FlyCapture 2.5
API Programming Reference

Revised April 13, 2013

Point Grey Research Inc.
12051 Riverside Way • Richmond, BC • Canada • V6W 1K7 • **T** (604) 242-9937 •
www.ptgrey.com

Copyright © 2013 Point Grey Research Inc. All Rights Reserved.

Software Warranty

Point Grey Research warrants to the Original Purchaser, for a period of one (1) year from date of purchase that:

1. The diskette on which the Software is furnished and the accompanying documentation are not defective;
2. The Software is properly recorded upon the diskettes enclosed;
3. The documentation is substantially complete and contains all the information Point Grey Research deems necessary to use the Software;
4. The Software functions substantially as described in the documentation.

Point Grey Research, Inc.'s entire liability and the Original Purchaser's exclusive remedy shall be the replacement of any diskette or documentation not meeting these warranties. On such an occasion, a copy of the paid receipt accompanied with the faulty diskette or documentation must be returned to Point Grey Research, Inc. or an authorized dealer.

Point Grey Research, Inc. expressly disclaims and excludes all other warranties, express, implied and statutory, including, but without limitation, warranty of merchantability and fitness for a particular application or purpose. In no event shall Point Grey Research, Inc. be liable to the Original Purchaser or any third party for direct, indirect, incidental, consequential, special or accidental damages, including without limitation damages for business interruption, loss of profits, revenue, data or bodily injury or death.

Software License Agreement

READ CAREFULLY: This is a legal agreement between you (an individual or a single entity) ("you") and Point Grey Research, Inc. ("PGR"). Before installing and using the FlyCapture® Software Development Kit and any updates to it that we may at our discretion provide to you (collectively, the "SDK"), you should read this agreement. If you do not agree with all of the terms of this agreement, do not install or use the SDK. PGR may change this agreement at any time and it is your responsibility to review the most updated version of it on PGR's website at http://www.ptgrey.com/support/kb/data/PGR-FlyCap-SDK-LA.pdf**. By continuing to use the SDK following such changes, you agree to be bound by them.**

1. **Grant of License:** Subject to the terms of this agreement, you are hereby granted a limited, terminable, non-transferable, non-exclusive license and right to use the SDK only in conjunction with: (a) those PGR cameras listed at http://www.ptgrey.com/products/index.asp (as such list may be amended by PGR at any time and from time to time) and owned by you; and (b) the images derived from such cameras.

2. **Free and Open Source Components:** Notwithstanding anything to the contrary herein, use, copying and distribution of components of the SDK licensed under free and open source license agreements are governed solely by the terms of those license agreements (which are contained in the electronic documentation for the SDK) and not this agreement.

3. **Restrictions:** Except as (and only if) explicitly permitted by Section 4 below, you will not, and will not permit any third parties to: (a) copy the SDK, other than a reasonable number of backup copies for your own use only, and such backup copies together with the original will be kept in your possession and control; (b) provide or disclose the SDK to any third party; (c) alter, modify, reverse engineer, decompile or disassemble the SDK, or attempt to do any of the foregoing; (d) grant sublicenses, leases, or any other rights in the SDK to any third party; or (e) remove, alter or obscure any proprietary rights notices (including any copyright and trademark notices) on and in the SDK.

4. **Additional OEM Rights:** If you are an original equipment manufacturer, then in addition to the rights set out in Section 1 above you are hereby granted a limited, terminable, non-transferable, non-exclusive license and right to use the SDK for the sole additional purpose of incorporating the libraries found in the SDK (collectively, the "**Libraries**") into new products developed by you, in whole or in part, using the SDK (collectively, the "**Derivative Products**") provided that you: (a) ensure that the components of any Derivative Product that derive functionality from any of the Libraries may only be used with PGR products, including the SDK, and images derived from such products; (b) may only redistribute drivers (.inf and .sys), dynamically linked libraries (.dlls and .so), executables (.exe) and documentation (.doc, .txt, .pdf and .chm) and only to the extent necessary to support your Derivative Products. For clarity, headers (.h), source (.c, .cpp,.cs and .vb) and statically linked libraries (.lib and .a) cannot be redistributed; (c) will prohibit any, and ensure that there is no, redistribution of any of the Libraries by any third party, including any end user customers; and (d) will include any PGR and third party proprietary rights legends or notices (including copyright and trademark notices), unaltered and unobscured, on all Derivative Products.

5. **Ownership:** PGR and third parties are the owners of and retain title to all proprietary and intellectual property rights (including all patent, copyright, trade secret and trademark rights) in and to the SDK. You have no right, title or interest in the SDK, except as specifically set forth herein, and no rights in any trade-marks of PGR. All rights not explicitly granted herein are hereby reserved.

6. **Indemnification:** You assume the entire risk relating to, and will indemnify, hold harmless and defend PGR from and against any claims, actions, lawsuits, or proceedings, and any losses, liabilities, damages and expenses (including attorney's fees and expenses) that arise or result from your activities under this agreement, including the distribution or use of the SDK (including the Libraries) and/or the development, distribution or use of any Derivative Product (including any intellectual property infringement claims relating thereto).

7. **No Warranties:** Your use of the SDK is solely at your own risk. The SDK is provided "as is" and "as available" without warranty or condition of any kind, either express, implied or statutory, including implied warranties of merchantability, fitness for a particular application or purpose, title and non-infringement, and PGR hereby expressly disclaims all such warranties and conditions. Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

8. **Limitation of Liability:** Notwithstanding any other provision herein, to the maximum extent permitted by applicable law, PGR will not under any circumstances be liable for any direct, indirect, incidental, consequential, punitive or special losses or damages (including damages for bodily injury or death, business interruption, loss or corruption of business information or data, loss of opportunity, loss of privacy, cost of recovery, financial loss, loss of revenue or loss of profits for any reason whatsoever) whether or not PGR has been advised or should have been aware of the possibility of any such losses or damage arising, and in no event will the aggregate and cumulative liability of PGR for any matters arising under this agreement or otherwise exceed $25.

9. **Changes and Support:** PGR may at its sole discretion elect to provide you with limited support services related to the SDK for such period of time as PGR at its sole discretion elects. PGR may change or cease to provide the SDK and the limited Support at any time and from time to time without notice to you and that PGR is not obligated to provide the SDK or any support. PGR may use any technical information, feedback or ideas you provide to PGR for PGR's business purposes, including product support and development.

10. **Termination:** PGR may terminate this agreement immediately on notice to you if you violate any of the terms of this agreement. PGR may also terminate this agreement for convenience provided that 30 days notice is delivered to you. Any notice given by PGR hereunder will be effective when sent by PGR to the email address you provided to PGR when you registered for a download account. On termination, for any reason, your rights hereunder will cease and you must immediately return all copies of the SDK to PGR and on termination for violation or default, you will be liable to PGR for all damages suffered as a result of the violation or default. Sections 6-8, 10 and 12 will survive any termination hereof.

11. **Export Laws:** This agreement involves products and/or technical data that may be controlled under laws and regulations of the United States and other countries, including the United States Export Administration Regulations, or any other applicable law, regulation, rule,

guideline or order (collectively, “**Export Laws**”). You will comply with all Export Laws to ensure that the SDK is not exported, directly or indirectly, in contravention of the Export Laws. You represent and warrant to PGR that you are not a person barred from receiving the SDK under any Export Laws.

12. **General:** This agreement is the entire agreement between you and PGR with respect to the subject matter of this agreement. If you are signing on behalf of an entity such as a corporation, you represent and warrant that you have the authority to bind such entity. This agreement and the rights granted hereunder are personal to you and you may not assign this agreement to a third party without the prior written consent of PGR. This agreement is governed exclusively by and will be enforced, construed, and interpreted exclusively in accordance with the laws of British Columbia (“BC”) and the laws of Canada applicable in BC. The courts of the Province of BC will have exclusive jurisdiction over any dispute arising under this agreement. You agree that termination and/or monetary damages may not be a sufficient remedy if you breach this agreement and that PGR will be entitled, without waiving any other rights or remedies, to injunctive or equitable relief as may be deemed proper by a court of competent jurisdiction in the event of a breach. If PGR does not exercise any legal right or remedy in this agreement or otherwise, this will not be taken to be a formal waiver by PGR of its rights, which rights will remain available to PGR. If any provision of this agreement is construed to be illegal or invalid, the illegal or invalid provisions will be deemed stricken and deleted herefrom to the same extent and effect as if never incorporated herein, but all other provisions hereof will continue in full force and effect.

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Module Documentation

6.1 Global constants

Variables

- static const unsigned int sk_maxStringLength = 512

  The maximum length that is allocated for a string.
- static const unsigned int sk_maxNumPorts = 32

  The maximum number of ports one device can have.

6.1.1 Variable Documentation

6.1.1.1 const unsigned int sk_maxNumPorts = 32 `[static]`

The maximum number of ports one device can have.

6.1.1.2 const unsigned int sk_maxStringLength = 512 `[static]`

The maximum length that is allocated for a string.

6.2 Enumerations

Enumerations

- enum ErrorType { PGRERROR_UNDEFINED = -1, PGRERROR_OK, PGRERROR_FAILED, PGRERROR_NOT_IMPLEMENTED, PGRERROR_FAILED_BUS_MASTER_CONNECTION, PGRERROR_NOT_CONNECTED, PGRERROR_INIT_FAILED, PGRERROR_NOT_INTITIALIZED, PGRERROR_INVALID_PARAMETER, PGRERROR_INVALID_SETTINGS, PGRERROR_INVALID_BUS_MANAGER, PGRERROR_MEMORY_ALLOCATION_FAILED, PGRERROR_LOW_LEVEL_FAILURE, PGRERROR_NOT_FOUND, PGRERROR_FAILED_GUID, PGRERROR_INVALID_PACKET_SIZE, PGRERROR_INVALID_MODE, PGRERROR_NOT_IN_FORMAT7, PGRERROR_NOT_SUPPORTED, PGRERROR_TIMEOUT, PGRERROR_BUS_MASTER_FAILED, PGRERROR_INVALID_GENERATION, PGRERROR_LUT_FAILED, PGRERROR_IIDC_FAILED, PGRERROR_STROBE_FAILED, PGRERROR_TRIGGER_FAILED, PGRERROR_PROPERTY_FAILED, PGRERROR_PROPERTY_NOT_PRESENT, PGRERROR_REGISTER_FAILED, PGRERROR_READ_REGISTER_FAILED, PGRERROR_WRITE_REGISTER_FAILED, PGRERROR_ISOCH_FAILED, PGRERROR_ISOCH_ALREADY_STARTED, PGRERROR_ISOCH_NOT_STARTED, PGRERROR_ISOCH_START_FAILED, PGRERROR_ISOCH_RETRIEVE_BUFFER_FAILED, PGRERROR_ISOCH_STOP_FAILED, PGRERROR_ISOCH_SYNC_FAILED, PGRERROR_ISOCH_BANDWIDTH_EXCEEDED, PGRERROR_IMAGE_CONVERSION_FAILED, PGRERROR_IMAGE_LIBRARY_FAILURE, PGRERROR_BUFFER_TOO_SMALL, PGRERROR_IMAGE_CONSISTENCY_ERROR, PGRERROR_FORCE_32BITS = FULL_32BIT_VALUE }

  The error types returned by functions.
- enum BusCallbackType { BUS_RESET, ARRIVAL, REMOVAL, CALLBACK_TYPE_FORCE_32BITS = FULL_32BIT_VALUE }

  The type of bus callback to register a callback function for.
- enum GrabMode { DROP_FRAMES, BUFFER_FRAMES, UNSPECIFIED_GRAB_MODE, GRAB_MODE_FORCE_32BITS = FULL_32BIT_VALUE }

  The grab strategy employed during image transfer.
- enum GrabTimeout { TIMEOUT_NONE = 0, TIMEOUT_INFINITE = -1, TIMEOUT_UNSPECIFIED = -2, GRAB_TIMEOUT_FORCE_32BITS = FULL_32BIT_VALUE }

  Timeout options for grabbing images.
- enum BandwidthAllocation { BANDWIDTH_ALLOCATION_OFF = 0, BANDWIDTH_ALLOCATION_ON = 1, BANDWIDTH_ALLOCATION_UNSUPPORTED = 2, BANDWIDTH_ALLOCATION_UNSPECIFIED = 3, BANDWIDTH_ALLOCATION_FORCE_32BITS = FULL_32BIT_VALUE }

  Bandwidth allocation options for 1394 devices.
- enum InterfaceType { INTERFACE_IEEE1394, INTERFACE_USB2, INTERFACE_USB3, INTERFACE_GIGE, INTERFACE_UNKNOWN, INTERFACE_TYPE_FORCE_32BITS = FULL_32BIT_VALUE }

  Interfaces that a camera may use to communicate with a host.

- enum PropertyType { BRIGHTNESS, AUTO_EXPOSURE, SHARPNESS, WH-ITE_BALANCE, HUE, SATURATION, GAMMA, IRIS, FOCUS, ZOOM, PAN, TILT, SHUTTER, GAIN, TRIGGER_MODE, TRIGGER_DELAY, FRAME_R-ATE, TEMPERATURE, UNSPECIFIED_PROPERTY_TYPE, PROPERTY_TY-PE_FORCE_32BITS = FULL_32BIT_VALUE }

  Camera properties.
- enum FrameRate { FRAMERATE_1_875, FRAMERATE_3_75, FRAMERATE-_7_5, FRAMERATE_15, FRAMERATE_30, FRAMERATE_60, FRAMERAT-E_120, FRAMERATE_240, FRAMERATE_FORMAT7, NUM_FRAMERATES, FRAMERATE_FORCE_32BITS = FULL_32BIT_VALUE }

  Frame rates in frames per second.
- enum VideoMode { VIDEOMODE_160x120YUV444, VIDEOMODE_320x240-YUV422, VIDEOMODE_640x480YUV411, VIDEOMODE_640x480YUV422, VIDEOMODE_640x480RGB, VIDEOMODE_640x480Y8, VIDEOMODE_-640x480Y16, VIDEOMODE_800x600YUV422, VIDEOMODE_800x600RGB, VIDEOMODE_800x600Y8, VIDEOMODE_800x600Y16, VIDEOMODE_-1024x768YUV422, VIDEOMODE_1024x768RGB, VIDEOMODE_1024x768Y8, VIDEOMODE_1024x768Y16, VIDEOMODE_1280x960YUV422, VIDEOMOD-E_1280x960RGB, VIDEOMODE_1280x960Y8, VIDEOMODE_1280x960Y16, VIDEOMODE_1600x1200YUV422, VIDEOMODE_1600x1200RGB, VIDEOM-ODE_1600x1200Y8, VIDEOMODE_1600x1200Y16, VIDEOMODE_FORMAT7, NUM_VIDEOMODES, VIDEOMODE_FORCE_32BITS = FULL_32BIT_VALUE }

  DCAM video modes.
- enum Mode { MODE_0 = 0, MODE_1, MODE_2, MODE_3, MODE_4, M-ODE_5, MODE_6, MODE_7, MODE_8, MODE_9, MODE_10, MODE_11, MODE_12, MODE_13, MODE_14, MODE_15, MODE_16, MODE_17, MOD-E_18, MODE_19, MODE_20, MODE_21, MODE_22, MODE_23, MODE_24, MODE_25, MODE_26, MODE_27, MODE_28, MODE_29, MODE_30, MO-DE_31, NUM_MODES, MODE_FORCE_32BITS = FULL_32BIT_VALUE }

  Camera modes for DCAM formats as well as Format7.
- enum PixelFormat { PIXEL_FORMAT_MONO8 = 0x80000000, PIXEL_FORMA-T_411YUV8 = 0x40000000, PIXEL_FORMAT_422YUV8 = 0x20000000, PIXE-L_FORMAT_444YUV8 = 0x10000000, PIXEL_FORMAT_RGB8 = 0x08000000, PIXEL_FORMAT_MONO16 = 0x04000000, PIXEL_FORMAT_RGB16 = 0x02000000, PIXEL_FORMAT_S_MONO16 = 0x01000000, PIXEL_FO-RMAT_S_RGB16 = 0x00800000, PIXEL_FORMAT_RAW8 = 0x00400000, PIXEL_FORMAT_RAW16 = 0x00200000, PIXEL_FORMAT_MONO12 = 0x00100000, PIXEL_FORMAT_RAW12 = 0x00080000, PIXEL_FORMAT_BGR = 0x80000008, PIXEL_FORMAT_BGRU = 0x40000008, PIXEL_FORMAT_RG-B = PIXEL_FORMAT_RGB8, PIXEL_FORMAT_RGBU = 0x40000002, PIXEL-_FORMAT_BGR16 = 0x02000001, PIXEL_FORMAT_BGRU16 = 0x02000002, PIXEL_FORMAT_422YUV8_JPEG = 0x40000001, NUM_PIXEL_FORMATS = 20, UNSPECIFIED_PIXEL_FORMAT = 0 }

  Pixel formats available for Format7 modes.
- enum BusSpeed { BUSSPEED_S100, BUSSPEED_S200, BUSSPEED_S400, BUSSPEED_S480, BUSSPEED_S800, BUSSPEED_S1600, BUSSPEED_-S3200, BUSSPEED_S5000, BUSSPEED_10BASE_T, BUSSPEED_100BA-

SE_T, BUSSPEED_1000BASE_T, BUSSPEED_10000BASE_T, BUSSPEED_S_FASTEST, BUSSPEED_ANY, BUSSPEED_SPEED_UNKNOWN = -1, BUSSPEED_FORCE_32BITS = FULL_32BIT_VALUE }

Bus speeds.

- enum PCIeBusSpeed { PCIE_BUSSPEED_2_5, PCIE_BUSSPEED_5_0, PCIE_BUSSPEED_UNKNOWN = -1, PCIE_BUSSPEED_FORCE_32BITS = FULL_32BIT_VALUE }
- enum DriverType { DRIVER_1394_CAM, DRIVER_1394_PRO, DRIVER_1394_JUJU, DRIVER_1394_VIDEO1394, DRIVER_1394_RAW1394, DRIVER_USB_NONE, DRIVER_USB_CAM, DRIVER_USB3_PRO, DRIVER_GIGE_NONE, DRIVER_GIGE_FILTER, DRIVER_GIGE_PRO, DRIVER_UNKNOWN = -1, DRIVER_FORCE_32BITS = FULL_32BIT_VALUE }

  Types of low level drivers that flycapture uses.
- enum ColorProcessingAlgorithm { DEFAULT, NO_COLOR_PROCESSING, NEAREST_NEIGHBOR, EDGE_SENSING, HQ_LINEAR, RIGOROUS, IPP, DIRECTIONAL_FILTER, COLOR_PROCESSING_ALGORITHM_FORCE_32BITS = FULL_32BIT_VALUE }

  Color processing algorithms.
- enum BayerTileFormat { NONE, RGGB, GRBG, GBRG, BGGR, BT_FORCE_32BITS = FULL_32BIT_VALUE }

  Bayer tile formats.
- enum ImageFileFormat { FROM_FILE_EXT = -1, PGM, PPM, BMP, JPEG, JPEG2000, TIFF, PNG, RAW, IMAGE_FILE_FORMAT_FORCE_32BITS = FULL_32BIT_VALUE }

  File formats to be used for saving images to disk.

6.2.1 Enumeration Type Documentation

6.2.1.1 enum BandwidthAllocation

Bandwidth allocation options for 1394 devices.

Enumerator:

BANDWIDTH_ALLOCATION_OFF Do not allocate bandwidth.

BANDWIDTH_ALLOCATION_ON Allocate bandwidth. This is the default setting.

BANDWIDTH_ALLOCATION_UNSUPPORTED Bandwidth allocation is not supported by either the camera or operating system.

BANDWIDTH_ALLOCATION_UNSPECIFIED Not specified. This leaves the current setting unchanged.

BANDWIDTH_ALLOCATION_FORCE_32BITS

6.2.1.2 enum BayerTileFormat

Bayer tile formats.

Enumerator:

NONE	No bayer tile format.
RGGB	Red-Green-Green-Blue.
GRBG	Green-Red-Blue-Green.
GBRG	Green-Blue-Red-Green.
BGGR	Blue-Green-Green-Red.
BT_FORCE_32BITS	

6.2.1.3 enum BusCallbackType

The type of bus callback to register a callback function for.

Enumerator:

BUS_RESET	Register for all bus events.
ARRIVAL	Register for arrivals only.
REMOVAL	Register for removals only.
CALLBACK_TYPE_FORCE_32BITS	

6.2.1.4 enum BusSpeed

Bus speeds.

Enumerator:

BUSSPEED_S100	100Mbits/sec.
BUSSPEED_S200	200Mbits/sec.
BUSSPEED_S400	400Mbits/sec.
BUSSPEED_S480	480Mbits/sec. Only for USB2 cameras.
BUSSPEED_S800	800Mbits/sec.
BUSSPEED_S1600	1600Mbits/sec.
BUSSPEED_S3200	3200Mbits/sec.
BUSSPEED_S5000	5000Mbits/sec. Only for USB3 cameras.
BUSSPEED_10BASE_T	10Base-T. Only for GigE Vision cameras.
BUSSPEED_100BASE_T	100Base-T. Only for GigE Vision cameras.
BUSSPEED_1000BASE_T	1000Base-T (Gigabit Ethernet). Only for GigE Vision cameras.

BUSSPEED_10000BASE_T 10000Base-T. Only for GigE Vision cameras.

BUSSPEED_S_FASTEST The fastest speed available.

BUSSPEED_ANY Any speed that is available.

BUSSPEED_SPEED_UNKNOWN Unknown bus speed.

BUSSPEED_FORCE_32BITS

6.2.1.5 enum ColorProcessingAlgorithm

Color processing algorithms.

Please refer to our knowledge base at article at http://www.ptgrey.-com/support/kb/index.asp?a=4&q=33 for complete details for each algorithm.

Enumerator:

DEFAULT Default method.

NO_COLOR_PROCESSING No color processing.

NEAREST_NEIGHBOR Fastest but lowest quality. Equivalent to FLYCAPTURE-_NEAREST_NEIGHBOR_FAST in FlyCapture.

EDGE_SENSING Weights surrounding pixels based on localized edge orientation.

HQ_LINEAR Well-balanced speed and quality.

RIGOROUS Slowest but produces good results.

IPP Multithreaded with similar results to edge sensing.

DIRECTIONAL_FILTER Best quality but much faster than rigorous.

COLOR_PROCESSING_ALGORITHM_FORCE_32BITS

6.2.1.6 enum DriverType

Types of low level drivers that flycapture uses.

Enumerator:

DRIVER_1394_CAM PGRCam.sys.

DRIVER_1394_PRO PGR1394.sys.

DRIVER_1394_JUJU firewire_core.

DRIVER_1394_VIDEO1394 video1394.

DRIVER_1394_RAW1394 raw1394.

DRIVER_USB_NONE No usb driver used just BSD stack. (Linux only)

DRIVER_USB_CAM PGRUsbCam.sys.

DRIVER_USB3_PRO PGRXHCI.sys.

DRIVER_GIGE_NONE no gige drivers used,MS/BSD stack.

DRIVER_GIGE_FILTER PGRGigE.sys.

DRIVER_GIGE_PRO PGRGigEPro.sys.

DRIVER_UNKNOWN Unknown driver type.

DRIVER_FORCE_32BITS

6.2.1.7 enum ErrorType

The error types returned by functions.

Enumerator:

PGRERROR_UNDEFINED Undefined.

PGRERROR_OK Function returned with no errors.

PGRERROR_FAILED General failure.

PGRERROR_NOT_IMPLEMENTED Function has not been implemented.

PGRERROR_FAILED_BUS_MASTER_CONNECTION Could not connect to - Bus Master.

PGRERROR_NOT_CONNECTED Camera has not been connected.

PGRERROR_INIT_FAILED Initialization failed.

PGRERROR_NOT_INTITIALIZED Camera has not been initialized.

PGRERROR_INVALID_PARAMETER Invalid parameter passed to function.

PGRERROR_INVALID_SETTINGS Setting set to camera is invalid.

PGRERROR_INVALID_BUS_MANAGER Invalid Bus Manager object.

PGRERROR_MEMORY_ALLOCATION_FAILED Could not allocate memory.

PGRERROR_LOW_LEVEL_FAILURE Low level error.

PGRERROR_NOT_FOUND Device not found.

PGRERROR_FAILED_GUID GUID failure.

PGRERROR_INVALID_PACKET_SIZE Packet size set to camera is invalid.

PGRERROR_INVALID_MODE Invalid mode has been passed to function.

PGRERROR_NOT_IN_FORMAT7 Error due to not being in Format7.

PGRERROR_NOT_SUPPORTED This feature is unsupported.

PGRERROR_TIMEOUT Timeout error.

PGRERROR_BUS_MASTER_FAILED Bus Master Failure.

PGRERROR_INVALID_GENERATION Generation Count Mismatch.

PGRERROR_LUT_FAILED Look Up Table failure.

PGRERROR_IIDC_FAILED IIDC failure.

PGRERROR_STROBE_FAILED Strobe failure.

PGRERROR_TRIGGER_FAILED Trigger failure.

PGRERROR_PROPERTY_FAILED Property failure.

PGRERROR_PROPERTY_NOT_PRESENT Property is not present.

PGRERROR_REGISTER_FAILED Register access failed.

PGRERROR_READ_REGISTER_FAILED Register read failed.

PGRERROR_WRITE_REGISTER_FAILED Register write failed.

PGRERROR_ISOCH_FAILED Isochronous failure.

PGRERROR_ISOCH_ALREADY_STARTED Isochronous transfer has already been started.

PGRERROR_ISOCH_NOT_STARTED Isochronous transfer has not been started.

PGRERROR_ISOCH_START_FAILED Isochronous start failed.

PGRERROR_ISOCH_RETRIEVE_BUFFER_FAILED Isochronous retrieve buffer failed.

PGRERROR_ISOCH_STOP_FAILED Isochronous stop failed.

PGRERROR_ISOCH_SYNC_FAILED Isochronous image synchronization failed.

PGRERROR_ISOCH_BANDWIDTH_EXCEEDED Isochronous bandwidth exceeded.

PGRERROR_IMAGE_CONVERSION_FAILED Image conversion failed.

PGRERROR_IMAGE_LIBRARY_FAILURE Image library failure.

PGRERROR_BUFFER_TOO_SMALL Buffer is too small.

PGRERROR_IMAGE_CONSISTENCY_ERROR There is an image consistency error.

PGRERROR_FORCE_32BITS

6.2.1.8 enum FrameRate

Frame rates in frames per second.

Enumerator:

FRAMERATE_1_875 1.875 fps.

FRAMERATE_3_75 3.75 fps.

FRAMERATE_7_5 7.5 fps.

FRAMERATE_15 15 fps.

FRAMERATE_30 30 fps.

FRAMERATE_60 60 fps.

FRAMERATE_120 120 fps.

FRAMERATE_240 240 fps.

FRAMERATE_FORMAT7 Custom frame rate for Format7 functionality.

NUM_FRAMERATES Number of possible camera frame rates.

FRAMERATE_FORCE_32BITS

6.2.1.9 enum GrabMode

The grab strategy employed during image transfer.

This type controls how images that stream off the camera accumulate in a user buffer for handling.

Enumerator:

DROP_FRAMES Grabs the newest image in the user buffer each time the -RetrieveBuffer() function is called. Older images are dropped instead of accumulating in the user buffer. Grabbing blocks if the camera has not finished transmitting the next available image. If the camera is transmitting images faster than the application can grab them, images may be dropped and only the most recent image is stored for grabbing. Note that this mode is the equivalent of flycaptureLockLatest in earlier versions of the FlyCapture SDK.

BUFFER_FRAMES Images accumulate in the user buffer, and the oldest image is grabbed for handling before being discarded. This member can be used to guarantee that each image is seen. However, image processing time must not exceed transmission time from the camera to the buffer. Grabbing blocks if the camera has not finished transmitting the next available image. The buffer size is controlled by the numBuffers parameter in the FC2Config struct. Note that this mode is the equivalent of flycaptureLockNext in earlier versions of the FlyCapture SDK.

UNSPECIFIED_GRAB_MODE Unspecified grab mode.

GRAB_MODE_FORCE_32BITS

6.2.1.10 enum GrabTimeout

Timeout options for grabbing images.

Enumerator:

TIMEOUT_NONE Non-blocking wait.

TIMEOUT_INFINITE Wait indefinitely.

TIMEOUT_UNSPECIFIED Unspecified timeout setting.

GRAB_TIMEOUT_FORCE_32BITS

6.2.1.11 enum ImageFileFormat

File formats to be used for saving images to disk.

Enumerator:

FROM_FILE_EXT Determine file format from file extension.

PGM Portable gray map.

PPM Portable pixmap.

BMP Bitmap.

JPEG JPEG.

JPEG2000 JPEG 2000.

TIFF Tagged image file format.

PNG Portable network graphics.

RAW Raw data.

IMAGE_FILE_FORMAT_FORCE_32BITS

6.2.1.12 enum InterfaceType

Interfaces that a camera may use to communicate with a host.

Enumerator:

INTERFACE_IEEE1394 IEEE-1394 (Includes 1394a and 1394b).

INTERFACE_USB2 USB 2.0.

INTERFACE_USB3 USB 3.0.

INTERFACE_GIGE GigE.

INTERFACE_UNKNOWN Unknown interface.

INTERFACE_TYPE_FORCE_32BITS

6.2.1.13 enum Mode

Camera modes for DCAM formats as well as Format7.

Enumerator:

MODE_0

MODE_1

MODE_2

MODE_3

MODE_4

MODE_5

MODE_6

MODE_7

MODE_8

MODE_9

MODE_10

MODE_11

MODE_12

MODE_13

MODE_14

MODE_15

MODE_16

MODE_17

MODE_18

MODE_19

MODE_20

MODE_21

MODE_22

MODE_23

MODE_24

MODE_25

MODE_26

MODE_27

MODE_28

MODE_29

MODE_30

MODE_31

NUM_MODES Number of modes.

MODE_FORCE_32BITS

6.2.1.14 enum PCIeBusSpeed

Enumerator:

PCIE_BUSSPEED_2_5

PCIE_BUSSPEED_5_0 2.5 Gb/s

PCIE_BUSSPEED_UNKNOWN 5.0 Gb/s

PCIE_BUSSPEED_FORCE_32BITS Speed is unknown.

6.2.1.15 enum PixelFormat

Pixel formats available for Format7 modes.

Enumerator:

PIXEL_FORMAT_MONO8 8 bits of mono information.

PIXEL_FORMAT_411YUV8 YUV 4:1:1.

PIXEL_FORMAT_422YUV8	YUV 4:2:2.
PIXEL_FORMAT_444YUV8	YUV 4:4:4.
PIXEL_FORMAT_RGB8	R = G = B = 8 bits.
PIXEL_FORMAT_MONO16	16 bits of mono information.
PIXEL_FORMAT_RGB16	R = G = B = 16 bits.
PIXEL_FORMAT_S_MONO16	16 bits of signed mono information.
PIXEL_FORMAT_S_RGB16	R = G = B = 16 bits signed.
PIXEL_FORMAT_RAW8	8 bit raw data output of sensor.
PIXEL_FORMAT_RAW16	16 bit raw data output of sensor.
PIXEL_FORMAT_MONO12	12 bits of mono information.
PIXEL_FORMAT_RAW12	12 bit raw data output of sensor.
PIXEL_FORMAT_BGR	24 bit BGR.
PIXEL_FORMAT_BGRU	32 bit BGRU.
PIXEL_FORMAT_RGB	24 bit RGB.
PIXEL_FORMAT_RGBU	32 bit RGBU.
PIXEL_FORMAT_BGR16	R = G = B = 16 bits.
PIXEL_FORMAT_BGRU16	64 bit BGRU.
PIXEL_FORMAT_422YUV8_JPEG	JPEG compressed stream.
NUM_PIXEL_FORMATS	Number of pixel formats.
UNSPECIFIED_PIXEL_FORMAT	Unspecified pixel format.

6.2.1.16 enum PropertyType

Camera properties.

Not all properties may be supported, depending on the camera model.

Enumerator:

BRIGHTNESS	Brightness.
AUTO_EXPOSURE	Auto exposure.
SHARPNESS	Sharpness.
WHITE_BALANCE	White balance.
HUE	Hue.
SATURATION	Saturation.
GAMMA	Gamma.
IRIS	Iris.
FOCUS	Focus.
ZOOM	Zoom.
PAN	Pan.

TILT Tilt.

SHUTTER Shutter.

GAIN Gain.

TRIGGER_MODE Trigger mode.

TRIGGER_DELAY Trigger delay.

FRAME_RATE Frame rate.

TEMPERATURE Temperature.

UNSPECIFIED_PROPERTY_TYPE Unspecified property type.

PROPERTY_TYPE_FORCE_32BITS

6.2.1.17 enum VideoMode

DCAM video modes.

Enumerator:

VIDEOMODE_160x120YUV444 160x120 YUV444.

VIDEOMODE_320x240YUV422 320x240 YUV422.

VIDEOMODE_640x480YUV411 640x480 YUV411.

VIDEOMODE_640x480YUV422 640x480 YUV422.

VIDEOMODE_640x480RGB 640x480 24-bit RGB.

VIDEOMODE_640x480Y8 640x480 8-bit.

VIDEOMODE_640x480Y16 640x480 16-bit.

VIDEOMODE_800x600YUV422 800x600 YUV422.

VIDEOMODE_800x600RGB 800x600 RGB.

VIDEOMODE_800x600Y8 800x600 8-bit.

VIDEOMODE_800x600Y16 800x600 16-bit.

VIDEOMODE_1024x768YUV422 1024x768 YUV422.

VIDEOMODE_1024x768RGB 1024x768 RGB.

VIDEOMODE_1024x768Y8 1024x768 8-bit.

VIDEOMODE_1024x768Y16 1024x768 16-bit.

VIDEOMODE_1280x960YUV422 1280x960 YUV422.

VIDEOMODE_1280x960RGB 1280x960 RGB.

VIDEOMODE_1280x960Y8 1280x960 8-bit.

VIDEOMODE_1280x960Y16 1280x960 16-bit.

VIDEOMODE_1600x1200YUV422 1600x1200 YUV422.

VIDEOMODE_1600x1200RGB 1600x1200 RGB.

VIDEOMODE_1600x1200Y8 1600x1200 8-bit.

VIDEOMODE_1600x1200Y16 1600x1200 16-bit.

VIDEOMODE_FORMAT7 Custom video mode for Format7 functionality.

NUM_VIDEOMODES Number of possible video modes.

VIDEOMODE_FORCE_32BITS

6.3 GigE specific enumerations

These enumerations are specific to GigE camera operation only.

Enumerations

- enum GigEPropertyType { HEARTBEAT, HEARTBEAT_TIMEOUT, PACKET_-SIZE, PACKET_DELAY }

  Possible properties that can be queried from the camera.

6.3.1 Detailed Description

These enumerations are specific to GigE camera operation only.

6.3.2 Enumeration Type Documentation

6.3.2.1 enum GigEPropertyType

Possible properties that can be queried from the camera.

Enumerator:

HEARTBEAT

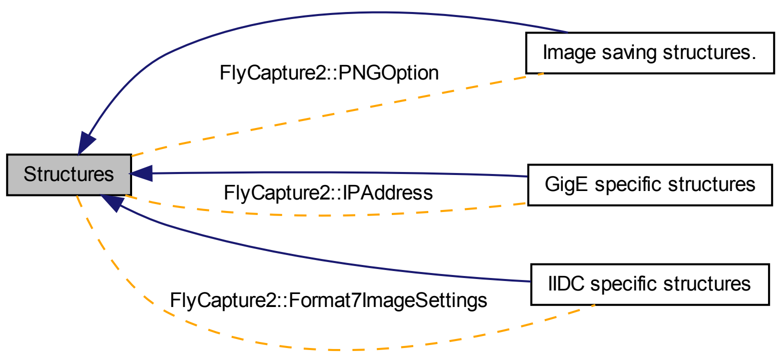
HEARTBEAT_TIMEOUT

PACKET_SIZE

PACKET_DELAY

6.4 Structures

Collaboration diagram for Structures:



Classes

- struct FC2Version

  The current version of the library.
- class PGRGuid

  A GUID to the camera.
- struct IPAddress

  IPv4 address.
- struct Format7ImageSettings

  Format 7 image settings.
- struct FC2Config

  Configuration for a camera.
- struct PropertyInfo

  Information about a specific camera property.
- struct Property

  A specific camera property.
- struct TriggerModeInfo

  Information about a camera trigger property.
- struct TriggerMode

  A camera trigger.
- struct StrobeInfo

  A camera strobe property.
- struct StrobeControl

A camera strobe.
- struct TimeStamp

  Timestamp information.
- struct ConfigROM

  Camera configuration ROM.
- struct CameraInfo

  Camera information.
- struct EmbeddedImageInfoProperty

  Properties of a single embedded image info property.
- struct EmbeddedImageInfo

  Properties of the possible embedded image information.
- struct ImageMetadata

  Metadata related to an image.
- struct LUTData

  Information about the camera's look up table.
- struct HostAdapterStats

  Information about the host adapter's statistics.
- struct CameraStats

  Camera diagnostic information.
- struct PNGOption

  Options for saving PNG images.

Modules

- GigE specific structures

  These structures are specific to GigE camera operation only.
- IIDC specific structures

  These structures are specific to IIDC camera operation only.
- Image saving structures.

  These structures define various parameters used for saving images.

Typedefs

- typedef PropertyInfo TriggerDelayInfo

  The TriggerDelayInfo structure is identical to PropertyInfo.
- typedef Property TriggerDelay

  The TriggerDelay structure is identical to Property.

6.4.1 Typedef Documentation

6.4.1.1 typedef Property TriggerDelay

The TriggerDelay structure is identical to Property.

6.4.1.2 typedef PropertyInfo TriggerDelayInfo

The TriggerDelayInfo structure is identical to PropertyInfo.

6.5 GigE specific structures

These structures are specific to GigE camera operation only.

Collaboration diagram for GigE specific structures:

GigE specific structures
FlyCapture2::IPAddress
Structures

Classes

- struct IPAddress

  IPv4 address.
- struct MACAddress

  MAC address.
- struct GigEProperty

  A GigE property.
- struct GigEStreamChannel

  Information about a single GigE stream channel.
- struct GigEConfig

  Configuration for a GigE camera.
- struct GigEImageSettingsInfo

  Format 7 information for a single mode.
- struct GigEImageSettings

  Image settings for a GigE camera.

