SIGNAL_CAM0_EXSYNC</b>

**Example 19.13. 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 */
}

```

### 19.10.3. FG\_TRIGGERCAMERA\_POLARITY

This is a legacy parameter. It is replaced by the parameters *FG\_TRIGGERCAMERA\_SOURCE\_CXP0* and *FG\_TRIGGERCAMERA\_SOURCE\_CXP1* as well as *FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP0* and *FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP1*. Before, the legacy parameter defined which edge of the trigger signal is used for CXP LinkTrigger and CXP LinkTrigger1. Now, this can be done individually for each CXP link trigger.

To keep the compatibility, when writing to this parameter, the value is copied to *FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP0* and sets the inverse value to *FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP1*.

Furthermore, the value of parameter *FG\_TRIGGERCAMERA\_SOURCE\_CXP0* is copied to *FG\_TRIGGERCAMERA\_SOURCE\_CXP1*.

Reading the value only represents the status of *FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP0* and can be ambiguous as the new parameters offer more possibilities which can't be represented with the legacy parameter.

This legacy parameter can't be used in configuration files anymore.

**Table 19.14. Parameter properties of FG\_TRIGGERCAMERA\_POLARITY**

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

**Example 19.14. 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 */
}

```

## 19.11. Debug

### 19.11.1. FG\_DEBUGSOURCE

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.15. 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</b> 0 <b>Maximum</b> 0 <b>Stepsize</b> 1
Default value	<b>0</b>

Example 19.15. 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 */
}

```

### 19.11.2. FG\_DEBUGSOURCENAME

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.16. Parameter properties of FG\_DEBUGSOURCENAME

Property	Value
Name	<b>FG_DEBUGSOURCENAME</b>
Display Name	<b>Debug Source Name</b>
Type	<b>String</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 19.16. Usage of FG\_DEBUGSOURCENAME

```

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

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

```

### 19.11.3. FG\_DEBUGSAVECONFIG

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.17. 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 19.17. 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 */
}

```

#### 19.11.4. FG\_DEBUG\_SLOWMODE

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.18. Parameter properties of FG\_DEBUG\_SLOWMODE

Property	Value
Name	<b>FG_DEBUG_SLOWMODE</b>
Display Name	<b>Debug Slow Output</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>FG_SLOW_OFF</b> Off <b>FG_SLOW_SOFTWARE</b> Software <b>FG_SLOW_PWM</b> PWM
Default value	<b>FG_SLOW_OFF</b>

Example 19.18. Usage of FG\_DEBUG\_SLOWMODE

```

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

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

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

```

#### 19.11.5. FG\_DEBUG\_SOFTWRAE\_SLOWGATE

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.19. Parameter properties of FG\_DEBUG\_SOFTWRAE\_SLOWGATE

Property	Value
Name	<b>FG_DEBUG_SOFTWRAE_SLOWGATE</b>
Display Name	<b>Debug Slow Software</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 <b>FG_PULSE</b> Pulse
Default value	<b>FG_OFF</b>

Example 19.19. Usage of FG\_DEBUG\_SOFTWRAE\_SLOWGATE

```

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

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

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

```

### 19.11.6. FG\_DEBUG\_PWM\_SLOWRATE

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.20. Parameter properties of FG\_DEBUG\_PWM\_SLOWRATE

Property	Value
Name	<b>FG_DEBUG_PWM_SLOWRATE</b>
Display Name	<b>Slowrate PWM</b>
Type	<b>Unsigned Integer (64 Bit)</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 144115188075855872 <b>Stepsize</b> 1
Default value	<b>10000000</b>

Example 19.20. Usage of FG\_DEBUG\_PWM\_SLOWRATE

```

int result = 0;
uint64_t value = 10000000;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT64_T;

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

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

```

### 19.11.7. FG\_DEBUG\_VERSION

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.21. 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 19.21. 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 */
}
```

### 19.11.8. FG\_DEBUG\_FRAMEID\_TO\_FIRSTPIXEL

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.22. Parameter properties of FG\_DEBUG\_FRAMEID\_TO\_FIRSTPIXEL

Property	Value
Name	<b>FG_DEBUG_FRAMEID_TO_FIRSTPIXEL</b>
Display Name	<b>FrameID mapped</b>
Type	<b>Enumeration</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Transient</b>
Allowed values	<b>FG_ON</b> On <b>FG_OFF</b> Off
Default value	<b>FG_OFF</b>

Example 19.22. Usage of FG\_DEBUG\_FRAMEID\_TO\_FIRSTPIXEL

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

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

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

### 19.11.9. Input

#### 19.11.9.1. FG\_DEBUGINENABLE

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.23. 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 19.23. 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 */
}

```

### 19.11.9.2. FG\_DEBUGFILE

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.24. 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 19.24. 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 */
}

```

### 19.11.9.3. FG\_DEBUGINSERT

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.25. 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 19.25. 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 */
}

```

#### 19.11.9.4. FG\_DEBUGWRITEPIXEL

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.26. 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</b> 0 <b>Maximum</b> 144115188075855872 <b>Stepsize</b> 1
Default value	<b>0</b>

Example 19.26. 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 */
}

```

#### 19.11.9.5. FG\_DEBUGWRITEFLAG



This parameter is for internal testing. Please DON'T use this parameter.

Table 19.27. 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 19.27. 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 */
}
```

### 19.11.9.6. FG\_DEBUGREADY

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.28. 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 19.28. 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 */
}
```

### 19.11.9.7. FG\_DEBUG\_FORCE\_FRAMEID

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.29. Parameter properties of FG\_DEBUG\_FORCE\_FRAMEID

Property	Value
Name	<b>FG_DEBUG_FORCE_FRAMEID</b>
Display Name	<b>Debug force FrameID</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 19.29. Usage of FG\_DEBUG\_FORCE\_FRAMEID

```

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

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

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

```

### 19.11.9.8. FG\_DEBUG\_FRAMEID

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.30. Parameter properties of FG\_DEBUG\_FRAMEID

Property	Value
Name	<b>FG_DEBUG_FRAMEID</b>
Display Name	<b>Debug FrameID</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read/Write/Change</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 65535 <b>Stepsize</b> 1
Default value	<b>0</b>

Example 19.30. Usage of FG\_DEBUG\_FRAMEID

```

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

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

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

```

### 19.11.9.9. FG\_DEBUG\_ENABLE\_SEQUENCEID

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.31. Parameter properties of FG\_DEBUG\_ENABLE\_SEQUENCEID

Property	Value
Name	<b>FG_DEBUG_ENABLE_SEQUENCEID</b>
Display Name	<b>Debug enable Sequence ID</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 19.31. Usage of FG\_DEBUG\_ENABLE\_SEQUENCEID

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

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

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

## 19.11.10. Output

### 19.11.10.1. FG\_DEBUGOUTENABLE

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.32. 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 19.32. 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 */
}
```

### 19.11.10.2. FG\_DEBUGOUTXPOS

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.33. 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</b> 48 <b>Maximum</b> 32768 <b>Stepsize</b> 1
Unit of measure	<b>pixel</b>

Example 19.33. 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 */
}
```

### 19.11.10.3. FG\_DEBUGOUTYPOS

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.34. 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</b> 1 <b>Maximum</b> 8388607 <b>Stepsize</b> 1
Unit of measure	<b>pixel</b>

Example 19.34. 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 */
}
```

### 19.11.10.4. FG\_DEBUGOUTPIXEL

This parameter is for internal testing. Please DON'T use this parameter.

