CHROMASENS

Offline User Manual for GCT Version 1.2.0

Table of Contents

Software > GCT > Introduction

Software > GCT > Installation

Software > GCT > GCT user interface

Software > GCT > GigE interface

Software > GCT > CXP interface

Software > GCT > GenICam-SDK

About this instruction manual

This instruction manual provides the necessary information for safe and efficient use of the product throughout its life cycle.

Representational tools

- Font markup **bold**: clickable areas in the software GCT.
- Font markup *italic*: windows and views to navigate to in the software GCT.

History of Documentation

Version 1.1.0

New features for GCT 2.5.0 release have been added, October 2023

Version 1.0.0

The initial version of this Documentation is July 2023.

History of GCT

GCT 2.7.1 (June 2024)

Release Notes:

Changes and new features

- Outlier suppression will be done only for pixels in ROI area between the left and right extrapolation columns.
- In DSNU/PRNU dialog, line plots and histograms will be updated and displayed for the the selected ROI region.
- GCT recognizes that a camera has been powered off and disconnected and informs the user with a dialog.
- User-set load button is disabled during streaming.

Bug fixes

- Fixed problem in generating sample projects with cmake.
- Histogram and line-plots are displayed for the image captured in Frame-trigger mode.
- Fixed Select-ROI feature of line plots in DSNU/PRNU generator dialog.
- Exposure time is updated properly when line-rate is changed.

- Single-channel view functions properly in the offline mode.
- Bugs in PRNU reference data creation with extrapolation are fixed.

GCT 2.7.0 (April 2024)

Release Notes:

Changes and new features

- New Functions added to CSI SDK
- SDK Functions to generate calibration data and upload it to camera and save the file in PC is added.
- A new sample to demonstrate usage of DSNU/PRNU generator SDK functions.
- Opening and displaying 10 bit and 12 bit images in GCT viewer.
- Images can be saved in LZW compressed TIFF format.
- In the Adjustment tool, the view type, single view channel is enabled.
- Mono image's line-plot and histogram feature is enabled.
- A message box for un-/successful load/save of user-set is displayed.

Bug fixes

- Spin boxes in Flash-controller plugin are disabled during streaming.
- Added "Image Count" section in the status bar of the central viewer.
- Added the possibility to save histogram and plot views.
- The image will be saved only if there is sufficient memory available in the PC.
- Right histogram and line-plots are displayed for last captured image, in frame-trigger mode.
- Fixed GCT-crash issue in master-slave mode device connection.
- The right device error code and description are displayed in a message box.
- The right range of axes values in lineplots and histograms can be set.

GCT 2.6.1 (January 2024)

Release Notes:

Bug Fixes

- fixed bug of setting led flash pattern duration incorrectly in the

configuration widget Flash controller.
- fixed bug of setting color transformation value incorrectly in the configuration widget Color.
- fixed bug of DSNU/PRNU not working for 16k image.

GCT 2.6.0 (December 2023)

Release Notes:

Changes and new features

- a new view type "vertical line plot" has been added.
- improved "horizontal line plot" ui.
- added link to online documentation.
- a function for saving 10- and 12-bit Tiff images in GCT has been added.
- added support for saving compressed TIFF files.
- refactored the line trigger plugin and frame trigger plugin.
- updated Kithara driver to 11.13.

Changed obsolete names

- Kithara driver from LakesightDrv to TKHVision.
- Kithara GenTL from ls_tl_gev_kithara.cti to tkh_tl_gev_kithara.cti.

Bug Fixes

- fixed streaming image issue with Kithara TL when stop acquisition.
- fixed general problem of image acquisition.
- fixed bug in configuration widget for sensor sensitivity of mono cameras.

GCT 2.5.0 (October 2023)

Download the Documentation for GCT Version 2.5.