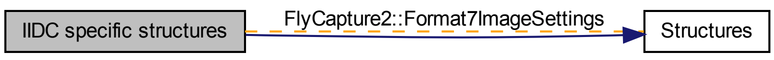
6.5.1 Detailed Description

These structures are specific to GigE camera operation only.

6.6 IIDC specific structures

These structures are specific to IIDC camera operation only.

Collaboration diagram for IIDC specific structures:



Classes

- struct Format7ImageSettings

  Format 7 image settings.
- struct Format7Info

  Format 7 information for a single mode.
- struct Format7PacketInfo

  Format 7 packet information.

6.6.1 Detailed Description

These structures are specific to IIDC camera operation only.

6.7 Image saving structures.

These structures define various parameters used for saving images.

Collaboration diagram for Image saving structures.:

Classes

- struct PNGOption

  Options for saving PNG images.
- struct PPMOption

  Options for saving PPM images.
- struct PGMOption

  Options for saving PGM images.
- struct TIFFOption

  Options for saving TIFF images.
- struct JPEGOption

  Options for saving JPEG image.
- struct JPG2Option

  Options for saving JPEG2000 image.
- struct BMPOption

  Options for saving Bitmap image.
- struct MJPGOption

  Options for saving MJPG files.
- struct H264Option

  Options for saving H264 files.
- struct AVIOption

  Options for saving AVI files.

6.7.1 Detailed Description

These structures define various parameters used for saving images.

Chapter 7

Namespace Documentation

7.1 FlyCapture2 Namespace Reference

Classes

- class AVIRecorder

    The AVIRecorder class provides the functionality for the user to record images to an AVI file.
- class BusManager

    The BusManager class provides the functionality for the user to get an PGRGuid for a desired camera or device easily.
- class Camera

    The Camera object represents a physical camera that uses the IIDC register set.
- class CameraBase

    The CameraBase class is an abstract base class that defines a general interface to a camera.
- class Error

    The Error object represents an error that is returned from the library.
- struct FC2Version

    The current version of the library.
- class PGRGuid

    A GUID to the camera.
- struct IPAddress

    IPv4 address.
- struct MACAddress

    MAC address.
- struct GigEProperty

    A GigE property.
- struct GigEStreamChannel

    Information about a single GigE stream channel.

- struct GigEConfig

  Configuration for a GigE camera.
- struct GigEImageSettingsInfo

  Format 7 information for a single mode.
- struct GigEImageSettings

  Image settings for a GigE camera.
- struct Format7ImageSettings

  Format 7 image settings.
- struct Format7Info

  Format 7 information for a single mode.
- struct Format7PacketInfo

  Format 7 packet information.
- struct FC2Config

  Configuration for a camera.
- struct PropertyInfo

  Information about a specific camera property.
- struct Property

  A specific camera property.
- struct TriggerModeInfo

  Information about a camera trigger property.
- struct TriggerMode

  A camera trigger.
- struct StrobeInfo

  A camera strobe property.
- struct StrobeControl

  A camera strobe.
- struct TimeStamp

  Timestamp information.
- struct ConfigROM

  Camera configuration ROM.
- struct CameraInfo

  Camera information.
- struct EmbeddedImageInfoProperty

  Properties of a single embedded image info property.
- struct EmbeddedImageInfo

  Properties of the possible embedded image information.
- struct ImageMetadata

  Metadata related to an image.
- struct LUTData

  Information about the camera's look up table.
- struct HostAdapterStats

  Information about the host adapter's statistics.
- struct CameraStats

Camera diagnostic information.
- struct PNGOption

  Options for saving PNG images.
- struct PPMOption

  Options for saving PPM images.
- struct PGMOption

  Options for saving PGM images.
- struct TIFFOption

  Options for saving TIFF images.
- struct JPEGOption

  Options for saving JPEG image.
- struct JPG2Option

  Options for saving JPEG2000 image.
- struct BMPOption

  Options for saving Bitmap image.
- struct MJPGOption

  Options for saving MJPG files.
- struct H264Option

  Options for saving H264 files.
- struct AVIOption

  Options for saving AVI files.
- class CameraControlDlg

  The CameraControlDlg object represents a dialog that provides a graphical interface to a specified camera.
- class CameraSelectionDlg

  The CameraSelectionDlg object represents a dialog that provides a graphical interface that lists the number of cameras available to the library.
- class GCCamera
- class GigECamera

  The GigECamera object represents a physical Gigabit Ethernet camera.
- class Image

  The Image class is used to retrieve images from a camera, convert between multiple pixel formats and save images to disk.
- class ImageStatistics

  The ImageStatistics object represents image statistics for an image.
- class Internal
- class NodeMap
- class TopologyNode

  The TopologyNode class contains topology information that can be used to generate a tree structure of all cameras and devices connected to a computer.
- struct SystemInfo

  Description of the system.
- class Utilities

  The Utility class is generally used to query for general system information such as operating system, available memory etc.

Typedefs

- typedef void(∗ BusEventCallback)(void ∗pParameter, unsigned int serial-Number)

  Bus event callback function prototype.
- typedef void ∗ CallbackHandle

  Handle that is returned when registering a callback.
- typedef void(∗ ImageEventCallback)(class Image ∗pImage, const void ∗p-CallbackData)

  Image event callback function prototype.
- typedef PropertyInfo TriggerDelayInfo

  The TriggerDelayInfo structure is identical to PropertyInfo.
- typedef Property TriggerDelay

  The TriggerDelay structure is identical to Property.
- typedef void(∗ AsyncCommandCallback)(class Error retError, void ∗pUser-Data)

  Async command callback function prototype.

Enumerations

- enum ErrorType { PGRERROR_UNDEFINED = -1, PGRERROR_OK, PGRE-RROR_FAILED, PGRERROR_NOT_IMPLEMENTED, PGRERROR_FAILED_-BUS_MASTER_CONNECTION, PGRERROR_NOT_CONNECTED, PGRERR-OR_INIT_FAILED, PGRERROR_NOT_INTITIALIZED, PGRERROR_INVALID-_PARAMETER, PGRERROR_INVALID_SETTINGS, PGRERROR_INVALID_-BUS_MANAGER, PGRERROR_MEMORY_ALLOCATION_FAILED, PGRERR-OR_LOW_LEVEL_FAILURE, PGRERROR_NOT_FOUND, PGRERROR_FAI-LED_GUID, PGRERROR_INVALID_PACKET_SIZE, PGRERROR_INVALID_-MODE, PGRERROR_NOT_IN_FORMAT7, PGRERROR_NOT_SUPPORTED, PGRERROR_TIMEOUT, PGRERROR_BUS_MASTER_FAILED, PGRERRO-R_INVALID_GENERATION, PGRERROR_LUT_FAILED, PGRERROR_IIDC-_FAILED, PGRERROR_STROBE_FAILED, PGRERROR_TRIGGER_FAILED, PGRERROR_PROPERTY_FAILED, PGRERROR_PROPERTY_NOT_PRES-ENT, PGRERROR_REGISTER_FAILED, PGRERROR_READ_REGISTER_F-AILED, PGRERROR_WRITE_REGISTER_FAILED, PGRERROR_ISOCH_FA-ILED, PGRERROR_ISOCH_ALREADY_STARTED, PGRERROR_ISOCH_NO-T_STARTED, PGRERROR_ISOCH_START_FAILED, PGRERROR_ISOCH_-RETRIEVE_BUFFER_FAILED, PGRERROR_ISOCH_STOP_FAILED, PGRE-RROR_ISOCH_SYNC_FAILED, PGRERROR_ISOCH_BANDWIDTH_EXCEE-DED, PGRERROR_IMAGE_CONVERSION_FAILED, PGRERROR_IMAGE_L-IBRARY_FAILURE, PGRERROR_BUFFER_TOO_SMALL, PGRERROR_IMA-GE_CONSISTENCY_ERROR, PGRERROR_FORCE_32BITS = FULL_32BIT_-VALUE }

  The error types returned by functions.
- enum BusCallbackType { BUS_RESET, ARRIVAL, REMOVAL, CALLBACK_-TYPE_FORCE_32BITS = FULL_32BIT_VALUE }

  The type of bus callback to register a callback function for.

- enum GrabMode { DROP_FRAMES, BUFFER_FRAMES, UNSPECIFIED_GRAB_MODE, GRAB_MODE_FORCE_32BITS = FULL_32BIT_VALUE }

  The grab strategy employed during image transfer.

- enum GrabTimeout { TIMEOUT_NONE = 0, TIMEOUT_INFINITE = -1, TIMEOUT_UNSPECIFIED = -2, GRAB_TIMEOUT_FORCE_32BITS = FULL_32BIT_VALUE }

  Timeout options for grabbing images.

- enum BandwidthAllocation { BANDWIDTH_ALLOCATION_OFF = 0, BANDWIDTH_ALLOCATION_ON = 1, BANDWIDTH_ALLOCATION_UNSUPPORTED = 2, BANDWIDTH_ALLOCATION_UNSPECIFIED = 3, BANDWIDTH_ALLOCATION_FORCE_32BITS = FULL_32BIT_VALUE }

  Bandwidth allocation options for 1394 devices.

- enum InterfaceType { INTERFACE_IEEE1394, INTERFACE_USB2, INTERFACE_USB3, INTERFACE_GIGE, INTERFACE_UNKNOWN, INTERFACE_TYPE_FORCE_32BITS = FULL_32BIT_VALUE }

  Interfaces that a camera may use to communicate with a host.

- enum PropertyType { BRIGHTNESS, AUTO_EXPOSURE, SHARPNESS, WHITE_BALANCE, HUE, SATURATION, GAMMA, IRIS, FOCUS, ZOOM, PAN, TILT, SHUTTER, GAIN, TRIGGER_MODE, TRIGGER_DELAY, FRAME_RATE, TEMPERATURE, UNSPECIFIED_PROPERTY_TYPE, PROPERTY_TYPE_FORCE_32BITS = FULL_32BIT_VALUE }

  Camera properties.

- enum FrameRate { FRAMERATE_1_875, FRAMERATE_3_75, FRAMERATE_7_5, FRAMERATE_15, FRAMERATE_30, FRAMERATE_60, FRAMERATE_120, FRAMERATE_240, FRAMERATE_FORMAT7, NUM_FRAMERATES, FRAMERATE_FORCE_32BITS = FULL_32BIT_VALUE }

  Frame rates in frames per second.

- enum VideoMode { VIDEOMODE_160x120YUV444, VIDEOMODE_320x240YUV422, VIDEOMODE_640x480YUV411, VIDEOMODE_640x480YUV422, VIDEOMODE_640x480RGB, VIDEOMODE_640x480Y8, VIDEOMODE_640x480Y16, VIDEOMODE_800x600YUV422, VIDEOMODE_800x600RGB, VIDEOMODE_800x600Y8, VIDEOMODE_800x600Y16, VIDEOMODE_1024x768YUV422, VIDEOMODE_1024x768RGB, VIDEOMODE_1024x768Y8, VIDEOMODE_1024x768Y16, VIDEOMODE_1280x960YUV422, VIDEOMODE_1280x960RGB, VIDEOMODE_1280x960Y8, VIDEOMODE_1280x960Y16, VIDEOMODE_1600x1200YUV422, VIDEOMODE_1600x1200RGB, VIDEOMODE_1600x1200Y8, VIDEOMODE_1600x1200Y16, VIDEOMODE_FORMAT7, NUM_VIDEOMODES, VIDEOMODE_FORCE_32BITS = FULL_32BIT_VALUE }

  DCAM video modes.

- enum Mode { MODE_0 = 0, MODE_1, MODE_2, MODE_3, MODE_4, MODE_5, MODE_6, MODE_7, MODE_8, MODE_9, MODE_10, MODE_11, MODE_12, MODE_13, MODE_14, MODE_15, MODE_16, MODE_17, MODE_18, MODE_19, MODE_20, MODE_21, MODE_22, MODE_23, MODE_24, MODE_25, MODE_26, MODE_27, MODE_28, MODE_29, MODE_30, MODE_31, NUM_MODES, MODE_FORCE_32BITS = FULL_32BIT_VALUE }

  Camera modes for DCAM formats as well as Format7.

- enum PixelFormat { PIXEL_FORMAT_MONO8 = 0x80000000, PIXEL_FORMAT_411YUV8 = 0x40000000, PIXEL_FORMAT_422YUV8 = 0x20000000, PIXEL_FORMAT_444YUV8 = 0x10000000, PIXEL_FORMAT_RGB8 = 0x08000000, PIXEL_FORMAT_MONO16 = 0x04000000, PIXEL_FORMAT_RGB16 = 0x02000000, PIXEL_FORMAT_S_MONO16 = 0x01000000, PIXEL_FORMAT_S_RGB16 = 0x00800000, PIXEL_FORMAT_RAW8 = 0x00400000, PIXEL_FORMAT_RAW16 = 0x00200000, PIXEL_FORMAT_MONO12 = 0x00100000, PIXEL_FORMAT_RAW12 = 0x00080000, PIXEL_FORMAT_BGR = 0x80000008, PIXEL_FORMAT_BGRU = 0x40000008, PIXEL_FORMAT_RGB = PIXEL_FORMAT_RGB8, PIXEL_FORMAT_RGBU = 0x40000002, PIXEL_FORMAT_BGR16 = 0x02000001, PIXEL_FORMAT_BGRU16 = 0x02000002, PIXEL_FORMAT_422YUV8_JPEG = 0x40000001, NUM_PIXEL_FORMATS = 20, UNSPECIFIED_PIXEL_FORMAT = 0 }

  Pixel formats available for Format7 modes.
- enum BusSpeed { BUSSPEED_S100, BUSSPEED_S200, BUSSPEED_S400, BUSSPEED_S480, BUSSPEED_S800, BUSSPEED_S1600, BUSSPEED_S3200, BUSSPEED_S5000, BUSSPEED_10BASE_T, BUSSPEED_100BASE_T, BUSSPEED_1000BASE_T, BUSSPEED_10000BASE_T, BUSSPEED_S_FASTEST, BUSSPEED_ANY, BUSSPEED_SPEED_UNKNOWN = -1, BUSSPEED_FORCE_32BITS = FULL_32BIT_VALUE }

  Bus speeds.
- enum PCIeBusSpeed { PCIE_BUSSPEED_2_5, PCIE_BUSSPEED_5_0, PCIE_BUSSPEED_UNKNOWN = -1, PCIE_BUSSPEED_FORCE_32BITS = FULL_32BIT_VALUE }
- enum DriverType { DRIVER_1394_CAM, DRIVER_1394_PRO, DRIVER_1394_JUJU, DRIVER_1394_VIDEO1394, DRIVER_1394_RAW1394, DRIVER_USB_NONE, DRIVER_USB_CAM, DRIVER_USB3_PRO, DRIVER_GIGE_NONE, DRIVER_GIGE_FILTER, DRIVER_GIGE_PRO, DRIVER_UNKNOWN = -1, DRIVER_FORCE_32BITS = FULL_32BIT_VALUE }

  Types of low level drivers that flycapture uses.
- enum ColorProcessingAlgorithm { DEFAULT, NO_COLOR_PROCESSING, NEAREST_NEIGHBOR, EDGE_SENSING, HQ_LINEAR, RIGOROUS, IPP, DIRECTIONAL_FILTER, COLOR_PROCESSING_ALGORITHM_FORCE_32BITS = FULL_32BIT_VALUE }

  Color processing algorithms.
- enum BayerTileFormat { NONE, RGGB, GRBG, GBRG, BGGR, BT_FORCE_32BITS = FULL_32BIT_VALUE }

  Bayer tile formats.
- enum ImageFileFormat { FROM_FILE_EXT = -1, PGM, PPM, BMP, JPEG, JPEG2000, TIFF, PNG, RAW, IMAGE_FILE_FORMAT_FORCE_32BITS = FULL_32BIT_VALUE }

  File formats to be used for saving images to disk.
- enum GigEPropertyType { HEARTBEAT, HEARTBEAT_TIMEOUT, PACKET_SIZE, PACKET_DELAY }

  Possible properties that can be queried from the camera.
- enum OSType { WINDOWS_X86, WINDOWS_X64, LINUX_X86, LINUX_X64, MAC, UNKNOWN_OS, OSTYPE_FORCE_32BITS = FULL_32BIT_VALUE }

Possible operating systems.

- enum ByteOrder { BYTE_ORDER_LITTLE_ENDIAN, BYTE_ORDER_BIG_ENDIAN, BYTE_ORDER_FORCE_32BITS = FULL_32BIT_VALUE }

  Possible byte orders.

Variables

- static const unsigned int sk_maxStringLength = 512

  The maximum length that is allocated for a string.

- static const unsigned int sk_maxNumPorts = 32

  The maximum number of ports one device can have.

7.1.1 Typedef Documentation

7.1.1.1 typedef void(∗ **AsyncCommandCallback**)(class **Error** retError, void ∗pUserData)

Async command callback function prototype.

Defines the syntax of the async command function that is passed into Launch-CommandAsync().

7.1.1.2 typedef void(∗ **BusEventCallback**)(void ∗pParameter, unsigned int serialNumber)

Bus event callback function prototype.

Defines the syntax of the callback function that is passed into RegisterCallback() and UnregisterCallback(). It is recommended that minimal handling be performed in this callback as it will block internal processing of bus events until it returns.

7.1.1.3 typedef void∗ **CallbackHandle**

Handle that is returned when registering a callback.

It is required when unregistering the callback.

7.1.1.4 typedef void(∗ **ImageEventCallback**)(class **Image** ∗pImage, const void ∗pCallbackData)

Image event callback function prototype.

Defines the syntax of the image callback function that is passed into StartCapture(). It is possible for this function to be called simultaneously. Therefore, users must make sure that code in the callback is thread safe.

7.1.2 Enumeration Type Documentation

7.1.2.1 enum ByteOrder

Possible byte orders.

Enumerator:

BYTE_ORDER_LITTLE_ENDIAN

BYTE_ORDER_BIG_ENDIAN

BYTE_ORDER_FORCE_32BITS

7.1.2.2 enum OSType

Possible operating systems.

Enumerator:

WINDOWS_X86 All Windows 32-bit variants.

WINDOWS_X64 All Windows 64-bit variants.

LINUX_X86 All Linux 32-bit variants.

LINUX_X64 All Linux 32-bit variants.

MAC Mac OSX.

UNKNOWN_OS Unknown operating system.

OSTYPE_FORCE_32BITS

7.2 MultiSyncLibrary Namespace Reference

Classes

- class SyncManager

Enumerations

- enum PGRSyncError { PGRSyncError_OK = 0, PGRSyncError_FAILED, PGRSyncError_ALREADY_STARTED, PGRSyncError_ALREADY_STOPPED, PGRSyncError_CAMERA_NOT_FOUND, PGRSyncError_UNKNOWN_ERROR }
- enum PGRSyncMessage { PGRSyncMessage_OK = 0, PGRSyncMessage_STARTED, PGRSyncMessage_STOPPED, PGRSyncMessage_SYNCING, PGRSyncMessage_NOMASTER, PGRSyncMessage_THREAD_ERROR, PGRSyncMessage_DEVICE_ERROR, PGRSyncMessage_NOT_ENOUGH_DEVICES, PGRSyncMessage_BUS_RESET, PGRSyncMessage_NOT_INITIALIZED, PGRSyncMessage_UNKNOWN_ERROR }

7.2.1 Enumeration Type Documentation

7.2.1.1 enum PGRSyncError

Enumerator:

PGRSyncError_OK

PGRSyncError_FAILED

PGRSyncError_ALREADY_STARTED

PGRSyncError_ALREADY_STOPPED

PGRSyncError_CAMERA_NOT_FOUND

PGRSyncError_UNKNOWN_ERROR

7.2.1.2 enum PGRSyncMessage

Enumerator:

PGRSyncMessage_OK

PGRSyncMessage_STARTED

PGRSyncMessage_STOPPED

PGRSyncMessage_SYNCING

PGRSyncMessage_NOMASTER

PGRSyncMessage_THREAD_ERROR

PGRSyncMessage_DEVICE_ERROR

PGRSyncMessage_NOT_ENOUGH_DEVICES

PGRSyncMessage_BUS_RESET

PGRSyncMessage_NOT_INITIALIZED

PGRSyncMessage_UNKNOWN_ERROR

Chapter 8

Class Documentation

8.1 AVIOption Struct Reference

Options for saving AVI files.

Public Member Functions

- AVIOption ()

Public Attributes

- float frameRate

  Frame rate of the stream.
- unsigned int reserved [256]

  Reserved for future use.

8.1.1 Detailed Description

Options for saving AVI files.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 AVIOption ( ) `[inline]`

8.1.3 Member Data Documentation

8.1.3.1 float frameRate

Frame rate of the stream.

8.1.3.2 unsigned int **reserved**[256]

Reserved for future use.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.2 AVIRecorder Class Reference

The AVIRecorder class provides the functionality for the user to record images to an AVI file.

Public Member Functions

- AVIRecorder ()

  Default constructor.
- virtual ∼AVIRecorder ()

  Default destructor.
- virtual Error AVIOpen (const char ∗pFileName, AVIOption ∗pOption)

  Open an AVI file in preparation for writing Images to disk.
- virtual Error AVIOpen (const char ∗pFileName, MJPGOption ∗pOption)

  Open an MJPEG AVI file in preparation for writing Images to disk.
- virtual Error AVIOpen (const char ∗pFileName, H264Option ∗pOption)

  Open an H264 MP4 file in preparation for writing Images to disk.
- virtual Error AVIAppend (Image ∗pImage)

  Append an image to the AVI/MP4 file.
- virtual Error AVIClose ()

  Close the AVI/MP4 file.

8.2.1 Detailed Description

The AVIRecorder class provides the functionality for the user to record images to an AVI file.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 **AVIRecorder ( )**

Default constructor.

8.2.2.2 virtual **∼AVIRecorder ( )** `[virtual]`

Default destructor.

8.2.3 Member Function Documentation

8.2.3.1 virtual Error AVIAppend (Image ∗ *pImage*) `[virtual]`

Append an image to the AVI/MP4 file.

Parameters

pImage	The image to append.

Returns

An Error indicating the success or failure of the function.

8.2.3.2 virtual Error AVIClose ( ) `[virtual]`

Close the AVI/MP4 file.

See also

AVIOpen()

Returns

An Error indicating the success or failure of the function.

8.2.3.3 virtual Error AVIOpen (const char ∗ *pFileName*, AVIOption ∗ *pOption*) `[virtual]`

Open an AVI file in preparation for writing Images to disk.

The size of AVI files is limited to 2GB. The filenames are automatically generated using the filename specified.

Parameters

pFileName	The filename of the AVI file.
pOption	Options to apply to the AVI file.

See also

AVIClose()

Returns

An Error indicating the success or failure of the function.

8.2.3.4 virtual Error AVIOpen (const char ∗ *pFileName,* MJPGOption ∗ *pOption*) [virtual]

Open an MJPEG AVI file in preparation for writing Images to disk.

The size of AVI files is limited to 2GB. The filenames are automatically generated using the filename specified.

Parameters

pFileName	The filename of the AVI file.
pOption	MJPEG options to apply to the AVI file.

See also

AVIClose()
MJPGOption

Returns

An Error indicating the success or failure of the function.

8.2.3.5 virtual Error AVIOpen (const char ∗ *pFileName,* H264Option ∗ *pOption*) [virtual]

Open an H264 MP4 file in preparation for writing Images to disk.

The size of MP4 files is limited to 2GB. The filenames are automatically generated using the filename specified.

Parameters

pFileName	The filename of the MP4 file.
pOption	H264 options to apply to the MP4 file.

See also

AVIClose()
H264Option

Returns

An Error indicating the success or failure of the function.

The documentation for this class was generated from the following file:

- AVIRecorder.h

8.3 BMPOption Struct Reference

Options for saving Bitmap image.

Public Member Functions

- BMPOption ()

Public Attributes

- bool indexedColor_8bit
- unsigned int reserved [16]

  Reserved for future use.

8.3.1 Detailed Description

Options for saving Bitmap image.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 BMPOption ( ) `[inline]`

8.3.3 Member Data Documentation

8.3.3.1 bool indexedColor_8bit

8.3.3.2 unsigned int reserved[16]

Reserved for future use.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.4 BusManager Class Reference

The BusManager class provides the functionality for the user to get an PGRGuid for a desired camera or device easily.

Public Member Functions

- BusManager ()

  Default constructor.

- virtual ∼BusManager ()

  Default destructor.
- virtual Error FireBusReset (PGRGuid ∗pGuid)

  Fire a bus reset.
- virtual Error GetNumOfCameras (unsigned int ∗pNumCameras)

  Gets the number of cameras attached to the PC.
- virtual Error GetCameraFromIPAddress (IPAddress ipAddress, PGRGuid ∗p-Guid)

  Gets the PGRGuid for a camera with the specified IPv4 address.
- virtual Error GetCameraFromIndex (unsigned int index, PGRGuid ∗pGuid)

  Gets the PGRGuid for a camera on the PC.
- virtual Error GetCameraFromSerialNumber (unsigned int serialNumber, PGR-Guid ∗pGuid)

  Gets the PGRGuid for a camera on the PC.
- virtual Error GetCameraSerialNumberFromIndex (unsigned int index, unsigned int ∗pSerialNumber)

  Gets the serial number of the camera with the specified index.
- virtual Error GetInterfaceTypeFromGuid (PGRGuid ∗pGuid, InterfaceType ∗p-InterfaceType)

  Gets the interface type associated with a PGRGuid.
- virtual Error GetNumOfDevices (unsigned int ∗pNumDevices)

  Gets the number of devices.
- virtual Error GetDeviceFromIndex (unsigned int index, PGRGuid ∗pGuid)

  Gets the PGRGuid for a device.
- virtual Error ReadPhyRegister (PGRGuid guid, unsigned int page, unsigned int port, unsigned int address, unsigned int ∗pValue)

  Read a phy register on the specified device.
- virtual Error WritePhyRegister (PGRGuid guid, unsigned int page, unsigned int port, unsigned int address, unsigned int value)

  Write a phy register on the specified device.
- virtual Error GetUsbLinkInfo (PGRGuid guid, unsigned int ∗pValue)

  Read usb link info for the port that the specified device is connected to.
- virtual Error GetUsbPortStatus (PGRGuid guid, unsigned int ∗pValue)

  Read usb port status for the port that the specified device is connected to.
- virtual Error GetTopology (TopologyNode ∗pNode)

  Gets the topology information for the PC.
- virtual Error RegisterCallback (BusEventCallback busEventCallback, Bus-CallbackType callbackType, void ∗pParameter, CallbackHandle ∗pCallback-Handle)

  Register a callback function that will be called when the specified callback event oc-curs.
- virtual Error UnregisterCallback (CallbackHandle callbackHandle)

  Unregister a callback function.
- virtual Error RescanBus ()

Force a rescan of the buses.

- Error IsCameraControlable (PGRGuid ∗pGuid, bool ∗pControlable)

  Query CCP status on camera with corresponding PGRGuid.

Static Public Member Functions

- static Error ForceIPAddressToCamera (MACAddress macAddress, IPAddress ipAddress, IPAddress subnetMask, IPAddress defaultGateway)

  Force the camera with the specific MAC address to the specified IP address, subnet mask and default gateway.
- static Error ForceAllIPAddressesAutomatically ()

  Force all cameras on the network to be assigned sequential IP addresses on the same subnet as the netowrk adapters that they are connected to.
- static Error ForceAllIPAddressesAutomatically (unsigned int serialNumber)

  Force a camera on the network to be assigned an IP address on the same subnet as the netowrk adapters that it is connected to.
- static Error DiscoverGigECameras (CameraInfo ∗gigECameras, unsigned int ∗arraySize)

  Discover all cameras connected to the network even if they reside on a different subnet.

8.4.1 Detailed Description

The BusManager class provides the functionality for the user to get an PGRGuid for a desired camera or device easily.

Once the camera or device token is found, it can then be used to connect to the camera or device through the camera class or device class. In addition, the BusManager class provides the ability to be notified when a camera or device is added or removed or some event occurs on the PC.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 BusManager ( )

Default constructor.

8.4.2.2 virtual ∼BusManager ( ) [virtual]

Default destructor.

8.4.3 Member Function Documentation

8.4.3.1 static Error DiscoverGigECameras (CameraInfo ∗ *gigECameras,* unsigned int ∗ *arraySize*) [static]

Discover all cameras connected to the network even if they reside on a different subnet.

This is useful in situations where a GigE camera is using Persistent IP and the application's subnet is different from the device subnet. After discovering the camera, it is easy to use ForceIPAddressToCamera() to set a different IP configuration.

Parameters

gigE-Cameras	Pointer to an array of CameraInfo structures.
arraySize	Size of the array. Number of discovered cameras is returned in the same value.

Returns

An Error indicating the success or failure of the function. If the error is PGRERROR_BUFFER_TOO_SMALL then arraySize will contain the minimum size needed for gigECameras array.

8.4.3.2 virtual Error FireBusReset (PGRGuid ∗ *pGuid*) [virtual]

Fire a bus reset.

The actual bus reset is only fired for the specified 1394 bus, but it will effectively cause a global bus reset for the library.

Parameters

pGuid	PGRGuid of the camera or the device to cause bus reset.

Returns

An Error indicating the success or failure of the function.

8.4.3.3 static Error ForceAllIPAddressesAutomatically ( ) [static]

Force all cameras on the network to be assigned sequential IP addresses on the same subnet as the netowrk adapters that they are connected to.

This is useful in situations where a GigE Vision cameras are using Persistent IP addresses and the application's subnet is different from the devices.

Returns

An Error indicating the success or failure of the function.

8.4.3.4 static Error ForceAllIPAddressesAutomatically (unsigned int *serialNumber*) `[static]`

Force a camera on the network to be assigned an IP address on the same subnet as the netowrk adapters that it is connected to.

This is useful in situations where a GigE Vision camera is using Persistent IP addresses and the application's subnet is different from the device.

Returns

An Error indicating the success or failure of the function.

8.4.3.5 static Error ForceIPAddressToCamera (MACAddress *macAddress,* IPAddress *ipAddress,* IPAddress *subnetMask,* IPAddress *defaultGateway*) `[static]`

Force the camera with the specific MAC address to the specified IP address, subnet mask and default gateway.

This is useful in situations where a GigE Vision camera is using Persistent IP and the application's subnet is different from the device subnet.

Parameters

macAddress	MAC address of the camera.
ipAddress	IP address to set on the camera.
subnetMask	Subnet mask to set on the camera.
default-Gateway	Default gateway to set on the camera.

Returns

An Error indicating the success or failure of the function.

8.4.3.6 virtual Error GetCameraFromIndex (unsigned int *index,* PGRGuid ∗ *pGuid*) `[virtual]`

Gets the PGRGuid for a camera on the PC.

It uniquely identifies the camera specified by the index and is used to identify the camera during a Camera::Connect() call.

Parameters

index	Zero based index of camera.
pGuid	Unique PGRGuid for the camera.

See also

GetCameraFromSerialNumber()

Returns

An Error indicating the success or failure of the function.

8.4.3.7 virtual Error GetCameraFromIPAddress (IPAddress *ipAddress,* PGRGuid * *pGuid*) [virtual]

Gets the PGRGuid for a camera with the specified IPv4 address.

Parameters

ipAddress	IP address to get GUID for.
pGuid	Unique PGRGuid for the camera.

Returns

An Error indicating the success or failure of the function.

8.4.3.8 virtual Error GetCameraFromSerialNumber (unsigned int *serialNumber,* PGRGuid * *pGuid*) [virtual]

Gets the PGRGuid for a camera on the PC.

It uniquely identifies the camera specified by the serial number and is used to identify the camera during a Camera::Connect() call.

Parameters

serial-Number	Serial number of camera.
pGuid	Unique PGRGuid for the camera.

See also

GetCameraFromIndex()

Returns

An Error indicating the success or failure of the function.

8.4.3.9 virtual Error GetCameraSerialNumberFromIndex (unsigned int *index,* unsigned int ∗ *pSerialNumber*) [virtual]

Gets the serial number of the camera with the specified index.

Parameters

index	Zero based index of desired camera.
pSerial-Number	Serial number of camera.

Returns

An Error indicating the success or failure of the function.

8.4.3.10 virtual Error GetDeviceFromIndex (unsigned int *index,* PGRGuid ∗ *pGuid*) [virtual]

Gets the PGRGuid for a device.

It uniquely identifies the device specified by the index.

Parameters

index	Zero based index of device.
pGuid	Unique PGRGuid for the device.

See also

GetNumOfDevices()

Returns

An Error indicating the success or failure of the function.

8.4.3.11 virtual Error GetInterfaceTypeFromGuid (PGRGuid ∗ *pGuid,* InterfaceType ∗ *pInterfaceType*) [virtual]

Gets the interface type associated with a PGRGuid.

This is useful in situations where there is a need to enumerate all cameras for a particular interface.

Parameters

pGuid	The PGRGuid to get the interface for.
pInterface-Type	The interface type of the PGRGuid.

Returns

An Error indicating the success or failure of the function.

8.4.3.12 virtual Error GetNumOfCameras (unsigned int ∗ *pNumCameras*) [virtual]

Gets the number of cameras attached to the PC.

Parameters

pNum-Cameras	The number of cameras attached.

Returns

An Error indicating the success or failure of the function.

8.4.3.13 virtual Error GetNumOfDevices (unsigned int ∗ *pNumDevices*) [virtual]

Gets the number of devices.

This may include hubs, host controllers and other hardware devices (including cameras).

Parameters

pNum-Devices	The number of devices found.

Returns

An Error indicating the success or failure of the function.

8.4.3.14 virtual Error GetTopology (TopologyNode ∗ *pNode*) [virtual]

Gets the topology information for the PC.

Parameters

pNode	TopologyNode object that will contain the topology information.

Returns

An Error indicating the success or failure of the function.

8.4.3.15 virtual Error GetUsbLinkInfo (PGRGuid *guid,* unsigned int ∗ *pValue*) [virtual]

Read usb link info for the port that the specified device is connected to.

Parameters

guid	PGRGuid of the device to read from.
pValue	Value read from the card register.

Returns

An Error indicating the success or failure of the function.

8.4.3.16 virtual Error GetUsbPortStatus (PGRGuid *guid,* unsigned int ∗ *pValue*) [virtual]

Read usb port status for the port that the specified device is connected to.

Parameters

guid	PGRGuid of the device to read from.
pValue	Value read from the card register.

Returns

An Error indicating the success or failure of the function.

8.4.3.17 Error IsCameraControlable (PGRGuid ∗ *pGuid,* bool ∗ *pControlable*)

Query CCP status on camera with corresponding PGRGuid.

This is useful to determine if a GigE camera can be controlled.

Parameters

pGuid	PGRGuid of the camera
pControlable	Indicates whether camera is controllable

Returns

An Error indicating the success or failure of the function.

8.4.3.18 virtual Error ReadPhyRegister (PGRGuid *guid,* unsigned int *page,* unsigned int *port,* unsigned int *address,* unsigned int ∗ *pValue*) [virtual]

Read a phy register on the specified device.

The full address to be read from is determined by the page, port and address.

Parameters

guid	PGRGuid of the device to read from.
page	Page to read from.
port	Port to read from.
address	Address to read from.
pValue	Value read from the phy register.

Returns

An Error indicating the success or failure of the function.

8.4.3.19 virtual Error RegisterCallback (BusEventCallback *busEventCallback*, BusCallbackType *callbackType*, void ∗ *pParameter*, CallbackHandle ∗ *pCallbackHandle*) `[virtual]`

Register a callback function that will be called when the specified callback event occurs.

Parameters

busEvent-Callback	Pointer to function that will receive the callback.
callbackType	Type of callback to register for.
pParameter	Callback parameter to be passed to callback.
pCallback-Handle	Unique callback handle used for unregistering callback.

See also

UnregisterCallback()

Returns

An Error indicating the success or failure of the function.

8.4.3.20 virtual Error RescanBus ( ) `[virtual]`

Force a rescan of the buses.

This does not trigger a bus reset. However, any current connections to a Camera object will be invalidated.

Returns

An Error indicating the success or failure of the function.

8.4.3.21 virtual Error UnregisterCallback (CallbackHandle *callbackHandle*) [virtual]

Unregister a callback function.

Parameters

callback-Handle	Unique callback handle.

See also

RegisterCallback()

Returns

An Error indicating the success or failure of the function.

8.4.3.22 virtual Error WritePhyRegister (PGRGuid *guid*, unsigned int *page*, unsigned int *port*, unsigned int *address*, unsigned int *value*) [virtual]

Write a phy register on the specified device.

The full address to be written to is determined by the page, port and address.

Parameters

guid	PGRGuid of the device to write to.
page	Page to write to.
port	Port to write to.
address	Address to write to.
value	Value to write to phy register.

Returns

An Error indicating the success or failure of the function.

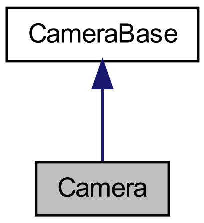
The documentation for this class was generated from the following file:

- BusManager.h

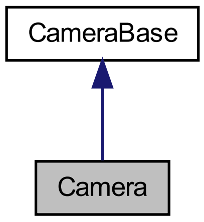
8.5 Camera Class Reference

The Camera object represents a physical camera that uses the IIDC register set.

Inheritance diagram for Camera:



Collaboration diagram for Camera:



Public Member Functions

- Camera ()

  Default constructor.
- virtual ∼Camera ()

  Default destructor.
- virtual Error Connect (PGRGuid ∗pGuid=NULL)

  The following functions are inherited from CameraBase.
- virtual Error Disconnect ()

  Disconnects the camera object from the camera.
- virtual bool IsConnected ()

  Checks if the camera object is currently connected to a physical camera.