Table 19.35. 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 19.35. 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 */
}
```

## 19.12. GenTL

### 19.12.1. FG\_GENTL\_INFO\_VERSION

This parameter gives the version of the GenTL description used by our GenTL producer. This parameter is of internal use only.

Table 19.36. Parameter properties of FG\_GENTL\_INFO\_VERSION

Property	Value
Name	<b>FG_GENTL_INFO_VERSION</b>
Display Name	<b>Version</b>
Type	<b>String</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Unit of measure	

Example 19.36. Usage of FG\_GENTL\_INFO\_VERSION

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

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

### 19.12.2. FG\_GENTL\_INFO\_IGNOREFGFORMAT

This parameter describes the handling of outputformats in the GenTL producer. If the parameter is set to 1 the handling of Output formats is done by the producer ignoring the Outputformatsettings of the Applet.

Table 19.37. Parameter properties of FG\_GENTL\_INFO\_IGNOREFGFORMAT

Property	Value
Name	<b>FG_GENTL_INFO_IGNOREFGFORMAT</b>
Display Name	<b>IgnoreFGFormat</b>
Type	<b>String</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 19.37. Usage of FG\_GENTL\_INFO\_IGNOREFGFORMAT

```

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

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

```

### 19.12.3. FG\_GENTL\_INFO\_OVERFLOWCAPABLE

This parameter informs the producer that the Applet is capable of extended overflow management.

Table 19.38. Parameter properties of FG\_GENTL\_INFO\_OVERFLOWCAPABLE

Property	Value
Name	<b>FG_GENTL_INFO_OVERFLOWCAPABLE</b>
Display Name	<b>OverflowCapable</b>
Type	<b>String</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 19.38. Usage of FG\_GENTL\_INFO\_OVERFLOWCAPABLE

```

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

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

```

# Chapter 20. Boardstatus

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

## 20.1. FG\_SYSTEMMONITOR\_CHANNEL\_CURRENT

Returns the power consumption of the CXP channel (PoCXP) in Ampere.

Table 20.1. Parameter properties of FG\_SYSTEMMONITOR\_CHANNEL\_CURRENT

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

Example 20.1. 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 */
}
}
```

## 20.2. FG\_SYSTEMMONITOR\_CHANNEL\_VOLTAGE

Returns the voltage of the CXP channel (PoCXP).

Table 20.2. Parameter properties of FG\_SYSTEMMONITOR\_CHANNEL\_VOLTAGE

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

Example 20.2. 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 */
    }
}
```

## 20.3. FG\_SYSTEMMONITOR\_MAPPED\_TO\_FG\_PORT

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

Table 20.3. Parameter properties of FG\_SYSTEMMONITOR\_MAPPED\_TO\_FG\_PORT

Property	Value
Name	<b>FG_SYSTEMMONITOR_MAPPED_TO_FG_PORT</b>
Display Name	<b>Systemmonitor Mapped to Fg Port</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 20.3. Usage of FG\_SYSTEMMONITOR\_MAPPED\_TO\_FG\_PORT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

## 20.4. FG\_DMASTATUS

Returns the status of the DMA transmission, i.e. the acquisition state. 0 = stopped DMA, 1 = started DMA.

Table 20.4. 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>
Allowed values	<b>Minimum</b> <b>0</b> <b>Maximum</b> <b>1</b> <b>Stepsize</b> <b>1</b>

Example 20.4. 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 */
}
```



}

## 20.5. FG\_SYSTEMMONITOR\_FPGA\_TEMPERATURE

Returns the current FGPA temperature.

Table 20.5. Parameter properties of FG\_SYSTEMMONITOR\_FPGA\_TEMPERATURE

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

Example 20.5. 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 */
}

```

## 20.6. FG\_SYSTEMMONITOR\_FPGA\_VCC\_INT

Returns the internal voltage of the FPGA.

Table 20.6. Parameter properties of FG\_SYSTEMMONITOR\_FPGA\_VCC\_INT

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

Example 20.6. 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 */
}

```

}

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

Returns the VCC auxiliary voltage of the FPGA.

Table 20.7. Parameter properties of FG\_SYSTEMMONITOR\_FPGA\_VCC\_AUX

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

Example 20.7. 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 */
}
```

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

Returns the VCC of the BlockRAM voltage of the FPGA.

Table 20.8. Parameter properties of FG\_SYSTEMMONITOR\_FPGA\_VCC\_BRAM

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

Example 20.8. 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 */
}
```

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

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

Table 20.9. 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</b> 0 <b>Maximum</b> 15 <b>Stepsize</b> 0
Unit of measure	<b>lanes</b>

Example 20.9. 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 */
}
```

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

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

Table 20.10. Parameter properties of FG\_SYSTEMMONITOR\_CURRENT\_LINK\_SPEED

Property	Value
Name	<b>FG_SYSTEMMONITOR_CURRENT_LINK_SPEED</b>
Display Name	<b>Systemmonitor Current Link Speed</b>
Type	<b>Double</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum</b> 0.0 <b>Maximum</b> 1000.0 <b>Stepsize</b> 0.5
Unit of measure	<b>Gb/s</b>

Example 20.10. 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 */
}
```

}

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

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

Table 20.11. Parameter properties of FG\_SYSTEMMONITOR\_PCIE\_TRAINED\_PAYLOAD\_SIZE

Property	Value
Name	<b>FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE</b>
Display Name	<b>Systemmonitor PCIe Trained Payload Size</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 1024 <b>Stepsize</b> 1
Unit of measure	<b>byte</b>

Example 20.11. 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 */
}
```

## 20.12. FG\_SYSTEMMONITOR\_PCIE\_TRAINED\_REQUEST\_SIZE

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

Table 20.12. Parameter properties of FG\_SYSTEMMONITOR\_PCIE\_TRAINED\_REQUEST\_SIZE

Property	Value
Name	<b>FG_SYSTEMMONITOR_PCIE_TRAINED_REQUEST_SIZE</b>
Display Name	<b>Systemmonitor PCIe Trained Request Size</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 4096 <b>Stepsize</b> 1
Unit of measure	<b>byte</b>

Example 20.12. Usage of FG\_SYSTEMMONITOR\_PCIE\_TRAINED\_REQUEST\_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_REQUEST_SIZE, &value, 0, type)) < 0) {
    /* error handling */
}
```

---

}

---

## 20.13. FG\_SYSTEMMONITOR\_EXTERNAL\_POWER

Indicates whether the external power connector is connected.

Table 20.13. Parameter properties of FG\_SYSTEMMONITOR\_EXTERNAL\_POWER

Property	Value
Name	<b>FG_SYSTEMMONITOR_EXTERNAL_POWER</b>
Display Name	<b>Systemmonitor External Power</b>
Type	<b>Enumeration</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>
Allowed values	<b>FG_GOOD</b> Good <b>FG_NO_POWER</b> No Power

Example 20.13. Usage of FG\_SYSTEMMONITOR\_EXTERNAL\_POWER

---

```

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

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

```

---

## 20.14. FG\_CXP\_INPUT\_MAPPED\_FW\_PORT\_PORT

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

Table 20.14. Parameter properties of FG\_CXP\_INPUT\_MAPPED\_FW\_PORT\_PORT

Property	Value
Name	<b>FG_CXP_INPUT_MAPPED_FW_PORT_PORT</b>
Display Name	<b>CXP Input Mapped to Firmware Port Port</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 20.14. Usage of FG\_CXP\_INPUT\_MAPPED\_FW\_PORT\_PORT

---

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

---

# Chapter 21. Errors

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

## 21.1. FG\_SYSTEMMONITOR\_DECODER\_8B\_10B\_ERROR

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

Table 21.1. Parameter properties of FG\_SYSTEMMONITOR\_DECODER\_8B\_10B\_ERROR

Property	Value
Name	FG_SYSTEMMONITOR_DECODER_8B_10B_ERROR
Display Name	Systemmonitor Decoder 8b10b Error
Type	Unsigned Integer Field (64 Bit)
Field Size	4
Access policy	Read-Only
Storage policy	Transient

Example 21.1. Usage of FG\_SYSTEMMONITOR\_DECODER\_8B\_10B\_ERROR

```
int result = 0;

FieldParameterAccess access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMACCESS;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_DECODER_8B_10B_ERROR, &access, 0, type)) < 0) {
    /* error handling */
}
}
```

## 21.2. FG\_SYSTEMMONITOR\_BYTE\_ALIGNMENT\_8B\_10B\_LOCKED

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

Table 21.2. Parameter properties of FG\_SYSTEMMONITOR\_BYTE\_ALIGNMENT\_8B\_10B\_LOCKED

Property	Value
Name	FG_SYSTEMMONITOR_BYTE_ALIGNMENT_8B_10B_LOCKED
Display Name	Systemmonitor Byte Alignment 8B 10 B Locked
Type	Unsigned Integer Field
Field Size	4
Access policy	Read-Only
Storage policy	Transient