Release Notes:

- Added split view functionality
- Improvements in DSNU/PRNU: Calculation improvement to suppress outliers like dust areas, Improvement to specify roi
- Added image buffer for image
- Updated Filter Driver to V2.7.2
- Updated Kithara Driver to 11.12
- Performance Tool using CSI API
- The SDK function "csiGetNextImage" has been modified. The second function argument is changed from

csiNewBufferEventData* to csiNewBufferEventData**. This output variable that contains the acquired image data must be modified accordingly

- A new view-type "Single-channel" is added. This view type displays just the selected channel's image data onto the image view
- Bugfixes

GCT 2.4.0 (May 2023)

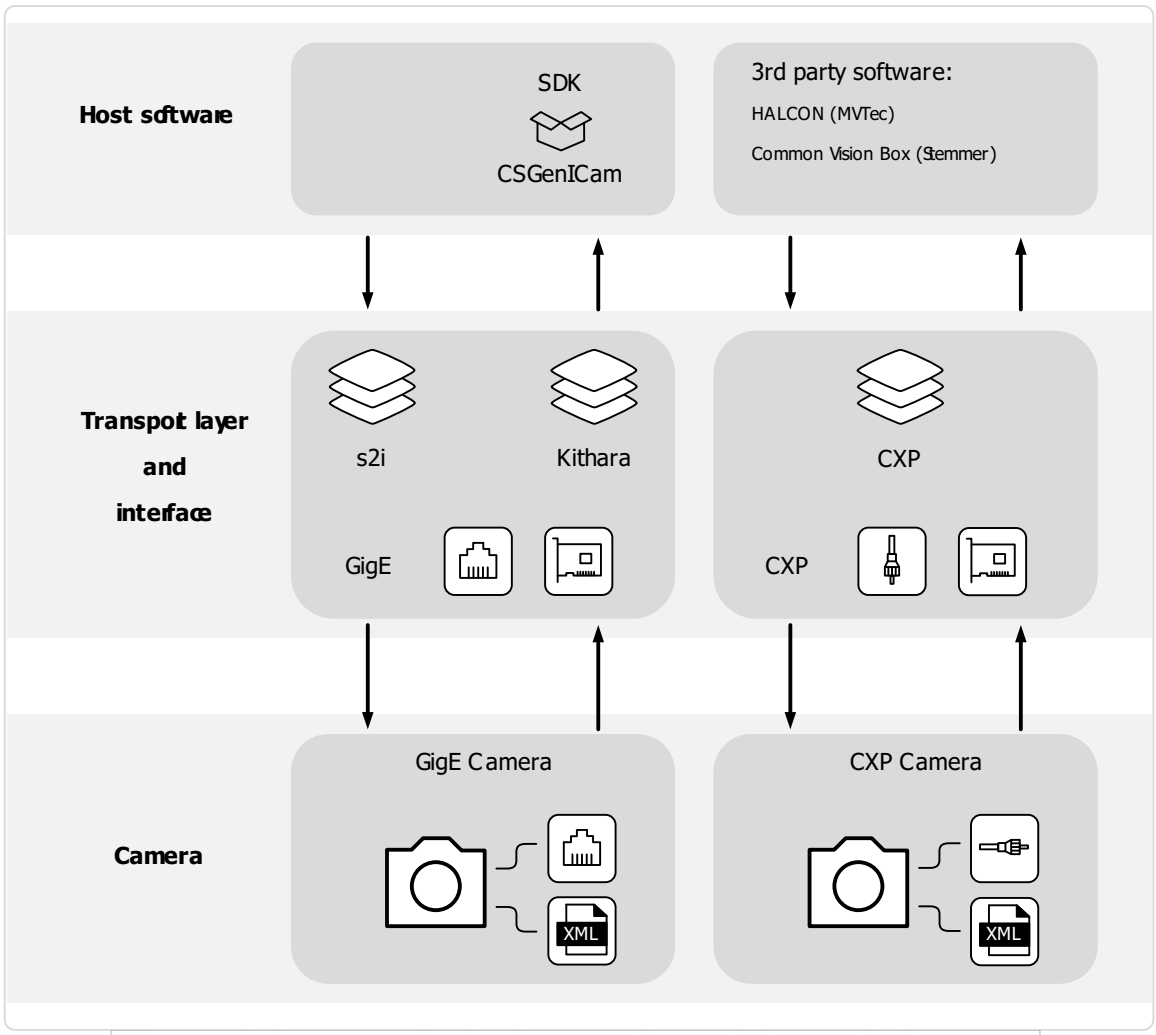
Download the Documentation for GCT Version 2.4.