- virtual Error SetCallback (ImageEventCallback callbackFn, const void ∗p-CallbackData=NULL)

  Sets the callback data to be used on completion of image transfer.
- virtual Error StartCapture (ImageEventCallback callbackFn=NULL, const void ∗p-CallbackData=NULL)

  Starts isochronous image capture.
- virtual Error RetrieveBuffer (Image ∗pImage)

  Retrieves the the next image object containing the next image.
- virtual Error StopCapture ()

  Stops isochronous image transfer and cleans up all associated resources.
- virtual Error WaitForBufferEvent (Image ∗pImage, unsigned int eventNumber)

  Retrieves the next image event containing the next part of the image.
- virtual Error SetUserBuffers (unsigned char ∗const pMemBuffers, int size, int numBuffers)

  Specify user allocated buffers to use as image data buffers.
- virtual Error GetConfiguration (FC2Config ∗pConfig)

  Get the configuration associated with the camera object.
- virtual Error SetConfiguration (const FC2Config ∗pConfig)

  Set the configuration associated with the camera object.
- virtual Error GetCameraInfo (CameraInfo ∗pCameraInfo)

  Retrieves information from the camera such as serial number, model name and other camera information.
- virtual Error GetPropertyInfo (PropertyInfo ∗pPropInfo)

  Retrieves information about the specified camera property.
- virtual Error GetProperty (Property ∗pProp)

  Reads the settings for the specified property from the camera.
- virtual Error SetProperty (const Property ∗pProp, bool broadcast=false)

  Writes the settings for the specified property to the camera.
- virtual Error GetGPIOPinDirection (unsigned int pin, unsigned int ∗pDirection)

  Get the GPIO pin direction for the specified pin.
- virtual Error SetGPIOPinDirection (unsigned int pin, unsigned int direction, bool broadcast=false)

  Set the GPIO pin direction for the specified pin.
- virtual Error GetTriggerModeInfo (TriggerModeInfo ∗pTriggerModeInfo)

  Retrieve trigger information from the camera.
- virtual Error GetTriggerMode (TriggerMode ∗pTriggerMode)

  Retrieve current trigger settings from the camera.
- virtual Error SetTriggerMode (const TriggerMode ∗pTriggerMode, bool broad-cast=false)

  Set the specified trigger settings to the camera.
- virtual Error FireSoftwareTrigger (bool broadcast=false)

  Fire the software trigger according to the DCAM specifications.
- virtual Error GetTriggerDelayInfo (TriggerDelayInfo ∗pTriggerDelayInfo)

  Retrieve trigger delay information from the camera.

- virtual Error GetTriggerDelay (TriggerDelay ∗pTriggerDelay)

    Retrieve current trigger delay settings from the camera.
- virtual Error SetTriggerDelay (const TriggerDelay ∗pTriggerDelay, bool broadcast=false)

    Set the specified trigger delay settings to the camera.
- virtual Error GetStrobeInfo (StrobeInfo ∗pStrobeInfo)

    Retrieve strobe information from the camera.
- virtual Error GetStrobe (StrobeControl ∗pStrobeControl)

    Retrieve current strobe settings from the camera.
- virtual Error SetStrobe (const StrobeControl ∗pStrobeControl, bool broadcast=false)

    Set current strobe settings to the camera.
- virtual Error GetLUTInfo (LUTData ∗pData)

    Query if LUT support is available on the camera.
- virtual Error GetLUTBankInfo (unsigned int bank, bool ∗pReadSupported, bool ∗pWriteSupported)

    Query the read/write status of a single LUT bank.
- virtual Error GetActiveLUTBank (unsigned int ∗pActiveBank)

    Get the LUT bank that is currently being used.
- virtual Error SetActiveLUTBank (unsigned int activeBank)

    Set the LUT bank that will be used.
- virtual Error EnableLUT (bool on)

    Enable or disable LUT functionality on the camera.
- virtual Error GetLUTChannel (unsigned int bank, unsigned int channel, unsigned int sizeEntries, unsigned int ∗pEntries)

    Get the LUT channel settings from the camera.
- virtual Error SetLUTChannel (unsigned int bank, unsigned int channel, unsigned int sizeEntries, const unsigned int ∗pEntries)

    Set the LUT channel settings to the camera.
- virtual Error GetMemoryChannel (unsigned int ∗pCurrentChannel)

    Retrieve the current memory channel from the camera.
- virtual Error SaveToMemoryChannel (unsigned int channel)

    Save the current settings to the specfied current memory channel.
- virtual Error RestoreFromMemoryChannel (unsigned int channel)

    Restore the specfied current memory channel.
- virtual Error GetMemoryChannelInfo (unsigned int ∗pNumChannels)

    Query the camera for memory channel support.
- virtual Error GetEmbeddedImageInfo (EmbeddedImageInfo ∗pInfo)

    Get the current status of the embedded image information register, as well as the availability of each embedded property.
- virtual Error SetEmbeddedImageInfo (EmbeddedImageInfo ∗pInfo)

    Sets the on/off values of the embedded image information structure to the camera.
- virtual Error WriteRegister (unsigned int address, unsigned int value, bool broadcast=false)

Write to the specified register on the camera.

- virtual Error ReadRegister (unsigned int address, unsigned int ∗pValue)

  Read the specified register from the camera.
- virtual Error WriteRegisterBlock (unsigned short addressHigh, unsigned int addressLow, const unsigned int ∗pBuffer, unsigned int length)

  Write to the specified register block on the camera.
- virtual Error ReadRegisterBlock (unsigned short addressHigh, unsigned int addressLow, unsigned int ∗pBuffer, unsigned int length)

  Read from the specified register block on the camera.
- virtual Error GetCycleTime (TimeStamp ∗timeStamp)

  Returns a Timestamp struct containing 1394 CYCLE_TIME information.
- virtual Error GetStats (CameraStats ∗pStats)
- virtual Error ResetStats ()

Static Public Member Functions

- static Error StartSyncCapture (unsigned int numCameras, const Camera ∗∗ppCameras, const ImageEventCallback ∗pCallbackFns=NULL, const void ∗∗pCallbackDataArray=NULL)
- static const char ∗ GetRegisterString (unsigned int registerVal)

  Returns a text representation of the register value.

DCAM Formats

These functions deal with DCAM video mode and frame rate on the camera.

- virtual Error GetVideoModeAndFrameRateInfo (VideoMode videoMode, FrameRate frameRate, bool ∗pSupported)

  Query the camera to determine if the specified video mode and frame rate is supported.
- virtual Error GetVideoModeAndFrameRate (VideoMode ∗pVideoMode, FrameRate ∗pFrameRate)

  Get the current video mode and frame rate from the camera.
- virtual Error SetVideoModeAndFrameRate (VideoMode videoMode, FrameRate frameRate)

  Set the specified video mode and frame rate to the camera.

Format7

These functions deal with Format7 custom image control on the camera.

- virtual Error GetFormat7Info (Format7Info ∗pInfo, bool ∗pSupported)

  Retrieve the availability of Format7 custom image mode and the camera capabilities for the specified Format7 mode.

- virtual Error ValidateFormat7Settings (const Format7ImageSettings ∗pImageSettings, bool ∗pSettingsAreValid, Format7PacketInfo ∗pPacketInfo)

  Validates Format7ImageSettings structure and returns valid packet size information if the image settings are valid.
- virtual Error GetFormat7Configuration (Format7ImageSettings ∗pImageSettings, unsigned int ∗pPacketSize, float ∗pPercentage)

  Get the current Format7 configuration from the camera.
- virtual Error SetFormat7Configuration (const Format7ImageSettings ∗pImageSettings, unsigned int packetSize)

  Set the current Format7 configuration to the camera.
- virtual Error SetFormat7Configuration (const Format7ImageSettings ∗pImageSettings, float percentSpeed)

  Set the current Format7 configuration to the camera.

8.5.1 Detailed Description

The Camera object represents a physical camera that uses the IIDC register set.

The object must first be connected to using Connect() before any other operations can proceed.

It is possible for more than 1 Camera object to connect to a single physical camera. However, isochronous transmission to more than 1 Camera object is not supported.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 Camera ( )

Default constructor.

8.5.2.2 virtual ∼Camera ( ) [virtual]

Default destructor.

8.5.3 Member Function Documentation

8.5.3.1 virtual Error Connect (PGRGuid ∗ *pGuid* = NULL) [virtual]

The following functions are inherited from CameraBase.

See CameraBase.h for further information.

Implements CameraBase.

8.5.3.2 virtual Error Disconnect ( ) `[virtual]`

Disconnects the camera object from the camera.

This allows another physical camera to be connected to the camera object.

See also

Connect()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.3 virtual Error EnableLUT (bool *on*) `[virtual]`

Enable or disable LUT functionality on the camera.

Parameters

on	Whether to enable or disable LUT.

See also

GetLUTInfo()
GetLUTChannel()
SetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.4 virtual Error FireSoftwareTrigger (bool *broadcast* = `false`) `[virtual]`

Fire the software trigger according to the DCAM specifications.

Parameters

broadcast	Whether the action should be broadcast.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.5 virtual **Error** GetActiveLUTBank (unsigned int ∗ *pActiveBank*) `[virtual]`

Get the LUT bank that is currently being used.

For cameras with PGR LUT, the active bank is always 0.

Parameters

pActiveBank	The currently active bank.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.6 virtual **Error** GetCameraInfo (**CameraInfo** ∗ *pCameraInfo*) `[virtual]`

Retrieves information from the camera such as serial number, model name and other camera information.

Parameters

pCameraInfo	Pointer to the camera information structure to be filled.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.7 virtual **Error** GetConfiguration (**FC2Config** ∗ *pConfig*) `[virtual]`

Get the configuration associated with the camera object.

Parameters

pConfig	Pointer to the configuration structure to be filled.

See also

SetConfiguration()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.8 virtual Error GetCycleTime (**TimeStamp** ∗ *timeStamp*) `[virtual]`

Returns a Timestamp struct containing 1394 CYCLE_TIME information.

Parameters

registerVal	The register value to query.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.9 virtual Error GetEmbeddedImageInfo (**EmbeddedImageInfo** ∗ *pInfo*) `[virtual]`

Get the current status of the embedded image information register, as well as the availability of each embedded property.

Parameters

pInfo	Structure to be filled.

See also

SetEmbeddedImageInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.10 virtual Error GetFormat7Configuration (**Format7ImageSettings** ∗ *pImageSettings*, unsigned int ∗ *pPacketSize*, float ∗ *pPercentage*) `[virtual]`

Get the current Format7 configuration from the camera.

This call will only succeed if the camera is already in Format7.

Parameters

pImage-Settings	Current image settings.
pPacketSize	Current packet size.
pPercentage	Current packet size as a percentage.

See also

GetFormat7Info()
ValidateFormat7Settings()
SetFormat7Configuration()
GetVideoModeAndFrameRate()

Returns

An Error indicating the success or failure of the function.

8.5.3.11 virtual Error GetFormat7Info (Format7Info ∗ *pInfo*, bool ∗ *pSupported*) [virtual]

Retrieve the availability of Format7 custom image mode and the camera capabilities for the specified Format7 mode.

The mode must be specified in the Format7Info structure in order for the function to succeed.

Parameters

pInfo	Structure to be filled with the capabilities of the specified mode and the current state in the specified mode.
pSupported	Whether the specified mode is supported.

See also

ValidateFormat7Settings()
GetFormat7Configuration()
SetFormat7Configuration()

Returns

An Error indicating the success or failure of the function.

8.5.3.12 virtual Error GetGPIOPinDirection (unsigned int *pin*, unsigned int ∗ *pDirection*) [virtual]

Get the GPIO pin direction for the specified pin.

This is not a required call when using the trigger or strobe functions as the pin direction is set automatically internally.

Parameters

pin	Pin to get the direction for.
pDirection	Direction of the pin. 0 for input, 1 for output.

See also

SetGPIOPinDirection()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.13 virtual Error GetLUTBankInfo (unsigned int *bank*, bool ∗ *pReadSupported*, bool ∗ *pWriteSupported*) `[virtual]`

Query the read/write status of a single LUT bank.

Parameters

bank	The bank to query.
pRead-Supported	Whether reading from the bank is supported.
pWrite-Supported	Whether writing to the bank is supported.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.14 virtual Error GetLUTChannel (unsigned int *bank*, unsigned int *channel*, unsigned int *sizeEntries*, unsigned int ∗ *pEntries*) `[virtual]`

Get the LUT channel settings from the camera.

Parameters

bank	Bank to retrieve.
channel	Channel to retrieve.
sizeEntries	Number of entries in LUT table to read.
pEntries	Array to store LUT entries.

See also

GetLUTInfo()
EnableLUT()
SetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.15 virtual Error GetLUTInfo (LUTData ∗ *pData*) [virtual]

Query if LUT support is available on the camera.

Note that some cameras may report support for the LUT and return an inputBitDepth of 0. In these cases use log2(numEntries) for the inputBitDepth.

Parameters

pData	The LUT structure to be filled.

See also

EnableLUT()
GetLUTChannel()
SetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.16 virtual Error GetMemoryChannel (unsigned int ∗ *pCurrentChannel*) [virtual]

Retrieve the current memory channel from the camera.

Parameters

pCurrent-Channel	Current memory channel.

See also

SaveToMemoryChannel()
RestoreFromMemoryChannel()
GetMemoryChannelInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.17 virtual Error GetMemoryChannelInfo (unsigned int ∗ *pNumChannels*) `[virtual]`

Query the camera for memory channel support.

If the number of channels is 0, then memory channel support is not available.

Parameters

pNum-Channels	Number of memory channels supported.

See also

GetMemoryChannel()
SaveToMemoryChannel()
RestoreFromMemoryChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.18 virtual Error GetProperty (Property ∗ *pProp*) `[virtual]`

Reads the settings for the specified property from the camera.

The property type must be specified in the Property structure passed into the function in order for the function to succeed. If auto is on, the integer and abs values returned may not be consistent with each other.

Parameters

pProp	Pointer to the Property structure to be filled.

See also

GetPropertyInfo()
SetProperty()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.19 virtual Error GetPropertyInfo (PropertyInfo ∗ *pPropInfo*) [virtual]

Retrieves information about the specified camera property.

The property type must be specified in the PropertyInfo structure passed into the function in order for the function to succeed.

Parameters

pPropInfo	Pointer to the PropertyInfo structure to be filled.

See also

GetProperty()
SetProperty()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.20 static const char∗ GetRegisterString (unsigned int *registerVal*) [static]

Returns a text representation of the register value.

Parameters

registerVal	The register value to query.

Returns

The text representation of the register.

Reimplemented from CameraBase.

8.5.3.21 virtual Error GetStats (CameraStats ∗ *pStats*) [virtual]

Implements CameraBase.

8.5.3.22 virtual Error GetStrobe (StrobeControl ∗ *pStrobeControl*) [virtual]

Retrieve current strobe settings from the camera.

The strobe pin must be specified in the structure before being passed in to the function.

Parameters

pStrobeControl	Structure to receive strobe settings.

See also

GetStrobeInfo()
SetStrobe()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.23 virtual Error GetStrobeInfo (StrobeInfo ∗ *pStrobeInfo*) [virtual]

Retrieve strobe information from the camera.

Parameters

pStrobeInfo	Structure to receive strobe information.

See also

GetStrobe()
SetStrobe()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.24 virtual Error GetTriggerDelay (TriggerDelay ∗ *pTriggerDelay*) [virtual]

Retrieve current trigger delay settings from the camera.

Parameters

pTriggerDelay	Structure to receive trigger delay settings.

See also

GetTriggerModeInfo()
GetTriggerMode()
SetTriggerMode()
GetTriggerDelayInfo()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.25 virtual Error GetTriggerDelayInfo (TriggerDelayInfo ∗ *pTriggerDelayInfo*) [virtual]

Retrieve trigger delay information from the camera.

Parameters

pTrigger-DelayInfo	Structure to receive trigger delay information.

See also

GetTriggerModeInfo()
GetTriggerMode()
SetTriggerMode()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.26 virtual Error GetTriggerMode (TriggerMode ∗ *pTriggerMode*) [virtual]

Retrieve current trigger settings from the camera.

Parameters

pTrigger-Mode	Structure to receive trigger mode settings.

See also

GetTriggerModeInfo()
SetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.27 virtual Error GetTriggerModeInfo (TriggerModeInfo ∗ *pTriggerModeInfo*) [virtual]

Retrieve trigger information from the camera.

Parameters

pTrigger-ModeInfo	Structure to receive trigger information.

See also

GetTriggerMode()
SetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.28 virtual Error GetVideoModeAndFrameRate (VideoMode ∗ *pVideoMode*, FrameRate ∗ *pFrameRate*) [virtual]

Get the current video mode and frame rate from the camera.

If the camera is in Format7, the video mode will be VIDEOMODE_FORMAT7 and the frame rate will be FRAMERATE_FORMAT7.

Parameters

pVideoMode	Current video mode.
pFrameRate	Current frame rate.

See also

GetVideoModeAndFrameRateInfo()
SetVideoModeAndFrameRate()

Returns

An Error indicating the success or failure of the function.

8.5.3.29 virtual Error GetVideoModeAndFrameRateInfo (VideoMode *videoMode*, FrameRate *frameRate*, bool ∗ *pSupported*) [virtual]

Query the camera to determine if the specified video mode and frame rate is supported.

Parameters

videoMode	Video mode to check.
frameRate	Frame rate to check.
pSupported	Whether the video mode and frame rate is supported.

See also

GetVideoModeAndFrameRate()
SetVideoModeAndFrameRate()

Returns

An Error indicating the success or failure of the function.

8.5.3.30 virtual bool IsConnected ( ) [virtual]

Checks if the camera object is currently connected to a physical camera.

See also

Connect()
Disconnect()

Returns

Whether the camera object is connected to a physical camera.

Implements CameraBase.

8.5.3.31 virtual **Error** ReadRegister (unsigned int *address*, unsigned int ∗ *pValue*) `[virtual]`

Read the specified register from the camera.

Parameters

address	DCAM address to be read from.
pValue	The value that is read.

See also

WriteRegister()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.32 virtual **Error** ReadRegisterBlock (unsigned short *addressHigh*, unsigned int *addressLow*, unsigned int ∗ *pBuffer*, unsigned int *length*) `[virtual]`

Read from the specified register block on the camera.

Parameters

addressHigh	Top 16 bits of the 48 bit absolute address to read from.
addressLow	Bottom 32 bits of the 48 bits absolute address to read from.
pBuffer	Array to store read data.
length	Size of array, in quadlets.

See also

WriteRegisterBlock()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.33 virtual **Error** ResetStats ( ) `[virtual]`

Implements CameraBase.

8.5.3.34 virtual Error RestoreFromMemoryChannel (unsigned int *channel*) [virtual]

Restore the specfied current memory channel.

Parameters

channel	Memory channel to restore from.

See also

GetMemoryChannel()
SaveToMemoryChannel()
GetMemoryChannelInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.35 virtual Error RetrieveBuffer (Image ∗ *pImage*) [virtual]

Retrieves the the next image object containing the next image.

If the grab mode has not been set, or has been set to DROP_FRAMES the default behavior is to requeue images for DMA if they have not been retrieved by the time the next image transfer completes. If BUFFER_FRAMES is specified, the next image in the sequence will be retrieved. Note that for the BUFFER_FRAMES case, if retrieval does not keep up with the DMA process, images will be lost. The default behavior is to perform DROP_FRAMES image retrieval.

Parameters

pImage	Pointer to Image object to store image data.

See also

StartCapture()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.36 virtual Error SaveToMemoryChannel (unsigned int *channel*) [virtual]

Save the current settings to the specfied current memory channel.

Parameters

channel	Memory channel to save to.

See also

GetMemoryChannel()
RestoreFromMemoryChannel()
GetMemoryChannelInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.37 virtual **Error** SetActiveLUTBank (unsigned int *activeBank*) `[virtual]`

Set the LUT bank that will be used.

Parameters

activeBank	The bank to be set as active.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.38 virtual **Error** SetCallback (**ImageEventCallback** *callbackFn*, const void * *pCallbackData* = `NULL`) `[virtual]`

Sets the callback data to be used on completion of image transfer.

To clear the current stored callback data, pass in NULL for both arguments.

Parameters

callbackFn	A function to be called when a new image is received.
pCallback-Data	A pointer to data that can be passed to the callback function.

See also

StartCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.39 virtual Error SetConfiguration (const FC2Config * *pConfig*) [virtual]

Set the configuration associated with the camera object.

Parameters

pConfig	Pointer to the configuration structure to be used.

See also

GetConfiguration()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.40 virtual Error SetEmbeddedImageInfo (EmbeddedImageInfo * *pInfo*) [virtual]

Sets the on/off values of the embedded image information structure to the camera.

Parameters

pInfo	Structure to be used.

See also

GetEmbeddedImageInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.41 virtual Error SetFormat7Configuration (const Format7ImageSettings * *pImageSettings*, unsigned int *packetSize*) [virtual]

Set the current Format7 configuration to the camera.

Parameters

pImage-Settings	Image settings to be written to the camera.
packetSize	Packet size to be written to the camera.

See also

GetFormat7Info()
ValidateFormat7Settings()
GetFormat7Configuration()

Returns

An Error indicating the success or failure of the function.

8.5.3.42 virtual Error SetFormat7Configuration (const Format7ImageSettings * *pImageSettings*, float *percentSpeed*) [virtual]

Set the current Format7 configuration to the camera.

Parameters

pImage-Settings	Image settings to be written to the camera.
percent-Speed	Percentage of packet size to be written to the camera.

See also

GetFormat7Info()
ValidateFormat7Settings()
GetFormat7Configuration()

Returns

An Error indicating the success or failure of the function.

8.5.3.43 virtual Error SetGPIOPinDirection (unsigned int *pin*, unsigned int *direction*, bool *broadcast* = `false`) [virtual]

Set the GPIO pin direction for the specified pin.

This is useful if there is a need to set the pin into an input pin (i.e. to read the voltage) off the pin without setting it as a trigger source. This is not a required call when using the trigger or strobe functions as the pin direction is set automatically internally.

Parameters

pin	Pin to get the direction for.
direction	Direction of the pin. 0 for input, 1 for output.
broadcast	Whether the action should be broadcast.

See also

GetGPIOPinDirection()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.44 virtual Error SetLUTChannel (unsigned int *bank,* unsigned int *channel,* unsigned int *sizeEntries,* const unsigned int ∗ *pEntries*) [virtual]

Set the LUT channel settings to the camera.

Parameters

bank	Bank to set.
channel	Channel to set.
sizeEntries	Number of entries in LUT table to write. This must be the same size as numEntries returned by GetLutInfo().
pEntries	Array containing LUT entries to write.

See also

GetLUTInfo()
EnableLUT()
GetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.45 virtual Error SetProperty (const Property ∗ *pProp,* bool *broadcast* = false) [virtual]

Writes the settings for the specified property to the camera.

The property type must be specified in the Property structure passed into the function in order for the function to succeed. The absControl flag controls whether the absolute or integer value is written to the camera.

Parameters

pProp	Pointer to the Property structure to be used.
broadcast	Whether the action should be broadcast.

See also

GetPropertyInfo()
GetProperty()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.46 virtual Error SetStrobe (const StrobeControl ∗ *pStrobeControl*, bool *broadcast* = `false`) `[virtual]`

Set current strobe settings to the camera.

The strobe pin must be specified in the structure before being passed in to the function.

Parameters

pStrobe-Control	Structure providing strobe settings.
broadcast	Whether the action should be broadcast.

See also

GetStrobeInfo()
GetStrobe()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.47 virtual Error SetTriggerDelay (const TriggerDelay ∗ *pTriggerDelay*, bool *broadcast* = `false`) `[virtual]`

Set the specified trigger delay settings to the camera.

Parameters

pTrigger-Delay	Structure providing trigger delay settings.
broadcast	Whether the action should be broadcast.

See also

GetTriggerModeInfo()
GetTriggerMode()
SetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.48 virtual **Error** SetTriggerMode (const **TriggerMode** ∗ *pTriggerMode*, bool *broadcast* = `false`) `[virtual]`

Set the specified trigger settings to the camera.

Parameters

pTrigger-Mode	Structure providing trigger mode settings.
broadcast	Whether the action should be broadcast.

See also

GetTriggerModeInfo()
GetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.49 virtual **Error** SetUserBuffers (unsigned char ∗const *pMemBuffers*, int *size*, int *numBuffers*) `[virtual]`

Specify user allocated buffers to use as image data buffers.

To prevent image tearing, the size of each buffer should be equal to ((unsigned int)(bufferSize + packetSize - 1)/packetSize) ∗ packetSize. The total size should be (size ∗ numBuffers) or larger.

Parameters

pMem-Buffers	Pointer to memory buffers to be written to.
size	The size of each buffer (in bytes).
numBuffers	Number of buffers in the array.

See also

StartCapture()
RetrieveBuffer()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.50 virtual Error SetVideoModeAndFrameRate (VideoMode *videoMode,* FrameRate *frameRate*) [virtual]

Set the specified video mode and frame rate to the camera.

It is not possible to set the camera to VIDEOMODE_FORMAT7 or FRAMERATE_FO-RMAT7. Use the Format7 functions to set the camera into Format7.

Parameters

videoMode	Video mode to set to camera.
frameRate	Frame rate to set to camera.

See also

GetVideoModeAndFrameRateInfo()
GetVideoModeAndFrameRate()

Returns

An Error indicating the success or failure of the function.

8.5.3.51 virtual Error StartCapture (ImageEventCallback *callbackFn* = NULL, const void ∗ *pCallbackData* = NULL) [virtual]

Starts isochronous image capture.

It will use either the current video mode or the most recently set video mode of the camera. The optional callback function parameter is called on completion of image transfer. Alternatively, the callback parameter can be set to NULL and RetrieveBuffer() can be called as a blocking call to get the image data.

Parameters

callbackFn	A function to be called when a new image is received.
pCallback-Data	A pointer to data that can be passed to the callback function.

See also

RetrieveBuffer()
StartSyncCapture()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.52 static Error StartSyncCapture (unsigned int *numCameras*, const Camera ∗∗ *ppCameras*, const ImageEventCallback ∗ *pCallbackFns* = `NULL`, const void ∗∗ *pCallbackDataArray* = `NULL`) `[static]`

8.5.3.53 virtual Error StopCapture ( ) `[virtual]`

Stops isochronous image transfer and cleans up all associated resources.

If an image callback function (specified in the StartCapture() call) is currently executing, StopCapture() will not return until after the callback has completed.

See also

StartCapture()
RetrieveBuffer()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.54 virtual Error ValidateFormat7Settings (const Format7ImageSettings ∗ *pImageSettings*, bool ∗ *pSettingsAreValid*, Format7PacketInfo ∗ *pPacketInfo*) `[virtual]`

Validates Format7ImageSettings structure and returns valid packet size information if the image settings are valid.

The current image settings are cached while validation is taking place. The cached settings are restored when validation is complete.

Parameters

pImage-Settings	Structure containing the image settings.
pSettings-AreValid	Whether the settings are valid.
pPacketInfo	Packet size information that can be used to determine a valid packet size.

See also

GetFormat7Info()
GetFormat7Configuration()
SetFormat7Configuration()

Returns

An Error indicating the success or failure of the function.

8.5.3.55 virtual Error WaitForBufferEvent (Image ∗ *pImage*, unsigned int *eventNumber*) [virtual]

Retrieves the next image event containing the next part of the image.

Parameters

pImage	Pointer to Image object to store image data.
event-Number	The event number to wait for.

See also

StartCapture()
RetrieveBuffer()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.56 virtual Error WriteRegister (unsigned int *address*, unsigned int *value*, bool *broadcast* = false) [virtual]

Write to the specified register on the camera.

Parameters

address	DCAM address to be written to.
value	The value to be written.
broadcast	Whether the action should be broadcast.

See also

ReadRegister()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.5.3.57 virtual Error WriteRegisterBlock (unsigned short *addressHigh,* unsigned int *addressLow,* const unsigned int * *pBuffer,* unsigned int *length*) [virtual]

Write to the specified register block on the camera.

Parameters

addressHigh	Top 16 bits of the 48 bit absolute address to write to.
addressLow	Bottom 32 bits of the 48 bits absolute address to write to.
pBuffer	Array containing data to be written.
length	Size of array, in quadlets.

See also

ReadRegisterBlock()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

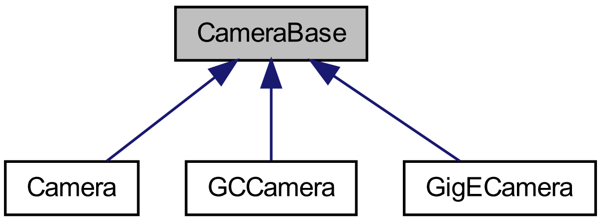
The documentation for this class was generated from the following file:

- Camera.h

8.6 CameraBase Class Reference

The CameraBase class is an abstract base class that defines a general interface to a camera.

Inheritance diagram for CameraBase:



Public Member Functions

- CameraBase ()

  Default constructor.
- virtual ∼CameraBase ()

  Default destructor.

Protected Attributes

- CameraData ∗ m_pCameraData

Connection and Image Retrieval

These functions deal with connections and image retrieval from the camera.

- virtual Error Connect (PGRGuid ∗pGuid=NULL)=0

  Connects the camera object to the camera specified by the GUID.
- virtual Error Disconnect ()=0

  Disconnects the camera object from the camera.
- virtual bool IsConnected ()=0

  Checks if the camera object is currently connected to a physical camera.
- virtual Error SetCallback (ImageEventCallback callbackFn, const void ∗p-CallbackData=NULL)=0

  Sets the callback data to be used on completion of image transfer.
- virtual Error StartCapture (ImageEventCallback callbackFn=NULL, const void ∗p-CallbackData=NULL)=0

  Starts isochronous image capture.

- virtual Error RetrieveBuffer (Image ∗pImage)=0

    Retrieves the the next image object containing the next image.
- virtual Error StopCapture ()=0

    Stops isochronous image transfer and cleans up all associated resources.
- virtual Error WaitForBufferEvent (Image ∗pImage, unsigned int eventNumber)=0

    Retrieves the next image event containing the next part of the image.
- virtual Error SetUserBuffers (unsigned char ∗const pMemBuffers, int size, int numBuffers)=0

    Specify user allocated buffers to use as image data buffers.
- virtual Error GetConfiguration (FC2Config ∗pConfig)=0

    Get the configuration associated with the camera object.
- virtual Error SetConfiguration (const FC2Config ∗pConfig)=0

    Set the configuration associated with the camera object.
- static Error StartSyncCapture (unsigned int numCameras, const CameraBase ∗∗ppCameras, const ImageEventCallback ∗pCallbackFns=NULL, const void ∗∗pCallbackDataArray=NULL)

    Starts isochronous image capture on multiple cameras.

Information and Properties

These functions deal with information and properties can be retrieved from the camera.

- virtual Error GetCameraInfo (CameraInfo ∗pCameraInfo)=0

    Retrieves information from the camera such as serial number, model name and other camera information.
- virtual Error GetPropertyInfo (PropertyInfo ∗pPropInfo)=0

    Retrieves information about the specified camera property.
- virtual Error GetProperty (Property ∗pProp)=0

    Reads the settings for the specified property from the camera.
- virtual Error SetProperty (const Property ∗pProp, bool broadcast=false)=0

    Writes the settings for the specified property to the camera.

General Purpose Input / Output

These functions deal with general GPIO pin control on the camera.

- virtual Error GetGPIOPinDirection (unsigned int pin, unsigned int ∗pDirection)=0

    Get the GPIO pin direction for the specified pin.
- virtual Error SetGPIOPinDirection (unsigned int pin, unsigned int direction, bool broadcast=false)=0

    Set the GPIO pin direction for the specified pin.

Trigger

These functions deal with trigger control on the camera.

- virtual Error GetTriggerModeInfo (TriggerModeInfo ∗pTriggerModeInfo)=0

  Retrieve trigger information from the camera.
- virtual Error GetTriggerMode (TriggerMode ∗pTriggerMode)=0

  Retrieve current trigger settings from the camera.
- virtual Error SetTriggerMode (const TriggerMode ∗pTriggerMode, bool broadcast=false)=0

  Set the specified trigger settings to the camera.
- virtual Error FireSoftwareTrigger (bool broadcast=false)=0

  Fire the software trigger according to the DCAM specifications.
- virtual Error GetTriggerDelayInfo (TriggerDelayInfo ∗pTriggerDelayInfo)=0

  Retrieve trigger delay information from the camera.
- virtual Error GetTriggerDelay (TriggerDelay ∗pTriggerDelay)=0

  Retrieve current trigger delay settings from the camera.
- virtual Error SetTriggerDelay (const TriggerDelay ∗pTriggerDelay, bool broadcast=false)=0

  Set the specified trigger delay settings to the camera.

Strobe

These functions deal with strobe control on the camera.

- virtual Error GetStrobeInfo (StrobeInfo ∗pStrobeInfo)=0

  Retrieve strobe information from the camera.
- virtual Error GetStrobe (StrobeControl ∗pStrobeControl)=0

  Retrieve current strobe settings from the camera.
- virtual Error SetStrobe (const StrobeControl ∗pStrobeControl, bool broadcast=false)=0

  Set current strobe settings to the camera.

Look Up Table

These functions deal with Look Up Table control on the camera.

- virtual Error GetLUTInfo (LUTData ∗pData)=0

  Query if LUT support is available on the camera.
- virtual Error GetLUTBankInfo (unsigned int bank, bool ∗pReadSupported, bool ∗pWriteSupported)=0

  Query the read/write status of a single LUT bank.
- virtual Error GetActiveLUTBank (unsigned int ∗pActiveBank)=0

  Get the LUT bank that is currently being used.

- virtual Error SetActiveLUTBank (unsigned int activeBank)=0

  Set the LUT bank that will be used.
- virtual Error EnableLUT (bool on)=0

  Enable or disable LUT functionality on the camera.
- virtual Error GetLUTChannel (unsigned int bank, unsigned int channel, unsigned int sizeEntries, unsigned int ∗pEntries)=0

  Get the LUT channel settings from the camera.
- virtual Error SetLUTChannel (unsigned int bank, unsigned int channel, unsigned int sizeEntries, const unsigned int ∗pEntries)=0

  Set the LUT channel settings to the camera.

Memory Channels

These functions deal with memory channel control on the camera.

- virtual Error GetMemoryChannel (unsigned int ∗pCurrentChannel)=0

  Retrieve the current memory channel from the camera.
- virtual Error SaveToMemoryChannel (unsigned int channel)=0

  Save the current settings to the specfied current memory channel.
- virtual Error RestoreFromMemoryChannel (unsigned int channel)=0

  Restore the specfied current memory channel.
- virtual Error GetMemoryChannelInfo (unsigned int ∗pNumChannels)=0

  Query the camera for memory channel support.

Embedded Image Information

These functions deal with embedded image information control on the camera.

- virtual Error GetEmbeddedImageInfo (EmbeddedImageInfo ∗pInfo)=0

  Get the current status of the embedded image information register, as well as the availability of each embedded property.
- virtual Error SetEmbeddedImageInfo (EmbeddedImageInfo ∗pInfo)=0

  Sets the on/off values of the embedded image information structure to the camera.

Register Operation

These functions deal with register operation on the camera.

- virtual Error WriteRegister (unsigned int address, unsigned int value, bool broadcast=false)=0

  Write to the specified register on the camera.
- virtual Error ReadRegister (unsigned int address, unsigned int ∗pValue)=0

  Read the specified register from the camera.

- virtual Error WriteRegisterBlock (unsigned short addressHigh, unsigned int addressLow, const unsigned int ∗pBuffer, unsigned int length)=0

  Write to the specified register block on the camera.
- virtual Error ReadRegisterBlock (unsigned short addressHigh, unsigned int addressLow, unsigned int ∗pBuffer, unsigned int length)=0

  Read from the specified register block on the camera.
- virtual Error GetCycleTime (TimeStamp ∗timeStamp)=0

  Returns a Timestamp struct containing 1394 CYCLE_TIME information.
- virtual Error GetStats (CameraStats ∗pStats)=0
- virtual Error ResetStats ()=0
- static const char ∗ GetRegisterString (unsigned int registerVal)

  Returns a text representation of the register value.

8.6.1 Detailed Description

The CameraBase class is an abstract base class that defines a general interface to a camera.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 CameraBase ( ) `[inline]`

Default constructor.

8.6.2.2 virtual ∼CameraBase ( ) `[inline, virtual]`

Default destructor.

8.6.3 Member Function Documentation

8.6.3.1 virtual Error Connect (PGRGuid ∗ *pGuid* = `NULL`) `[pure virtual]`

Connects the camera object to the camera specified by the GUID.

If the guid is omitted or set to NULL, the connection will be made to the first camera detected on the PC (i.e. index = 0).

Parameters

pGuid	The unique identifier for a specific camera on the PC.

See also

BusManager::GetCameraFromIndex()
BusManager::GetCameraFromSerialNumber()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.2 virtual Error Disconnect ( ) `[pure virtual]`

Disconnects the camera object from the camera.

This allows another physical camera to be connected to the camera object.

See also

Connect()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.3 virtual Error EnableLUT (bool *on*) `[pure virtual]`

Enable or disable LUT functionality on the camera.

Parameters

on	Whether to enable or disable LUT.

See also

GetLUTInfo()
GetLUTChannel()
SetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.4 virtual Error FireSoftwareTrigger (bool *broadcast* = `false`) `[pure virtual]`

Fire the software trigger according to the DCAM specifications.

Parameters

broadcast	Whether the action should be broadcast.

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.5 virtual **Error** GetActiveLUTBank (unsigned int ∗ *pActiveBank*) `[pure virtual]`

Get the LUT bank that is currently being used.

For cameras with PGR LUT, the active bank is always 0.

Parameters

pActiveBank	The currently active bank.

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.6 virtual **Error** GetCameraInfo (**CameraInfo** ∗ *pCameraInfo*) `[pure virtual]`

Retrieves information from the camera such as serial number, model name and other camera information.

Parameters

pCameraInfo	Pointer to the camera information structure to be filled.

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.7 virtual **Error** GetConfiguration (**FC2Config** ∗ *pConfig*) `[pure virtual]`

Get the configuration associated with the camera object.

Parameters

pConfig	Pointer to the configuration structure to be filled.

See also

SetConfiguration()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.8 virtual Error GetCycleTime (TimeStamp ∗ *timeStamp*) `[pure virtual]`

Returns a Timestamp struct containing 1394 CYCLE_TIME information.

Parameters

registerVal	The register value to query.

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.9 virtual Error GetEmbeddedImageInfo (EmbeddedImageInfo ∗ *pInfo*) `[pure virtual]`

Get the current status of the embedded image information register, as well as the availability of each embedded property.