Example 21.2. Usage of FG\_SYSTEMMONITOR\_BYTE\_ALIGNMENT\_8B\_10B\_LOCKED

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;
```

```

    if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_BYTE_ALIGNMENT_8B_10B_LOCKED, &access, 0, type)) < 0) {
        /* error handling */
    }
}

```

## 21.3. FG\_SYSTEMMONITOR\_RX\_STREAM\_INCOMPLETE\_COUNT

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

Table 21.3. Parameter properties of FG\_SYSTEMMONITOR\_RX\_STREAM\_INCOMPLETE\_COUNT

Property	Value
Name	<b>FG_SYSTEMMONITOR_RX_STREAM_INCOMPLETE_COUNT</b>
Display Name	<b>Systemmonitor Rx Stream Incomplete Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.3. Usage of FG\_SYSTEMMONITOR\_RX\_STREAM\_INCOMPLETE\_COUNT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

## 21.4. FG\_SYSTEMMONITOR\_RX\_UNKNOWN\_DATA\_RECEIVED\_COUNT

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

Table 21.4. Parameter properties of FG\_SYSTEMMONITOR\_RX\_UNKNOWN\_DATA\_RECEIVED\_COUNT

Property	Value
Name	<b>FG_SYSTEMMONITOR_RX_UNKNOWN_DATA_RECEIVED_COUNT</b>
Display Name	<b>Systemmonitor Rx Unknown Data Received Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.4. Usage of FG\_SYSTEMMONITOR\_RX\_UNKNOWN\_DATA\_RECEIVED\_COUNT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

}

## 21.5. FG\_CXP\_OVERTRIGGER\_REQUEST\_PULSECOUNT

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

Table 21.5. Parameter properties of FG\_CXP\_OVERTRIGGER\_REQUEST\_PULSECOUNT

Property	Value
Name	<b>FG_CXP_OVERTRIGGER_REQUEST_PULSECOUNT</b>
Display Name	<b>CXP Overtrigger Request Pulse Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.5. Usage of FG\_CXP\_OVERTRIGGER\_REQUEST\_PULSECOUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

## 21.6. FG\_CXP\_TRIGGER\_ACK\_MISSING\_COUNT

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

Table 21.6. Parameter properties of FG\_CXP\_TRIGGER\_ACK\_MISSING\_COUNT

Property	Value
Name	<b>FG_CXP_TRIGGER_ACK_MISSING_COUNT</b>
Display Name	<b>CXP Lost Trigger ACK Missing Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.6. Usage of FG\_CXP\_TRIGGER\_ACK\_MISSING\_COUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

if ((result = Fg_getParameterWithType(fg, FG_CXP_TRIGGER_ACK_MISSING_COUNT, &access, 0, type)) < 0) {
```



```

    } /* error handling */
}

```

## 21.7. FG\_CXP\_CONTROL\_ACK\_LOST\_COUNT

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

Table 21.7. Parameter properties of FG\_CXP\_CONTROL\_ACK\_LOST\_COUNT

Property	Value
Name	<b>FG_CXP_CONTROL_ACK_LOST_COUNT</b>
Display Name	<b>CXP Control ACK Lost Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.7. Usage of FG\_CXP\_CONTROL\_ACK\_LOST\_COUNT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

## 21.8. FG\_CXP\_CONTROL\_TAG\_ERROR\_COUNT

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

Table 21.8. Parameter properties of FG\_CXP\_CONTROL\_TAG\_ERROR\_COUNT

Property	Value
Name	<b>FG_CXP_CONTROL_TAG_ERROR_COUNT</b>
Display Name	<b>CXP Control Tag Error Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.8. Usage of FG\_CXP\_CONTROL\_TAG\_ERROR\_COUNT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

```

```

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

```

## 21.9. FG\_CXP\_CONTROL\_ACK\_INCOMPLETE\_COUNT

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

Table 21.9. Parameter properties of FG\_CXP\_CONTROL\_ACK\_INCOMPLETE\_COUNT

Property	Value
Name	<b>FG_CXP_CONTROL_ACK_INCOMPLETE_COUNT</b>
Display Name	<b>CXP Control ACK Incomplete Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.9. Usage of FG\_CXP\_CONTROL\_ACK\_INCOMPLETE\_COUNT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

## 21.10. FG\_CXP\_HEARTBEAT\_INCOMPLETE\_COUNT

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

Table 21.10. Parameter properties of FG\_CXP\_HEARTBEAT\_INCOMPLETE\_COUNT

Property	Value
Name	<b>FG_CXP_HEARTBEAT_INCOMPLETE_COUNT</b>
Display Name	<b>CXP Heartbeat Incomplete Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.10. Usage of FG\_CXP\_HEARTBEAT\_INCOMPLETE\_COUNT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

```

```

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

```

## 21.11. FG\_CXP\_HEARTBEAT\_MAX\_PERIOD\_VIOLATION\_COUNT

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

Table 21.11. Parameter properties of FG\_CXP\_HEARTBEAT\_MAX\_PERIOD\_VIOLATION\_COUNT

Property	Value
Name	<b>FG_CXP_HEARTBEAT_MAX_PERIOD_VIOLATION_COUNT</b>
Display Name	<b>CXP Hearbeat Max Period Violation Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.11. Usage of FG\_CXP\_HEARTBEAT\_MAX\_PERIOD\_VIOLATION\_COUNT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

## 21.12. FG\_PACKET\_TAG\_ERROR\_COUNT

The parameter counts the number of lost CXP stream packets.

Table 21.12. Parameter properties of FG\_PACKET\_TAG\_ERROR\_COUNT

Property	Value
Name	<b>FG_PACKET_TAG_ERROR_COUNT</b>
Display Name	<b>Packet Tag Error Count</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 4095 <b>Stepsize</b> 1

Example 21.12. Usage of FG\_PACKET\_TAG\_ERROR\_COUNT

```

int result = 0;
unsigned int value = 0;

```

```

const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

```

## 21.13. FG\_SYSTEMMONITOR\_PACKETBUFFER\_OVERFLOW\_COUNT

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

Table 21.13. Parameter properties of FG\_SYSTEMMONITOR\_PACKETBUFFER\_OVERFLOW\_COUNT

Property	Value
Name	<b>FG_SYSTEMMONITOR_PACKETBUFFER_OVERFLOW_COUNT</b>
Display Name	<b>Systemmonitor Packetbuffer Overflow Count</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 4095 <b>Stepsize</b> 1

Example 21.13. Usage of FG\_SYSTEMMONITOR\_PACKETBUFFER\_OVERFLOW\_COUNT

```

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

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

```

## 21.14. FG\_SYSTEMMONITOR\_PACKETBUFFER\_OVERFLOW\_SOURCE

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

Table 21.14. Parameter properties of FG\_SYSTEMMONITOR\_PACKETBUFFER\_OVERFLOW\_SOURCE

Property	Value
Name	<b>FG_SYSTEMMONITOR_PACKETBUFFER_OVERFLOW_SOURCE</b>
Display Name	<b>Systemmonitor Packetbuffer Overflow Source</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 15 <b>Stepsize</b> 1

Example 21.14. Usage of FG\_SYSTEMMONITOR\_PACKETBUFFER\_OVERFLOW\_SOURCE

```

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

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

```

---

}

---

## 21.15. FG\_CXP\_IMAGETAG\_ERROR\_COUNT

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

Table 21.15. Parameter properties of FG\_CXP\_IMAGETAG\_ERROR\_COUNT

Property	Value
Name	<b>FG_CXP_IMAGETAG_ERROR_COUNT</b>
Display Name	<b>CXP Image Tag Error Count</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 8191 <b>Stepsize</b> 1

Example 21.15. Usage of FG\_CXP\_IMAGETAG\_ERROR\_COUNT

---

```

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

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

```

---

## 21.16. FG\_CXP\_STREAMID\_ERROR\_COUNT

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

Table 21.16. Parameter properties of FG\_CXP\_STREAMID\_ERROR\_COUNT

Property	Value
Name	<b>FG_CXP_STREAMID_ERROR_COUNT</b>
Display Name	<b>CXP Stream ID Error Count</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 8191 <b>Stepsize</b> 1

Example 21.16. Usage of FG\_CXP\_STREAMID\_ERROR\_COUNT

---

```

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

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