GCT 2.3.0

Download the Documentation for GCT Version 2.3.

Overview

The GenICam Control Tool (GCT) permits communication with cameras that fulfill the GenICam standards. The tool allows one to set up a camera, browse and adjust parameters, perform camera calibration tasks, and visualize and analyze acquired images.
The following graphic shows an overview of the connection between the GCT software and the camera. The host software uses the CSGenICam SDK to connect to the camera. Regarding the GenIcam GenTL standard, the software and your program are interface-independent. The only difference is that the appropriate transport layer should be used for each interface type. This means if you are using a GigE camera, you need a network adapter and the s2i or Kithara transport layer, if you want to use a CoaXPress camera you need a frame grabber and the CoaXPress transport layer (delivered from the frame grabber vendor).



Scheme of the communication between a host software and the camera

Intended use

- The device is designed for machines and systems which are used for commercial and industrial applications.
- The device is designed for contactless optical detection of primarily two dimensional objects.
- The device may only be connected or used as described in this manual.
- Do not use the device in safety relevant control circuits and potentially explosive environment.

System Requirements

Hardware requirements

- TKH Vision GenICam-standard camera
- Intel Core i7 or higher
- 16GB RAM or higher
- Min. PCIe 3.0 lane with min. 8 lanes to install your frame grabber or network card

Software requirements

- Driver for 10-Gigabit Ethernet network interface card
- Driver for frame grabber software

Supported operating systems

- Windows (GCT_win64_v2.7.1.exe)
    - Windows 10 22H2
    - Windows 11 23H2
- Debian (csgenicam-2.7.2.deb)
    - Ubuntu 20.04
    - Ubuntu 22.04
- RPM (csgenicam-2.7.2×86_64.rpm)
    - OpenSUSE Leap 15.4
    - AlmaLinux 9.4

Example for tested systems

allPIXA neo 10GigE

With the appropriated hardware, the allPIXA neo was operated without any limitations.

Camera	Network Adapter	PC	CPU	Memory
allPIXA neo 10GigE CP000660-XX	Broadcom P210 TP	Precision 3680 Tower Workstation	Intel Core i7 14700 14. generation	16 GB