Parameters

pInfo	Structure to be filled.

See also

SetEmbeddedImageInfo()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.10 virtual Error GetGPIOPinDirection (unsigned int *pin*, unsigned int ∗ *pDirection*) `[pure virtual]`

Get the GPIO pin direction for the specified pin.

This is not a required call when using the trigger or strobe functions as the pin direction is set automatically internally.

Parameters

pin	Pin to get the direction for.
pDirection	Direction of the pin. 0 for input, 1 for output.

See also

SetGPIOPinDirection()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.11 virtual Error GetLUTBankInfo (unsigned int *bank*, bool ∗ *pReadSupported*, bool ∗ *pWriteSupported*) `[pure virtual]`

Query the read/write status of a single LUT bank.

Parameters

bank	The bank to query.
pRead-Supported	Whether reading from the bank is supported.
pWrite-Supported	Whether writing to the bank is supported.

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.12 virtual Error GetLUTChannel (unsigned int *bank*, unsigned int *channel*, unsigned int *sizeEntries*, unsigned int ∗ *pEntries*) `[pure virtual]`

Get the LUT channel settings from the camera.

Parameters

bank	Bank to retrieve.
channel	Channel to retrieve.
sizeEntries	Number of entries in LUT table to read.
pEntries	Array to store LUT entries.

See also

GetLUTInfo()
EnableLUT()
SetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.13 virtual Error GetLUTInfo (LUTData ∗ *pData*) `[pure virtual]`

Query if LUT support is available on the camera.

Note that some cameras may report support for the LUT and return an inputBitDepth of 0. In these cases use log2(numEntries) for the inputBitDepth.

Parameters

pData	The LUT structure to be filled.

See also

EnableLUT()
GetLUTChannel()
SetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.14 virtual Error GetMemoryChannel (unsigned int ∗ *pCurrentChannel*) `[pure virtual]`

Retrieve the current memory channel from the camera.

Parameters

pCurrent-Channel	Current memory channel.

See also

SaveToMemoryChannel()
RestoreFromMemoryChannel()
GetMemoryChannelInfo()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.15 virtual Error GetMemoryChannelInfo (unsigned int ∗ *pNumChannels*) `[pure virtual]`

Query the camera for memory channel support.

If the number of channels is 0, then memory channel support is not available.

Parameters

pNum-Channels	Number of memory channels supported.

See also

GetMemoryChannel()
SaveToMemoryChannel()
RestoreFromMemoryChannel()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.16 virtual Error GetProperty (Property ∗ *pProp*) `[pure virtual]`

Reads the settings for the specified property from the camera.

The property type must be specified in the Property structure passed into the function in order for the function to succeed. If auto is on, the integer and abs values returned may not be consistent with each other.

Parameters

pProp	Pointer to the Property structure to be filled.

See also

GetPropertyInfo()
SetProperty()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.17 virtual Error GetPropertyInfo (PropertyInfo ∗ *pPropInfo*) [pure virtual]

Retrieves information about the specified camera property.

The property type must be specified in the PropertyInfo structure passed into the function in order for the function to succeed.

Parameters

pPropInfo	Pointer to the PropertyInfo structure to be filled.

See also

GetProperty()
SetProperty()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.18 static const char∗ GetRegisterString (unsigned int *registerVal*) [static]

Returns a text representation of the register value.

Parameters

registerVal	The register value to query.

Returns

The text representation of the register.

Reimplemented in GigECamera, Camera, and GCCamera.

8.6.3.19 virtual Error GetStats (CameraStats ∗ *pStats*) [pure virtual]

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.20 virtual Error GetStrobe (StrobeControl ∗ *pStrobeControl*) [pure virtual]

Retrieve current strobe settings from the camera.

The strobe pin must be specified in the structure before being passed in to the function.

Parameters

pStrobe-Control	Structure to receive strobe settings.

See also

GetStrobeInfo()
SetStrobe()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.21 virtual Error GetStrobeInfo (StrobeInfo ∗ *pStrobeInfo*) [pure virtual]

Retrieve strobe information from the camera.

Parameters

pStrobeInfo	Structure to receive strobe information.

See also

GetStrobe()
SetStrobe()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.22 virtual Error GetTriggerDelay (TriggerDelay * *pTriggerDelay*) [pure virtual]

Retrieve current trigger delay settings from the camera.

Parameters

pTrigger-Delay	Structure to receive trigger delay settings.

See also

GetTriggerModeInfo()
GetTriggerMode()
SetTriggerMode()
GetTriggerDelayInfo()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.23 virtual Error GetTriggerDelayInfo (TriggerDelayInfo * *pTriggerDelayInfo*) [pure virtual]

Retrieve trigger delay information from the camera.

Parameters

pTrigger-DelayInfo	Structure to receive trigger delay information.

See also

GetTriggerModeInfo()
GetTriggerMode()
SetTriggerMode()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.24 virtual Error GetTriggerMode (TriggerMode ∗ *pTriggerMode*) `[pure virtual]`

Retrieve current trigger settings from the camera.

Parameters

pTrigger-Mode	Structure to receive trigger mode settings.

See also

GetTriggerModeInfo()
SetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.25 virtual Error GetTriggerModeInfo (TriggerModeInfo ∗ *pTriggerModeInfo*) `[pure virtual]`

Retrieve trigger information from the camera.

Parameters

pTrigger-ModeInfo	Structure to receive trigger information.

See also

GetTriggerMode()
SetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.26 virtual bool IsConnected ( ) `[pure virtual]`

Checks if the camera object is currently connected to a physical camera.

See also

Connect()
Disconnect()

Returns

Whether the camera object is connected to a physical camera.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.27 virtual Error ReadRegister (unsigned int *address*, unsigned int * *pValue*) `[pure virtual]`

Read the specified register from the camera.

Parameters

address	DCAM address to be read from.
pValue	The value that is read.

See also

WriteRegister()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.28 virtual Error ReadRegisterBlock (unsigned short *addressHigh*, unsigned int *addressLow*, unsigned int * *pBuffer*, unsigned int *length*) `[pure virtual]`

Read from the specified register block on the camera.

Parameters

addressHigh	Top 16 bits of the 48 bit absolute address to read from.
addressLow	Bottom 32 bits of the 48 bits absolute address to read from.
pBuffer	Array to store read data.
length	Size of array, in quadlets.

See also

WriteRegisterBlock()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.29 virtual Error ResetStats ( ) `[pure virtual]`

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.30 virtual Error RestoreFromMemoryChannel (unsigned int *channel*) `[pure virtual]`

Restore the specfied current memory channel.

Parameters

channel	Memory channel to restore from.

See also

GetMemoryChannel()
SaveToMemoryChannel()
GetMemoryChannelInfo()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.31 virtual Error RetrieveBuffer (Image ∗ *pImage*) `[pure virtual]`

Retrieves the the next image object containing the next image.

If the grab mode has not been set, or has been set to DROP_FRAMES the default behavior is to requeue images for DMA if they have not been retrieved by the time the next image transfer completes. If BUFFER_FRAMES is specified, the next image in the sequence will be retrieved. Note that for the BUFFER_FRAMES case, if retrieval does not keep up with the DMA process, images will be lost. The default behavior is to perform DROP_FRAMES image retrieval.

Parameters

pImage	Pointer to Image object to store image data.

See also

StartCapture()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.32 virtual Error SaveToMemoryChannel (unsigned int *channel*) `[pure virtual]`

Save the current settings to the specfied current memory channel.

Parameters

channel	Memory channel to save to.

See also

GetMemoryChannel()
RestoreFromMemoryChannel()
GetMemoryChannelInfo()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.33 virtual Error SetActiveLUTBank (unsigned int *activeBank*) `[pure virtual]`

Set the LUT bank that will be used.

Parameters

activeBank	The bank to be set as active.

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.34 virtual Error SetCallback (ImageEventCallback *callbackFn*, const void ∗ *pCallbackData* = NULL) [pure virtual]

Sets the callback data to be used on completion of image transfer.

To clear the current stored callback data, pass in NULL for both arguments.

Parameters

callbackFn	A function to be called when a new image is received.
pCallback-Data	A pointer to data that can be passed to the callback function.

See also

StartCapture()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.35 virtual Error SetConfiguration (const FC2Config ∗ *pConfig*) [pure virtual]

Set the configuration associated with the camera object.

Parameters

pConfig	Pointer to the configuration structure to be used.

See also

GetConfiguration()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.36 virtual Error SetEmbeddedImageInfo (EmbeddedImageInfo ∗ *pInfo*) [pure virtual]

Sets the on/off values of the embedded image information structure to the camera.

Parameters

pInfo	Structure to be used.

See also

GetEmbeddedImageInfo()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.37 virtual Error SetGPIOPinDirection (unsigned int *pin*, unsigned int *direction*, bool *broadcast* = false) [pure virtual]

Set the GPIO pin direction for the specified pin.

This is useful if there is a need to set the pin into an input pin (i.e. to read the voltage) off the pin without setting it as a trigger source. This is not a required call when using the trigger or strobe functions as the pin direction is set automatically internally.

Parameters

pin	Pin to get the direction for.
direction	Direction of the pin. 0 for input, 1 for output.
broadcast	Whether the action should be broadcast.

See also

GetGPIOPinDirection()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.38 virtual Error SetLUTChannel (unsigned int *bank*, unsigned int *channel*, unsigned int *sizeEntries*, const unsigned int ∗ *pEntries*) [pure virtual]

Set the LUT channel settings to the camera.

Parameters

bank	Bank to set.
channel	Channel to set.
sizeEntries	Number of entries in LUT table to write. This must be the same size as numEntries returned by GetLutInfo().
pEntries	Array containing LUT entries to write.

See also

GetLUTInfo()
EnableLUT()
GetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.39 virtual Error SetProperty (const Property * *pProp*, bool *broadcast* = `false`) `[pure virtual]`

Writes the settings for the specified property to the camera.

The property type must be specified in the Property structure passed into the function in order for the function to succeed. The absControl flag controls whether the absolute or integer value is written to the camera.

Parameters

pProp	Pointer to the Property structure to be used.
broadcast	Whether the action should be broadcast.

See also

GetPropertyInfo()
GetProperty()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.40 virtual Error SetStrobe (const StrobeControl * *pStrobeControl*, bool *broadcast* = `false`) `[pure virtual]`

Set current strobe settings to the camera.

The strobe pin must be specified in the structure before being passed in to the function.

Parameters

pStrobe-Control	Structure providing strobe settings.
broadcast	Whether the action should be broadcast.

See also

GetStrobeInfo()
GetStrobe()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.41 virtual Error SetTriggerDelay (const TriggerDelay * *pTriggerDelay*, bool *broadcast* = false) [pure virtual]

Set the specified trigger delay settings to the camera.

Parameters

pTrigger-Delay	Structure providing trigger delay settings.
broadcast	Whether the action should be broadcast.

See also

GetTriggerModeInfo()
GetTriggerMode()
SetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.42 virtual Error SetTriggerMode (const TriggerMode * *pTriggerMode*, bool *broadcast* = false) [pure virtual]

Set the specified trigger settings to the camera.

Parameters

pTrigger-Mode	Structure providing trigger mode settings.
broadcast	Whether the action should be broadcast.

See also

GetTriggerModeInfo()
GetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.43 virtual Error SetUserBuffers (unsigned char ∗const *pMemBuffers,* int *size,* int *numBuffers*) `[pure virtual]`

Specify user allocated buffers to use as image data buffers.

To prevent image tearing, the size of each buffer should be equal to ((unsigned int)(bufferSize + packetSize - 1)/packetSize) ∗ packetSize. The total size should be (size ∗ numBuffers) or larger.

Parameters

pMem-Buffers	Pointer to memory buffers to be written to.
size	The size of each buffer (in bytes).
numBuffers	Number of buffers in the array.

See also

StartCapture()
RetrieveBuffer()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.44 virtual **Error** StartCapture (**ImageEventCallback** *callbackFn* = `NULL`, const void ∗ *pCallbackData* = `NULL`) `[pure virtual]`

Starts isochronous image capture.

It will use either the current video mode or the most recently set video mode of the camera. The optional callback function parameter is called on completion of image transfer. Alternatively, the callback parameter can be set to NULL and RetrieveBuffer() can be called as a blocking call to get the image data.

Parameters

callbackFn	A function to be called when a new image is received.
pCallback-Data	A pointer to data that can be passed to the callback function.

See also

RetrieveBuffer()
StartSyncCapture()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.45 static **Error** StartSyncCapture (unsigned int *numCameras*, const **CameraBase** ∗∗ *ppCameras*, const **ImageEventCallback** ∗ *pCallbackFns* = `NULL`, const void ∗∗ *pCallbackDataArray* = `NULL`) `[static]`

Starts isochronous image capture on multiple cameras.

On each frame, the time stamps across the cameras are aligned which means the frames are synchronized. Note that the cameras must be synchronized by external means in order for this function to work. This means that the cameras should either be on the same bus, hardware synchronized (e.g. through triggering) or Multisync is running.

Parameters

num-Cameras	Number of Camera objects in the ppCameras array.
ppCameras	Array of pointers to Camera objects containing the cameras to be started and synchronized.
pCallback-Fns	Array of callback functions for each camera.
pCallback-DataArray	Array of callback data pointers.

See also

RetrieveBuffer()
StartCapture()
StopCapture()

Returns

An Error indicating the success or failure of the function.

8.6.3.46 virtual Error StopCapture ( ) `[pure virtual]`

Stops isochronous image transfer and cleans up all associated resources.

If an image callback function (specified in the StartCapture() call) is currently executing, StopCapture() will not return until after the callback has completed.

See also

StartCapture()
RetrieveBuffer()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.47 virtual Error WaitForBufferEvent (Image ∗ *pImage*, unsigned int *eventNumber*) `[pure virtual]`

Retrieves the next image event containing the next part of the image.

Parameters

pImage	Pointer to Image object to store image data.
event-Number	The event number to wait for.

See also

StartCapture()
RetrieveBuffer()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.48 virtual Error WriteRegister (unsigned int *address,* unsigned int *value,* bool *broadcast* = false) [pure virtual]

Write to the specified register on the camera.

Parameters

address	DCAM address to be written to.
value	The value to be written.
broadcast	Whether the action should be broadcast.

See also

ReadRegister()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.3.49 virtual Error WriteRegisterBlock (unsigned short *addressHigh,* unsigned int *addressLow,* const unsigned int $*$ *pBuffer,* unsigned int *length*) [pure virtual]

Write to the specified register block on the camera.

Parameters

addressHigh	Top 16 bits of the 48 bit absolute address to write to.
addressLow	Bottom 32 bits of the 48 bits absolute address to write to.
pBuffer	Array containing data to be written.
length	Size of array, in quadlets.

See also

ReadRegisterBlock()

Returns

An Error indicating the success or failure of the function.

Implemented in GigECamera, Camera, and GCCamera.

8.6.4 Member Data Documentation

8.6.4.1 CameraData∗ m_pCameraData [protected]

The documentation for this class was generated from the following file:

- CameraBase.h

8.7 CameraControlDlg Class Reference

The CameraControlDlg object represents a dialog that provides a graphical interface to a specified camera.

Public Member Functions

- CameraControlDlg ()

  Default constructor.
- ∼CameraControlDlg ()

  Default destructor.
- void Connect (CameraBase ∗pCamera)

  Connect dialog to a camera.
- void Disconnect ()

  Disconnect a connected camera from the dialog.
- void Show ()

  Show the dialog.
- void Show (void ∗pParent)

  Show the dialog.
- void ShowModal ()

  Show the modal dialog.
- void ShowModal (void ∗pParent)

  Show the modal dialog.
- void Hide ()

  Hide the dialog.
- bool IsVisible ()

  Get the visibility of the dialog.
- void SetTitle (const char ∗title)

  Change the title of the window.

8.7.1 Detailed Description

The CameraControlDlg object represents a dialog that provides a graphical interface to a specified camera.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 CameraControlDlg ( )

Default constructor.

8.7.2.2 ∼CameraControlDlg ( )

Default destructor.

8.7.3 Member Function Documentation

8.7.3.1 void Connect (CameraBase ∗ *pCamera*)

Connect dialog to a camera.

Parameters

pCamera	Camera object to connect the dialog to.

8.7.3.2 void Disconnect ( )

Disconnect a connected camera from the dialog.

8.7.3.3 void Hide ( )

Hide the dialog.

8.7.3.4 bool IsVisible ( )

Get the visibility of the dialog.

Returns

Whether the dialog is visible.

8.7.3.5 void SetTitle (const char ∗ *title*)

Change the title of the window.

This has to be called after calling Connect().

Parameters

title	Null-terminated string representing the title.

8.7.3.6 void Show ( )

Show the dialog.

8.7.3.7 void Show (void ∗ *pParent*)

Show the dialog.

8.7.3.8 void ShowModal ( )

Show the modal dialog.

8.7.3.9 void ShowModal (void ∗ *pParent*)

Show the modal dialog.

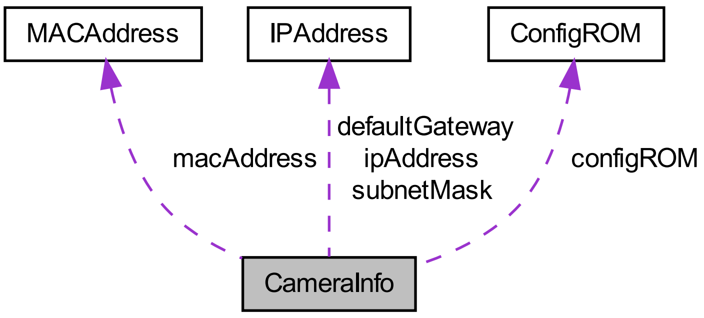
The documentation for this class was generated from the following file:

- FlyCapture2GUI.h

8.8 CameraInfo Struct Reference

Camera information.

Collaboration diagram for CameraInfo:



Public Member Functions

- CameraInfo ()

Public Attributes

- unsigned int serialNumber

    Device serial number.
- InterfaceType interfaceType

    Interface type.
- DriverType driverType

    Driver type.
- bool isColorCamera

    Flag indicating if this is a color camera.
- char modelName [sk_maxStringLength]

    Device model name.
- char vendorName [sk_maxStringLength]

    Device vendor name.
- char sensorInfo [sk_maxStringLength]

    String detailing the sensor information.
- char sensorResolution [sk_maxStringLength]

    String providing the sensor resolution.
- char driverName [sk_maxStringLength]

    Driver name of driver being used.
- char firmwareVersion [sk_maxStringLength]

    Firmware version of camera.
- char firmwareBuildTime [sk_maxStringLength]

    Firmware build time.
- BusSpeed maximumBusSpeed

    Maximum bus speed.
- BayerTileFormat bayerTileFormat

    Bayer tile format.
- unsigned short busNumber

    Bus number, set to 0 for GigE and USB cameras.
- unsigned short nodeNumber

    ieee1394 Node number, set to 0 for GigE and USB cameras
- PCIeBusSpeed pcieBusSpeed

    PCIe Bus Speed, set to PCIE_BUSSPEED_UNKNOWN for unsupported drivers.
- unsigned int reserved [16]

    Reserved for future use.

IIDC specific information

- unsigned int iidcVer
  DCAM version.
- ConfigROM configROM
  Configuration ROM data.

GigE specific information

- unsigned int gigEMajorVersion
  GigE Vision version.
- unsigned int gigEMinorVersion
  GigE Vision minor version.
- char userDefinedName [sk_maxStringLength]
  User defined name.
- char xmlURL1 [sk_maxStringLength]
  XML URL 1.
- char xmlURL2 [sk_maxStringLength]
  XML URL 2.
- MACAddress macAddress
  MAC address.
- IPAddress ipAddress
  IP address.
- IPAddress subnetMask
  Subnet mask.
- IPAddress defaultGateway
  Default gateway.
- unsigned int ccpStatus
  Status/Content of CCP register.
- unsigned int applicationIPAddress
  Local Application IP Address.
- unsigned int applicationPort
  Local Application port.

8.8.1 Detailed Description

Camera information.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 CameraInfo ( ) `[inline]`

8.8.3 Member Data Documentation

8.8.3.1 unsigned int applicationIPAddress

Local Application IP Address.

8.8.3.2 unsigned int applicationPort

Local Application port.

8.8.3.3 BayerTileFormat bayerTileFormat

Bayer tile format.

8.8.3.4 unsigned short busNumber

Bus number, set to 0 for GigE and USB cameras.

8.8.3.5 unsigned int ccpStatus

Status/Content of CCP register.

8.8.3.6 ConfigROM configROM

Configuration ROM data.

8.8.3.7 IPAddress defaultGateway

Default gateway.

8.8.3.8 char driverName[sk_maxStringLength]

Driver name of driver being used.

8.8.3.9 DriverType driverType

Driver type.

8.8.3.10 char firmwareBuildTime[sk_maxStringLength]

Firmware build time.

8.8.3.11 char firmwareVersion[sk_maxStringLength]

Firmware version of camera.

8.8.3.12 unsigned int gigEMajorVersion

GigE Vision version.

8.8.3.13 unsigned int gigEMinorVersion

GigE Vision minor version.

8.8.3.14 unsigned int iidcVer

DCAM version.

8.8.3.15 InterfaceType interfaceType

Interface type.

8.8.3.16 IPAddress ipAddress

IP address.

8.8.3.17 bool isColorCamera

Flag indicating if this is a color camera.

8.8.3.18 MACAddress macAddress

MAC address.

8.8.3.19 BusSpeed maximumBusSpeed

Maximum bus speed.

8.8.3.20 char modelName[sk_maxStringLength]

Device model name.

8.8.3.21 unsigned short nodeNumber

ieee1394 Node number, set to 0 for GigE and USB cameras

8.8.3.22 PCIeBusSpeed pcieBusSpeed

PCIe Bus Speed, set to PCIE_BUSSPEED_UNKNOWN for unsupported drivers.

8.8.3.23 unsigned int reserved[16]

Reserved for future use.

8.8.3.24 char sensorInfo[sk_maxStringLength]

String detailing the sensor information.

8.8.3.25 char sensorResolution[sk_maxStringLength]

String providing the sensor resolution.

8.8.3.26 unsigned int serialNumber

Device serial number.

8.8.3.27 IPAddress subnetMask

Subnet mask.

8.8.3.28 char userDefinedName[sk_maxStringLength]

User defined name.

8.8.3.29 char vendorName[sk_maxStringLength]

Device vendor name.

8.8.3.30 char xmlURL1[sk_maxStringLength]

XML URL 1.

8.8.3.31 char xmlURL2[sk_maxStringLength]

XML URL 2.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.9 CameraSelectionDlg Class Reference

The CameraSelectionDlg object represents a dialog that provides a graphical interface that lists the number of cameras available to the library.

Public Member Functions

- CameraSelectionDlg ()

  Default constructor.
- ∼CameraSelectionDlg ()

  Default destructor.
- void ShowModal (bool ∗pOk, PGRGuid ∗pGuid, unsigned int ∗pSize)

  Show the CameraSelectionDlg.
- void SetTitle (const char ∗title)

  Set the window title.

8.9.1 Detailed Description

The CameraSelectionDlg object represents a dialog that provides a graphical interface that lists the number of cameras available to the library.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 CameraSelectionDlg ( )

Default constructor.

8.9.2.2 ∼CameraSelectionDlg ( )

Default destructor.

8.9.3 Member Function Documentation

8.9.3.1 void SetTitle (const char ∗ *title*)

Set the window title.

Parameters

title	Null-terminated string representing the title.

8.9.3.2 void ShowModal (bool ∗ *pOk*, PGRGuid ∗ *pGuid*, unsigned int ∗ *pSize*)

Show the CameraSelectionDlg.

Parameters

pOk	Whether Ok (true) or Cancel (false) was clicked.
pGuid	Array of PGRGuids containing the selected cameras.
pSize	Size of PGRGuid array.

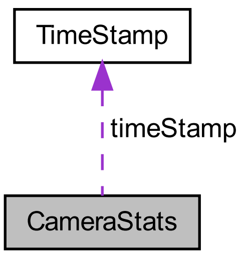
The documentation for this class was generated from the following file:

- FlyCapture2GUI.h

8.10 CameraStats Struct Reference

Camera diagnostic information.

Collaboration diagram for CameraStats:



Public Member Functions

- CameraStats ()

Public Attributes

- unsigned int imageDropped
- unsigned int imageCorrupt
- unsigned int imageXmitFailed
- unsigned int imageDriverDropped

- unsigned int regReadFailed
- unsigned int regWriteFailed
- unsigned int portErrors
- bool cameraPowerUp
- float cameraVoltages [8]
- unsigned int numVoltages

  The number of voltage registers available.
- float cameraCurrents [8]
- unsigned int numCurrents

  The number of current registers available.
- unsigned int temperature
- unsigned int timeSinceInitialization
- unsigned int timeSinceBusReset
- TimeStamp timeStamp
- unsigned int numResendPacketsRequested
- unsigned int numResendPacketsReceived
- unsigned int reserved [16]

  Reserved for future use.

8.10.1 Detailed Description

Camera diagnostic information.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 CameraStats ( ) `[inline]`

8.10.3 Member Data Documentation

8.10.3.1 float cameraCurrents[8]

8.10.3.2 bool cameraPowerUp

8.10.3.3 float cameraVoltages[8]

8.10.3.4 unsigned int imageCorrupt

8.10.3.5 unsigned int imageDriverDropped

8.10.3.6 unsigned int imageDropped

8.10.3.7 unsigned int imageXmitFailed

8.10.3.8 unsigned int numCurrents

The number of current registers available.

0: the values in cameraCurrents[] are invalid.

8.10.3.9 unsigned int **numResendPacketsReceived**

8.10.3.10 unsigned int **numResendPacketsRequested**

8.10.3.11 unsigned int **numVoltages**

The number of voltage registers available.

0: the values in cameraVoltages[] are invalid.

8.10.3.12 unsigned int **portErrors**

8.10.3.13 unsigned int **regReadFailed**

8.10.3.14 unsigned int **regWriteFailed**

8.10.3.15 unsigned int **reserved**[16]

Reserved for future use.

8.10.3.16 unsigned int **temperature**

8.10.3.17 unsigned int **timeSinceBusReset**

8.10.3.18 unsigned int **timeSinceInitialization**

8.10.3.19 **TimeStamp timeStamp**

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.11 ConfigROM Struct Reference

Camera configuration ROM.

Public Member Functions

- ConfigROM ()

Public Attributes

- unsigned int nodeVendorId

  Vendor ID of a node.
- unsigned int chipIdHi

  Chip ID (high part).
- unsigned int chipIdLo

  Chip ID (low part).
- unsigned int unitSpecId

  Unit Spec ID, usually 0xa02d.
- unsigned int unitSWVer

  Unit software version.
- unsigned int unitSubSWVer

  Unit sub software version.
- unsigned int vendorUniqueInfo_0

  Vendor unique info 0.
- unsigned int vendorUniqueInfo_1

  Vendor unique info 1.
- unsigned int vendorUniqueInfo_2

  Vendor unique info 2.
- unsigned int vendorUniqueInfo_3

  Vendor unique info 3.
- char pszKeyword [sk_maxStringLength]

  Keyword.
- unsigned int reserved [16]

  Reserved for future use.

8.11.1 Detailed Description

Camera configuration ROM.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 ConfigROM ( ) `[inline]`

8.11.3 Member Data Documentation

8.11.3.1 unsigned int chipIdHi

Chip ID (high part).

8.11.3.2 unsigned int chipIdLo

Chip ID (low part).

8.11.3.3 unsigned int **nodeVendorId**

Vendor ID of a node.

8.11.3.4 char **pszKeyword[sk_maxStringLength]**

Keyword.

8.11.3.5 unsigned int **reserved**[16]

Reserved for future use.

8.11.3.6 unsigned int **unitSpecId**

Unit Spec ID, usually 0xa02d.

8.11.3.7 unsigned int **unitSubSWVer**

Unit sub software version.

8.11.3.8 unsigned int **unitSWVer**

Unit software version.

8.11.3.9 unsigned int **vendorUniqueInfo_0**

Vendor unique info 0.

8.11.3.10 unsigned int **vendorUniqueInfo_1**

Vendor unique info 1.

8.11.3.11 unsigned int **vendorUniqueInfo_2**

Vendor unique info 2.

8.11.3.12 unsigned int **vendorUniqueInfo_3**

Vendor unique info 3.

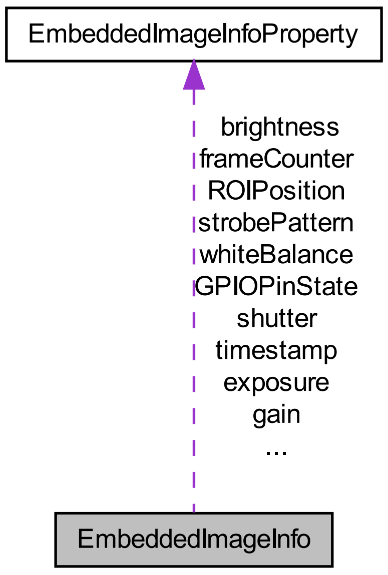
The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.12 EmbeddedImageInfo Struct Reference

Properties of the possible embedded image information.

Collaboration diagram for EmbeddedImageInfo:



Public Attributes

- EmbeddedImageInfoProperty timestamp
- EmbeddedImageInfoProperty gain
- EmbeddedImageInfoProperty shutter
- EmbeddedImageInfoProperty brightness
- EmbeddedImageInfoProperty exposure
- EmbeddedImageInfoProperty whiteBalance
- EmbeddedImageInfoProperty frameCounter
- EmbeddedImageInfoProperty strobePattern
- EmbeddedImageInfoProperty GPIOPinState
- EmbeddedImageInfoProperty ROIPosition

8.12.1 Detailed Description

Properties of the possible embedded image information.

8.12.2 Member Data Documentation

8.12.2.1 EmbeddedImageInfoProperty brightness

8.12.2.2 EmbeddedImageInfoProperty exposure

8.12.2.3 EmbeddedImageInfoProperty frameCounter

8.12.2.4 EmbeddedImageInfoProperty gain

8.12.2.5 EmbeddedImageInfoProperty GPIOPinState

8.12.2.6 EmbeddedImageInfoProperty ROIPosition

8.12.2.7 EmbeddedImageInfoProperty shutter

8.12.2.8 EmbeddedImageInfoProperty strobePattern

8.12.2.9 EmbeddedImageInfoProperty timestamp

8.12.2.10 EmbeddedImageInfoProperty whiteBalance

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.13 EmbeddedImageInfoProperty Struct Reference

Properties of a single embedded image info property.

Public Member Functions

- EmbeddedImageInfoProperty ()

Public Attributes

- bool available

  Whether this property is available.
- bool onOff

  Whether this property is on or off.

8.13.1 Detailed Description

Properties of a single embedded image info property.

8.13.2 Constructor & Destructor Documentation

8.13.2.1 EmbeddedImageInfoProperty ( ) [inline]

8.13.3 Member Data Documentation

8.13.3.1 bool available

Whether this property is available.

8.13.3.2 bool onOff

Whether this property is on or off.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.14 Error Class Reference

The Error object represents an error that is returned from the library.

Public Member Functions

- Error ()

  Default constructor.
- Error (const Error &error)

  Copy constructor.
- virtual ∼Error ()

  Default destructor.
- virtual Error & operator= (const Error &error)

  Assignment operator.
- virtual bool operator== (const Error &error) const

  Equality operator.
- virtual bool operator== (const ErrorType &errorType) const

  Equality operator.
- virtual bool operator!= (const Error &error) const

  Inequality operator.
- virtual bool operator!= (const ErrorType &errorType) const

  Inequality operator.
- virtual ErrorType GetType () const

  Retrieve the ErrorType of the error.
- virtual const char $*$ GetDescription () const

Retrieve the top level description of the error that occurred.

- virtual unsigned int GetLine () const

  Retrieve the line number where the error originated.

- virtual const char ∗ GetFilename () const

  Retrieve the source filename where the error originated.

- virtual Error GetCause () const

  Get the error which caused this error.

- virtual const char ∗ GetBuildDate () const

  Retrieve the build date of the file where the error originated.

- virtual const char ∗ CollectSupportInformation () const

  Retrieve the support information.

- virtual void PrintErrorTrace () const

  Print a formatted log trace to stderr.

Friends

- class InternalError

8.14.1 Detailed Description

The Error object represents an error that is returned from the library.

Overloaded operators allow comparisons against other Error objects or the ErrorType enumeration.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 Error ( )

Default constructor.

8.14.2.2 Error (const Error & *error*)

Copy constructor.

8.14.2.3 virtual ∼Error ( ) `[virtual]`

Default destructor.

8.14.3 Member Function Documentation

8.14.3.1 virtual const char∗ CollectSupportInformation ( ) const [virtual]

Retrieve the support information.

It is not implemented in this release.

Returns

A string containing support information.

8.14.3.2 virtual const char∗ GetBuildDate ( ) const [virtual]

Retrieve the build date of the file where the error originated.

Returns

A string with the build date and time.

8.14.3.3 virtual Error GetCause ( ) const [virtual]

Get the error which caused this error.

Returns

An error object representing the cause of this error.

8.14.3.4 virtual const char∗ GetDescription ( ) const [virtual]

Retrieve the top level description of the error that occurred.

Returns

A string with the error description.

8.14.3.5 virtual const char∗ GetFilename ( ) const [virtual]

Retrieve the source filename where the error originated.

Returns

A string with the file name.

8.14.3.6 virtual unsigned int GetLine ( ) const [virtual]

Retrieve the line number where the error originated.

Returns

The line number.

8.14.3.7 virtual ErrorType GetType ( ) const [virtual]

Retrieve the ErrorType of the error.

Returns

The ErrorType of the error.

8.14.3.8 virtual bool operator!= (const Error & *error*) const [virtual]

Inequality operator.

8.14.3.9 virtual bool operator!= (const ErrorType & *errorType*) const [virtual]

Inequality operator.

This overloaded operator compares the ErrorType of the Error against the specified ErrorType.

8.14.3.10 virtual Error& operator= (const Error & *error*) [virtual]

Assignment operator.

8.14.3.11 virtual bool operator== (const Error & *error*) const [virtual]

Equality operator.

8.14.3.12 virtual bool operator== (const ErrorType & *errorType*) const [virtual]

Equality operator.

This overloaded operator compares the ErrorType of the Error against the specified ErrorType.

8.14.3.13 virtual void PrintErrorTrace ( ) const [virtual]

Print a formatted log trace to stderr.

8.14.4 Friends And Related Function Documentation

8.14.4.1 friend class InternalError `[friend]`

The documentation for this class was generated from the following file:

- Error.h

8.15 FC2Config Struct Reference

Configuration for a camera.

Public Member Functions

- FC2Config ()

Public Attributes

- unsigned int numBuffers

    Number of buffers used by the FlyCapture2 library to grab images.
- unsigned int numImageNotifications

    Number of notifications per image.
- unsigned int minNumImageNotifications

    Minimum number of notifications needed for the current image settings on the camera.
- int grabTimeout

    Time in milliseconds that RetrieveBuffer() and WaitForBufferEvent() will wait for an image before timing out and returning.
- GrabMode grabMode

    Grab mode for the camera.
- bool highPerformanceRetrieveBuffer

    This parameter enables RetireveBuffer to run in high performance mode.
- BusSpeed isochBusSpeed

    Isochronous bus speed.
- BusSpeed asyncBusSpeed

    Asynchronous bus speed.
- BandwidthAllocation bandwidthAllocation

    Bandwidth allocation flag that tells the camera the bandwidth allocation strategy to employ.
- unsigned int registerTimeoutRetries

    Number of retries to perform when a register read/write timeout is received by the library.
- unsigned int registerTimeout

    Register read/write timeout value, in microseconds.
- unsigned int reserved [16]

    Reserved for future use.

8.15.1 Detailed Description

Configuration for a camera.

These options are options that are generally should be set before starting isochronous transfer.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 FC2Config ( ) `[inline]`

8.15.3 Member Data Documentation

8.15.3.1 BusSpeed asyncBusSpeed

Asynchronous bus speed.

8.15.3.2 BandwidthAllocation bandwidthAllocation

Bandwidth allocation flag that tells the camera the bandwidth allocation strategy to employ.

8.15.3.3 GrabMode grabMode

Grab mode for the camera.

The default is DROP_FRAMES.

8.15.3.4 int grabTimeout

Time in milliseconds that RetrieveBuffer() and WaitForBufferEvent() will wait for an image before timing out and returning.

8.15.3.5 bool highPerformanceRetrieveBuffer

This parameter enables RetireveBuffer to run in high performance mode.

This means that any interaction with the camera, other then grabbing the image is disabled. Currently Retrieve buffer reads registers on the camera to determine which embedded image information settings have been enabled, and it reads what the bayer tile is currently set to. When High Performance mode is on, these reads are disabled. - This means that any changes to the Bayer Tile or to the Embedded image info after StartCapture() will not be tracked when made using direct register writes. If the corresponding SetEmbededImageInfo() and GetEmbededImageInfo() calls are used then the changes will be appropriately reflected. This also means that changes to embedded image info from other processes will not be updated either.

8.15.3.6 BusSpeed isochBusSpeed

Isochronous bus speed.

8.15.3.7 unsigned int minNumImageNotifications

Minimum number of notifications needed for the current image settings on the camera.

Read-only value.

8.15.3.8 unsigned int numBuffers

Number of buffers used by the FlyCapture2 library to grab images.

8.15.3.9 unsigned int numImageNotifications

Number of notifications per image.

This value should only be set after the image settings to be used is set to the camera. The default number of notifications is 1.

There are 4 general scenarios:

- 1 notification - End of image
- 2 notifications - After first packet and end of image
- 3 notifications - After first packet, middle of image, end of image
- x notifications - After first packet, (x -2) spread evenly, end of image

Specifying zero for the number of notifications will be ignored (the current value will not be modified).

Note that the event numbers start at 0. Ex. when 3 notifications are used, the three events will be 0, 1 and 2.

8.15.3.10 unsigned int registerTimeout

Register read/write timeout value, in microseconds.

The default value is dependent on the interface type.

8.15.3.11 unsigned int registerTimeoutRetries

Number of retries to perform when a register read/write timeout is received by the library.

The default value is 0.

8.15.3.12 unsigned int **reserved**[16]

Reserved for future use.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.16 FC2Version Struct Reference

The current version of the library.