```

---

}

## 21.17. FG\_CXP\_CAMERA\_MARKER\_ERROR\_COUNT

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

Table 21.17. Parameter properties of FG\_CXP\_CAMERA\_MARKER\_ERROR\_COUNT

Property	Value
Name	<b>FG_CXP_CAMERA_MARKER_ERROR_COUNT</b>
Display Name	<b>CXP Camera Marker Error Count</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 8191 <b>Stepsize</b> 1

Example 21.17. Usage of FG\_CXP\_CAMERA\_MARKER\_ERROR\_COUNT

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

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

## 21.18. FG\_CXP\_CAMERA\_UNEXPECTED\_STARTUP\_DATA

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

Table 21.18. Parameter properties of FG\_CXP\_CAMERA\_UNEXPECTED\_STARTUP\_DATA

Property	Value
Name	<b>FG_CXP_CAMERA_UNEXPECTED_STARTUP_DATA</b>
Display Name	<b>CXP Camera Unexpected Startup Data Status</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 21.18. Usage of FG\_CXP\_CAMERA\_UNEXPECTED\_STARTUP\_DATA

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

if ((result = Fg_getParameterWithType(fg, FG_CXP_CAMERA_UNEXPECTED_STARTUP_DATA, &value, 0, type)) < 0) {
```

```

    /* error handling */
}

```

## 21.19. FG\_CXP\_CAMERA\_FRAME\_LOST\_COUNT

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

Table 21.19. Parameter properties of FG\_CXP\_CAMERA\_FRAME\_LOST\_COUNT

Property	Value
Name	<b>FG_CXP_CAMERA_FRAME_LOST_COUNT</b>
Display Name	<b>CXP Camera Frame Lost Count</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 33554431 <b>Stepsize</b> 1

Example 21.19. Usage of FG\_CXP\_CAMERA\_FRAME\_LOST\_COUNT

```

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

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

```

## 21.20. FG\_CXP\_CAMERA\_FRAME\_CORRUPT\_COUNT

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

Table 21.20. Parameter properties of FG\_CXP\_CAMERA\_FRAME\_CORRUPT\_COUNT

Property	Value
Name	<b>FG_CXP_CAMERA_FRAME_CORRUPT_COUNT</b>
Display Name	<b>CXP Camera Frame Corrupt Count</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 33554431 <b>Stepsize</b> 1

Example 21.20. Usage of FG\_CXP\_CAMERA\_FRAME\_CORRUPT\_COUNT

```

int result = 0;

```

```

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

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

```

## 21.21. CRC

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

### 21.21.1. FG\_SYSTEMMONITOR\_RX\_PACKET\_CRC\_ERROR\_COUNT

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

Table 21.21. Parameter properties of FG\_SYSTEMMONITOR\_RX\_PACKET\_CRC\_ERROR\_COUNT

Property	Value
Name	<b>FG_SYSTEMMONITOR_RX_PACKET_CRC_ERROR_COUNT</b>
Display Name	<b>Systemmonitor Rx Packet CRC Error Count</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.21. Usage of FG\_SYSTEMMONITOR\_RX\_PACKET\_CRC\_ERROR\_COUNT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

### 21.21.2. FG\_CXP\_STREAMPACKET\_CRC\_ERROR

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

Table 21.22. Parameter properties of FG\_CXP\_STREAMPACKET\_CRC\_ERROR

Property	Value
Name	<b>FG_CXP_STREAMPACKET_CRC_ERROR</b>
Display Name	<b>CXP Stream Packet CRC Error</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.22. Usage of FG\_CXP\_STREAMPACKET\_CRC\_ERROR

```

int result = 0;

FieldParameterInt access;

```



```

const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

### 21.21.3. FG\_CXP\_CONTROL\_ACK\_PACKET\_CRC\_ERROR

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

Table 21.23. Parameter properties of FG\_CXP\_CONTROL\_ACK\_PACKET\_CRC\_ERROR

Property	Value
Name	FG_CXP_CONTROL_ACK_PACKET_CRC_ERROR
Display Name	CXP Control ACK Packet CRC Error
Type	Unsigned Integer Field
Field Size	4
Access policy	Read-Only
Storage policy	Transient

Example 21.23. Usage of FG\_CXP\_CONTROL\_ACK\_PACKET\_CRC\_ERROR

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

## 21.22. LengthErrors

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

### 21.22.1. FG\_SYSTEMMONITOR\_RX\_LENGTH\_ERROR\_COUNT

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

Table 21.24. Parameter properties of FG\_SYSTEMMONITOR\_RX\_LENGTH\_ERROR\_COUNT

Property	Value
Name	FG_SYSTEMMONITOR_RX_LENGTH_ERROR_COUNT
Display Name	Systemmonitor Rx Length Error Count
Type	Unsigned Integer Field
Field Size	4
Access policy	Read-Only
Storage policy	Transient

Example 21.24. Usage of FG\_SYSTEMMONITOR\_RX\_LENGTH\_ERROR\_COUNT

```

int result = 0;

```

```
FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

## 21.22.2. FG\_CXP\_STREAMPACKET\_LENGTH\_ERROR

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

Table 21.25. Parameter properties of FG\_CXP\_STREAMPACKET\_LENGTH\_ERROR

Property	Value
Name	FG_CXP_STREAMPACKET_LENGTH_ERROR
Display Name	CXP Stream Packet Length Error
Type	Unsigned Integer Field
Field Size	4
Access policy	Read-Only
Storage policy	Transient

Example 21.25. Usage of FG\_CXP\_STREAMPACKET\_LENGTH\_ERROR

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

## 21.23. ReceivedPacketsCorrected

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

### 21.23.1. FG\_CXP\_ERROR\_CORRECTED

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

Table 21.26. Parameter properties of FG\_CXP\_ERROR\_CORRECTED

Property	Value
Name	FG_CXP_ERROR_CORRECTED
Display Name	CXP Error Corrected
Type	Unsigned Integer Field
Field Size	4
Access policy	Read-Only
Storage policy	Transient

**Example 21.26. Usage of FG\_CXP\_ERROR\_CORRECTED**

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

### 21.23.2. FG\_CXP\_ERROR\_CORRECTED\_TRIGGER

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

**Table 21.27. Parameter properties of FG\_CXP\_ERROR\_CORRECTED\_TRIGGER**

Property	Value
Name	<b>FG_CXP_ERROR_CORRECTED_TRIGGER</b>
Display Name	<b>CXP Error Corrected Trigger</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

**Example 21.27. Usage of FG\_CXP\_ERROR\_CORRECTED\_TRIGGER**

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

### 21.23.3. FG\_CXP\_ERROR\_CORRECTED\_TRIGGER\_ACK

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

**Table 21.28. Parameter properties of FG\_CXP\_ERROR\_CORRECTED\_TRIGGER\_ACK**

Property	Value
Name	<b>FG_CXP_ERROR_CORRECTED_TRIGGER_ACK</b>
Display Name	<b>CXP Error Corrected Trigger ACK</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

**Example 21.28. Usage of FG\_CXP\_ERROR\_CORRECTED\_TRIGGER\_ACK**

```

int result = 0;

```

```
FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

#### 21.23.4. FG\_CXP\_ERROR\_CORRECTED\_STREAM

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

Table 21.29. Parameter properties of FG\_CXP\_ERROR\_CORRECTED\_STREAM

Property	Value
Name	<b>FG_CXP_ERROR_CORRECTED_STREAM</b>
Display Name	<b>CXP Error Corrected Stream</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.29. Usage of FG\_CXP\_ERROR\_CORRECTED\_STREAM

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

#### 21.23.5. FG\_CXP\_ERROR\_CORRECTED\_CONTROL\_ACK

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

Table 21.30. Parameter properties of FG\_CXP\_ERROR\_CORRECTED\_CONTROL\_ACK

Property	Value
Name	<b>FG_CXP_ERROR_CORRECTED_CONTROL_ACK</b>
Display Name	<b>CXP Error Corrected Control ACK</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.30. Usage of FG\_CXP\_ERROR\_CORRECTED\_CONTROL\_ACK

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;
```

```

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

```

### 21.23.6. FG\_CXP\_ERROR\_CORRECTED\_LINKTEST

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

Table 21.31. Parameter properties of FG\_CXP\_ERROR\_CORRECTED\_LINKTEST

Property	Value
Name	<b>FG_CXP_ERROR_CORRECTED_LINKTEST</b>
Display Name	<b>CXP Error Corrected Link Test</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.31. Usage of FG\_CXP\_ERROR\_CORRECTED\_LINKTEST

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

### 21.23.7. FG\_CXP\_ERROR\_CORRECTED\_HEARTBEAT

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

Table 21.32. Parameter properties of FG\_CXP\_ERROR\_CORRECTED\_HEARTBEAT

Property	Value
Name	<b>FG_CXP_ERROR_CORRECTED_HEARTBEAT</b>
Display Name	<b>CXP Error Corrected Heartbeat</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.32. Usage of FG\_CXP\_ERROR\_CORRECTED\_HEARTBEAT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

```

    }
}

```

### 21.23.8. FG\_CORRECTED\_ERROR\_COUNT

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

Table 21.33. Parameter properties of FG\_CORRECTED\_ERROR\_COUNT

Property	Value
Name	<b>FG_CORRECTED_ERROR_COUNT</b>
Display Name	<b>Corrected Error Count</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 4095 <b>Stepsize</b> 1

Example 21.33. Usage of FG\_CORRECTED\_ERROR\_COUNT

```

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

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

```

## 21.24. ReceivedPacketsUncorrected

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

### 21.24.1. FG\_CXP\_ERROR\_UNCORRECTED

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

Table 21.34. Parameter properties of FG\_CXP\_ERROR\_UNCORRECTED

Property	Value
Name	<b>FG_CXP_ERROR_UNCORRECTED</b>
Display Name	<b>CXP Error Uncorrected</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.34. Usage of FG\_CXP\_ERROR\_UNCORRECTED

```

int result = 0;

```

```
FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

### 21.24.2. FG\_CXP\_ERROR\_UNCORRECTED\_TRIGGER

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

Table 21.35. Parameter properties of FG\_CXP\_ERROR\_UNCORRECTED\_TRIGGER

Property	Value
Name	<b>FG_CXP_ERROR_UNCORRECTED_TRIGGER</b>
Display Name	<b>CXP Error Uncorrected Trigger</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.35. Usage of FG\_CXP\_ERROR\_UNCORRECTED\_TRIGGER

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

### 21.24.3. FG\_CXP\_ERROR\_UNCORRECTED\_TRIGGER\_ACK

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

Table 21.36. Parameter properties of FG\_CXP\_ERROR\_UNCORRECTED\_TRIGGER\_ACK

Property	Value
Name	<b>FG_CXP_ERROR_UNCORRECTED_TRIGGER_ACK</b>
Display Name	<b>CXP Error Uncorrected Trigger ACK</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.36. Usage of FG\_CXP\_ERROR\_UNCORRECTED\_TRIGGER\_ACK

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;
```

```

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

```

#### 21.24.4. FG\_CXP\_ERROR\_UNCORRECTED\_STREAM

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

Table 21.37. Parameter properties of FG\_CXP\_ERROR\_UNCORRECTED\_STREAM

Property	Value
Name	<b>FG_CXP_ERROR_UNCORRECTED_STREAM</b>
Display Name	<b>CXP Error Uncorrected Stream</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.37. Usage of FG\_CXP\_ERROR\_UNCORRECTED\_STREAM

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

#### 21.24.5. FG\_CXP\_ERROR\_UNCORRECTED\_CONTROL\_ACK

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

Table 21.38. Parameter properties of FG\_CXP\_ERROR\_UNCORRECTED\_CONTROL\_ACK

Property	Value
Name	<b>FG_CXP_ERROR_UNCORRECTED_CONTROL_ACK</b>
Display Name	<b>CXP Error Uncorrected Control ACK</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.38. Usage of FG\_CXP\_ERROR\_UNCORRECTED\_CONTROL\_ACK

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```



```

    }
}

```

### 21.24.6. FG\_CXP\_ERROR\_UNCORRECTED\_LINKTEST

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

Table 21.39. Parameter properties of FG\_CXP\_ERROR\_UNCORRECTED\_LINKTEST

Property	Value
Name	<b>FG_CXP_ERROR_UNCORRECTED_LINKTEST</b>
Display Name	<b>CXP Error Uncorrected Link Test</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.39. Usage of FG\_CXP\_ERROR\_UNCORRECTED\_LINKTEST

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

### 21.24.7. FG\_CXP\_ERROR\_UNCORRECTED\_HEARTBEAT

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

Table 21.40. Parameter properties of FG\_CXP\_ERROR\_UNCORRECTED\_HEARTBEAT

Property	Value
Name	<b>FG_CXP_ERROR_UNCORRECTED_HEARTBEAT</b>
Display Name	<b>CXP Error Uncorrected Heartbeat</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.40. Usage of FG\_CXP\_ERROR\_UNCORRECTED\_HEARTBEAT

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

## 21.24.8. FG\_UNCORRECTED\_ERROR\_COUNT

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

Table 21.41. Parameter properties of FG\_UNCORRECTED\_ERROR\_COUNT

Property	Value
Name	<b>FG_UNCORRECTED_ERROR_COUNT</b>
Display Name	<b>Uncorrected Error Count</b>
Type	<b>Unsigned Integer</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Persistent</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 4095 <b>Stepsize</b> 1

Example 21.41. Usage of FG\_UNCORRECTED\_ERROR\_COUNT

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

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

## 21.25. UnsupportedPackets

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

### 21.25.1. FG\_SYSTEMMONITOR\_RX\_UNSUPPORTED\_PACKET\_COUNT

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

Table 21.42. Parameter properties of FG\_SYSTEMMONITOR\_RX\_UNSUPPORTED\_PACKET\_COUNT

Property	Value
Name	<b>FG_SYSTEMMONITOR_RX_UNSUPPORTED_PACKET_COUNT</b>
Display Name	<b>Systemmonitor Rx Unsupported Packet Unit</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.42. Usage of FG\_SYSTEMMONITOR\_RX\_UNSUPPORTED\_PACKET\_COUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

---

}

---

### 21.25.2. FG\_CXP\_UNSUPPORTED\_GPIO\_RECEIVED

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

Table 21.43. Parameter properties of FG\_CXP\_UNSUPPORTED\_GPIO\_RECEIVED

Property	Value
Name	<b>FG_CXP_UNSUPPORTED_GPIO_RECEIVED</b>
Display Name	<b>CXP Unsupported GPIO Received</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.43. Usage of FG\_CXP\_UNSUPPORTED\_GPIO\_RECEIVED

---

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

---

### 21.25.3. FG\_CXP\_UNSUPPORTED\_EVENT\_RECEIVED

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

Table 21.44. Parameter properties of FG\_CXP\_UNSUPPORTED\_EVENT\_RECEIVED

Property	Value
Name	<b>FG_CXP_UNSUPPORTED_EVENT_RECEIVED</b>
Display Name	<b>CXP Unsupported Event Received</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.44. Usage of FG\_CXP\_UNSUPPORTED\_EVENT\_RECEIVED

---

```

int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```

---

### 21.25.4. FG\_CXP\_UNSUPPORTED\_HEARTBEAT\_RECEIVED

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

Table 21.45. Parameter properties of FG\_CXP\_UNSUPPORTED\_HEARTBEAT\_RECEIVED

Property	Value
Name	<b>FG_CXP_UNSUPPORTED_HEARTBEAT_RECEIVED</b>
Display Name	<b>CXP Unsupported Hearbeat Received</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.45. Usage of FG\_CXP\_UNSUPPORTED\_HEARTBEAT\_RECEIVED

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

### 21.25.5. FG\_CXP\_UNSUPPORTED\_GPIO\_ACK\_RECEIVED

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

Table 21.46. Parameter properties of FG\_CXP\_UNSUPPORTED\_GPIO\_ACK\_RECEIVED

Property	Value
Name	<b>FG_CXP_UNSUPPORTED_GPIO_ACK_RECEIVED</b>
Display Name	<b>CXP Unsupported GPIO ACK Received</b>
Type	<b>Unsigned Integer Field</b>
Field Size	<b>4</b>
Access policy	<b>Read-Only</b>
Storage policy	<b>Transient</b>

Example 21.46. Usage of FG\_CXP\_UNSUPPORTED\_GPIO\_ACK\_RECEIVED

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

### 21.25.6. FG\_CXP\_UNSUPPORTED\_GPIO\_REQUEST\_RECEIVED

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

Table 21.47. Parameter properties of FG\_CXP\_UNSUPPORTED\_GPIO\_REQUEST\_RECEIVED

Property	Value
Name	FG_CXP_UNSUPPORTED_GPIO_REQUEST_RECEIVED
Display Name	CXP Unsupported GPIO Request Received
Type	Unsigned Integer Field
Field Size	4
Access policy	Read-Only
Storage policy	Transient

Example 21.47. Usage of FG\_CXP\_UNSUPPORTED\_GPIO\_REQUEST\_RECEIVED

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

# Chapter 22. Revision History

Revision history of acquisition applet releases.