Public Attributes

- unsigned int major

    Major version number.
- unsigned int minor

    Minor version number.
- unsigned int type

    Type version number.
- unsigned int build

    Build version number.

8.16.1 Detailed Description

The current version of the library.

8.16.2 Member Data Documentation

8.16.2.1 unsigned int **build**

Build version number.

8.16.2.2 unsigned int **major**

Major version number.

8.16.2.3 unsigned int **minor**

Minor version number.

8.16.2.4 unsigned int type

Type version number.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.17 Format7ImageSettings Struct Reference

Format 7 image settings.

Public Member Functions

- Format7ImageSettings ()

Public Attributes

- Mode mode

  Format 7 mode.
- unsigned int offsetX

  Horizontal image offset.
- unsigned int offsetY

  Vertical image offset.
- unsigned int width

  Width of image.
- unsigned int height

  Height of image.
- PixelFormat pixelFormat

  Pixel format of image.
- unsigned int reserved [8]

  Reserved for future use.

8.17.1 Detailed Description

Format 7 image settings.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 Format7ImageSettings ( ) `[inline]`

8.17.3 Member Data Documentation

8.17.3.1 unsigned int height

Height of image.

8.17.3.2 Mode mode

Format 7 mode.

8.17.3.3 unsigned int offsetX

Horizontal image offset.

8.17.3.4 unsigned int offsetY

Vertical image offset.

8.17.3.5 PixelFormat pixelFormat

Pixel format of image.

8.17.3.6 unsigned int reserved[8]

Reserved for future use.

8.17.3.7 unsigned int width

Width of image.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.18 Format7Info Struct Reference

Format 7 information for a single mode.

Public Member Functions

- Format7Info ()

Public Attributes

- Mode mode

  Format 7 mode.
- unsigned int maxWidth

  Maximum image width.
- unsigned int maxHeight

  Maximum image height.
- unsigned int offsetHStepSize

  Horizontal step size for the offset.
- unsigned int offsetVStepSize

  Vertical step size for the offset.
- unsigned int imageHStepSize

  Horizontal step size for the image.
- unsigned int imageVStepSize

  Vertical step size for the image.
- unsigned int pixelFormatBitField

  Supported pixel formats in a bit field.
- unsigned int vendorPixelFormatBitField

  Vendor unique pixel formats in a bit field.
- unsigned int packetSize

  Current packet size in bytes.
- unsigned int minPacketSize

  Minimum packet size in bytes for current mode.
- unsigned int maxPacketSize

  Maximum packet size in bytes for current mode.
- float percentage

  Current packet size as a percentage of maximum packet size.
- unsigned int reserved [16]

  Reserved for future use.

8.18.1 Detailed Description

Format 7 information for a single mode.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 Format7Info ( ) `[inline]`

8.18.3 Member Data Documentation

8.18.3.1 unsigned int imageHStepSize

Horizontal step size for the image.

8.18.3.2 unsigned int imageVStepSize

Vertical step size for the image.

8.18.3.3 unsigned int maxHeight

Maximum image height.

8.18.3.4 unsigned int maxPacketSize

Maximum packet size in bytes for current mode.

8.18.3.5 unsigned int maxWidth

Maximum image width.

8.18.3.6 unsigned int minPacketSize

Minimum packet size in bytes for current mode.

8.18.3.7 Mode mode

Format 7 mode.

8.18.3.8 unsigned int offsetHStepSize

Horizontal step size for the offset.

8.18.3.9 unsigned int offsetVStepSize

Vertical step size for the offset.

8.18.3.10 unsigned int packetSize

Current packet size in bytes.

8.18.3.11 float percentage

Current packet size as a percentage of maximum packet size.

8.18.3.12 unsigned int **pixelFormatBitField**

Supported pixel formats in a bit field.

8.18.3.13 unsigned int **reserved**[16]

Reserved for future use.

8.18.3.14 unsigned int **vendorPixelFormatBitField**

Vendor unique pixel formats in a bit field.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.19 Format7PacketInfo Struct Reference

Format 7 packet information.

Public Member Functions

- Format7PacketInfo ()

Public Attributes

- unsigned int recommendedBytesPerPacket

  Recommended bytes per packet.
- unsigned int maxBytesPerPacket

  Maximum bytes per packet.
- unsigned int unitBytesPerPacket

  Minimum bytes per packet.
- unsigned int reserved [8]

  Reserved for future use.

8.19.1 Detailed Description

Format 7 packet information.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 **Format7PacketInfo** ( ) `[inline]`

8.19.3 Member Data Documentation

8.19.3.1 unsigned int **maxBytesPerPacket**

Maximum bytes per packet.

8.19.3.2 unsigned int **recommendedBytesPerPacket**

Recommended bytes per packet.

8.19.3.3 unsigned int **reserved**[8]

Reserved for future use.

8.19.3.4 unsigned int **unitBytesPerPacket**

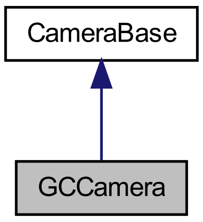
Minimum bytes per packet.

The documentation for this struct was generated from the following file:

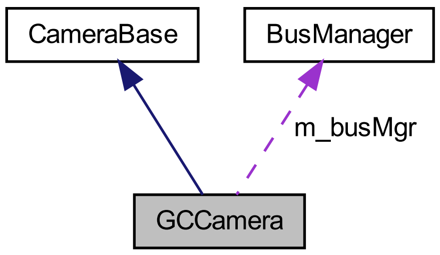
- FlyCapture2Defs.h

8.20 GCCamera Class Reference

Inheritance diagram for GCCamera:



Collaboration diagram for GCCamera:



Public Member Functions

- GCCamera (void)
- virtual ∼GCCamera (void)
- ::GenApi::INodeMap ∗ GetNodeMap ()
- Error SetCamera (CameraBase ∗camera)
- Error SetCamera (CameraBase ∗camera, const char ∗filepath=NULL)
- std::string GCCamera::GetXML ()
- virtual Error WriteGVCPRegister (unsigned int address, unsigned int value, bool broadcast=false)
- virtual Error ReadGVCPRegister (unsigned int address, unsigned int ∗pValue)
- virtual Error WriteGVCPRegisterBlock (unsigned int address, const unsigned int ∗pBuffer, unsigned int length)
- virtual Error ReadGVCPRegisterBlock (unsigned int address, unsigned int ∗p-Buffer, unsigned int length)
- virtual Error WriteGVCPMemory (unsigned int address, const unsigned char ∗p-Buffer, unsigned int length)
- virtual Error ReadGVCPMemory (unsigned int address, unsigned char ∗pBuffer, unsigned int length)
- virtual Error Connect (PGRGuid ∗pGuid=NULL)

  The following functions are inherited from CameraBase.
- Error Connect (PGRGuid ∗pGuid=NULL, const char ∗filepath=NULL)
- virtual Error Disconnect ()

  Disconnects the camera object from the camera.
- virtual bool IsConnected ()

  Checks if the camera object is currently connected to a physical camera.
- virtual Error SetCallback (ImageEventCallback callbackFn, const void ∗p-CallbackData=NULL)

Sets the callback data to be used on completion of image transfer.

- virtual Error StartCapture (ImageEventCallback callbackFn=NULL, const void ∗pCallbackData=NULL)

  Starts isochronous image capture.
- virtual Error RetrieveBuffer (Image ∗pImage)

  Retrieves the the next image object containing the next image.
- virtual Error StopCapture ()

  Stops isochronous image transfer and cleans up all associated resources.
- virtual Error WaitForBufferEvent (Image ∗pImage, unsigned int eventNumber)

  Retrieves the next image event containing the next part of the image.
- virtual Error SetUserBuffers (unsigned char ∗const pMemBuffers, int size, int numBuffers)

  Specify user allocated buffers to use as image data buffers.
- virtual Error GetConfiguration (FC2Config ∗pConfig)

  Get the configuration associated with the camera object.
- virtual Error SetConfiguration (const FC2Config ∗pConfig)

  Set the configuration associated with the camera object.
- virtual Error GetCameraInfo (CameraInfo ∗pCameraInfo)

  Retrieves information from the camera such as serial number, model name and other camera information.
- virtual Error GetPropertyInfo (PropertyInfo ∗pPropInfo)

  Retrieves information about the specified camera property.
- virtual Error GetProperty (Property ∗pProp)

  Reads the settings for the specified property from the camera.
- virtual Error SetProperty (const Property ∗pProp, bool broadcast=false)

  Writes the settings for the specified property to the camera.
- virtual Error GetGPIOPinDirection (unsigned int pin, unsigned int ∗pDirection)

  Get the GPIO pin direction for the specified pin.
- virtual Error SetGPIOPinDirection (unsigned int pin, unsigned int direction, bool broadcast=false)

  Set the GPIO pin direction for the specified pin.
- virtual Error GetTriggerModeInfo (TriggerModeInfo ∗pTriggerModeInfo)

  Retrieve trigger information from the camera.
- virtual Error GetTriggerMode (TriggerMode ∗pTriggerMode)

  Retrieve current trigger settings from the camera.
- virtual Error SetTriggerMode (const TriggerMode ∗pTriggerMode, bool broadcast=false)

  Set the specified trigger settings to the camera.
- virtual Error FireSoftwareTrigger (bool broadcast=false)

  Fire the software trigger according to the DCAM specifications.
- virtual Error GetTriggerDelayInfo (TriggerDelayInfo ∗pTriggerDelayInfo)

  Retrieve trigger delay information from the camera.
- virtual Error GetTriggerDelay (TriggerDelay ∗pTriggerDelay)

  Retrieve current trigger delay settings from the camera.

- virtual Error SetTriggerDelay (const TriggerDelay ∗pTriggerDelay, bool broadcast=false)

  Set the specified trigger delay settings to the camera.
- virtual Error GetStrobeInfo (StrobeInfo ∗pStrobeInfo)

  Retrieve strobe information from the camera.
- virtual Error GetStrobe (StrobeControl ∗pStrobeControl)

  Retrieve current strobe settings from the camera.
- virtual Error SetStrobe (const StrobeControl ∗pStrobeControl, bool broadcast=false)

  Set current strobe settings to the camera.
- virtual Error GetLUTInfo (LUTData ∗pData)

  Query if LUT support is available on the camera.
- virtual Error GetLUTBankInfo (unsigned int bank, bool ∗pReadSupported, bool ∗pWriteSupported)

  Query the read/write status of a single LUT bank.
- virtual Error GetActiveLUTBank (unsigned int ∗pActiveBank)

  Get the LUT bank that is currently being used.
- virtual Error SetActiveLUTBank (unsigned int activeBank)

  Set the LUT bank that will be used.
- virtual Error EnableLUT (bool on)

  Enable or disable LUT functionality on the camera.
- virtual Error GetLUTChannel (unsigned int bank, unsigned int channel, unsigned int sizeEntries, unsigned int ∗pEntries)

  Get the LUT channel settings from the camera.
- virtual Error SetLUTChannel (unsigned int bank, unsigned int channel, unsigned int sizeEntries, const unsigned int ∗pEntries)

  Set the LUT channel settings to the camera.
- virtual Error GetMemoryChannel (unsigned int ∗pCurrentChannel)

  Retrieve the current memory channel from the camera.
- virtual Error SaveToMemoryChannel (unsigned int channel)

  Save the current settings to the specfied current memory channel.
- virtual Error RestoreFromMemoryChannel (unsigned int channel)

  Restore the specfied current memory channel.
- virtual Error GetMemoryChannelInfo (unsigned int ∗pNumChannels)

  Query the camera for memory channel support.
- virtual Error GetEmbeddedImageInfo (EmbeddedImageInfo ∗pInfo)

  Get the current status of the embedded image information register, as well as the availability of each embedded property.
- virtual Error SetEmbeddedImageInfo (EmbeddedImageInfo ∗pInfo)

  Sets the on/off values of the embedded image information structure to the camera.
- virtual Error WriteRegister (unsigned int address, unsigned int value, bool broadcast=false)

  Write to the specified register on the camera.
- virtual Error ReadRegister (unsigned int address, unsigned int ∗pValue)

Read the specified register from the camera.

- virtual Error WriteRegisterBlock (unsigned short addressHigh, unsigned int addressLow, const unsigned int ∗pBuffer, unsigned int length)

  Write to the specified register block on the camera.
- virtual Error ReadRegisterBlock (unsigned short addressHigh, unsigned int addressLow, unsigned int ∗pBuffer, unsigned int length)

  Read from the specified register block on the camera.
- Error GetCycleTime (TimeStamp ∗timeStamp)

  Returns a Timestamp struct containing 1394 CYCLE_TIME information.
- InterfaceType GetInterfaceType ()
- virtual Error GetStats (CameraStats ∗pStats)
- virtual Error ResetStats ()

Static Public Member Functions

- static Error StartSyncCapture (unsigned int numCameras, const GigECamera ∗∗ppCameras, const ImageEventCallback ∗pCallbackFns=NULL, const void ∗∗p-CallbackDataArray=NULL)
- static const char ∗ GetRegisterString (unsigned int registerVal)

  Returns a text representation of the register value.

Protected Member Functions

- void TestGainNode ()

Protected Attributes

- BusManager m_busMgr

8.20.1 Constructor & Destructor Documentation

8.20.1.1 GCCamera (void)

8.20.1.2 virtual ∼GCCamera (void) [virtual]

8.20.2 Member Function Documentation

8.20.2.1 virtual Error Connect (PGRGuid ∗ *pGuid* = NULL) [virtual]

The following functions are inherited from CameraBase.

See CameraBase.h for further information.

Implements CameraBase.

8.20.2.2 Error Connect (PGRGuid ∗ *pGuid* = `NULL`, const char ∗ *filepath* = `NULL`)

8.20.2.3 virtual Error Disconnect ( ) `[virtual]`

Disconnects the camera object from the camera.

This allows another physical camera to be connected to the camera object.

See also

Connect()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.4 virtual Error EnableLUT (bool *on*) `[virtual]`

Enable or disable LUT functionality on the camera.

Parameters

on	Whether to enable or disable LUT.

See also

GetLUTInfo()
GetLUTChannel()
SetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.5 virtual Error FireSoftwareTrigger (bool *broadcast* = `false`) `[virtual]`

Fire the software trigger according to the DCAM specifications.

Parameters

broadcast	Whether the action should be broadcast.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.6 std::string GCCamera::GetXML ( )

8.20.2.7 virtual Error GetActiveLUTBank (unsigned int ∗ *pActiveBank*) [virtual]

Get the LUT bank that is currently being used.

For cameras with PGR LUT, the active bank is always 0.

Parameters

pActiveBank	The currently active bank.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.8 virtual Error GetCameraInfo (CameraInfo ∗ *pCameraInfo*) [virtual]

Retrieves information from the camera such as serial number, model name and other camera information.

Parameters

pCameraInfo	Pointer to the camera information structure to be filled.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.9 virtual Error GetConfiguration (FC2Config ∗ *pConfig*) [virtual]

Get the configuration associated with the camera object.

Parameters

pConfig	Pointer to the configuration structure to be filled.

See also

SetConfiguration()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.10 Error GetCycleTime (TimeStamp ∗ *timeStamp*) [virtual]

Returns a Timestamp struct containing 1394 CYCLE_TIME information.

Parameters

registerVal	The register value to query.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.11 virtual Error GetEmbeddedImageInfo (EmbeddedImageInfo ∗ *pInfo*) [virtual]

Get the current status of the embedded image information register, as well as the availability of each embedded property.

Parameters

pInfo	Structure to be filled.

See also

SetEmbeddedImageInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.12 virtual Error GetGPIOPinDirection (unsigned int *pin,* unsigned int ∗ *pDirection*) [virtual]

Get the GPIO pin direction for the specified pin.

This is not a required call when using the trigger or strobe functions as the pin direction is set automatically internally.

Parameters

pin	Pin to get the direction for.
pDirection	Direction of the pin. 0 for input, 1 for output.

See also

SetGPIOPinDirection()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.13 InterfaceType GetInterfaceType ( )

8.20.2.14 virtual Error GetLUTBankInfo (unsigned int *bank,* bool ∗ *pReadSupported,* bool ∗ *pWriteSupported*) `[virtual]`

Query the read/write status of a single LUT bank.

Parameters

bank	The bank to query.
pRead-Supported	Whether reading from the bank is supported.
pWrite-Supported	Whether writing to the bank is supported.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.15 virtual Error GetLUTChannel (unsigned int *bank,* unsigned int *channel,* unsigned int *sizeEntries,* unsigned int ∗ *pEntries*) `[virtual]`

Get the LUT channel settings from the camera.

Parameters

bank	Bank to retrieve.
channel	Channel to retrieve.

sizeEntries	Number of entries in LUT table to read.
pEntries	Array to store LUT entries.

See also

GetLUTInfo()
EnableLUT()
SetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.16 virtual **Error** GetLUTInfo (**LUTData** ∗ *pData*) [virtual]

Query if LUT support is available on the camera.

Note that some cameras may report support for the LUT and return an inputBitDepth of 0. In these cases use log2(numEntries) for the inputBitDepth.

Parameters

pData	The LUT structure to be filled.

See also

EnableLUT()
GetLUTChannel()
SetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.17 virtual **Error** GetMemoryChannel (unsigned int ∗ *pCurrentChannel*) [virtual]

Retrieve the current memory channel from the camera.

Parameters

pCurrent-Channel	Current memory channel.

See also

SaveToMemoryChannel()
RestoreFromMemoryChannel()
GetMemoryChannelInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.18 virtual Error GetMemoryChannelInfo (unsigned int ∗ *pNumChannels*) `[virtual]`

Query the camera for memory channel support.

If the number of channels is 0, then memory channel support is not available.

Parameters

pNum-Channels	Number of memory channels supported.

See also

GetMemoryChannel()
SaveToMemoryChannel()
RestoreFromMemoryChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.19 ::GenApi::INodeMap∗ GetNodeMap ( )

8.20.2.20 virtual Error GetProperty (Property ∗ *pProp*) `[virtual]`

Reads the settings for the specified property from the camera.

The property type must be specified in the Property structure passed into the function in order for the function to succeed. If auto is on, the integer and abs values returned may not be consistent with each other.

Parameters

pProp	Pointer to the Property structure to be filled.

See also

GetPropertyInfo()
SetProperty()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.21 virtual Error GetPropertyInfo (PropertyInfo ∗ *pPropInfo*) [virtual]

Retrieves information about the specified camera property.

The property type must be specified in the PropertyInfo structure passed into the function in order for the function to succeed.

Parameters

pPropInfo	Pointer to the PropertyInfo structure to be filled.

See also

GetProperty()
SetProperty()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.22 static const char∗ GetRegisterString (unsigned int *registerVal*) [static]

Returns a text representation of the register value.

Parameters

registerVal	The register value to query.

Returns

The text representation of the register.

Reimplemented from CameraBase.

8.20.2.23 virtual Error GetStats (CameraStats * *pStats*) [virtual]

Implements CameraBase.

8.20.2.24 virtual Error GetStrobe (StrobeControl * *pStrobeControl*) [virtual]

Retrieve current strobe settings from the camera.

The strobe pin must be specified in the structure before being passed in to the function.

Parameters

pStrobe-Control	Structure to receive strobe settings.

See also

GetStrobeInfo()
SetStrobe()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.25 virtual Error GetStrobeInfo (StrobeInfo * *pStrobeInfo*) [virtual]

Retrieve strobe information from the camera.

Parameters

pStrobeInfo	Structure to receive strobe information.

See also

GetStrobe()
SetStrobe()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.26 virtual Error GetTriggerDelay (TriggerDelay * *pTriggerDelay*) [virtual]

Retrieve current trigger delay settings from the camera.

Parameters

pTrigger-Delay	Structure to receive trigger delay settings.

See also

GetTriggerModeInfo()
GetTriggerMode()
SetTriggerMode()
GetTriggerDelayInfo()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.27 virtual Error GetTriggerDelayInfo (TriggerDelayInfo ∗ *pTriggerDelayInfo*) [virtual]

Retrieve trigger delay information from the camera.

Parameters

pTrigger-DelayInfo	Structure to receive trigger delay information.

See also

GetTriggerModeInfo()
GetTriggerMode()
SetTriggerMode()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.28 virtual Error GetTriggerMode (TriggerMode ∗ *pTriggerMode*) [virtual]

Retrieve current trigger settings from the camera.

Parameters

pTrigger-Mode	Structure to receive trigger mode settings.

See also

GetTriggerModeInfo()
SetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.29 virtual Error GetTriggerModeInfo (TriggerModeInfo ∗ *pTriggerModeInfo*) [virtual]

Retrieve trigger information from the camera.

Parameters

pTrigger-ModeInfo	Structure to receive trigger information.

See also

GetTriggerMode()
SetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.30 virtual bool IsConnected ( ) [virtual]

Checks if the camera object is currently connected to a physical camera.

See also

Connect()
Disconnect()

Returns

Whether the camera object is connected to a physical camera.

Implements CameraBase.

8.20.2.31 virtual Error ReadGVCPMemory (unsigned int *address*, unsigned char * *pBuffer*, unsigned int *length*) `[virtual]`

8.20.2.32 virtual Error ReadGVCPRegister (unsigned int *address*, unsigned int * *pValue*) `[virtual]`

8.20.2.33 virtual Error ReadGVCPRegisterBlock (unsigned int *address*, unsigned int * *pBuffer*, unsigned int *length*) `[virtual]`

8.20.2.34 virtual Error ReadRegister (unsigned int *address*, unsigned int * *pValue*) `[virtual]`

Read the specified register from the camera.

Parameters

address	DCAM address to be read from.
pValue	The value that is read.

See also

WriteRegister()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.35 virtual Error ReadRegisterBlock (unsigned short *addressHigh*, unsigned int *addressLow*, unsigned int * *pBuffer*, unsigned int *length*) `[virtual]`

Read from the specified register block on the camera.

Parameters

addressHigh	Top 16 bits of the 48 bit absolute address to read from.
addressLow	Bottom 32 bits of the 48 bits absolute address to read from.
pBuffer	Array to store read data.
length	Size of array, in quadlets.

See also

WriteRegisterBlock()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.36 virtual Error ResetStats ( ) [virtual]

Implements CameraBase.

8.20.2.37 virtual Error RestoreFromMemoryChannel (unsigned int *channel*) [virtual]

Restore the specfied current memory channel.

Parameters

channel	Memory channel to restore from.

See also

GetMemoryChannel()
SaveToMemoryChannel()
GetMemoryChannelInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.38 virtual Error RetrieveBuffer (Image * *pImage*) [virtual]

Retrieves the the next image object containing the next image.

If the grab mode has not been set, or has been set to DROP_FRAMES the default behavior is to requeue images for DMA if they have not been retrieved by the time the next image transfer completes. If BUFFER_FRAMES is specified, the next image in the sequence will be retrieved. Note that for the BUFFER_FRAMES case, if retrieval does not keep up with the DMA process, images will be lost. The default behavior is to perform DROP_FRAMES image retrieval.

Parameters

pImage	Pointer to Image object to store image data.

See also

StartCapture()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.39 virtual Error SaveToMemoryChannel (unsigned int *channel*) `[virtual]`

Save the current settings to the specfied current memory channel.

Parameters

channel	Memory channel to save to.

See also

GetMemoryChannel()
RestoreFromMemoryChannel()
GetMemoryChannelInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.40 virtual Error SetActiveLUTBank (unsigned int *activeBank*) `[virtual]`

Set the LUT bank that will be used.

Parameters

activeBank	The bank to be set as active.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.41 virtual Error SetCallback (ImageEventCallback *callbackFn*, const void ∗ *pCallbackData* = `NULL`) `[virtual]`

Sets the callback data to be used on completion of image transfer.

To clear the current stored callback data, pass in NULL for both arguments.

Parameters

callbackFn	A function to be called when a new image is received.
pCallback-Data	A pointer to data that can be passed to the callback function.

See also

StartCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.42 Error SetCamera (CameraBase ∗ *camera*)

8.20.2.43 Error SetCamera (CameraBase ∗ *camera*, const char ∗ *filepath* = `NULL`)

8.20.2.44 virtual Error SetConfiguration (const FC2Config ∗ *pConfig*) `[virtual]`

Set the configuration associated with the camera object.

Parameters

pConfig	Pointer to the configuration structure to be used.

See also

GetConfiguration()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.45 virtual Error SetEmbeddedImageInfo (EmbeddedImageInfo ∗ *pInfo*) [virtual]

Sets the on/off values of the embedded image information structure to the camera.

Parameters

pInfo	Structure to be used.

See also

GetEmbeddedImageInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.46 virtual Error SetGPIOPinDirection (unsigned int *pin,* unsigned int *direction,* bool *broadcast* = false) [virtual]

Set the GPIO pin direction for the specified pin.

This is useful if there is a need to set the pin into an input pin (i.e. to read the voltage) off the pin without setting it as a trigger source. This is not a required call when using the trigger or strobe functions as the pin direction is set automatically internally.

Parameters

pin	Pin to get the direction for.
direction	Direction of the pin. 0 for input, 1 for output.
broadcast	Whether the action should be broadcast.

See also

GetGPIOPinDirection()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.47 virtual Error SetLUTChannel (unsigned int *bank,* unsigned int *channel,* unsigned int *sizeEntries,* const unsigned int ∗ *pEntries*) [virtual]

Set the LUT channel settings to the camera.

Parameters

bank	Bank to set.
channel	Channel to set.
sizeEntries	Number of entries in LUT table to write. This must be the same size as numEntries returned by GetLutInfo().
pEntries	Array containing LUT entries to write.

See also

GetLUTInfo()
EnableLUT()
GetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.48 virtual Error SetProperty (const Property ∗ *pProp*, bool *broadcast* = `false`) `[virtual]`

Writes the settings for the specified property to the camera.

The property type must be specified in the Property structure passed into the function in order for the function to succeed. The absControl flag controls whether the absolute or integer value is written to the camera.

Parameters

pProp	Pointer to the Property structure to be used.
broadcast	Whether the action should be broadcast.

See also

GetPropertyInfo()
GetProperty()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.49 virtual Error SetStrobe (const StrobeControl ∗ *pStrobeControl*, bool *broadcast* = `false`) `[virtual]`

Set current strobe settings to the camera.

The strobe pin must be specified in the structure before being passed in to the function.

Parameters

pStrobe-Control	Structure providing strobe settings.
broadcast	Whether the action should be broadcast.

See also

GetStrobeInfo()
GetStrobe()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.50 virtual Error SetTriggerDelay (const TriggerDelay * *pTriggerDelay*, bool *broadcast* = `false`) `[virtual]`

Set the specified trigger delay settings to the camera.

Parameters

pTrigger-Delay	Structure providing trigger delay settings.
broadcast	Whether the action should be broadcast.

See also

GetTriggerModeInfo()
GetTriggerMode()
SetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.51 virtual Error SetTriggerMode (const TriggerMode * *pTriggerMode*, bool *broadcast* = `false`) `[virtual]`

Set the specified trigger settings to the camera.

Parameters

pTrigger-Mode	Structure providing trigger mode settings.
broadcast	Whether the action should be broadcast.

See also

GetTriggerModeInfo()
GetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.52 virtual Error SetUserBuffers (unsigned char ∗const *pMemBuffers,* int *size,* int *numBuffers*) [virtual]

Specify user allocated buffers to use as image data buffers.

To prevent image tearing, the size of each buffer should be equal to ((unsigned int)(bufferSize + packetSize - 1)/packetSize) ∗ packetSize. The total size should be (size ∗ numBuffers) or larger.

Parameters

pMem-Buffers	Pointer to memory buffers to be written to.
size	The size of each buffer (in bytes).
numBuffers	Number of buffers in the array.

See also

StartCapture()
RetrieveBuffer()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.53 virtual Error StartCapture (ImageEventCallback *callbackFn* = `NULL`, const void ∗ *pCallbackData* = `NULL`) `[virtual]`

Starts isochronous image capture.

It will use either the current video mode or the most recently set video mode of the camera. The optional callback function parameter is called on completion of image transfer. Alternatively, the callback parameter can be set to NULL and RetrieveBuffer() can be called as a blocking call to get the image data.

Parameters

callbackFn	A function to be called when a new image is received.
pCallback-Data	A pointer to data that can be passed to the callback function.

See also

RetrieveBuffer()
StartSyncCapture()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.54 static Error StartSyncCapture (unsigned int *numCameras*, const GigECamera ∗∗ *ppCameras*, const ImageEventCallback ∗ *pCallbackFns* = `NULL`, const void ∗∗ *pCallbackDataArray* = `NULL`) `[static]`

8.20.2.55 virtual Error StopCapture ( ) `[virtual]`

Stops isochronous image transfer and cleans up all associated resources.

If an image callback function (specified in the StartCapture() call) is currently executing, StopCapture() will not return until after the callback has completed.

See also

StartCapture()
RetrieveBuffer()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.56 void TestGainNode ( ) [protected]

8.20.2.57 virtual Error WaitForBufferEvent (Image * *pImage*, unsigned int *eventNumber*) [virtual]

Retrieves the next image event containing the next part of the image.

Parameters

pImage	Pointer to Image object to store image data.
event-Number	The event number to wait for.

See also

StartCapture()
RetrieveBuffer()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.58 virtual Error WriteGVCPMemory (unsigned int *address*, const unsigned char * *pBuffer*, unsigned int *length*) [virtual]

8.20.2.59 virtual Error WriteGVCPRegister (unsigned int *address*, unsigned int *value*, bool *broadcast* = false) [virtual]

8.20.2.60 virtual Error WriteGVCPRegisterBlock (unsigned int *address*, const unsigned int * *pBuffer*, unsigned int *length*) [virtual]

8.20.2.61 virtual Error WriteRegister (unsigned int *address*, unsigned int *value*, bool *broadcast* = false) [virtual]

Write to the specified register on the camera.

Parameters

address	DCAM address to be written to.
value	The value to be written.
broadcast	Whether the action should be broadcast.

See also

ReadRegister()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.2.62 virtual Error WriteRegisterBlock (unsigned short *addressHigh*, unsigned int *addressLow*, const unsigned int ∗ *pBuffer*, unsigned int *length*) [virtual]

Write to the specified register block on the camera.

Parameters

addressHigh	Top 16 bits of the 48 bit absolute address to write to.
addressLow	Bottom 32 bits of the 48 bits absolute address to write to.
pBuffer	Array containing data to be written.
length	Size of array, in quadlets.

See also

ReadRegisterBlock()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.20.3 Member Data Documentation

8.20.3.1 BusManager m_busMgr [protected]

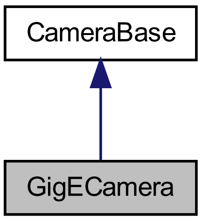
The documentation for this class was generated from the following file:

- GCCamera.h

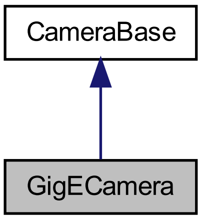
8.21 GigECamera Class Reference

The GigECamera object represents a physical Gigabit Ethernet camera.

Inheritance diagram for GigECamera:



Collaboration diagram for GigECamera:



Public Member Functions

- GigECamera ()

    Default constructor.
- virtual ∼GigECamera ()

    Default destructor.
- virtual Error Connect (PGRGuid ∗pGuid=NULL)

    The following functions are inherited from CameraBase.
- virtual Error Disconnect ()

    Disconnects the camera object from the camera.
- virtual bool IsConnected ()

    Checks if the camera object is currently connected to a physical camera.

- virtual Error SetCallback (ImageEventCallback callbackFn, const void ∗p-CallbackData=NULL)

  Sets the callback data to be used on completion of image transfer.
- virtual Error StartCapture (ImageEventCallback callbackFn=NULL, const void ∗p-CallbackData=NULL)

  Starts isochronous image capture.
- virtual Error RetrieveBuffer (Image ∗pImage)

  Retrieves the the next image object containing the next image.
- virtual Error StopCapture ()

  Stops isochronous image transfer and cleans up all associated resources.
- virtual Error WaitForBufferEvent (Image ∗pImage, unsigned int eventNumber)

  Retrieves the next image event containing the next part of the image.
- virtual Error SetUserBuffers (unsigned char ∗const pMemBuffers, int size, int numBuffers)

  Specify user allocated buffers to use as image data buffers.
- virtual Error GetConfiguration (FC2Config ∗pConfig)

  Get the configuration associated with the camera object.
- virtual Error SetConfiguration (const FC2Config ∗pConfig)

  Set the configuration associated with the camera object.
- virtual Error GetCameraInfo (CameraInfo ∗pCameraInfo)

  Retrieves information from the camera such as serial number, model name and other camera information.
- virtual Error GetPropertyInfo (PropertyInfo ∗pPropInfo)

  Retrieves information about the specified camera property.
- virtual Error GetProperty (Property ∗pProp)

  Reads the settings for the specified property from the camera.
- virtual Error SetProperty (const Property ∗pProp, bool broadcast=false)

  Writes the settings for the specified property to the camera.
- virtual Error GetGPIOPinDirection (unsigned int pin, unsigned int ∗pDirection)

  Get the GPIO pin direction for the specified pin.
- virtual Error SetGPIOPinDirection (unsigned int pin, unsigned int direction, bool broadcast=false)

  Set the GPIO pin direction for the specified pin.
- virtual Error GetTriggerModeInfo (TriggerModeInfo ∗pTriggerModeInfo)

  Retrieve trigger information from the camera.
- virtual Error GetTriggerMode (TriggerMode ∗pTriggerMode)

  Retrieve current trigger settings from the camera.
- virtual Error SetTriggerMode (const TriggerMode ∗pTriggerMode, bool broad-cast=false)

  Set the specified trigger settings to the camera.
- virtual Error FireSoftwareTrigger (bool broadcast=false)

  Fire the software trigger according to the DCAM specifications.
- virtual Error GetTriggerDelayInfo (TriggerDelayInfo ∗pTriggerDelayInfo)

  Retrieve trigger delay information from the camera.

- virtual Error GetTriggerDelay (TriggerDelay ∗pTriggerDelay)

  Retrieve current trigger delay settings from the camera.
- virtual Error SetTriggerDelay (const TriggerDelay ∗pTriggerDelay, bool broadcast=false)

  Set the specified trigger delay settings to the camera.
- virtual Error GetStrobeInfo (StrobeInfo ∗pStrobeInfo)

  Retrieve strobe information from the camera.
- virtual Error GetStrobe (StrobeControl ∗pStrobeControl)

  Retrieve current strobe settings from the camera.
- virtual Error SetStrobe (const StrobeControl ∗pStrobeControl, bool broadcast=false)

  Set current strobe settings to the camera.
- virtual Error GetLUTInfo (LUTData ∗pData)

  Query if LUT support is available on the camera.
- virtual Error GetLUTBankInfo (unsigned int bank, bool ∗pReadSupported, bool ∗pWriteSupported)

  Query the read/write status of a single LUT bank.
- virtual Error GetActiveLUTBank (unsigned int ∗pActiveBank)

  Get the LUT bank that is currently being used.
- virtual Error SetActiveLUTBank (unsigned int activeBank)

  Set the LUT bank that will be used.
- virtual Error EnableLUT (bool on)

  Enable or disable LUT functionality on the camera.
- virtual Error GetLUTChannel (unsigned int bank, unsigned int channel, unsigned int sizeEntries, unsigned int ∗pEntries)

  Get the LUT channel settings from the camera.
- virtual Error SetLUTChannel (unsigned int bank, unsigned int channel, unsigned int sizeEntries, const unsigned int ∗pEntries)

  Set the LUT channel settings to the camera.
- virtual Error GetMemoryChannel (unsigned int ∗pCurrentChannel)

  Retrieve the current memory channel from the camera.
- virtual Error SaveToMemoryChannel (unsigned int channel)

  Save the current settings to the specfied current memory channel.
- virtual Error RestoreFromMemoryChannel (unsigned int channel)

  Restore the specfied current memory channel.
- virtual Error GetMemoryChannelInfo (unsigned int ∗pNumChannels)

  Query the camera for memory channel support.
- virtual Error GetEmbeddedImageInfo (EmbeddedImageInfo ∗pInfo)

  Get the current status of the embedded image information register, as well as the availability of each embedded property.
- virtual Error SetEmbeddedImageInfo (EmbeddedImageInfo ∗pInfo)

  Sets the on/off values of the embedded image information structure to the camera.
- virtual Error WriteRegister (unsigned int address, unsigned int value, bool broadcast=false)

Write to the specified register on the camera.

- virtual Error ReadRegister (unsigned int address, unsigned int ∗pValue)

  Read the specified register from the camera.

- virtual Error WriteRegisterBlock (unsigned short addressHigh, unsigned int addressLow, const unsigned int ∗pBuffer, unsigned int length)

  Write to the specified register block on the camera.

- virtual Error ReadRegisterBlock (unsigned short addressHigh, unsigned int addressLow, unsigned int ∗pBuffer, unsigned int length)

  Read from the specified register block on the camera.

- Error GetCycleTime (TimeStamp ∗timeStamp)

  Returns a Timestamp struct containing 1394 CYCLE_TIME information.

- virtual Error GetStats (CameraStats ∗pStats)
- virtual Error ResetStats ()

Static Public Member Functions

- static Error StartSyncCapture (unsigned int numCameras, const GigECamera ∗∗ppCameras, const ImageEventCallback ∗pCallbackFns=NULL, const void ∗∗p-CallbackDataArray=NULL)
- static const char ∗ GetRegisterString (unsigned int registerVal)

  Returns a text representation of the register value.

GVCP Register Operation

These functions deal with GVCP register operation on the camera.

- virtual Error WriteGVCPRegister (unsigned int address, unsigned int value, bool broadcast=false)

  Write a GVCP register.

- virtual Error ReadGVCPRegister (unsigned int address, unsigned int ∗pValue)

  Read a GVCP register.

- virtual Error WriteGVCPRegisterBlock (unsigned int address, const unsigned int ∗pBuffer, unsigned int length)

  Write a GVCP register block.

- virtual Error ReadGVCPRegisterBlock (unsigned int address, unsigned int ∗p-Buffer, unsigned int length)

  Read a GVCP register block.

- virtual Error WriteGVCPMemory (unsigned int address, const unsigned char ∗p-Buffer, unsigned int length)

  Write a GVCP Memory block.

- virtual Error ReadGVCPMemory (unsigned int address, unsigned char ∗pBuffer, unsigned int length)

  Read a GVCP memory block.