Applet Version	Release Date	Change Log	Delivered with
1.0.1.0	22 Dec 2023	Initial version of this applet. The features are exactly the same as in the applets for imaWorx CXP-12 Quad.	Framegrabber SDK 5.11.2
1.1.1.0	31 Jul 2024	<ul style="list-style-type: none"><li>This applet has been extended with additional parameters and events for problem analysis, e.g. to find errors in the CXP stream. The new parameters and events have been added to the parameter tree.</li><li>Bug fixes. See Section 22.1, 'Fixed Issues' for a detailed list of fixed issues.</li></ul>	Framegrabber SDK 5.11.3
1.2.2.0	25 Feb 2025	Bug fixes. See Section 22.1, 'Fixed Issues' for a detailed list of fixed issues.	Framegrabber SDK 5.11.4

## 22.1. Fixed Issues

### 22.1.1. Fixed in Version 1.2.2.0

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

### 22.1.2. Fixed in Version 1.1.1.0

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

## 22.2. Known Issues

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

---

# Glossary

Area of Interest (AOI)	See Region of Interest.
Board	A Basler hardware. Usually, a board is represented by a frame grabber. Boards might comprise multiple devices.
Board ID Number	An identification number of a Basler board in a PC system. The number is not fixed to a specific hardware but has to be unique in a PC system.
Camera Index	The index of a camera connected to a frame grabber. The first camera will have index zero. Mind the difference between the camera index and the frame grabber camera port. See also Camera Port.
Camera Port	The Basler frame grabber connectors for cameras are called camera ports. They are numbered {0, 1, 2, ...} or enumerated {A, B, C, ...}. Depending on the interface one camera could be connected to multiple camera ports. Also, multiple cameras could be connected to one camera port.
Camera Tap	See Tap.
Device	A board can consist of multiple devices. Devices are numbered. The first device usually has number one.
Direct Memory Access (DMA)	<p>A DMA transfer allows hardware subsystems within the computer to access the system memory independently of the central processing unit (CPU).</p> <p>Basler uses DMAs for data transfer such as image data between a board e.g. a frame grabber and a PC. Data transfers can be established in multiple directions i.e. from a frame grabber to the PC (download) and from the PC to a frame grabber (upload). Multiple DMA channels may exist for one board. Control and configuration data usually do not use DMA channels.</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>Basler uses hardware interrupts for the event transfer and processing is absolutely optimized for low latency. These interrupts are only produced by the frame grabber if an event is registered and activated by software. If an event is fired at a very high frequency this may influence the system performance.</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	<p>Simple programming of FPGA-based image processing devices.</p> <p>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.</p>

---

# Index

## A

Area of Interest, 22

## B

Bandwidth, 3

Boardstatus, 143

## C

Camera, 16

Events, 16

Format, 7

Interface, 4, 16

Camera Simulator, 112, 112

Camera Trigger Source, 26, 34, 38, 39

Camera::Events, 16

CoaXPress, 7

CoaXPress::Test, 12

Color Converter, 92

## D

Debugging, 75

Digital I/O, 26, 26

Digital I/O::Camera, 26

Digital I/O::Event Source, 39

Digital I/O::Events, 42

Digital I/O::GPI, 38

Digital I/O::GPO, 34

## E

Errors, 150

Errors::CRC, 160

Errors::LengthErrors, 161

Errors::ReceivedPacketsCorrected, 162

Errors::ReceivedPacketsUncorrected, 166

Errors::UnsupportedPackets, 170

Events

Camera, 16

Overflow, 87

Trigger, 42

## F

Features, 1

FG\_APPLET\_BUILD\_TIME, 124

FG\_APPLET\_ID, 123

FG\_APPLET\_REVISION, 127

FG\_APPLET\_VERSION, 127

FG\_BITALIGNMENT, 109

FG\_CAMERASIMULATOR\_ACTIVE, 121

FG\_CAMERASIMULATOR\_ENABLE, 112

FG\_CAMERASIMULATOR\_FRAMERATE, 119

FG\_CAMERASIMULATOR\_FRAME\_GAP, 115

FG\_CAMERASIMULATOR\_HEIGHT, 114

FG\_CAMERASIMULATOR\_LINERATE, 119

FG\_CAMERASIMULATOR\_LINE\_GAP, 114

FG\_CAMERASIMULATOR\_PASSIVE, 121  
FG\_CAMERASIMULATOR\_PATTERN, 116  
FG\_CAMERASIMULATOR\_PATTERN\_OFFSET, 116  
FG\_CAMERASIMULATOR\_PIXEL\_FREQUENCY, 118  
FG\_CAMERASIMULATOR\_ROLL, 117  
FG\_CAMERASIMULATOR\_SELECT\_MODE, 118  
FG\_CAMERASIMULATOR\_TRIGGER\_MODE, 120  
FG\_CAMERASIMULATOR\_WIDTH, 113  
FG\_CAMERA\_STREAM\_STATUS0, 16  
FG\_CAMSTATUS, 124  
FG\_CAMSTATUS\_EXTENDED, 125  
FG\_CORRECTED\_ERROR\_COUNT, 166  
FG\_CUSTOM\_BIT\_SHIFT\_RIGHT, 110  
FG\_CUSTOM\_SIGNAL\_EVENT\_0, 42  
FG\_CUSTOM\_SIGNAL\_EVENT\_0\_POLARITY, 40  
FG\_CUSTOM\_SIGNAL\_EVENT\_0\_SOURCE, 39  
FG\_CUSTOM\_SIGNAL\_EVENT\_1, 43  
FG\_CUSTOM\_SIGNAL\_EVENT\_1\_POLARITY, 41  
FG\_CUSTOM\_SIGNAL\_EVENT\_1\_SOURCE, 40  
FG\_CXP\_CAMERA\_FRAME\_CORRUPT\_COUNT, 159  
FG\_CXP\_CAMERA\_FRAME\_LOST\_COUNT, 159  
FG\_CXP\_CAMERA\_MARKER\_ERROR\_COUNT, 158  
FG\_CXP\_CAMERA\_UNEXPECTED\_STARTUP\_DATA, 158  
FG\_CXP\_CLEAR\_TEST\_STATISTIC\_PORT, 14  
FG\_CXP\_CONTROL\_ACK\_INCOMPLETE\_COUNT, 154  
FG\_CXP\_CONTROL\_ACK\_LOST\_COUNT, 153  
FG\_CXP\_CONTROL\_ACK\_PACKET\_CRC\_ERROR, 161  
FG\_CXP\_CONTROL\_TAG\_ERROR\_COUNT, 153  
FG\_CXP\_CORRUPTED\_WORD\_COUNT, 13  
FG\_CXP\_ERROR\_CORRECTED, 162  
FG\_CXP\_ERROR\_CORRECTED\_CONTROL\_ACK, 164  
FG\_CXP\_ERROR\_CORRECTED\_HEARTBEAT, 165  
FG\_CXP\_ERROR\_CORRECTED\_LINKTEST, 165  
FG\_CXP\_ERROR\_CORRECTED\_STREAM, 164  
FG\_CXP\_ERROR\_CORRECTED\_TRIGGER, 163  
FG\_CXP\_ERROR\_CORRECTED\_TRIGGER\_ACK, 163  
FG\_CXP\_ERROR\_UNCORRECTED, 166  
FG\_CXP\_ERROR\_UNCORRECTED\_CONTROL\_ACK, 168  
FG\_CXP\_ERROR\_UNCORRECTED\_HEARTBEAT, 169  
FG\_CXP\_ERROR\_UNCORRECTED\_LINKTEST, 169  
FG\_CXP\_ERROR\_UNCORRECTED\_STREAM, 168  
FG\_CXP\_ERROR\_UNCORRECTED\_TRIGGER, 167  
FG\_CXP\_ERROR\_UNCORRECTED\_TRIGGER\_ACK, 167  
FG\_CXP\_HEARTBEAT\_INCOMPLETE\_COUNT, 154  
FG\_CXP\_HEARTBEAT\_MAX\_PERIOD\_VIOLATION\_COUNT, 155  
FG\_CXP\_IMAGETAG\_ERROR\_COUNT, 157  
FG\_CXP\_INPUT\_MAPPED\_FW\_PORT\_PORT, 149  
FG\_CXP\_OVERTRIGGER\_REQUEST\_PULSECOUNT, 152  
FG\_CXP\_PACKET\_LENGTH\_ERROR\_COUNT, 14  
FG\_CXP\_RECEIVED\_PACKET\_COUNT, 13  
FG\_CXP\_STREAMID\_ERROR\_COUNT, 157  
FG\_CXP\_STREAMPACKET\_CRC\_ERROR, 160  
FG\_CXP\_STREAMPACKET\_LENGTH\_ERROR, 162  
FG\_CXP\_STREAM\_PACKET\_COUNT, 10  
FG\_CXP\_TRANSMITTED\_PACKET\_COUNT, 12  
FG\_CXP\_TRIGGER\_ACK\_MISSING\_COUNT, 152  
FG\_CXP\_TRIGGER\_PACKET\_MODE, 128  
FG\_CXP\_UNSUPPORTED\_EVENT\_RECEIVED, 171

FG\_CXP\_UNSUPPORTED\_GPIO\_ACK\_RECEIVED, 172  
FG\_CXP\_UNSUPPORTED\_GPIO\_RECEIVED, 171  
FG\_CXP\_UNSUPPORTED\_GPIO\_REQUEST\_RECEIVED, 172  
FG\_CXP\_UNSUPPORTED\_HEARTBEAT\_RECEIVED, 171  
FG\_DEBUGFILE, 135  
FG\_DEBUGINENABLE, 134  
FG\_DEBUGINSERT, 135  
FG\_DEBUGOUTENABLE, 139  
FG\_DEBUGOUTPIXEL, 140  
FG\_DEBUGOUTXPOS, 140  
FG\_DEBUGOUTYPOS, 140  
FG\_DEBUGREADY, 137  
FG\_DEBUGSAVECONFIG, 131  
FG\_DEBUGSOURCE, 131  
FG\_DEBUGSOURCENAME, 131  
FG\_DEBUGWRITEFLAG, 136  
FG\_DEBUGWRITEPIXEL, 136  
FG\_DEBUG\_ENABLE\_SEQUENCEID, 138  
FG\_DEBUG\_FORCE\_FRAMEID, 137  
FG\_DEBUG\_FRAMEID, 138  
FG\_DEBUG\_FRAMEID\_TO\_FIRSTPIXEL, 134  
FG\_DEBUG\_PWM\_SLOWRATE, 133  
FG\_DEBUG\_SLOWMODE, 132  
FG\_DEBUG\_SOFTWRAE\_SLOWGATE, 132  
FG\_DEBUG\_VERSION, 133  
FG\_DIGIO\_INPUT, 38  
FG\_DMASTATUS, 144  
FG\_END\_OF\_FRAME\_CAM\_PORT\_0, 18  
FG\_END\_OF\_LINE\_CAM\_PORT\_0, 18  
FG\_EXSYNCON, 45  
FG\_EXSYNCPOLARITY, 64  
FG\_FILLLEVEL, 82  
FG\_FLASHON, 68  
FG\_FLASH\_POLARITY, 73  
FG\_FORMAT, 106  
FG\_GENTL\_INFO\_IGNOREFGFORMAT, 141  
FG\_GENTL\_INFO\_OVERFLOWCAPABLE, 142  
FG\_GENTL\_INFO\_VERSION, 141  
FG\_HAP\_FILE, 124  
FG\_HEIGHT, 24  
FG\_IMAGE\_TAG, 5  
FG\_IMGTRIGGERDEBOUNCING, 71  
FG\_IMGTRIGGERGATEDELAY, 71  
FG\_IMGTRIGGERINPOLARITY, 70  
FG\_IMGTRIGGERINSRC, 70  
FG\_IMGTRIGGERMODE, 67  
FG\_IMGTRIGGERON, 67  
FG\_IMGTRIGGER\_ASYNC\_HEIGHT, 68  
FG\_IMGTRIGGER\_IS\_BUSY, 69  
FG\_IMG\_SELECT, 89  
FG\_IMG\_SELECT\_PERIOD, 88  
FG\_LINEEXPOSURE, 63  
FG\_LINEPERIODE, 62  
FG\_LINETRIGGERDEBOUNCING, 49  
FG\_LINETRIGGERDELAY, 65  
FG\_LINETRIGGERINPOLARITY, 48  
FG\_LINETRIGGERINSRC, 47  
FG\_LINETRIGGERMODE, 44

FG\_LINE\_DOWNSCALE, 49  
FG\_LINE\_DOWNSCALEINIT, 50  
FG\_LUT\_CUSTOM\_FILE, 97  
FG\_LUT\_ENABLE, 93  
FG\_LUT\_IMPLEMENTATION\_TYPE, 99  
FG\_LUT\_IN\_BITS, 99  
FG\_LUT\_OUT\_BITS, 100  
FG\_LUT\_SAVE\_FILE, 98  
FG\_LUT\_TYPE, 94  
FG\_LUT\_VALUE, 94  
FG\_LUT\_VALUE\_BLUE, 96  
FG\_LUT\_VALUE\_GREEN, 95  
FG\_LUT\_VALUE\_RED, 95  
FG\_OVERFLOW, 83  
FG\_OVERFLOW\_CAM0, 87  
FG\_OVERFLOW\_EVENT\_SELECT, 85  
FG\_OVERFLOW\_OFF\_THRESHOLD, 83  
FG\_OVERFLOW\_ON\_SYNC\_THRESHOLD, 85  
FG\_OVERFLOW\_ON\_THRESHOLD, 84  
FG\_PACKET\_TAG\_ERROR\_COUNT, 155  
FG\_PIXELDEPTH, 109  
FG\_PIXELFORMAT, 10  
FG\_PROCESSING\_GAIN, 102  
FG\_PROCESSING\_GAMMA, 103  
FG\_PROCESSING\_INVERT, 104  
FG\_PROCESSING\_OFFSET, 102  
FG\_SCALINGFACTOR\_BLUE, 91  
FG\_SCALINGFACTOR\_GREEN, 90  
FG\_SCALINGFACTOR\_RED, 90  
FG\_SENDSOFTWARETRIGGER, 73  
FG\_SENSORHEIGHT, 20  
FG\_SENSORWIDTH, 19  
FG\_SETSOFTWARETRIGGER, 74  
FG\_SHAFTENCODERINSRC, 53  
FG\_SHAFTENCODERLEADING, 54  
FG\_SHAFTENCODERMODE, 52  
FG\_SHAFTENCODERON, 51  
FG\_SHAFTENCODER\_COMPENSATION\_COUNT, 56  
FG\_SHAFTENCODER\_COMPENSATION\_ENABLE, 55  
FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_CURRENT, 77  
FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_MAX, 78  
FG\_SIGNAL\_ANALYZER\_0\_PERIOD\_MIN, 78  
FG\_SIGNAL\_ANALYZER\_0\_POLARITY, 76  
FG\_SIGNAL\_ANALYZER\_0\_PULSE\_COUNT, 79  
FG\_SIGNAL\_ANALYZER\_0\_SOURCE, 75  
FG\_SIGNAL\_ANALYZER\_1\_PERIOD\_CURRENT, 77  
FG\_SIGNAL\_ANALYZER\_1\_PERIOD\_MAX, 78  
FG\_SIGNAL\_ANALYZER\_1\_PERIOD\_MIN, 78  
FG\_SIGNAL\_ANALYZER\_1\_POLARITY, 76  
FG\_SIGNAL\_ANALYZER\_1\_PULSE\_COUNT, 79  
FG\_SIGNAL\_ANALYZER\_1\_SOURCE, 75  
FG\_SIGNAL\_ANALYZER\_CLEAR, 80  
FG\_SIGNAL\_ANALYZER\_PULSE\_COUNT\_DIFFERENCE, 80  
FG\_START\_OF\_FRAME\_CAM\_PORT\_0, 18  
FG\_START\_OF\_LINE\_CAM\_PORT\_0, 18  
FG\_STROBEPULSEDELAY, 72  
FG\_SYSTEMMONITOR\_BYTE\_ALIGNMENT\_8B\_10B\_LOCKED, 150  
FG\_SYSTEMMONITOR\_CHANNEL\_CURRENT, 143