GigE property manipulation

These functions deal with GigE properties.

- virtual Error GetGigEProperty (GigEProperty ∗pGigEProp)

  Get the specified GigEProperty.
- virtual Error SetGigEProperty (const GigEProperty ∗pGigEProp)

  Set the specified GigEProperty.
- virtual Error DiscoverGigEPacketSize (unsigned int ∗packetSize)

  Discover the largest packet size that works for the network link between the PC and the camera.

GigE image settings

These functions deal with GigE image setting.

- virtual Error QueryGigEImagingMode (Mode mode, bool ∗isSupported)

  Check if the particular imaging mode is supported by the camera.
- virtual Error GetGigEImagingMode (Mode ∗mode)

  Get the current imaging mode on the camera.
- virtual Error SetGigEImagingMode (Mode mode)

  Set the current imaging mode to the camera.
- virtual Error GetGigEImageSettingsInfo (GigEImageSettingsInfo ∗pInfo)

  Get information about the image settings possible on the camera.
- virtual Error GetGigEImageSettings (GigEImageSettings ∗pImageSettings)

  Get the current image settings on the camera.
- virtual Error SetGigEImageSettings (const GigEImageSettings ∗pImage-Settings)

  Set the image settings specified to the camera.

GigE image binning settings

These functions deal with GigE image binning settings.

- virtual Error GetGigEImageBinningSettings (unsigned int ∗horzBinnningValue, unsigned int ∗vertBinnningValue)

  Get the current binning settings on the camera.
- virtual Error SetGigEImageBinningSettings (unsigned int horzBinnningValue, unsigned int vertBinnningValue)

  Set the specified binning values to the camera.

GigE image stream configuration

These functions deal with GigE image stream configuration.

- virtual Error GetNumStreamChannels (unsigned int ∗numChannels)

  Get the number of stream channels present on the camera.
- virtual Error GetGigEStreamChannelInfo (unsigned int channel, GigEStreamChannel ∗pChannel)

  Get the stream channel information for the specified channel.
- virtual Error SetGigEStreamChannelInfo (unsigned int channel, GigEStreamChannel ∗pChannel)

  Set the stream channel information for the specified channel.
- virtual Error GetGigEConfig (GigEConfig ∗pGigEConfig)

  Get the current gige config on the camera.
- virtual Error SetGigEConfig (const GigEConfig ∗pGigEConfig)

  Set the gige config specified to the camera.

8.21.1 Detailed Description

The GigECamera object represents a physical Gigabit Ethernet camera.

The object must first be connected to using Connect() before any other operations can proceed.

Please see Camera.h for basic functions that this class inherits from.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 GigECamera ( )

Default constructor.

8.21.2.2 virtual ∼GigECamera ( ) [virtual]

Default destructor.

8.21.3 Member Function Documentation

8.21.3.1 virtual Error Connect (PGRGuid ∗ *pGuid* = NULL) [virtual]

The following functions are inherited from CameraBase.

See CameraBase.h for further information.

Implements CameraBase.

8.21.3.2 virtual Error Disconnect ( ) [virtual]

Disconnects the camera object from the camera.

This allows another physical camera to be connected to the camera object.

See also

Connect()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.3 virtual Error DiscoverGigEPacketSize (unsigned int ∗ *packetSize*) [virtual]

Discover the largest packet size that works for the network link between the PC and the camera.

This is useful in cases where there may be multiple links between the PC and the camera and there is a possiblity of a component not supporting the recommended jumbo frame packet size of 9000.

Parameters

packetSize	The maximum packet size supported by the link.

Returns

An Error indicating the success or failure of the function.

8.21.3.4 virtual Error EnableLUT (bool *on*) [virtual]

Enable or disable LUT functionality on the camera.

Parameters

on	Whether to enable or disable LUT.

See also

GetLUTInfo()
GetLUTChannel()
SetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.5 virtual **Error** FireSoftwareTrigger (bool *broadcast* = `false`) `[virtual]`

Fire the software trigger according to the DCAM specifications.

Parameters

broadcast	Whether the action should be broadcast.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.6 virtual **Error** GetActiveLUTBank (unsigned int ∗ *pActiveBank*) `[virtual]`

Get the LUT bank that is currently being used.

For cameras with PGR LUT, the active bank is always 0.

Parameters

pActiveBank	The currently active bank.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.7 virtual **Error** GetCameraInfo (**CameraInfo** ∗ *pCameraInfo*) `[virtual]`

Retrieves information from the camera such as serial number, model name and other camera information.

Parameters

pCameraInfo	Pointer to the camera information structure to be filled.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.8 virtual Error GetConfiguration (**FC2Config** ∗ *pConfig*) `[virtual]`

Get the configuration associated with the camera object.

Parameters

pConfig	Pointer to the configuration structure to be filled.

See also

SetConfiguration()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.9 Error GetCycleTime (**TimeStamp** ∗ *timeStamp*) `[virtual]`

Returns a Timestamp struct containing 1394 CYCLE_TIME information.

Parameters

registerVal	The register value to query.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.10 virtual Error GetEmbeddedImageInfo (**EmbeddedImageInfo** ∗ *pInfo*) `[virtual]`

Get the current status of the embedded image information register, as well as the availability of each embedded property.

Parameters

pInfo	Structure to be filled.

See also

SetEmbeddedImageInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.11 virtual **Error** GetGigEConfig (**GigEConfig** ∗ *pGigEConfig*) [virtual]

Get the current gige config on the camera.

Parameters

pGigEConfig	Current configuration on camera.

Returns

An Error indicating the success or failure of the function.

8.21.3.12 virtual **Error** GetGigEImageBinningSettings (unsigned int ∗ *horzBinnningValue,* unsigned int ∗ *vertBinnningValue*) [virtual]

Get the current binning settings on the camera.

Parameters

horz-Binning-Value	Current horizontal binning value.
vert-Binning-Value	Current vertical binning value.

Returns

An Error indicating the success or failure of the function.

8.21.3.13 virtual **Error** GetGigEImageSettings (**GigEImageSettings** ∗ *pImageSettings*) [virtual]

Get the current image settings on the camera.

Parameters

pImage-Settings	Current image settings on camera.

Returns

An Error indicating the success or failure of the function.

8.21.3.14 virtual Error GetGigEImageSettingsInfo (GigEImageSettingsInfo ∗ *pInfo*) [virtual]

Get information about the image settings possible on the camera.

Parameters

pInfo	Image settings information.

Returns

An Error indicating the success or failure of the function.

8.21.3.15 virtual Error GetGigEImagingMode (Mode ∗ *mode*) [virtual]

Get the current imaging mode on the camera.

Parameters

mode	Current imaging mode on the camera.

Returns

An Error indicating the success or failure of the function.

8.21.3.16 virtual Error GetGigEProperty (GigEProperty ∗ *pGigEProp*) [virtual]

Get the specified GigEProperty.

The GigEPropertyType field must be set in order for this function to succeed.

Parameters

pGigEProp	The GigE property to get.

Returns

An Error indicating the success or failure of the function.

8.21.3.17 virtual Error GetGigEStreamChannelInfo (unsigned int *channel*, GigEStreamChannel ∗ *pChannel*) [virtual]

Get the stream channel information for the specified channel.

Parameters

channel	Channel number to use.
pChannel	Stream channel information for the specified channel.

Returns

An Error indicating the success or failure of the function.

8.21.3.18 virtual Error GetGPIOPinDirection (unsigned int *pin*, unsigned int ∗ *pDirection*) [virtual]

Get the GPIO pin direction for the specified pin.

This is not a required call when using the trigger or strobe functions as the pin direction is set automatically internally.

Parameters

pin	Pin to get the direction for.
pDirection	Direction of the pin. 0 for input, 1 for output.

See also

SetGPIOPinDirection()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.19 virtual Error GetLUTBankInfo (unsigned int *bank*, bool ∗ *pReadSupported*, bool ∗ *pWriteSupported*) [virtual]

Query the read/write status of a single LUT bank.

Parameters

bank	The bank to query.
pRead-Supported	Whether reading from the bank is supported.
pWrite-Supported	Whether writing to the bank is supported.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.20 virtual Error GetLUTChannel (unsigned int *bank*, unsigned int *channel*, unsigned int *sizeEntries*, unsigned int ∗ *pEntries*) `[virtual]`

Get the LUT channel settings from the camera.

Parameters

bank	Bank to retrieve.
channel	Channel to retrieve.
sizeEntries	Number of entries in LUT table to read.
pEntries	Array to store LUT entries.

See also

GetLUTInfo()
EnableLUT()
SetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.21 virtual Error GetLUTInfo (LUTData ∗ *pData*) `[virtual]`

Query if LUT support is available on the camera.

Note that some cameras may report support for the LUT and return an inputBitDepth of 0. In these cases use log2(numEntries) for the inputBitDepth.

Parameters

pData	The LUT structure to be filled.

See also

EnableLUT()
GetLUTChannel()
SetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.22 virtual Error GetMemoryChannel (unsigned int ∗ *pCurrentChannel*) [virtual]

Retrieve the current memory channel from the camera.

Parameters

pCurrent-Channel	Current memory channel.

See also

SaveToMemoryChannel()
RestoreFromMemoryChannel()
GetMemoryChannelInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.23 virtual Error GetMemoryChannelInfo (unsigned int ∗ *pNumChannels*) [virtual]

Query the camera for memory channel support.

If the number of channels is 0, then memory channel support is not available.

Parameters

pNum-Channels	Number of memory channels supported.

See also

GetMemoryChannel()
SaveToMemoryChannel()
RestoreFromMemoryChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.24 virtual Error GetNumStreamChannels (unsigned int ∗ *numChannels*) `[virtual]`

Get the number of stream channels present on the camera.

Parameters

num-Channels	Number of stream channels present.

Returns

An Error indicating the success or failure of the function.

8.21.3.25 virtual Error GetProperty (Property ∗ *pProp*) `[virtual]`

Reads the settings for the specified property from the camera.

The property type must be specified in the Property structure passed into the function in order for the function to succeed. If auto is on, the integer and abs values returned may not be consistent with each other.

Parameters

pProp	Pointer to the Property structure to be filled.

See also

GetPropertyInfo()
SetProperty()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.26 virtual Error GetPropertyInfo (**PropertyInfo** ∗ *pPropInfo*) [virtual]

Retrieves information about the specified camera property.

The property type must be specified in the PropertyInfo structure passed into the function in order for the function to succeed.

Parameters

pPropInfo	Pointer to the PropertyInfo structure to be filled.

See also

GetProperty()
SetProperty()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.27 static const char∗ GetRegisterString (unsigned int *registerVal*) [static]

Returns a text representation of the register value.

Parameters

registerVal	The register value to query.

Returns

The text representation of the register.

Reimplemented from CameraBase.

8.21.3.28 virtual Error GetStats (**CameraStats** ∗ *pStats*) [virtual]

Implements CameraBase.

8.21.3.29 virtual Error GetStrobe (**StrobeControl** ∗ *pStrobeControl*) [virtual]

Retrieve current strobe settings from the camera.

The strobe pin must be specified in the structure before being passed in to the function.

Parameters

pStrobe-Control	Structure to receive strobe settings.

See also

GetStrobeInfo()
SetStrobe()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.30 virtual Error GetStrobeInfo (StrobeInfo ∗ *pStrobeInfo*) [virtual]

Retrieve strobe information from the camera.

Parameters

pStrobeInfo	Structure to receive strobe information.

See also

GetStrobe()
SetStrobe()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.31 virtual Error GetTriggerDelay (TriggerDelay ∗ *pTriggerDelay*) [virtual]

Retrieve current trigger delay settings from the camera.

Parameters

pTrigger-Delay	Structure to receive trigger delay settings.

See also

GetTriggerModeInfo()
GetTriggerMode()

SetTriggerMode()
GetTriggerDelayInfo()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.32 virtual Error GetTriggerDelayInfo (TriggerDelayInfo ∗ *pTriggerDelayInfo*) [virtual]

Retrieve trigger delay information from the camera.

Parameters

pTrigger-DelayInfo	Structure to receive trigger delay information.

See also

GetTriggerModeInfo()
GetTriggerMode()
SetTriggerMode()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.33 virtual Error GetTriggerMode (TriggerMode ∗ *pTriggerMode*) [virtual]

Retrieve current trigger settings from the camera.

Parameters

pTrigger-Mode	Structure to receive trigger mode settings.

See also

GetTriggerModeInfo()
SetTriggerMode()

GetTriggerDelayInfo()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.34 virtual Error GetTriggerModeInfo (TriggerModeInfo ∗ *pTriggerModeInfo*) `[virtual]`

Retrieve trigger information from the camera.

Parameters

pTrigger-ModeInfo	Structure to receive trigger information.

See also

GetTriggerMode()
SetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.35 virtual bool IsConnected ( ) `[virtual]`

Checks if the camera object is currently connected to a physical camera.

See also

Connect()
Disconnect()

Returns

Whether the camera object is connected to a physical camera.

Implements CameraBase.

8.21.3.36 virtual Error QueryGigEImagingMode (Mode *mode*, bool ∗ *isSupported*) [virtual]

Check if the particular imaging mode is supported by the camera.

Parameters

mode	The mode to check.
isSupported	Whether the mode is supported.

Returns

An Error indicating the success or failure of the function.

8.21.3.37 virtual Error ReadGVCPMemory (unsigned int *address*, unsigned char ∗ *pBuffer*, unsigned int *length*) [virtual]

Read a GVCP memory block.

Parameters

address	GVCP address to be read from.
pBuffer	Array for data to be read into.
length	Size of array, in quadlets.

Returns

An Error indicating the success or failure of the function.

8.21.3.38 virtual Error ReadGVCPRegister (unsigned int *address*, unsigned int ∗ *pValue*) [virtual]

Read a GVCP register.

Parameters

address	GVCP address to be read from.
pValue	The value that is read.

Returns

An Error indicating the success or failure of the function.

8.21.3.39 virtual Error ReadGVCPRegisterBlock (unsigned int *address*, unsigned int ∗ *pBuffer*, unsigned int *length*) `[virtual]`

Read a GVCP register block.

Parameters

address	GVCP address to be read from.
pBuffer	Array for data to be read into.
length	Size of array, in quadlets.

Returns

An Error indicating the success or failure of the function.

8.21.3.40 virtual Error ReadRegister (unsigned int *address*, unsigned int ∗ *pValue*) `[virtual]`

Read the specified register from the camera.

Parameters

address	DCAM address to be read from.
pValue	The value that is read.

See also

WriteRegister()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.41 virtual Error ReadRegisterBlock (unsigned short *addressHigh*, unsigned int *addressLow*, unsigned int ∗ *pBuffer*, unsigned int *length*) `[virtual]`

Read from the specified register block on the camera.

Parameters

addressHigh	Top 16 bits of the 48 bit absolute address to read from.
addressLow	Bottom 32 bits of the 48 bits absolute address to read from.
pBuffer	Array to store read data.
length	Size of array, in quadlets.

See also

WriteRegisterBlock()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.42 virtual Error ResetStats ( ) `[virtual]`

Implements CameraBase.

8.21.3.43 virtual Error RestoreFromMemoryChannel (unsigned int *channel*) `[virtual]`

Restore the specfied current memory channel.

Parameters

channel	Memory channel to restore from.

See also

GetMemoryChannel()
SaveToMemoryChannel()
GetMemoryChannelInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.44 virtual Error RetrieveBuffer (Image $*$ *pImage*) `[virtual]`

Retrieves the the next image object containing the next image.

If the grab mode has not been set, or has been set to DROP_FRAMES the default behavior is to requeue images for DMA if they have not been retrieved by the time the next image transfer completes. If BUFFER_FRAMES is specified, the next image in the sequence will be retrieved. Note that for the BUFFER_FRAMES case, if retrieval does not keep up with the DMA process, images will be lost. The default behavior is to perform DROP_FRAMES image retrieval.

Parameters

pImage	Pointer to Image object to store image data.

See also

StartCapture()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.45 virtual Error SaveToMemoryChannel (unsigned int *channel*) [virtual]

Save the current settings to the specfied current memory channel.

Parameters

channel	Memory channel to save to.

See also

GetMemoryChannel()
RestoreFromMemoryChannel()
GetMemoryChannelInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.46 virtual Error SetActiveLUTBank (unsigned int *activeBank*) [virtual]

Set the LUT bank that will be used.

Parameters

activeBank	The bank to be set as active.

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.47 virtual **Error** SetCallback (**ImageEventCallback** *callbackFn*, const void ∗ *pCallbackData* = `NULL`) `[virtual]`

Sets the callback data to be used on completion of image transfer.

To clear the current stored callback data, pass in NULL for both arguments.

Parameters

callbackFn	A function to be called when a new image is received.
pCallback-Data	A pointer to data that can be passed to the callback function.

See also

StartCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.48 virtual **Error** SetConfiguration (const **FC2Config** ∗ *pConfig*) `[virtual]`

Set the configuration associated with the camera object.

Parameters

pConfig	Pointer to the configuration structure to be used.

See also

GetConfiguration()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.49 virtual Error SetEmbeddedImageInfo (EmbeddedImageInfo ∗ *pInfo*) [virtual]

Sets the on/off values of the embedded image information structure to the camera.

Parameters

pInfo	Structure to be used.

See also

GetEmbeddedImageInfo()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.50 virtual Error SetGigEConfig (const GigEConfig ∗ *pGigEConfig*) [virtual]

Set the gige config specified to the camera.

Parameters

pGigEConfig	configuration to set to camera.

Returns

An Error indicating the success or failure of the function.

8.21.3.51 virtual Error SetGigEImageBinningSettings (unsigned int *horzBinnningValue*, unsigned int *vertBinnningValue*) [virtual]

Set the specified binning values to the camera.

It is recommended that GetGigEImageSettingsInfo() be called after this function succeeds to retrieve the new image settings information for the new binning mode.

Parameters

horz-Binnning-Value	Horizontal binning value.
vert-Binnning-Value	Vertical binning value.

Returns

An Error indicating the success or failure of the function.

8.21.3.52 virtual Error SetGigEImageSettings (const GigEImageSettings ∗ *pImageSettings*) [virtual]

Set the image settings specified to the camera.

Parameters

pImage-Settings	Image settings to set to camera.

Returns

An Error indicating the success or failure of the function.

8.21.3.53 virtual Error SetGigEImagingMode (Mode *mode*) [virtual]

Set the current imaging mode to the camera.

This should only be done when the camera is not streaming images.

Parameters

mode	Imaging mode to set to the camera.

Returns

An Error indicating the success or failure of the function.

8.21.3.54 virtual Error SetGigEProperty (const GigEProperty ∗ *pGigEProp*) [virtual]

Set the specified GigEProperty.

The GigEPropertyType field must be set in order for this function to succeed.

Parameters

pGigEProp	The GigE property to set.

Returns

An Error indicating the success or failure of the function.

8.21.3.55 virtual Error SetGigEStreamChannelInfo (unsigned int *channel*, GigEStreamChannel ∗ *pChannel*) [virtual]

Set the stream channel information for the specified channel.

Note that the source UDP port of the stream channel is read-only.

Parameters

channel	Channel number to use.
pChannel	Stream channel information to use for the specified channel.

Returns

An Error indicating the success or failure of the function.

8.21.3.56 virtual Error SetGPIOPinDirection (unsigned int *pin*, unsigned int *direction*, bool *broadcast* = false) [virtual]

Set the GPIO pin direction for the specified pin.

This is useful if there is a need to set the pin into an input pin (i.e. to read the voltage) off the pin without setting it as a trigger source. This is not a required call when using the trigger or strobe functions as the pin direction is set automatically internally.

Parameters

pin	Pin to get the direction for.
direction	Direction of the pin. 0 for input, 1 for output.
broadcast	Whether the action should be broadcast.

See also

GetGPIOPinDirection()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.57 virtual Error SetLUTChannel (unsigned int *bank*, unsigned int *channel*, unsigned int *sizeEntries*, const unsigned int ∗ *pEntries*) [virtual]

Set the LUT channel settings to the camera.

Parameters

bank	Bank to set.
channel	Channel to set.
sizeEntries	Number of entries in LUT table to write. This must be the same size as numEntries returned by GetLutInfo().
pEntries	Array containing LUT entries to write.

See also

GetLUTInfo()
EnableLUT()
GetLUTChannel()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.58 virtual Error SetProperty (const Property * *pProp*, bool *broadcast* = `false`) `[virtual]`

Writes the settings for the specified property to the camera.

The property type must be specified in the Property structure passed into the function in order for the function to succeed. The absControl flag controls whether the absolute or integer value is written to the camera.

Parameters

pProp	Pointer to the Property structure to be used.
broadcast	Whether the action should be broadcast.

See also

GetPropertyInfo()
GetProperty()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.59 virtual Error SetStrobe (const StrobeControl * *pStrobeControl*, bool *broadcast* = `false`) `[virtual]`

Set current strobe settings to the camera.

The strobe pin must be specified in the structure before being passed in to the function.

Parameters

pStrobe-Control	Structure providing strobe settings.
broadcast	Whether the action should be broadcast.

See also

GetStrobeInfo()
GetStrobe()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.60 virtual **Error** SetTriggerDelay (const **TriggerDelay** $*$ *pTriggerDelay,* bool *broadcast* = false) [virtual]

Set the specified trigger delay settings to the camera.

Parameters

pTrigger-Delay	Structure providing trigger delay settings.
broadcast	Whether the action should be broadcast.

See also

GetTriggerModeInfo()
GetTriggerMode()
SetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.61 virtual **Error** SetTriggerMode (const **TriggerMode** $*$ *pTriggerMode,* bool *broadcast* = false) [virtual]

Set the specified trigger settings to the camera.

Parameters

pTrigger-Mode	Structure providing trigger mode settings.
broadcast	Whether the action should be broadcast.

See also

GetTriggerModeInfo()
GetTriggerMode()
GetTriggerDelayInfo()
GetTriggerDelay()
SetTriggerDelay()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.62 virtual Error SetUserBuffers (unsigned char ∗const *pMemBuffers,* int *size,* int *numBuffers*) `[virtual]`

Specify user allocated buffers to use as image data buffers.

To prevent image tearing, the size of each buffer should be equal to ((unsigned int)(bufferSize + packetSize - 1)/packetSize) ∗ packetSize. The total size should be (size ∗ numBuffers) or larger.

Parameters

pMem-Buffers	Pointer to memory buffers to be written to.
size	The size of each buffer (in bytes).
numBuffers	Number of buffers in the array.

See also

StartCapture()
RetrieveBuffer()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.63 virtual Error StartCapture (ImageEventCallback *callbackFn* = NULL, const void * *pCallbackData* = NULL) [virtual]

Starts isochronous image capture.

It will use either the current video mode or the most recently set video mode of the camera. The optional callback function parameter is called on completion of image transfer. Alternatively, the callback parameter can be set to NULL and RetrieveBuffer() can be called as a blocking call to get the image data.

Parameters

callbackFn	A function to be called when a new image is received.
pCallback-Data	A pointer to data that can be passed to the callback function.

See also

RetrieveBuffer()
StartSyncCapture()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.64 static Error StartSyncCapture (unsigned int *numCameras*, const GigECamera ** *ppCameras*, const ImageEventCallback * *pCallbackFns* = NULL, const void ** *pCallbackDataArray* = NULL) [static]

8.21.3.65 virtual Error StopCapture ( ) [virtual]

Stops isochronous image transfer and cleans up all associated resources.

If an image callback function (specified in the StartCapture() call) is currently executing, StopCapture() will not return until after the callback has completed.

See also

StartCapture()
RetrieveBuffer()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.66 virtual **Error** WaitForBufferEvent (**Image** ∗ ***pImage,*** unsigned int ***eventNumber***) `[virtual]`

Retrieves the next image event containing the next part of the image.

Parameters

pImage	Pointer to Image object to store image data.
event-Number	The event number to wait for.

See also

StartCapture()
RetrieveBuffer()
StopCapture()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.67 virtual **Error** WriteGVCPMemory (unsigned int ***address,*** const unsigned char ∗ ***pBuffer,*** unsigned int ***length***) `[virtual]`

Write a GVCP Memory block.

Parameters

address	GVCP address to be write to.
pBuffer	Array containing data to be written in increments.
length	Size of array, in quadlets.

Returns

An Error indicating the success or failure of the function.

8.21.3.68 virtual **Error** WriteGVCPRegister (unsigned int ***address,*** unsigned int ***value,*** bool ***broadcast*** = `false`) `[virtual]`

Write a GVCP register.

Parameters

address	GVCP address to be written to.
value	The value to be written.
broadcast	Whether the action should be broadcast.

Returns

An Error indicating the success or failure of the function.

8.21.3.69 virtual Error WriteGVCPRegisterBlock (unsigned int *address*, const unsigned int ∗ *pBuffer*, unsigned int *length*) [virtual]

Write a GVCP register block.

Parameters

address	GVCP address to be write to.
pBuffer	Array containing data to be written.
length	Size of array, in quadlets.

Returns

An Error indicating the success or failure of the function.

8.21.3.70 virtual Error WriteRegister (unsigned int *address*, unsigned int *value*, bool *broadcast* = false) [virtual]

Write to the specified register on the camera.

Parameters

address	DCAM address to be written to.
value	The value to be written.
broadcast	Whether the action should be broadcast.

See also

ReadRegister()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

8.21.3.71 virtual Error WriteRegisterBlock (unsigned short *addressHigh*, unsigned int *addressLow*, const unsigned int ∗ *pBuffer*, unsigned int *length*) [virtual]

Write to the specified register block on the camera.

Parameters

addressHigh	Top 16 bits of the 48 bit absolute address to write to.
addressLow	Bottom 32 bits of the 48 bits absolute address to write to.
pBuffer	Array containing data to be written.
length	Size of array, in quadlets.

See also

ReadRegisterBlock()

Returns

An Error indicating the success or failure of the function.

Implements CameraBase.

The documentation for this class was generated from the following file:

- GigECamera.h

8.22 GigEConfig Struct Reference

Configuration for a GigE camera.

Public Member Functions

- GigEConfig ()

Public Attributes

- bool enablePacketResend

  Turn on/off packet resend functionality.
- unsigned int registerTimeoutRetries

  Number of retries to perform when a register read/write timeout is received by the library.
- unsigned int registerTimeout

  Register read/write timeout value, in microseconds.

8.22.1 Detailed Description

Configuration for a GigE camera.

These options are options that are generally should be set before starting isochronous transfer.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 GigEConfig ( ) [inline]

8.22.3 Member Data Documentation

8.22.3.1 bool enablePacketResend

Turn on/off packet resend functionality.

8.22.3.2 unsigned int registerTimeout

Register read/write timeout value, in microseconds.

The default value is dependent on the interface type.

8.22.3.3 unsigned int registerTimeoutRetries

Number of retries to perform when a register read/write timeout is received by the library.

The default value is 0.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.23 GigEImageSettings Struct Reference

Image settings for a GigE camera.

Public Member Functions

- GigEImageSettings ()

Public Attributes

- unsigned int offsetX

  Horizontal image offset.
- unsigned int offsetY

  Vertical image offset.
- unsigned int width

  Width of image.
- unsigned int height

  Height of image.

- PixelFormat pixelFormat

  Pixel format of image.
- unsigned int reserved [8]

  Reserved for future use.

8.23.1 Detailed Description

Image settings for a GigE camera.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 GigEImageSettings ( ) [inline]

8.23.3 Member Data Documentation

8.23.3.1 unsigned int height

Height of image.

8.23.3.2 unsigned int offsetX

Horizontal image offset.

8.23.3.3 unsigned int offsetY

Vertical image offset.

8.23.3.4 PixelFormat pixelFormat

Pixel format of image.

8.23.3.5 unsigned int reserved[8]

Reserved for future use.

8.23.3.6 unsigned int width

Width of image.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.24 GigEImageSettingsInfo Struct Reference

Format 7 information for a single mode.

Public Member Functions

- GigEImageSettingsInfo ()

Public Attributes

- unsigned int maxWidth

  Maximum image width.
- unsigned int maxHeight

  Maximum image height.
- unsigned int offsetHStepSize

  Horizontal step size for the offset.
- unsigned int offsetVStepSize

  Vertical step size for the offset.
- unsigned int imageHStepSize

  Horizontal step size for the image.
- unsigned int imageVStepSize

  Vertical step size for the image.
- unsigned int pixelFormatBitField

  Supported pixel formats in a bit field.
- unsigned int vendorPixelFormatBitField

  Vendor unique pixel formats in a bit field.
- unsigned int reserved [16]

  Reserved for future use.

8.24.1 Detailed Description

Format 7 information for a single mode.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 GigEImageSettingsInfo ( ) `[inline]`

8.24.3 Member Data Documentation

8.24.3.1 unsigned int imageHStepSize

Horizontal step size for the image.

8.24.3.2 unsigned int imageVStepSize

Vertical step size for the image.

8.24.3.3 unsigned int maxHeight

Maximum image height.

8.24.3.4 unsigned int maxWidth

Maximum image width.

8.24.3.5 unsigned int offsetHStepSize

Horizontal step size for the offset.

8.24.3.6 unsigned int offsetVStepSize

Vertical step size for the offset.

8.24.3.7 unsigned int pixelFormatBitField

Supported pixel formats in a bit field.

8.24.3.8 unsigned int reserved[16]

Reserved for future use.

8.24.3.9 unsigned int vendorPixelFormatBitField

Vendor unique pixel formats in a bit field.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.25 GigEProperty Struct Reference

A GigE property.

Public Attributes

- GigEPropertyType propType

  The type of property.
- bool isReadable

  Whether the property is readable.
- bool isWritable

  Whether the property is writable.
- unsigned int min

  Minimum value.
- unsigned int max

  Maximum value.
- unsigned int value

  Current value.

8.25.1 Detailed Description

A GigE property.

8.25.2 Member Data Documentation

8.25.2.1 bool **isReadable**

Whether the property is readable.

If this is false, then no other value in this structure is valid.

8.25.2.2 bool **isWritable**

Whether the property is writable.

8.25.2.3 unsigned int **max**

Maximum value.

8.25.2.4 unsigned int **min**

Minimum value.

8.25.2.5 GigEPropertyType **propType**

The type of property.

8.25.2.6 unsigned int value

Current value.

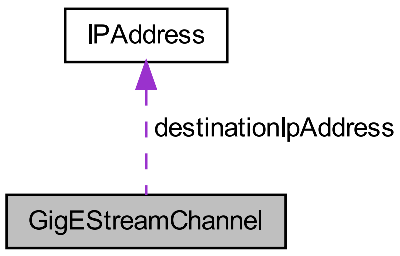
The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.26 GigEStreamChannel Struct Reference

Information about a single GigE stream channel.

Collaboration diagram for GigEStreamChannel:



Public Member Functions

- GigEStreamChannel ()

Public Attributes

- unsigned int networkInterfaceIndex

    Network interface index used (or to use).
- unsigned int hostPost

    Host port on the PC where the camera will send the data stream.
- bool doNotFragment

    Disable IP fragmentation of packets.
- unsigned int packetSize

    Packet size, in bytes.
- unsigned int interPacketDelay

Inter packet delay, in timestamp counter units.
- IPAddress destinationIpAddress

  Destination IP address.
- unsigned int sourcePort

  Source UDP port of the stream channel.

8.26.1 Detailed Description

Information about a single GigE stream channel.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 GigEStreamChannel ( ) `[inline]`

8.26.3 Member Data Documentation

8.26.3.1 IPAddress destinationIpAddress

Destination IP address.

It can be a multicast or unicast address.

8.26.3.2 bool doNotFragment

Disable IP fragmentation of packets.

8.26.3.3 unsigned int hostPost

Host port on the PC where the camera will send the data stream.

8.26.3.4 unsigned int interPacketDelay

Inter packet delay, in timestamp counter units.

8.26.3.5 unsigned int networkInterfaceIndex

Network interface index used (or to use).

8.26.3.6 unsigned int packetSize

Packet size, in bytes.

8.26.3.7 unsigned int sourcePort

Source UDP port of the stream channel.

Read only.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.27 H264Option Struct Reference

Options for saving H264 files.

Public Member Functions

- H264Option ()

Public Attributes

- float frameRate

    Frame rate of the stream.
- unsigned int width

    Width of source image.
- unsigned int height

    Height of source image.
- unsigned int bitrate

    Bitrate to encode at.
- unsigned int reserved [256]

    Reserved for future use.

8.27.1 Detailed Description

Options for saving H264 files.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 H264Option ( ) `[inline]`

8.27.3 Member Data Documentation

8.27.3.1 unsigned int bitrate

Bitrate to encode at.

8.27.3.2 float frameRate

Frame rate of the stream.

8.27.3.3 unsigned int height

Height of source image.

8.27.3.4 unsigned int reserved[256]

Reserved for future use.

8.27.3.5 unsigned int width

Width of source image.

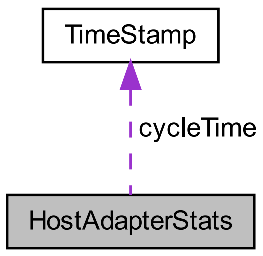
The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.28 HostAdapterStats Struct Reference

Information about the host adapter's statistics.

Collaboration diagram for HostAdapterStats:



Public Member Functions

- HostAdapterStats ()

Public Attributes

- char vendor [sk_maxStringLength]
- unsigned int numPorts
- unsigned int portErrors [sk_maxNumPorts]
- unsigned int fifoOverflows
- unsigned int busResets
- unsigned int deviceArrivals
- unsigned int deviceRemovals
- unsigned int busErrors
- unsigned int gapCount
- unsigned int busNumber
- bool AnotherDeviceConnected
- TimeStamp cycleTime

8.28.1 Detailed Description

Information about the host adapter's statistics.

8.28.2 Constructor & Destructor Documentation

8.28.2.1 HostAdapterStats ( ) `[inline]`

8.28.3 Member Data Documentation

8.28.3.1 bool **AnotherDeviceConnected**

8.28.3.2 unsigned int **busErrors**

8.28.3.3 unsigned int **busNumber**

8.28.3.4 unsigned int **busResets**

8.28.3.5 **TimeStamp cycleTime**

8.28.3.6 unsigned int **deviceArrivals**

8.28.3.7 unsigned int **deviceRemovals**

8.28.3.8 unsigned int **fifoOverflows**

8.28.3.9 unsigned int **gapCount**

8.28.3.10 unsigned int **numPorts**

8.28.3.11 unsigned int **portErrors[sk_maxNumPorts]**

8.28.3.12 char vendor[sk_maxStringLength]

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.29 Image Class Reference

The Image class is used to retrieve images from a camera, convert between multiple pixel formats and save images to disk.

Public Member Functions

- Image ()

  Default constructor.
- Image (unsigned int rows, unsigned int cols, unsigned int stride, unsigned char ∗pData, unsigned int dataSize, PixelFormat format, BayerTileFormat bayerFormat=NONE)

  Construct an Image object with the specified arguments.
- Image (unsigned int rows, unsigned int cols, unsigned int stride, unsigned char ∗pData, unsigned int dataSize, unsigned int receivedDataSize, PixelFormat format, BayerTileFormat bayerFormat=NONE)

  Construct an Image object with the specified arguments.
- Image (unsigned char ∗pData, unsigned int dataSize)

  Construct an Image object with the specified arguments.
- Image (unsigned int rows, unsigned int cols, PixelFormat format, BayerTileFormat bayerFormat=NONE)

  Construct an Image object with the specified arguments.
- Image (const Image &image)

  Copy constructor.
- virtual ∼Image ()

  Default destructor.
- virtual Image & operator= (const Image &image)

  Assignment operator.
- virtual unsigned char ∗ operator[] (unsigned int index)

  Indexing operator.
- virtual unsigned char ∗ operator() (unsigned int row, unsigned int col)

  Indexing operator.
- virtual Error DeepCopy (const Image ∗pImage)

  Perform a deep copy of the Image.
- virtual Error SetDimensions (unsigned int rows, unsigned int cols, unsigned int stride, PixelFormat pixelFormat, BayerTileFormat bayerFormat)

  Sets the dimensions of the image object.

- virtual Error SetData (const unsigned char ∗pData, unsigned int dataSize)

  Set the data of the Image object.
- virtual Error SetBlockId (const unsigned int blockId)

  Set the block id of the Image object.
- virtual unsigned int GetBlockId ()

  get the block id of the Image object.
- virtual PixelFormat GetPixelFormat () const

  Get the current pixel format.
- virtual ColorProcessingAlgorithm GetColorProcessing () const

  Get the current color processing algorithm.
- virtual Error SetColorProcessing (ColorProcessingAlgorithm colorProc)

  Set the color processing algorithm.
- virtual unsigned int GetCols () const

  Get the number of columns in the image.
- virtual unsigned int GetRows () const

  Get the number of rows in the image.
- virtual unsigned int GetStride () const

  Get the stride in the image.
- virtual unsigned int GetBitsPerPixel () const

  Get the bits per pixel of the image.
- virtual BayerTileFormat GetBayerTileFormat () const

  Get the Bayer tile format of the image.
- virtual unsigned int GetDataSize () const

  Get the size of the buffer associated with the image, in bytes.
- virtual unsigned int GetReceivedDataSize () const

  Get the size of the compressed data, in bytes.
- virtual void GetDimensions (unsigned int ∗pRows, unsigned int ∗pCols=NUL-L, unsigned int ∗pStride=NULL, PixelFormat ∗pPixelFormat=NULL, BayerTile-Format ∗pBayerFormat=NULL) const

  Get the image dimensions associated with the image.
- virtual unsigned char ∗ GetData ()

  Get a pointer to the data associated with the image.
- virtual unsigned char ∗const GetData () const
- virtual ImageMetadata GetMetadata () const

  Get the metadata associated with the image.
- virtual Error CalculateStatistics (ImageStatistics ∗pStatistics)

  Calculate statistics associated with the image.
- virtual TimeStamp GetTimeStamp () const

  Get the timestamp data associated with the image.
- virtual Error Save (const char ∗pFilename, ImageFileFormat format=FROM_FIL-E_EXT)

  Save the image to the specified file name with the file format specified.
- virtual Error Save (const char ∗pFilename, PNGOption ∗pOption)

Save the image to the specified file name with the options specified.