FG\_SYSTEMMONITOR\_CHANNEL\_VOLTAGE, 143  
FG\_SYSTEMMONITOR\_CURRENT\_LINK\_SPEED, 147  
FG\_SYSTEMMONITOR\_CURRENT\_LINK\_WIDTH, 147  
FG\_SYSTEMMONITOR\_CXP\_IMAGE\_LINE\_MODE, 12  
FG\_SYSTEMMONITOR\_CXP\_STANDARD, 9  
FG\_SYSTEMMONITOR\_DECODER\_8B\_10B\_ERROR, 150  
FG\_SYSTEMMONITOR\_EXTERNAL\_POWER, 149  
FG\_SYSTEMMONITOR\_FPGA\_DNA\_HIGH, 126  
FG\_SYSTEMMONITOR\_FPGA\_DNA\_LOW, 126  
FG\_SYSTEMMONITOR\_FPGA\_TEMPERATURE, 145  
FG\_SYSTEMMONITOR\_FPGA\_VCC\_AUX, 146  
FG\_SYSTEMMONITOR\_FPGA\_VCC\_BRAM, 146  
FG\_SYSTEMMONITOR\_FPGA\_VCC\_INT, 145  
FG\_SYSTEMMONITOR\_MAPPED\_TO\_FG\_PORT, 144  
FG\_SYSTEMMONITOR\_PACKETBUFFER\_OVERFLOW\_COUNT, 156  
FG\_SYSTEMMONITOR\_PACKETBUFFER\_OVERFLOW\_SOURCE, 156  
FG\_SYSTEMMONITOR\_PCIE\_TRAINED\_PAYLOAD\_SIZE, 148  
FG\_SYSTEMMONITOR\_PCIE\_TRAINED\_REQUEST\_SIZE, 148  
FG\_SYSTEMMONITOR\_PORT\_BIT\_RATE, 8  
FG\_SYSTEMMONITOR\_POWER\_OVER\_CXP\_CONTROLLER\_ENABLED, 7  
FG\_SYSTEMMONITOR\_POWER\_OVER\_CXP\_STATE, 7  
FG\_SYSTEMMONITOR\_RX\_LENGTH\_ERROR\_COUNT, 161  
FG\_SYSTEMMONITOR\_RX\_PACKET\_CRC\_ERROR\_COUNT, 160  
FG\_SYSTEMMONITOR\_RX\_STREAM\_INCOMPLETE\_COUNT, 151  
FG\_SYSTEMMONITOR\_RX\_UNKNOWN\_DATA\_RECEIVED\_COUNT, 151  
FG\_SYSTEMMONITOR\_RX\_UNSUPPORTED\_PACKET\_COUNT, 170  
FG\_SYSTEMMONITOR\_STREAM\_PACKET\_SIZE, 8  
FG\_SYSTEMMONITOR\_USED\_CXP\_CONNECTIONS, 11  
FG\_TIMEOUT, 123  
FG\_TRIGGERCAMERA\_POLARITY, 130  
FG\_TRIGGERCAMERA\_SOURCE, 129  
FG\_TRIGGERCAMERA\_SOURCE\_CXP0, 27  
FG\_TRIGGERCAMERA\_SOURCE\_CXP1, 28  
FG\_TRIGGERCAMERA\_SOURCE\_CXP2, 30  
FG\_TRIGGERCAMERA\_SOURCE\_CXP3, 32  
FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP0, 27  
FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP1, 29  
FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP2, 31  
FG\_TRIGGERCAMERA\_SOURCE\_EDGE\_CXP3, 33  
FG\_TRIGGEROUT\_FRONT\_GPO\_0\_POLARITY, 37  
FG\_TRIGGEROUT\_FRONT\_GPO\_0\_SOURCE, 36  
FG\_TRIGGEROUT\_FRONT\_GPO\_1\_POLARITY, 37  
FG\_TRIGGEROUT\_FRONT\_GPO\_1\_SOURCE, 36  
FG\_TRIGGEROUT\_GPO\_0\_POLARITY, 35  
FG\_TRIGGEROUT\_GPO\_0\_SOURCE, 34  
FG\_TRIGGEROUT\_GPO\_1\_POLARITY, 35  
FG\_TRIGGEROUT\_GPO\_1\_SOURCE, 34  
FG\_TRIGGEROUT\_GPO\_2\_POLARITY, 35  
FG\_TRIGGEROUT\_GPO\_2\_SOURCE, 34  
FG\_TRIGGEROUT\_GPO\_3\_POLARITY, 35  
FG\_TRIGGEROUT\_GPO\_3\_SOURCE, 34  
FG\_TRIGGEROUT\_GPO\_4\_POLARITY, 35  
FG\_TRIGGEROUT\_GPO\_4\_SOURCE, 34  
FG\_TRIGGEROUT\_GPO\_5\_POLARITY, 35  
FG\_TRIGGEROUT\_GPO\_5\_SOURCE, 34  
FG\_TRIGGEROUT\_GPO\_6\_POLARITY, 35  
FG\_TRIGGEROUT\_GPO\_6\_SOURCE, 34  
FG\_TRIGGEROUT\_GPO\_7\_POLARITY, 35

FG\_TRIGGEROUT\_GPO\_7\_SOURCE, 34  
FG\_TRIGGER\_INPUT0\_FALLING, 42  
FG\_TRIGGER\_INPUT0\_RISING, 42  
FG\_UNCORRECTED\_ERROR\_COUNT, 170  
FG\_VANTAGEPOINT, 19  
FG\_VISUALAPPLETS\_BUILD\_VERSION, 127  
FG\_WIDTH, 23  
FG\_XOFFSET, 24  
FG\_YOFFSET, 25  
Format, 106  
Frame ID, 4

## G

Generator, 112

## I

Image Select, 88  
Image Selector, 88  
Image Tag, 5  
Image Transfer, 5  
Image Trigger / Flash, 66  
Image Trigger / Flash::Image Trigger Input, 69  
Image Trigger / Flash::Image Trigger Input::Flash, 73  
Image Trigger / Flash::Image Trigger Input::Software Trigger, 73

## L

Line Trigger / ExSync, 44  
Line Trigger / ExSync::ExSync Output, 62  
Line Trigger / ExSync::Line Trigger Input, 46  
Line Trigger / ExSync::Line Trigger Input::Downscale, 49  
Line Trigger / ExSync::Shaft Encoder A/B Filter, 51  
Lookup Table, 93, 93  
Lookup Table::Applet Properties, 99

## M

Miscellaneous, 123  
Miscellaneous::Debug, 130  
Miscellaneous::Debug::Input, 134  
Miscellaneous::Debug::Output, 139  
Miscellaneous::GenTL, 141  
Miscellaneous::Legacy, 128  
Miscellaneous::Version, 126

## O

Output Format, 106  
Overflow, 82, 82  
    Events, 87  
Overflow::Events, 86

## P

PC Interface, 5  
Pixel Format, 7  
Processing, 101  
Processor, 101

## R

Region of Interest, 22



ROI, 22

## **S**

Sensor Geometry, 19, 19

Signal Analyzer, 75, 75

Software Interface, 6

Specifications, 1

## **T**

Trigger

    Digital Input, 38

    Events, 42

    Input, 38

## **W**

White Balance, 90, 90