- virtual Error Save (const char ∗pFilename, PPMOption ∗pOption)

  Save the image to the specified file name with the options specified.
- virtual Error Save (const char ∗pFilename, PGMOption ∗pOption)

  Save the image to the specified file name with the options specified.
- virtual Error Save (const char ∗pFilename, TIFFOption ∗pOption)

  Save the image to the specified file name with the options specified.
- virtual Error Save (const char ∗pFilename, JPEGOption ∗pOption)

  Save the image to the specified file name with the options specified.
- virtual Error Save (const char ∗pFilename, JPG2Option ∗pOption)

  Save the image to the specified file name with the options specified.
- virtual Error Save (const char ∗pFilename, BMPOption ∗pOption)

  Save the image to the specified file name with the options specified.
- virtual Error Convert (PixelFormat format, Image ∗pDestImage) const

  Converts the current image buffer to the specified output format and stores the result in the specified image.
- virtual Error Convert (Image ∗pDestImage) const

  Converts the current image buffer to the specified output format and stores the result in the specified image.
- virtual Error ReleaseBuffer ()

  Release the buffer associated with the Image.

Static Public Member Functions

- static Error SetDefaultColorProcessing (ColorProcessingAlgorithm defaultMethod)

  Set the default color processing algorithm.
- static ColorProcessingAlgorithm GetDefaultColorProcessing ()

  Get the default color processing algorithm.
- static Error SetDefaultOutputFormat (PixelFormat format)

  Set the default output pixel format.
- static PixelFormat GetDefaultOutputFormat ()

  Get the default output pixel format.
- static unsigned int DetermineBitsPerPixel (PixelFormat format)

  Calculate the bits per pixel for the specified pixel format.

Friends

- class Iso

8.29.1 Detailed Description

The Image class is used to retrieve images from a camera, convert between multiple pixel formats and save images to disk.

Operations on Image objects are not guaranteed to be thread safe. It is recommended that operations on Image objects be protected by thread synchronization constructs such as mutexes.

8.29.2 Constructor & Destructor Documentation

8.29.2.1 Image ( )

Default constructor.

8.29.2.2 Image (unsigned int *rows,* unsigned int *cols,* unsigned int *stride,* unsigned char ∗ *pData,* unsigned int *dataSize,* PixelFormat *format,* BayerTileFormat *bayerFormat* = `NONE`)

Construct an Image object with the specified arguments.

Ownership of the image buffer is not transferred to the Image object. It is the user's responsibility to delete the buffer when it is no longer in use.

Parameters

rows	Rows in the image.
cols	Columns in the image.
stride	Stride of the image buffer.
pData	Pointer to the image buffer.
dataSize	Size of the image buffer.
format	Pixel format.
bayerFormat	Format of the Bayer tiled raw image.

8.29.2.3 Image (unsigned int *rows,* unsigned int *cols,* unsigned int *stride,* unsigned char ∗ *pData,* unsigned int *dataSize,* unsigned int *receivedDataSize,* PixelFormat *format,* BayerTileFormat *bayerFormat* = `NONE`)

Construct an Image object with the specified arguments.

Ownership of the image buffer is not transferred to the Image object. It is the user's responsibility to delete the buffer when it is no longer in use.

Parameters

rows	Rows in the image.
cols	Columns in the image.
stride	Stride of the image buffer.
pData	Pointer to the image buffer.

dataSize	Size of the image buffer.
received-DataSize	Actual size of data.
format	Pixel format.
bayerFormat	Format of the Bayer tiled raw image.

8.29.2.4 Image (unsigned char ∗ *pData,* unsigned int *dataSize*)

Construct an Image object with the specified arguments.

Ownership of the image buffer is not transferred to the Image object. It is the user's responsibility to delete the buffer when it is no longer in use.

Parameters

pData	Pointer to the image buffer.
dataSize	Size of the image buffer.

8.29.2.5 Image (unsigned int *rows,* unsigned int *cols,* PixelFormat *format,* BayerTileFormat *bayerFormat* = `NONE`)

Construct an Image object with the specified arguments.

Parameters

rows	Rows in the image.
cols	Columns in the image.
format	Pixel format.
bayerFormat	Format of the Bayer tiled raw image.

8.29.2.6 Image (const Image & *image*)

Copy constructor.

Both images will point to the same image buffer internally.

8.29.2.7 virtual ∼Image ( ) `[virtual]`

Default destructor.

The internal image buffer will be released if there are no other Image objects holding a reference to it. This will also allow the buffer to be requeued internally.

8.29.3 Member Function Documentation

8.29.3.1 virtual Error CalculateStatistics (ImageStatistics ∗ *pStatistics*) [virtual]

Calculate statistics associated with the image.

In order to collect statistics for a particular channel, the enabled flag for the channel must be set to true. Statistics can only be collected for images in Mono8, Mono16, RGB, RGBU, BGR and BGRU.

Parameters

pStatistics	The ImageStatistics object to hold the statistics.

Returns

An Error indicating the success or failure of the function.

8.29.3.2 virtual Error Convert (PixelFormat *format,* Image ∗ *pDestImage*) const [virtual]

Converts the current image buffer to the specified output format and stores the result in the specified image.

The destination image does not need to be configured in any way before the call is made.

Parameters

format	Output format of the converted image.
pDestImage	Destination image.

Returns

An Error indicating the success or failure of the function.

8.29.3.3 virtual Error Convert (Image ∗ *pDestImage*) const [virtual]

Converts the current image buffer to the specified output format and stores the result in the specified image.

The destination image does not need to be configured in anyway before the call is made.

Parameters

pDestImage	Destination image.

Returns

An Error indicating the success or failure of the function.

8.29.3.4 virtual Error DeepCopy (const Image * *pImage*) [virtual]

Perform a deep copy of the Image.

After this operation, the image contents and member variables will be the same. The Images will not share a buffer. The Image's current buffer will not be released.

Parameters

pImage	The Image to copy the data from.

Returns

An Error indicating the success or failure of the function.

8.29.3.5 static unsigned int DetermineBitsPerPixel (PixelFormat *format*) [static]

Calculate the bits per pixel for the specified pixel format.

Parameters

format	The pixel format.

Returns

The bits per pixel.

8.29.3.6 virtual BayerTileFormat GetBayerTileFormat ( ) const [virtual]

Get the Bayer tile format of the image.

Returns

The Bayer tile format.

8.29.3.7 virtual unsigned int GetBitsPerPixel ( ) const [virtual]

Get the bits per pixel of the image.

Returns

The bits per pixel.

8.29.3.8 virtual unsigned int GetBlockId ( ) [virtual]

get the block id of the Image object.

Returns

The blockId assigned to the image.

8.29.3.9 virtual ColorProcessingAlgorithm GetColorProcessing ( ) const [virtual]

Get the current color processing algorithm.

See also

SetColorProcessing()

Returns

The current color processing algorithm.

8.29.3.10 virtual unsigned int GetCols ( ) const [virtual]

Get the number of columns in the image.

Returns

The number of columns.

8.29.3.11 virtual unsigned char∗ GetData ( ) [virtual]

Get a pointer to the data associated with the image.

This function is considered unsafe. The pointer returned could be invalidated if the buffer is resized or released. The pointer may also be invalidated if the Image object is passed to Camera::RetrieveBuffer(). It is recommended that a Image::DeepCopy() be performed if a seperate copy of the Image data is required for further processing.

Returns

A pointer to the image data.

8.29.3.12 virtual unsigned char∗ const GetData ( ) const [virtual]

8.29.3.13 virtual unsigned int GetDataSize ( ) const [virtual]

Get the size of the buffer associated with the image, in bytes.

Returns

The size of the buffer, in bytes.

8.29.3.14 static ColorProcessingAlgorithm GetDefaultColorProcessing ( ) [static]

Get the default color processing algorithm.

See also

SetDefaultColorProcessing()

Returns

The default color processing algorithm.

8.29.3.15 static PixelFormat GetDefaultOutputFormat ( ) [static]

Get the default output pixel format.

See also

SetDefaultOutputFormat()

Returns

The default pixel format.

8.29.3.16 virtual void GetDimensions (unsigned int ∗ *pRows*, unsigned int ∗ *pCols* = NULL, unsigned int ∗ *pStride* = NULL, PixelFormat ∗ *pPixelFormat* = NULL, BayerTileFormat ∗ *pBayerFormat* = NULL) const [virtual]

Get the image dimensions associated with the image.

Parameters

pRows	Number of rows.
pCols	Number of columns.
pStride	The stride.
pPixel-Format	Pixel format.
pBayer-Format	Bayer tile format.

8.29.3.17 virtual ImageMetadata GetMetadata ( ) const [virtual]

Get the metadata associated with the image.

This includes embedded image information.

Returns

Metadata associated with the image.

8.29.3.18 virtual PixelFormat GetPixelFormat ( ) const [virtual]

Get the current pixel format.

Returns

The current pixel format.

8.29.3.19 virtual unsigned int GetReceivedDataSize ( ) const [virtual]

Get the size of the compressed data, in bytes.

A compressed image will have a maximum size equal to GetDataSize(), but may actually contain less data, depending on the compression level. For uncompressed images, a value smaller than the data size may indicate lost data.

Returns

The size of the compressed data, in bytes. 0 when camera not sending compressed data.

8.29.3.20 virtual unsigned int GetRows ( ) const [virtual]

Get the number of rows in the image.

Returns

The number of rows.

8.29.3.21 virtual unsigned int GetStride ( ) const [virtual]

Get the stride in the image.

Returns

The stride (The number of bytes between rows of the image).

8.29.3.22 virtual TimeStamp GetTimeStamp ( ) const [virtual]

Get the timestamp data associated with the image.

Returns

Timestamp data associated with the image.

8.29.3.23 virtual unsigned char∗ operator() (unsigned int *row*, unsigned int *col*) [virtual]

Indexing operator.

Parameters

row	The row of the pixel to return.
col	The column of the pixel to return.

Returns

The address of the specified byte from the image data.

8.29.3.24 virtual Image& operator= (const Image & *image*) [virtual]

Assignment operator.

Both images will point to the same image buffer internally. If the Image already has a buffer attached to it, it will will be released.

Parameters

image	The image to copy from.

8.29.3.25 virtual unsigned char∗ operator[] (unsigned int *index*) [virtual]

Indexing operator.

Parameters

index	The index of the byte to return.

Returns

The address of the specified byte from the image data.

8.29.3.26 virtual Error ReleaseBuffer ( ) [virtual]

Release the buffer associated with the Image.

If no buffer is associated, the function does nothing.

Returns

An Error indicating the success or failure of the function.

8.29.3.27 virtual Error Save (const char ∗ *pFilename*, ImageFileFormat *format* = FROM_FILE_EXT) [virtual]

Save the image to the specified file name with the file format specified.

Parameters

pFilename	Filename to save image with.
format	File format to save in.

Returns

An Error indicating the success or failure of the function.

8.29.3.28 virtual Error Save (const char ∗ *pFilename*, PNGOption ∗ *pOption*) [virtual]

Save the image to the specified file name with the options specified.

Parameters

pFilename	Filename to save image with.
pOption	Options to use while saving image.

Returns

An Error indicating the success or failure of the function.

8.29.3.29 virtual Error Save (const char ∗ *pFilename*, PPMOption ∗ *pOption*) [virtual]

Save the image to the specified file name with the options specified.

Parameters

pFilename	Filename to save image with.
pOption	Options to use while saving image.

Returns

An Error indicating the success or failure of the function.

8.29.3.30 virtual Error Save (const char ∗ *pFilename*, PGMOption ∗ *pOption*) [virtual]

Save the image to the specified file name with the options specified.

Parameters

pFilename	Filename to save image with.
pOption	Options to use while saving image.

Returns

An Error indicating the success or failure of the function.

8.29.3.31 virtual Error Save (const char ∗ *pFilename*, TIFFOption ∗ *pOption*) [virtual]

Save the image to the specified file name with the options specified.

Parameters

pFilename	Filename to save image with.
pOption	Options to use while saving image.

Returns

An Error indicating the success or failure of the function.

8.29.3.32 virtual Error Save (const char ∗ *pFilename*, JPEGOption ∗ *pOption*) [virtual]

Save the image to the specified file name with the options specified.

Parameters

pFilename	Filename to save image with.
pOption	Options to use while saving image.

Returns

An Error indicating the success or failure of the function.

8.29.3.33 virtual Error Save (const char ∗ *pFilename*, JPG2Option ∗ *pOption*) [virtual]

Save the image to the specified file name with the options specified.

Parameters

pFilename	Filename to save image with.
pOption	Options to use while saving image.

Returns

An Error indicating the success or failure of the function.

8.29.3.34 virtual Error Save (const char ∗ *pFilename*, BMPOption ∗ *pOption*) [virtual]

Save the image to the specified file name with the options specified.

Parameters

pFilename	Filename to save image with.
pOption	Options to use while saving image.

Returns

An Error indicating the success or failure of the function.

8.29.3.35 virtual Error SetBlockId (const unsigned int *blockId*) [virtual]

Set the block id of the Image object.

Parameters

blockId	The blockId to assign to the image.

8.29.3.36 virtual Error SetColorProcessing (ColorProcessingAlgorithm *colorProc*) [virtual]

Set the color processing algorithm.

This should be set on the input Image object.

Parameters

colorProc	The color processing algorithm to use.

See also

GetColorProcessing()

Returns

An Error indicating the success or failure of the function.

8.29.3.37 virtual Error SetData (const unsigned char $*$ *pData*, unsigned int *dataSize*) [virtual]

Set the data of the Image object.

Ownership of the image buffer is not transferred to the Image object. It is the user's responsibility to delete the buffer when it is no longer in use.

Parameters

pData	Pointer to the image buffer.
dataSize	Size of the image buffer.

8.29.3.38 static Error SetDefaultColorProcessing (ColorProcessingAlgorithm *defaultMethod*) [static]

Set the default color processing algorithm.

This method will be used for any image with the DEFAULT algorithm set. The method used is determined at the time of the Convert() call, therefore the most recent execution of this function will take precedence. The default setting is shared within the current process.

Parameters

default-Method	The color processing algorithm to set.

See also

GetDefaultColorProcessing()

Returns

An Error indicating the success or failure of the function.

8.29.3.39 static Error SetDefaultOutputFormat (PixelFormat *format*) [static]

Set the default output pixel format.

This format will be used for any call to Convert() that does not specify an output format. The format used will be determined at the time of the Convert() call, therefore the most recent execution of this function will take precedence. The default is shared within the current process.

Parameters

format	The output pixel format to set.

See also

GetDefaultOutputFormat()

Returns

The default color processing algorithm.

8.29.3.40 virtual Error SetDimensions (unsigned int *rows*, unsigned int *cols*, unsigned int *stride*, PixelFormat *pixelFormat*, BayerTileFormat *bayerFormat*) [virtual]

Sets the dimensions of the image object.

Parameters

rows	Number of rows to set.
cols	Number of cols to set.
stride	Stride to set.
pixelFormat	Pixel format to set.
bayerFormat	Bayer tile format to set.

See also

GetDimensions()

Returns

An Error indicating the success or failure of the function.

8.29.4 Friends And Related Function Documentation

8.29.4.1 friend class Iso [friend]

The documentation for this class was generated from the following file:

- Image.h

8.30 ImageMetadata Struct Reference

Metadata related to an image.

Public Member Functions

- ImageMetadata ()

Public Attributes

- unsigned int embeddedTimeStamp

  Embedded timestamp.
- unsigned int embeddedGain

  Embedded gain.
- unsigned int embeddedShutter

  Embedded shutter.
- unsigned int embeddedBrightness

  Embedded brightness.
- unsigned int embeddedExposure

  Embedded exposure.
- unsigned int embeddedWhiteBalance

  Embedded white balance.
- unsigned int embeddedFrameCounter

  Embedded frame counter.
- unsigned int embeddedStrobePattern

  Embedded strobe pattern.
- unsigned int embeddedGPIOPinState

  Embedded GPIO pin state.
- unsigned int embeddedROIPosition

  Embedded ROI position.
- unsigned int reserved [31]

  Reserved for future use.

8.30.1 Detailed Description

Metadata related to an image.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 ImageMetadata ( ) `[inline]`

8.30.3 Member Data Documentation

8.30.3.1 unsigned int embeddedBrightness

Embedded brightness.

8.30.3.2 unsigned int embeddedExposure

Embedded exposure.

8.30.3.3 unsigned int **embeddedFrameCounter**

Embedded frame counter.

8.30.3.4 unsigned int **embeddedGain**

Embedded gain.

8.30.3.5 unsigned int **embeddedGPIOPinState**

Embedded GPIO pin state.

8.30.3.6 unsigned int **embeddedROIPosition**

Embedded ROI position.

8.30.3.7 unsigned int **embeddedShutter**

Embedded shutter.

8.30.3.8 unsigned int **embeddedStrobePattern**

Embedded strobe pattern.

8.30.3.9 unsigned int **embeddedTimeStamp**

Embedded timestamp.

8.30.3.10 unsigned int **embeddedWhiteBalance**

Embedded white balance.

8.30.3.11 unsigned int **reserved**[31]

Reserved for future use.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.31 ImageStatistics Class Reference

The ImageStatistics object represents image statistics for an image.

Public Types

- enum StatisticsChannel { GREY, RED, GREEN, BLUE, HUE, SATURATION, LIGHTNESS, NUM_STATISTICS_CHANNELS }

  Channels that allow statistics to be calculated.

Public Member Functions

- ImageStatistics ()

  Default constructor.
- virtual ∼ImageStatistics ()

  Default destructor.
- ImageStatistics (const ImageStatistics &other)

  Copy constructor.
- ImageStatistics & operator= (const ImageStatistics &other)

  Assignment operator.
- Error EnableAll ()

  Enable all channels.
- Error DisableAll ()

  Disable all channels.
- Error EnableGreyOnly ()

  Enable only the grey channel.
- Error EnableRGBOnly ()

  Enable only the RGB channels.
- Error EnableHSLOnly ()

  Enable only the HSL channels.
- Error GetChannelStatus (StatisticsChannel channel, bool ∗pEnabled) const

  Get the status of a statistics channel.
- Error SetChannelStatus (StatisticsChannel channel, bool enabled)

  Set the status of a statistics channel.
- Error GetRange (StatisticsChannel channel, unsigned int ∗pMin, unsigned int ∗p-Max) const

  Get the range of a statistics channel.
- Error GetPixelValueRange (StatisticsChannel channel, unsigned int ∗pPixel-ValueMin, unsigned int ∗pPixelValueMax) const

  Get the range of a statistics channel.
- Error GetNumPixelValues (StatisticsChannel channel, unsigned int ∗pNumPixel-Values) const

  Get the number of unique pixel values in the image.
- Error GetMean (StatisticsChannel channel, float ∗pPixelValueMean) const

  Get the mean of the image.
- Error GetHistogram (StatisticsChannel channel, int ∗∗ppHistogram) const

  Get the histogram for the image.

- Error GetStatistics (StatisticsChannel channel, unsigned int ∗pRangeMin=NULL, unsigned int ∗pRangeMax=NULL, unsigned int ∗pPixelValueMin=NULL, unsigned int ∗pPixelValueMax=NULL, unsigned int ∗pNumPixelValues=NULL, float ∗pPixelValueMean=NULL, int ∗∗ppHistogram=NULL) const

  Get all statistics for the image.

Friends

- class ImageStatsCalculator

8.31.1 Detailed Description

The ImageStatistics object represents image statistics for an image.

8.31.2 Member Enumeration Documentation

8.31.2.1 enum StatisticsChannel

Channels that allow statistics to be calculated.

Enumerator:

GREY

RED

GREEN

BLUE

HUE

SATURATION

LIGHTNESS

NUM_STATISTICS_CHANNELS

8.31.3 Constructor & Destructor Documentation

8.31.3.1 ImageStatistics ( )

Default constructor.

8.31.3.2 virtual ∼ImageStatistics ( ) [virtual]

Default destructor.

8.31.3.3 ImageStatistics (const ImageStatistics & *other*)

Copy constructor.

8.31.4 Member Function Documentation

8.31.4.1 Error DisableAll ( )

Disable all channels.

Returns

An Error indicating the success or failure of the function.

8.31.4.2 Error EnableAll ( )

Enable all channels.

Returns

An Error indicating the success or failure of the function.

8.31.4.3 Error EnableGreyOnly ( )

Enable only the grey channel.

Returns

An Error indicating the success or failure of the function.

8.31.4.4 Error EnableHSLOnly ( )

Enable only the HSL channels.

Returns

An Error indicating the success or failure of the function.

8.31.4.5 Error EnableRGBOnly ( )

Enable only the RGB channels.

Returns

An Error indicating the success or failure of the function.

8.31.4.6 Error GetChannelStatus (StatisticsChannel *channel*, bool * *pEnabled*) const

Get the status of a statistics channel.

Parameters

channel	The statistics channel.
pEnabled	Whether the channel is enabled.

See also

SetChannelStatus()

Returns

An Error indicating the success or failure of the function.

8.31.4.7 Error GetHistogram (StatisticsChannel *channel*, int ** *ppHistogram*) const

Get the histogram for the image.

Parameters

channel	The statistics channel.
ppHistogram	Pointer to an array containing the histogram.

Returns

An Error indicating the success or failure of the function.

8.31.4.8 Error GetMean (StatisticsChannel *channel*, float * *pPixelValueMean*) const

Get the mean of the image.

Parameters

channel	The statistics channel.
pPixelValue-Mean	The mean of the image.

Returns

An Error indicating the success or failure of the function.

8.31.4.9 Error GetNumPixelValues (StatisticsChannel *channel*, unsigned int ∗ *pNumPixelValues*) const

Get the number of unique pixel values in the image.

Parameters

channel	The statistics channel.
pNumPixel-Values	The number of unique pixel values.

Returns

An Error indicating the success or failure of the function.

8.31.4.10 Error GetPixelValueRange (StatisticsChannel *channel*, unsigned int ∗ *pPixelValueMin*, unsigned int ∗ *pPixelValueMax*) const

Get the range of a statistics channel.

The values returned are the maximum values recorded for all pixels in the image.

Parameters

channel	The statistics channel.
pPixelValue-Min	The minimum pixel value.
pPixelValue-Max	The maximum pixel value.

Returns

An Error indicating the success or failure of the function.

8.31.4.11 Error GetRange (StatisticsChannel *channel*, unsigned int ∗ *pMin*, unsigned int ∗ *pMax*) const

Get the range of a statistics channel.

The values returned are the maximum possible values for any given pixel in the image. This is generally 0-255 for 8 bit images, and 0-65535 for 16 bit images.

Parameters

channel	The statistics channel.
pMin	The minimum possible value.
pMax	The maximum possible value.

Returns

An Error indicating the success or failure of the function.

8.31.4.12 Error GetStatistics (StatisticsChannel *channel*, unsigned int ∗ *pRangeMin* = NULL, unsigned int ∗ *pRangeMax* = NULL, unsigned int ∗ *pPixelValueMin* = NULL, unsigned int ∗ *pPixelValueMax* = NULL, unsigned int ∗ *pNumPixelValues* = NULL, float ∗ *pPixelValueMean* = NULL, int ∗∗ *ppHistogram* = NULL) const

Get all statistics for the image.

Parameters

channel	The statistics channel.
pRangeMin	The minimum possible value.
pRangeMax	The maximum possible value.
pPixelValue-Min	The minimum pixel value.
pPixelValue-Max	The maximum pixel value.
pNumPixel-Values	The number of unique pixel values.
pPixelValue-Mean	The mean of the image.
ppHistogram	Pointer to an array containing the histogram.

Returns

An Error indicating the success or failure of the function.

8.31.4.13 ImageStatistics& operator= (const ImageStatistics & *other*)

Assignment operator.

Parameters

other	The ImageStatistics object to copy from.

8.31.4.14 Error SetChannelStatus (StatisticsChannel *channel*, bool *enabled*)

Set the status of a statistics channel.

Parameters

channel	The statistics channel.
enabled	Whether the channel should be enabled.

See also

GetChannelStatus()

Returns

An Error indicating the success or failure of the function.

8.31.5 Friends And Related Function Documentation

8.31.5.1 friend class ImageStatsCalculator `[friend]`

The documentation for this class was generated from the following file:

- ImageStatistics.h

8.32 Internal Class Reference

Static Public Member Functions

- static void $*$ GetInternal (unsigned int index)

8.32.1 Member Function Documentation

8.32.1.1 static void$*$ GetInternal (unsigned int *index*) `[static]`

The documentation for this class was generated from the following file:

- Internal.h

8.33 IPAddress Struct Reference

IPv4 address.

Public Member Functions

- IPAddress ()
- IPAddress (unsigned int ipAddressVal)
- bool operator== (const IPAddress &address) const

  Equality operator.
- bool operator!= (const IPAddress &address)

  Inequality operator.

Public Attributes

- unsigned char octets [4]

8.33.1 Detailed Description

IPv4 address.

8.33.2 Constructor & Destructor Documentation

8.33.2.1 IPAddress ( ) [inline]

8.33.2.2 IPAddress (unsigned int *ipAddressVal*) [inline]

8.33.3 Member Function Documentation

8.33.3.1 bool operator!= (const IPAddress & *address*) [inline]

Inequality operator.

8.33.3.2 bool operator== (const IPAddress & *address*) const [inline]

Equality operator.

8.33.4 Member Data Documentation

8.33.4.1 unsigned char octets[4]

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.34 JPEGOption Struct Reference

Options for saving JPEG image.

Public Member Functions

- JPEGOption ()

Public Attributes

- bool progressive

  Whether to save as a progressive JPEG file.
- unsigned int quality

  JPEG image quality in range (0-100).
- unsigned int reserved [16]

  Reserved for future use.

8.34.1 Detailed Description

Options for saving JPEG image.

8.34.2 Constructor & Destructor Documentation

8.34.2.1 JPEGOption ( ) [inline]

8.34.3 Member Data Documentation

8.34.3.1 bool progressive

Whether to save as a progressive JPEG file.

8.34.3.2 unsigned int quality

JPEG image quality in range (0-100).

- 100 - Superb quality.
- 75 - Good quality.
- 50 - Normal quality.
- 10 - Poor quality.

8.34.3.3 unsigned int reserved[16]

Reserved for future use.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.35 JPG2Option Struct Reference

Options for saving JPEG2000 image.

Public Member Functions

- JPG2Option ()

Public Attributes

- unsigned int quality

  JPEG saving quality in range (1-512).
- unsigned int reserved [16]

  Reserved for future use.

8.35.1 Detailed Description

Options for saving JPEG2000 image.

8.35.2 Constructor & Destructor Documentation

8.35.2.1 JPG2Option ( ) `[inline]`

8.35.3 Member Data Documentation

8.35.3.1 unsigned int **quality**

JPEG saving quality in range (1-512).

8.35.3.2 unsigned int **reserved**[16]

Reserved for future use.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.36 LUTData Struct Reference

Information about the camera's look up table.

Public Member Functions

- LUTData ()

Public Attributes

- bool supported

  Flag indicating if LUT is supported.
- bool enabled

  Flag indicating if LUT is enabled.
- unsigned int numBanks

  The number of LUT banks available (Always 1 for PGR LUT).
- unsigned int numChannels

  The number of LUT channels per bank available.
- unsigned int inputBitDepth

  The input bit depth of the LUT.
- unsigned int outputBitDepth

  The output bit depth of the LUT.
- unsigned int numEntries

  The number of entries in the LUT.
- unsigned int reserved [8]

  Reserved for future use.

8.36.1 Detailed Description

Information about the camera's look up table.

8.36.2 Constructor & Destructor Documentation

8.36.2.1 LUTData ( ) `[inline]`

8.36.3 Member Data Documentation

8.36.3.1 bool enabled

Flag indicating if LUT is enabled.

8.36.3.2 unsigned int inputBitDepth

The input bit depth of the LUT.

8.36.3.3 unsigned int **numBanks**

The number of LUT banks available (Always 1 for PGR LUT).

8.36.3.4 unsigned int **numChannels**

The number of LUT channels per bank available.

8.36.3.5 unsigned int **numEntries**

The number of entries in the LUT.

8.36.3.6 unsigned int **outputBitDepth**

The output bit depth of the LUT.

8.36.3.7 unsigned int **reserved**[8]

Reserved for future use.

8.36.3.8 bool **supported**

Flag indicating if LUT is supported.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.37 MACAddress Struct Reference

MAC address.

Public Member Functions

- MACAddress ()
- MACAddress (unsigned int macAddressValHigh, unsigned int macAddressVal-Low)
- bool operator== (const MACAddress &address) const

  Equality operator.
- bool operator!= (const MACAddress &address)

  Inequality operator.

Public Attributes

- unsigned char octets [6]

8.37.1 Detailed Description

MAC address.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 **MACAddress** ( ) `[inline]`

8.37.2.2 **MACAddress** (unsigned int *macAddressValHigh*, unsigned int *macAddressValLow*) `[inline]`

8.37.3 Member Function Documentation

8.37.3.1 bool operator!= (const **MACAddress** & *address*) `[inline]`

Inequality operator.

8.37.3.2 bool operator== (const **MACAddress** & *address*) const `[inline]`

Equality operator.

8.37.4 Member Data Documentation

8.37.4.1 unsigned char **octets**[6]

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.38 MJPGOption Struct Reference

Options for saving MJPG files.

Public Member Functions

- MJPGOption ()

Public Attributes

- float frameRate

    Frame rate of the stream.
- unsigned int quality

    Image quality (1-100)
- unsigned int reserved [256]

8.38.1 Detailed Description

Options for saving MJPG files.

8.38.2 Constructor & Destructor Documentation

8.38.2.1 MJPGOption ( ) [inline]

8.38.3 Member Data Documentation

8.38.3.1 float frameRate

Frame rate of the stream.

8.38.3.2 unsigned int quality

Image quality (1-100)

8.38.3.3 unsigned int reserved[256]

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.39 NodeMap Class Reference

Public Member Functions

- NodeMap (GenApi::CNodeMapRef ∗ref)
- virtual ∼NodeMap (void)
- GenICam::gcstring _GetDeviceName ()

    Get device name.
- void _Poll (int64_t ElapsedTime)

    Fires nodes which have a polling time.

- void _GetNodes (NodeList_t &Nodes)

  Retrieves all nodes in the node map.
- INode ∗ _GetNode (const GenICam::gcstring &key)

  Retrieves the node from the central map by name.
- void _InvalidateNodes () const

  Invalidates all nodes.

8.39.1 Constructor & Destructor Documentation

8.39.1.1 NodeMap (GenApi::CNodeMapRef ∗ *ref*)

8.39.1.2 virtual ∼NodeMap (void) [virtual]

8.39.2 Member Function Documentation

8.39.2.1 GenICam::gcstring _GetDeviceName ( )

Get device name.

8.39.2.2 INode∗ _GetNode (const GenICam::gcstring & *key*)

Retrieves the node from the central map by name.

8.39.2.3 void _GetNodes (NodeList_t & *Nodes*)

Retrieves all nodes in the node map.

8.39.2.4 void _InvalidateNodes ( ) const

Invalidates all nodes.

8.39.2.5 void _Poll (int64_t *ElapsedTime*)

Fires nodes which have a polling time.

The documentation for this class was generated from the following file:

- NodeMap.h

8.40 PGMOption Struct Reference

Options for saving PGM images.

Public Member Functions

- PGMOption ()

Public Attributes

- bool binaryFile

  Whether to save the PPM as a binary file.
- unsigned int reserved [16]

  Reserved for future use.

8.40.1 Detailed Description

Options for saving PGM images.

8.40.2 Constructor & Destructor Documentation

8.40.2.1 PGMOption ( ) `[inline]`

8.40.3 Member Data Documentation

8.40.3.1 bool binaryFile

Whether to save the PPM as a binary file.

8.40.3.2 unsigned int reserved[16]

Reserved for future use.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.41 PGRGuid Class Reference

A GUID to the camera.

Public Member Functions

- PGRGuid ()

  Constructor.
- bool operator== (const PGRGuid &guid) const

  Equality operator.

- bool operator!= (const PGRGuid &guid)

    Inequality operator.

Public Attributes

- unsigned int value [4]

8.41.1 Detailed Description

A GUID to the camera.

It is used to uniquely identify a camera.

8.41.2 Constructor & Destructor Documentation

8.41.2.1 PGRGuid ( ) [inline]

Constructor.

8.41.3 Member Function Documentation

8.41.3.1 bool operator!= (const PGRGuid & *guid*) [inline]

Inequality operator.

8.41.3.2 bool operator== (const PGRGuid & *guid*) const [inline]

Equality operator.

8.41.4 Member Data Documentation

8.41.4.1 unsigned int value[4]

The documentation for this class was generated from the following file:

- FlyCapture2Defs.h

8.42 PNGOption Struct Reference

Options for saving PNG images.

Public Member Functions

- PNGOption ()

Public Attributes

- bool interlaced

  Whether to save the PNG as interlaced.
- unsigned int compressionLevel

  Compression level (0-9).
- unsigned int reserved [16]

  Reserved for future use.

8.42.1 Detailed Description

Options for saving PNG images.

8.42.2 Constructor & Destructor Documentation

8.42.2.1 PNGOption ( ) `[inline]`

8.42.3 Member Data Documentation

8.42.3.1 unsigned int compressionLevel

Compression level (0-9).

0 is no compression, 9 is best compression.

8.42.3.2 bool interlaced

Whether to save the PNG as interlaced.

8.42.3.3 unsigned int reserved[16]

Reserved for future use.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.43 PPMOption Struct Reference

Options for saving PPM images.

Public Member Functions

- PPMOption ()

Public Attributes

- bool binaryFile

  Whether to save the PPM as a binary file.
- unsigned int reserved [16]

  Reserved for future use.

8.43.1 Detailed Description

Options for saving PPM images.

8.43.2 Constructor & Destructor Documentation

8.43.2.1 PPMOption ( ) [inline]

8.43.3 Member Data Documentation

8.43.3.1 bool binaryFile

Whether to save the PPM as a binary file.

8.43.3.2 unsigned int reserved[16]

Reserved for future use.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.44 Property Struct Reference

A specific camera property.

Public Member Functions

- Property ()
- Property (PropertyType propType)

Public Attributes

- PropertyType type

  Property info type.
- bool present

  Flag indicating if the property is present.
- bool absControl

  Flag controlling absolute mode.
- bool onePush

  Flag controlling one push.
- bool onOff

  Flag controlling on/off.
- bool autoManualMode

  Flag controlling auto.
- unsigned int valueA

  Value A (integer).
- unsigned int valueB

  Value B (integer).
- float absValue

  Floating point value.
- unsigned int reserved [8]

  Reserved for future use.

8.44.1 Detailed Description

A specific camera property.

For example, to set the gain to 12dB, set the following values:

- *type* - `GAIN`
- *absControl* - `true`
- *onePush* - `false`
- *onOff* - `true`
- *autoManualMode* - `false`
- *absValue* - `12.0`

8.44.2 Constructor & Destructor Documentation

8.44.2.1 Property ( ) `[inline]`

8.44.2.2 Property (PropertyType *propType*) `[inline]`

8.44.3 Member Data Documentation

8.44.3.1 bool **absControl**

Flag controlling absolute mode.

8.44.3.2 float **absValue**

Floating point value.

8.44.3.3 bool **autoManualMode**

Flag controlling auto.

8.44.3.4 bool **onePush**

Flag controlling one push.

8.44.3.5 bool **onOff**

Flag controlling on/off.

8.44.3.6 bool **present**

Flag indicating if the property is present.

8.44.3.7 unsigned int **reserved**[8]

Reserved for future use.

8.44.3.8 PropertyType **type**

Property info type.

8.44.3.9 unsigned int **valueA**

Value A (integer).

8.44.3.10 unsigned int valueB

Value B (integer).

Applies only to the white balance blue value. Use Value A for the red value.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.45 PropertyInfo Struct Reference

Information about a specific camera property.

Public Member Functions

- PropertyInfo ()
- PropertyInfo (PropertyType propType)

Public Attributes

- PropertyType type

  Property info type.
- bool present

  Flag indicating if the property is present.
- bool autoSupported

  Flag indicating if auto is supported.
- bool manualSupported

  Flag indicating if manual is supported.
- bool onOffSupported

  Flag indicating if on/off is supported.
- bool onePushSupported

  Flag indicating if one push is supported.
- bool absValSupported

  Flag indicating if absolute mode is supported.
- bool readOutSupported

  Flag indicating if property value can be read out.
- unsigned int min

  Minimum value (as an integer).
- unsigned int max

  Maximum value (as an integer).
- float absMin

  Minimum value (as a floating point value).
- float absMax

Maximum value (as a floating point value).

- char pUnits [sk_maxStringLength]

  Textual description of units.
- char pUnitAbbr [sk_maxStringLength]

  Abbreviated textual description of units.
- unsigned int reserved [8]

  Reserved for future use.

8.45.1 Detailed Description

Information about a specific camera property.

This structure is also also used as the TriggerDelayInfo structure.

8.45.2 Constructor & Destructor Documentation

8.45.2.1 **PropertyInfo ( )** `[inline]`

8.45.2.2 **PropertyInfo (PropertyType *propType*)** `[inline]`

8.45.3 Member Data Documentation

8.45.3.1 float **absMax**

Maximum value (as a floating point value).

8.45.3.2 float **absMin**

Minimum value (as a floating point value).

8.45.3.3 bool **absValSupported**

Flag indicating if absolute mode is supported.

8.45.3.4 bool **autoSupported**

Flag indicating if auto is supported.

8.45.3.5 bool **manualSupported**

Flag indicating if manual is supported.

8.45.3.6 unsigned int **max**

Maximum value (as an integer).

8.45.3.7 unsigned int **min**

Minimum value (as an integer).

8.45.3.8 bool **onePushSupported**

Flag indicating if one push is supported.

8.45.3.9 bool **onOffSupported**

Flag indicating if on/off is supported.

8.45.3.10 bool **present**

Flag indicating if the property is present.

8.45.3.11 char **pUnitAbbr[sk_maxStringLength]**

Abbreviated textual description of units.

8.45.3.12 char **pUnits[sk_maxStringLength]**

Textual description of units.

8.45.3.13 bool **readOutSupported**

Flag indicating if property value can be read out.

8.45.3.14 unsigned int **reserved**[8]

Reserved for future use.

8.45.3.15 **PropertyType type**

Property info type.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.46 StrobeControl Struct Reference

A camera strobe.

Public Member Functions

- StrobeControl ()

Public Attributes

- unsigned int source

  Source value.
- bool onOff

  Flag controlling on/off.
- unsigned int polarity

  Signal polarity.
- float delay

  Signal delay (in ms).
- float duration

  Signal duration (in ms).
- unsigned int reserved [8]

  Reserved for future use.

8.46.1 Detailed Description

A camera strobe.

8.46.2 Constructor & Destructor Documentation

8.46.2.1 StrobeControl ( ) `[inline]`

8.46.3 Member Data Documentation

8.46.3.1 float delay

Signal delay (in ms).

8.46.3.2 float duration

Signal duration (in ms).

8.46.3.3 bool onOff

Flag controlling on/off.

8.46.3.4 unsigned int polarity

Signal polarity.

8.46.3.5 unsigned int reserved[8]

Reserved for future use.

8.46.3.6 unsigned int source

Source value.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.47 StrobeInfo Struct Reference

A camera strobe property.

Public Member Functions

- StrobeInfo ()

Public Attributes

- unsigned int source

  Source value.
- bool present

  Presence of strobe.
- bool readOutSupported

  Flag indicating if strobe value can be read out.
- bool onOffSupported

  Flag indicating if on/off is supported.
- bool polaritySupported

  Flag indicating if polarity is supported.
- float minValue

  Minimum value.

- float maxValue

  Maximum value.
- unsigned int reserved [8]

  Reserved for future use.

8.47.1 Detailed Description

A camera strobe property.

8.47.2 Constructor & Destructor Documentation

8.47.2.1 StrobeInfo ( ) [inline]

8.47.3 Member Data Documentation

8.47.3.1 float maxValue

Maximum value.

8.47.3.2 float minValue

Minimum value.

8.47.3.3 bool onOffSupported

Flag indicating if on/off is supported.

8.47.3.4 bool polaritySupported

Flag indicating if polarity is supported.

8.47.3.5 bool present

Presence of strobe.

8.47.3.6 bool readOutSupported

Flag indicating if strobe value can be read out.

8.47.3.7 unsigned int reserved[8]

Reserved for future use.

8.47.3.8 unsigned int **source**

Source value.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.48 SyncManager Class Reference

Public Member Functions

- MULTISYNCLIBRARY_API SyncManager ()
- MULTISYNCLIBRARY_API ∼SyncManager ()
- MULTISYNCLIBRARY_API PGRSyncError Start ()
- MULTISYNCLIBRARY_API PGRSyncError Stop ()
- MULTISYNCLIBRARY_API PGRSyncError RescanMasterTimingBus ()
- MULTISYNCLIBRARY_API PGRSyncMessage GetSyncStatus ()
- MULTISYNCLIBRARY_API double GetTimeSinceSynced ()
- MULTISYNCLIBRARY_API bool IsTimingBusConnected ()
- MULTISYNCLIBRARY_API bool EnableCrossPCSynchronization ()
- MULTISYNCLIBRARY_API bool DisableCrossPCSynchronization ()
- MULTISYNCLIBRARY_API bool QueryCrossPCSynchronizationSetting ()

8.48.1 Constructor & Destructor Documentation

8.48.1.1 MULTISYNCLIBRARY_API **SyncManager** ( )

8.48.1.2 MULTISYNCLIBRARY_API ∼**SyncManager** ( )

8.48.2 Member Function Documentation

8.48.2.1 MULTISYNCLIBRARY_API bool DisableCrossPCSynchronization ( )

8.48.2.2 MULTISYNCLIBRARY_API bool EnableCrossPCSynchronization ( )

8.48.2.3 MULTISYNCLIBRARY_API **PGRSyncMessage** GetSyncStatus ( )

8.48.2.4 MULTISYNCLIBRARY_API double GetTimeSinceSynced ( )

8.48.2.5 MULTISYNCLIBRARY_API bool IsTimingBusConnected ( )

8.48.2.6 MULTISYNCLIBRARY_API bool QueryCrossPCSynchronizationSetting ( )

8.48.2.7 MULTISYNCLIBRARY_API **PGRSyncError** RescanMasterTimingBus ( )

8.48.2.8 MULTISYNCLIBRARY_API PGRSyncError Start ( )

8.48.2.9 MULTISYNCLIBRARY_API PGRSyncError Stop ( )

The documentation for this class was generated from the following file:

- MultiSyncLibraryDefs.h

8.49 SystemInfo Struct Reference

Description of the system.

Public Attributes

- OSType osType

  Operating system type as described by OSType.
- char osDescription [sk_maxStringLength]

  Detailed description of the operating system.
- ByteOrder byteOrder

  Byte order of the system.
- size_t sysMemSize

  Amount of memory available on the system.
- char cpuDescription [sk_maxStringLength]

  Detailed description of the CPU.
- size_t numCpuCores

  Number of cores on all CPUs on the system.
- char driverList [sk_maxStringLength]

  List of drivers used.
- char libraryList [sk_maxStringLength]

  List of libraries used.
- char gpuDescription [sk_maxStringLength]

  Detailed description of the GPU.
- size_t screenWidth

  Screen resolution width in pixels.
- size_t screenHeight

  Screen resolution height in pixels.
- unsigned int reserved [16]

  Reserved for future use.

8.49.1 Detailed Description

Description of the system.

8.49.2 Member Data Documentation

8.49.2.1 ByteOrder byteOrder

Byte order of the system.

8.49.2.2 char cpuDescription[sk_maxStringLength]

Detailed description of the CPU.

8.49.2.3 char driverList[sk_maxStringLength]

List of drivers used.

8.49.2.4 char gpuDescription[sk_maxStringLength]

Detailed description of the GPU.

8.49.2.5 char libraryList[sk_maxStringLength]

List of libraries used.

8.49.2.6 size_t numCpuCores

Number of cores on all CPUs on the system.

8.49.2.7 char osDescription[sk_maxStringLength]

Detailed description of the operating system.

8.49.2.8 OSType osType

Operating system type as described by OSType.

8.49.2.9 unsigned int reserved[16]

Reserved for future use.

8.49.2.10 size_t screenHeight

Screen resolution height in pixels.

8.49.2.11 size_t screenWidth

Screen resolution width in pixels.

8.49.2.12 size_t sysMemSize

Amount of memory available on the system.

The documentation for this struct was generated from the following file:

- Utilities.h

8.50 TIFFOption Struct Reference

Options for saving TIFF images.

Public Types

- enum CompressionMethod { NONE = 1, PACKBITS, DEFLATE, ADOBE_DEFLATE, CCITTFAX3, CCITTFAX4, LZW, JPEG }

Public Member Functions

- TIFFOption ()

Public Attributes

- CompressionMethod compression

  Compression method to use for encoding TIFF images.
- unsigned int reserved [16]

  Reserved for future use.

8.50.1 Detailed Description

Options for saving TIFF images.

8.50.2 Member Enumeration Documentation

8.50.2.1 enum CompressionMethod

Enumerator:

NONE Save without any compression.

PACKBITS Save using PACKBITS compression.

DEFLATE Save using DEFLATE compression (ZLIB compression).

ADOBE_DEFLATE Save using ADOBE DEFLATE compression.

CCITTFAX3 Save using CCITT Group 3 fax encoding. This is only valid for 1-bit images only. Default to LZW for other bit depths.

CCITTFAX4 Save using CCITT Group 4 fax encoding. This is only valid for 1-bit images only. Default to LZW for other bit depths.

LZW Save using LZW compression.

JPEG Save using JPEG compression. This is only valid for 8-bit greyscale and 24-bit only. Default to LZW for other bit depths.

8.50.3 Constructor & Destructor Documentation

8.50.3.1 TIFFOption ( ) `[inline]`

8.50.4 Member Data Documentation

8.50.4.1 CompressionMethod compression

Compression method to use for encoding TIFF images.

8.50.4.2 unsigned int reserved[16]

Reserved for future use.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.51 TimeStamp Struct Reference

Timestamp information.

Public Member Functions

- TimeStamp ()

Public Attributes

- long long seconds

    Seconds.
- unsigned int microSeconds

Microseconds.
- unsigned int cycleSeconds

  1394 cycle time seconds.
- unsigned int cycleCount

  1394 cycle time count.
- unsigned int cycleOffset

  1394 cycle time offset.
- unsigned int reserved [8]

  Reserved for future use.

8.51.1 Detailed Description

Timestamp information.

8.51.2 Constructor & Destructor Documentation

8.51.2.1 TimeStamp ( ) `[inline]`

8.51.3 Member Data Documentation

8.51.3.1 unsigned int **cycleCount**

1394 cycle time count.

8.51.3.2 unsigned int **cycleOffset**

1394 cycle time offset.

8.51.3.3 unsigned int **cycleSeconds**

1394 cycle time seconds.

8.51.3.4 unsigned int **microSeconds**

Microseconds.

8.51.3.5 unsigned int **reserved**[8]

Reserved for future use.

8.51.3.6 long long seconds

Seconds.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.52 TopologyNode Class Reference

The TopologyNode class contains topology information that can be used to generate a tree structure of all cameras and devices connected to a computer.

Public Types

- enum PortType { NOT_CONNECTED = 1, CONNECTED_TO_PARENT, CONNECTED_TO_CHILD }

  Possible states of a port on a node.
- enum NodeType { COMPUTER, BUS, CAMERA, NODE }

  Type of node.

Public Member Functions

- TopologyNode ()

  Default constructor.
- TopologyNode (PGRGuid guid, int deviceId, NodeType nodeType, InterfaceType interfaceType)

  Constructor.
- virtual ∼TopologyNode ()

  Default destructor.
- TopologyNode (const TopologyNode &other)

  Copy constructor.
- virtual TopologyNode & operator= (const TopologyNode &other)

  Assignment operator.
- virtual PGRGuid GetGuid ()

  Get the PGRGuid associated with the node.
- virtual int GetDeviceId ()

  Get the device ID associated with the node.
- virtual NodeType GetNodeType ()

  Get the node type associated with the node.
- virtual InterfaceType GetInterfaceType ()

  Get the interface type associated with the node.
- virtual unsigned int GetNumChildren ()

Get the number of child nodes.

- virtual TopologyNode GetChild (unsigned int position)

  Get child node located at the specified position.
- virtual void AddChild (TopologyNode childNode)

  Add the specified TopologyNode as a child of the node.
- virtual unsigned int GetNumPorts ()

  Get the number of ports.
- virtual PortType GetPortType (unsigned int position)

  Get type of port located at the specified position.
- virtual void AddPort (PortType childPort)

  Add the specified PortType as a port of the node.
- virtual bool AssignGuidToNode (PGRGuid guid, int deviceId)

  Assign a PGRGuid and device ID to the node.
- virtual bool AssignGuidToNode (PGRGuid guid, int deviceId, NodeType nodeType)

  Assign a PGRGuid, device ID and nodeType to the node.

8.52.1 Detailed Description

The TopologyNode class contains topology information that can be used to generate a tree structure of all cameras and devices connected to a computer.

8.52.2 Member Enumeration Documentation

8.52.2.1 enum NodeType

Type of node.

Enumerator:

COMPUTER

BUS

CAMERA

NODE

8.52.2.2 enum PortType

Possible states of a port on a node.

Enumerator:

NOT_CONNECTED

CONNECTED_TO_PARENT

CONNECTED_TO_CHILD

8.52.3 Constructor & Destructor Documentation

8.52.3.1 TopologyNode ( )

Default constructor.

8.52.3.2 TopologyNode (PGRGuid *guid*, int *deviceId*, NodeType *nodeType*, InterfaceType *interfaceType*)

Constructor.

Parameters

guid	The PGRGuid of the node (if applicable).
deviceId	Device ID of the node.
nodeType	Type of the node.
interface-Type	Interface type of the node.

8.52.3.3 virtual ∼TopologyNode ( ) `[virtual]`

Default destructor.

8.52.3.4 TopologyNode (const TopologyNode & *other*)

Copy constructor.

8.52.4 Member Function Documentation

8.52.4.1 virtual void AddChild (TopologyNode *childNode*) `[virtual]`

Add the specified TopologyNode as a child of the node.

Parameters

childNode	The TopologyNode to add.

8.52.4.2 virtual void AddPort (PortType *childPort*) `[virtual]`

Add the specified PortType as a port of the node.

Parameters

childPort	The port to add.

8.52.4.3 virtual bool AssignGuidToNode (PGRGuid *guid*, int *deviceId*) [virtual]

Assign a PGRGuid and device ID to the node.

Parameters

guid	PGRGuid to be assigned.
deviceId	Device ID to be assigned.

Returns

Whether the data was successfully set to the node.

8.52.4.4 virtual bool AssignGuidToNode (PGRGuid *guid*, int *deviceId*, NodeType *nodeType*) [virtual]

Assign a PGRGuid, device ID and nodeType to the node.

Parameters

guid	PGRGuid to be assigned.
deviceId	Device ID to be assigned.
nodeType	NodeType to be assigned

Returns

Whether the data was successfully set to the node.

8.52.4.5 virtual TopologyNode GetChild (unsigned int *position*) [virtual]

Get child node located at the specified position.

Parameters

position	Position of the node.

Returns

TopologyNode at the specified position.

8.52.4.6 virtual int GetDeviceId ( ) [virtual]

Get the device ID associated with the node.

Returns

Device ID of the node.

8.52.4.7 virtual PGRGuid GetGuid ( ) [virtual]

Get the PGRGuid associated with the node.

Returns

PGRGuid of the node.

8.52.4.8 virtual InterfaceType GetInterfaceType ( ) [virtual]

Get the interface type associated with the node.

Returns

Interface type of the node.

8.52.4.9 virtual NodeType GetNodeType ( ) [virtual]

Get the node type associated with the node.

Returns

Node type of the node.

8.52.4.10 virtual unsigned int GetNumChildren ( ) [virtual]

Get the number of child nodes.

Returns

Number of child nodes.

8.52.4.11 virtual unsigned int GetNumPorts ( ) [virtual]

Get the number of ports.

Returns

Number of ports.

8.52.4.12 virtual **PortType** GetPortType (unsigned int *position*) [virtual]

Get type of port located at the specified position.

Parameters

position	Position of the port.

Returns

PortType at the specified position.

8.52.4.13 virtual **TopologyNode&** operator= (const **TopologyNode** & *other*) [virtual]

Assignment operator.

Parameters

other	The TopologyNode to copy from.

The documentation for this class was generated from the following file:

- TopologyNode.h

8.53 TriggerMode Struct Reference

A camera trigger.

Public Member Functions

- TriggerMode ()

Public Attributes

- bool onOff

  Flag controlling on/off.
- unsigned int polarity

  Polarity value.
- unsigned int source

  Source value.
- unsigned int mode

  Mode value.
- unsigned int parameter

Parameter value.

- unsigned int reserved [8]

    Reserved for future use.

8.53.1 Detailed Description

A camera trigger.

8.53.2 Constructor & Destructor Documentation

8.53.2.1 TriggerMode ( ) `[inline]`

8.53.3 Member Data Documentation

8.53.3.1 unsigned int **mode**

Mode value.

8.53.3.2 bool **onOff**

Flag controlling on/off.

8.53.3.3 unsigned int **parameter**

Parameter value.

8.53.3.4 unsigned int **polarity**

Polarity value.

8.53.3.5 unsigned int **reserved**[8]

Reserved for future use.

8.53.3.6 unsigned int **source**

Source value.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.54 TriggerModeInfo Struct Reference

Information about a camera trigger property.

Public Member Functions

- TriggerModeInfo ()

Public Attributes

- bool present

  Presence of trigger mode.
- bool readOutSupported

  Flag indicating if trigger value can be read out.
- bool onOffSupported

  Flag indicating if on/off is supported.
- bool polaritySupported

  Flag indicating if polarity is supported.
- bool valueReadable

  Flag indicating if the value is readable.
- unsigned int sourceMask

  Source mask.
- bool softwareTriggerSupported

  Flag indicating if software trigger is supported.
- unsigned int modeMask

  Mode mask.
- unsigned int reserved [8]

  Reserved for future use.

8.54.1 Detailed Description

Information about a camera trigger property.

8.54.2 Constructor & Destructor Documentation

8.54.2.1 TriggerModeInfo ( ) `[inline]`

8.54.3 Member Data Documentation

8.54.3.1 unsigned int modeMask

Mode mask.

8.54.3.2 bool onOffSupported

Flag indicating if on/off is supported.

8.54.3.3 bool polaritySupported

Flag indicating if polarity is supported.

8.54.3.4 bool present

Presence of trigger mode.

8.54.3.5 bool readOutSupported

Flag indicating if trigger value can be read out.

8.54.3.6 unsigned int reserved[8]

Reserved for future use.

8.54.3.7 bool softwareTriggerSupported

Flag indicating if software trigger is supported.

8.54.3.8 unsigned int sourceMask

Source mask.

8.54.3.9 bool valueReadable

Flag indicating if the value is readable.

The documentation for this struct was generated from the following file:

- FlyCapture2Defs.h

8.55 Utilities Class Reference

The Utility class is generally used to query for general system information such as operating system, available memory etc.

Static Public Member Functions

- static Error GetSystemInfo (SystemInfo ∗pSystemInfo)

  Get system information.
- static Error GetLibraryVersion (FC2Version ∗pVersion)

  Get library version.
- static Error LaunchBrowser (const char ∗pAddress)

  Launch a URL in the system default browser.
- static Error LaunchHelp (const char ∗pFileName)

  Open a CHM file in the system default CHM viewer.
- static Error LaunchCommand (const char ∗pCommand)

  Execute a command in the terminal.
- static Error LaunchCommandAsync (const char ∗pCommand, AsyncCommand-Callback pCallback, void ∗pUserData)

  Execute a command in the terminal.

8.55.1 Detailed Description

The Utility class is generally used to query for general system information such as operating system, available memory etc.

It can also be used to launch browsers, CHM viewers or terminal commands.

8.55.2 Member Function Documentation

8.55.2.1 static Error GetLibraryVersion (FC2Version ∗ *pVersion*) `[static]`

Get library version.

Parameters

pVersion	Structure to receive the library version.

Returns

An Error indicating the success or failure of the function.

8.55.2.2 static Error GetSystemInfo (SystemInfo ∗ *pSystemInfo*) `[static]`

Get system information.

Parameters

pSystemInfo	Structure to receive system information.

Returns

An Error indicating the success or failure of the function.

8.55.2.3 static Error LaunchBrowser (const char * *pAddress*) [static]

Launch a URL in the system default browser.

Parameters

pAddress	URL to open in browser.

Returns

An Error indicating the success or failure of the function.

8.55.2.4 static Error LaunchCommand (const char * *pCommand*) [static]

Execute a command in the terminal.

This is a blocking call that will return when the command completes.

Parameters

pCommand	Command to execute.

See also

LaunchCommandAsync()

Returns

An Error indicating the success or failure of the function.

8.55.2.5 static Error LaunchCommandAsync (const char * *pCommand*, AsyncCommandCallback *pCallback*, void * *pUserData*) [static]

Execute a command in the terminal.

This is a non-blocking call that will return immediately. The return value of the command can be retrieved in the callback.

Parameters

pCommand	Command to execute.
pCallback	Callback to fire when command is complete.
pUserData	Data pointer to pass to callback.

See also

LaunchCommand()

Returns

An Error indicating the success or failure of the function.

8.55.2.6 static Error LaunchHelp (const char ∗ *pFileName*) `[static]`

Open a CHM file in the system default CHM viewer.

Parameters

pFileName	Filename of CHM file to open.

Returns

An Error indicating the success or failure of the function.

The documentation for this class was generated from the following file:

- Utilities.h

Chapter 9

File Documentation

9.1 AVIRecorder.h File Reference

Classes

- class AVIRecorder

  The AVIRecorder class provides the functionality for the user to record images to an AVI file.

Namespaces

- namespace FlyCapture2

9.2 BusManager.h File Reference

Classes

- class BusManager

  The BusManager class provides the functionality for the user to get an PGRGuid for a desired camera or device easily.

Namespaces

- namespace FlyCapture2

Typedefs

- typedef void(∗ BusEventCallback)(void ∗pParameter, unsigned int serialNumber)

Bus event callback function prototype.

- typedef void ∗ CallbackHandle

  Handle that is returned when registering a callback.

9.3 Camera.h File Reference

Classes

- class Camera

  The Camera object represents a physical camera that uses the IIDC register set.

Namespaces

- namespace FlyCapture2

9.4 CameraBase.h File Reference

Classes

- class CameraBase

  The CameraBase class is an abstract base class that defines a general interface to a camera.

Namespaces

- namespace FlyCapture2

Typedefs

- typedef void(∗ ImageEventCallback)(class Image ∗pImage, const void ∗p-CallbackData)

  Image event callback function prototype.

9.5 Error.h File Reference

Classes

- class Error

  The Error object represents an error that is returned from the library.

Namespaces

- namespace FlyCapture2

9.6 FlyCapture2.h File Reference

9.7 FlyCapture2Defs.h File Reference

Classes

- struct FC2Version

  The current version of the library.
- class PGRGuid

  A GUID to the camera.
- struct IPAddress

  IPv4 address.
- struct MACAddress

  MAC address.
- struct GigEProperty

  A GigE property.
- struct GigEStreamChannel

  Information about a single GigE stream channel.
- struct GigEConfig

  Configuration for a GigE camera.
- struct GigEImageSettingsInfo

  Format 7 information for a single mode.
- struct GigEImageSettings

  Image settings for a GigE camera.
- struct Format7ImageSettings

  Format 7 image settings.
- struct Format7Info

  Format 7 information for a single mode.
- struct Format7PacketInfo

  Format 7 packet information.
- struct FC2Config

  Configuration for a camera.
- struct PropertyInfo

  Information about a specific camera property.
- struct Property

  A specific camera property.
- struct TriggerModeInfo

  Information about a camera trigger property.

- struct TriggerMode

    A camera trigger.
- struct StrobeInfo

    A camera strobe property.
- struct StrobeControl

    A camera strobe.
- struct TimeStamp

    Timestamp information.
- struct ConfigROM

    Camera configuration ROM.
- struct CameraInfo

    Camera information.
- struct EmbeddedImageInfoProperty

    Properties of a single embedded image info property.
- struct EmbeddedImageInfo

    Properties of the possible embedded image information.
- struct ImageMetadata

    Metadata related to an image.
- struct LUTData

    Information about the camera's look up table.
- struct HostAdapterStats

    Information about the host adapter's statistics.
- struct CameraStats

    Camera diagnostic information.
- struct PNGOption

    Options for saving PNG images.
- struct PPMOption

    Options for saving PPM images.
- struct PGMOption

    Options for saving PGM images.
- struct TIFFOption

    Options for saving TIFF images.
- struct JPEGOption

    Options for saving JPEG image.
- struct JPG2Option

    Options for saving JPEG2000 image.
- struct BMPOption

    Options for saving Bitmap image.
- struct MJPGOption

    Options for saving MJPG files.
- struct H264Option

    Options for saving H264 files.
- struct AVIOption

    Options for saving AVI files.

Namespaces

- namespace FlyCapture2

Defines

- #define NULL 0
- #define FULL_32BIT_VALUE 0x7FFFFFFF

Typedefs

- typedef PropertyInfo TriggerDelayInfo

  The TriggerDelayInfo structure is identical to PropertyInfo.
- typedef Property TriggerDelay

  The TriggerDelay structure is identical to Property.

Enumerations

- enum ErrorType { PGRERROR_UNDEFINED = -1, PGRERROR_OK, PGRERROR_FAILED, PGRERROR_NOT_IMPLEMENTED, PGRERROR_FAILED_BUS_MASTER_CONNECTION, PGRERROR_NOT_CONNECTED, PGRERROR_INIT_FAILED, PGRERROR_NOT_INTITIALIZED, PGRERROR_INVALID_PARAMETER, PGRERROR_INVALID_SETTINGS, PGRERROR_INVALID_BUS_MANAGER, PGRERROR_MEMORY_ALLOCATION_FAILED, PGRERROR_LOW_LEVEL_FAILURE, PGRERROR_NOT_FOUND, PGRERROR_FAILED_GUID, PGRERROR_INVALID_PACKET_SIZE, PGRERROR_INVALID_MODE, PGRERROR_NOT_IN_FORMAT7, PGRERROR_NOT_SUPPORTED, PGRERROR_TIMEOUT, PGRERROR_BUS_MASTER_FAILED, PGRERROR_INVALID_GENERATION, PGRERROR_LUT_FAILED, PGRERROR_IIDC_FAILED, PGRERROR_STROBE_FAILED, PGRERROR_TRIGGER_FAILED, PGRERROR_PROPERTY_FAILED, PGRERROR_PROPERTY_NOT_PRESENT, PGRERROR_REGISTER_FAILED, PGRERROR_READ_REGISTER_FAILED, PGRERROR_WRITE_REGISTER_FAILED, PGRERROR_ISOCH_FAILED, PGRERROR_ISOCH_ALREADY_STARTED, PGRERROR_ISOCH_NOT_STARTED, PGRERROR_ISOCH_START_FAILED, PGRERROR_ISOCH_RETRIEVE_BUFFER_FAILED, PGRERROR_ISOCH_STOP_FAILED, PGRERROR_ISOCH_SYNC_FAILED, PGRERROR_ISOCH_BANDWIDTH_EXCEEDED, PGRERROR_IMAGE_CONVERSION_FAILED, PGRERROR_IMAGE_LIBRARY_FAILURE, PGRERROR_BUFFER_TOO_SMALL, PGRERROR_IMAGE_CONSISTENCY_ERROR, PGRERROR_FORCE_32BITS = FULL_32BIT_VALUE }

  The error types returned by functions.
- enum BusCallbackType { BUS_RESET, ARRIVAL, REMOVAL, CALLBACK_TYPE_FORCE_32BITS = FULL_32BIT_VALUE }

  The type of bus callback to register a callback function for.

- enum GrabMode { DROP_FRAMES, BUFFER_FRAMES, UNSPECIFIED_GR-AB_MODE, GRAB_MODE_FORCE_32BITS = FULL_32BIT_VALUE }

  The grab strategy employed during image transfer.
- enum GrabTimeout { TIMEOUT_NONE = 0, TIMEOUT_INFINITE = -1, TIME-OUT_UNSPECIFIED = -2, GRAB_TIMEOUT_FORCE_32BITS = FULL_32BIT-_VALUE }

  Timeout options for grabbing images.
- enum BandwidthAllocation { BANDWIDTH_ALLOCATION_OFF = 0, BANDWI-DTH_ALLOCATION_ON = 1, BANDWIDTH_ALLOCATION_UNSUPPORTED = 2, BANDWIDTH_ALLOCATION_UNSPECIFIED = 3, BANDWIDTH_ALLOCAT-ION_FORCE_32BITS = FULL_32BIT_VALUE }

  Bandwidth allocation options for 1394 devices.
- enum InterfaceType { INTERFACE_IEEE1394, INTERFACE_USB2, INTERF-ACE_USB3, INTERFACE_GIGE, INTERFACE_UNKNOWN, INTERFACE_T-YPE_FORCE_32BITS = FULL_32BIT_VALUE }

  Interfaces that a camera may use to communicate with a host.
- enum PropertyType { BRIGHTNESS, AUTO_EXPOSURE, SHARPNESS, WH-ITE_BALANCE, HUE, SATURATION, GAMMA, IRIS, FOCUS, ZOOM, PAN, TILT, SHUTTER, GAIN, TRIGGER_MODE, TRIGGER_DELAY, FRAME_R-ATE, TEMPERATURE, UNSPECIFIED_PROPERTY_TYPE, PROPERTY_TY-PE_FORCE_32BITS = FULL_32BIT_VALUE }

  Camera properties.
- enum FrameRate { FRAMERATE_1_875, FRAMERATE_3_75, FRAMERATE-_7_5, FRAMERATE_15, FRAMERATE_30, FRAMERATE_60, FRAMERAT-E_120, FRAMERATE_240, FRAMERATE_FORMAT7, NUM_FRAMERATES, FRAMERATE_FORCE_32BITS = FULL_32BIT_VALUE }

  Frame rates in frames per second.
- enum VideoMode { VIDEOMODE_160x120YUV444, VIDEOMODE_320x240-YUV422, VIDEOMODE_640x480YUV411, VIDEOMODE_640x480YUV422, VIDEOMODE_640x480RGB, VIDEOMODE_640x480Y8, VIDEOMODE_-640x480Y16, VIDEOMODE_800x600YUV422, VIDEOMODE_800x600RGB, VIDEOMODE_800x600Y8, VIDEOMODE_800x600Y16, VIDEOMODE_-1024x768YUV422, VIDEOMODE_1024x768RGB, VIDEOMODE_1024x768Y8, VIDEOMODE_1024x768Y16, VIDEOMODE_1280x960YUV422, VIDEOMOD-E_1280x960RGB, VIDEOMODE_1280x960Y8, VIDEOMODE_1280x960Y16, VIDEOMODE_1600x1200YUV422, VIDEOMODE_1600x1200RGB, VIDEOM-ODE_1600x1200Y8, VIDEOMODE_1600x1200Y16, VIDEOMODE_FORMAT7, NUM_VIDEOMODES, VIDEOMODE_FORCE_32BITS = FULL_32BIT_VALUE }

  DCAM video modes.
- enum Mode { MODE_0 = 0, MODE_1, MODE_2, MODE_3, MODE_4, M-ODE_5, MODE_6, MODE_7, MODE_8, MODE_9, MODE_10, MODE_11, MODE_12, MODE_13, MODE_14, MODE_15, MODE_16, MODE_17, MOD-E_18, MODE_19, MODE_20, MODE_21, MODE_22, MODE_23, MODE_24, MODE_25, MODE_26, MODE_27, MODE_28, MODE_29, MODE_30, MO-DE_31, NUM_MODES, MODE_FORCE_32BITS = FULL_32BIT_VALUE }

  Camera modes for DCAM formats as well as Format7.

- enum PixelFormat { PIXEL_FORMAT_MONO8 = 0x80000000, PIXEL_FORMAT_411YUV8 = 0x40000000, PIXEL_FORMAT_422YUV8 = 0x20000000, PIXEL_FORMAT_444YUV8 = 0x10000000, PIXEL_FORMAT_RGB8 = 0x08000000, PIXEL_FORMAT_MONO16 = 0x04000000, PIXEL_FORMAT_RGB16 = 0x02000000, PIXEL_FORMAT_S_MONO16 = 0x01000000, PIXEL_FORMAT_S_RGB16 = 0x00800000, PIXEL_FORMAT_RAW8 = 0x00400000, PIXEL_FORMAT_RAW16 = 0x00200000, PIXEL_FORMAT_MONO12 = 0x00100000, PIXEL_FORMAT_RAW12 = 0x00080000, PIXEL_FORMAT_BGR = 0x80000008, PIXEL_FORMAT_BGRU = 0x40000008, PIXEL_FORMAT_RGB = PIXEL_FORMAT_RGB8, PIXEL_FORMAT_RGBU = 0x40000002, PIXEL_FORMAT_BGR16 = 0x02000001, PIXEL_FORMAT_BGRU16 = 0x02000002, PIXEL_FORMAT_422YUV8_JPEG = 0x40000001, NUM_PIXEL_FORMATS = 20, UNSPECIFIED_PIXEL_FORMAT = 0 }

  Pixel formats available for Format7 modes.
- enum BusSpeed { BUSSPEED_S100, BUSSPEED_S200, BUSSPEED_S400, BUSSPEED_S480, BUSSPEED_S800, BUSSPEED_S1600, BUSSPEED_S3200, BUSSPEED_S5000, BUSSPEED_10BASE_T, BUSSPEED_100BASE_T, BUSSPEED_1000BASE_T, BUSSPEED_10000BASE_T, BUSSPEED_S_FASTEST, BUSSPEED_ANY, BUSSPEED_SPEED_UNKNOWN = -1, BUSSPEED_FORCE_32BITS = FULL_32BIT_VALUE }

  Bus speeds.
- enum PCIeBusSpeed { PCIE_BUSSPEED_2_5, PCIE_BUSSPEED_5_0, PCIE_BUSSPEED_UNKNOWN = -1, PCIE_BUSSPEED_FORCE_32BITS = FULL_32BIT_VALUE }
- enum DriverType { DRIVER_1394_CAM, DRIVER_1394_PRO, DRIVER_1394_JUJU, DRIVER_1394_VIDEO1394, DRIVER_1394_RAW1394, DRIVER_USB_NONE, DRIVER_USB_CAM, DRIVER_USB3_PRO, DRIVER_GIGE_NONE, DRIVER_GIGE_FILTER, DRIVER_GIGE_PRO, DRIVER_UNKNOWN = -1, DRIVER_FORCE_32BITS = FULL_32BIT_VALUE }

  Types of low level drivers that flycapture uses.
- enum ColorProcessingAlgorithm { DEFAULT, NO_COLOR_PROCESSING, NEAREST_NEIGHBOR, EDGE_SENSING, HQ_LINEAR, RIGOROUS, IPP, DIRECTIONAL_FILTER, COLOR_PROCESSING_ALGORITHM_FORCE_32BITS = FULL_32BIT_VALUE }

  Color processing algorithms.
- enum BayerTileFormat { NONE, RGGB, GRBG, GBRG, BGGR, BT_FORCE_32BITS = FULL_32BIT_VALUE }

  Bayer tile formats.
- enum ImageFileFormat { FROM_FILE_EXT = -1, PGM, PPM, BMP, JPEG, JPEG2000, TIFF, PNG, RAW, IMAGE_FILE_FORMAT_FORCE_32BITS = FULL_32BIT_VALUE }

  File formats to be used for saving images to disk.
- enum GigEPropertyType { HEARTBEAT, HEARTBEAT_TIMEOUT, PACKET_SIZE, PACKET_DELAY }

  Possible properties that can be queried from the camera.

Variables

- static const unsigned int sk_maxStringLength = 512

  The maximum length that is allocated for a string.
- static const unsigned int sk_maxNumPorts = 32

  The maximum number of ports one device can have.

9.7.1 Define Documentation

9.7.1.1 #define FULL_32BIT_VALUE 0x7FFFFFFF

9.7.1.2 #define NULL 0

9.8 FlyCapture2GUI.h File Reference

Classes

- class CameraControlDlg

  The CameraControlDlg object represents a dialog that provides a graphical interface to a specified camera.
- class CameraSelectionDlg

  The CameraSelectionDlg object represents a dialog that provides a graphical interface that lists the number of cameras available to the library.

Namespaces

- namespace FlyCapture2

9.9 FlyCapture2Platform.h File Reference

Defines

- #define FLYCAPTURE2_API __attribute__ ((visibility ("default")))
- #define FLYCAPTURE2_LOCAL __attribute__ ((visibility ("hidden")))

9.9.1 Define Documentation

9.9.1.1 #define FLYCAPTURE2_API __attribute__ ((visibility (”default”)))

9.9.1.2 #define FLYCAPTURE2_LOCAL __attribute__ ((visibility (”hidden”)))

9.10 GCCamera.h File Reference

Classes

- class GCCamera

Namespaces

- namespace FlyCapture2

9.11 GigECamera.h File Reference

Classes

- class GigECamera

  The GigECamera object represents a physical Gigabit Ethernet camera.

Namespaces

- namespace FlyCapture2

9.12 Image.h File Reference

Classes

- class Image

  The Image class is used to retrieve images from a camera, convert between multiple pixel formats and save images to disk.

Namespaces

- namespace FlyCapture2

9.13 ImageStatistics.h File Reference

Classes

- class ImageStatistics

  The ImageStatistics object represents image statistics for an image.

Namespaces

- namespace FlyCapture2

9.14 Internal.h File Reference

Classes

- class Internal

Namespaces

- namespace FlyCapture2

9.15 MultiSyncLibrary.h File Reference

9.16 MultiSyncLibraryDefs.h File Reference

Classes

- class SyncManager

Namespaces

- namespace MultiSyncLibrary

Enumerations

- enum PGRSyncError { PGRSyncError_OK = 0, PGRSyncError_FAILED, PGR-SyncError_ALREADY_STARTED, PGRSyncError_ALREADY_STOPPED, PG-RSyncError_CAMERA_NOT_FOUND, PGRSyncError_UNKNOWN_ERROR }
- enum PGRSyncMessage { PGRSyncMessage_OK = 0, PGRSyncMessage_-STARTED, PGRSyncMessage_STOPPED, PGRSyncMessage_SYNCING, P-GRSyncMessage_NOMASTER, PGRSyncMessage_THREAD_ERROR, PGR-SyncMessage_DEVICE_ERROR, PGRSyncMessage_NOT_ENOUGH_DEVIC-ES, PGRSyncMessage_BUS_RESET, PGRSyncMessage_NOT_INITIALIZED, PGRSyncMessage_UNKNOWN_ERROR }

9.17 MultiSyncLibraryPlatform.h File Reference

Defines

- #define MULTISYNCLIBRARY_API __attribute__ ((visibility ("default")))
- #define MULTISYNCLIBRARY_LOCAL __attribute__ ((visibility ("hidden")))

9.17.1 Define Documentation

9.17.1.1 #define MULTISYNCLIBRARY_API __attribute__ ((visibility ("default")))

9.17.1.2 #define MULTISYNCLIBRARY_LOCAL __attribute__ ((visibility ("hidden")))

9.18 NodeMap.h File Reference

Classes

- class NodeMap

Namespaces

- namespace FlyCapture2

9.19 TopologyNode.h File Reference

Classes

- class TopologyNode

  The TopologyNode class contains topology information that can be used to generate a tree structure of all cameras and devices connected to a computer.

Namespaces

- namespace FlyCapture2

9.20 Utilities.h File Reference

Classes

- struct SystemInfo

  Description of the system.
- class Utilities

  The Utility class is generally used to query for general system information such as operating system, available memory etc.

Namespaces

- namespace FlyCapture2

Typedefs

- typedef void(∗ AsyncCommandCallback)(class Error retError, void ∗pUser-Data)

  Async command callback function prototype.

Enumerations

- enum OSType { WINDOWS_X86, WINDOWS_X64, LINUX_X86, LINUX_X64, MAC, UNKNOWN_OS, OSTYPE_FORCE_32BITS = FULL_32BIT_VALUE }

  Possible operating systems.
- enum ByteOrder { BYTE_ORDER_LITTLE_ENDIAN, BYTE_ORDER_BIG_ENDIAN, BYTE_ORDER_FORCE_32BITS = FULL_32BIT_VALUE }

  Possible byte orders.

Index