Installation GigE

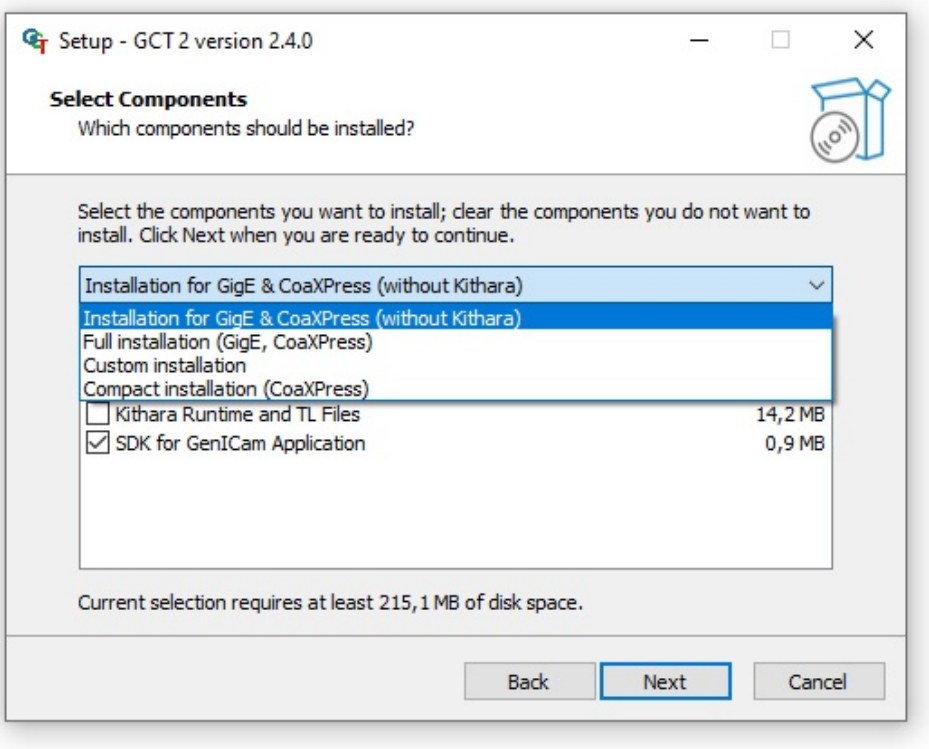
Note GigE Transport Layer

GCT delivers two different Transport Layers (TL) in the installer. You can decide between the *s2i* TL and the *Kithara* TL. For a real-time application with high frame rates, it is recommended to use the *Kitahra* TL. Furthermore, it is recommended to set up your system with the *s2i* TL and if necessary activate the *Kitahra* TL later on. With this **installation**, you can also connect a **CXP** camera with **GCT**

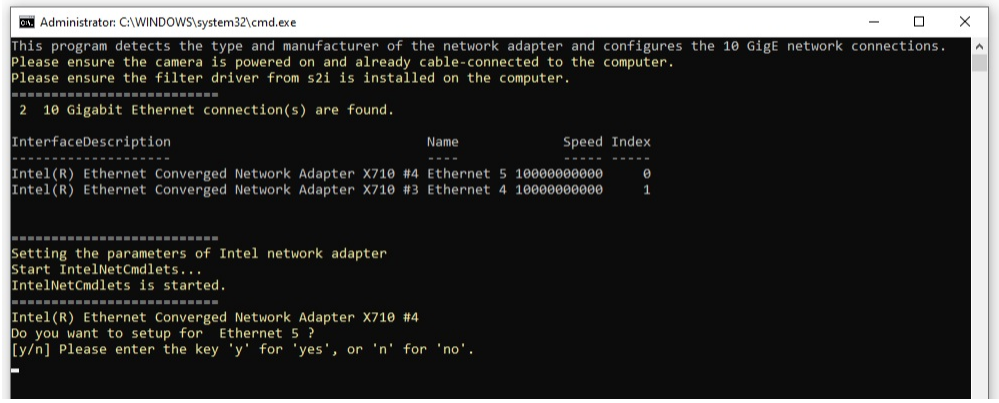
For the allPixa neo and a Broadcom network adapter, you can use the *s2i* TL.

To install the GCT software for Windows:

1. Download the installer from our homepage.
2. Install the NIC-10GigE-Driver Files, please refer to Network adapters and transceivers.
3. Connect the camera to the PC and turn it on.
4. Start the GCT installer and follow the instructions.
5. At the window *Select Components:* Select the **Full installation**, if you want to install *Kithara* or use the **Installation for GigE & CXP** if you only need the *s2i* TL.



6. For *Kithara* a License Dongle is required
7. During installation, you will be asked if you want to configure the **GigE network adapters**, if you are using a network adapter with an *Intel* chipset, you should answer with **Yes**
    a. The command line window lists all available ethernet adapters and you can configure the network adapter by entering a **Y** into the script. Now the script setup your network card.
    b. For each adapter, you will be asked in the script.



> **Note**
>
> If an error occurs during the execution of the script you have to configure the network adapter manually. Make sure that the **parameters** of your **network adapter** are set correctly. For more information, see **Configure the network adapter**.

8. After the configuration of the network parameter, the script can help you to set a static IP Address of the network adapter. *We recommend disagreeing* by entering **N** into the window. If you want to set a static IP Address for your network, you can answer with **Y**.
9. Restart the PC
10. If you want to use Kithara please configure Kithara after the restart. To set up *Kithara,* please follow the instructions 10 GigE with Kithara transport layer.

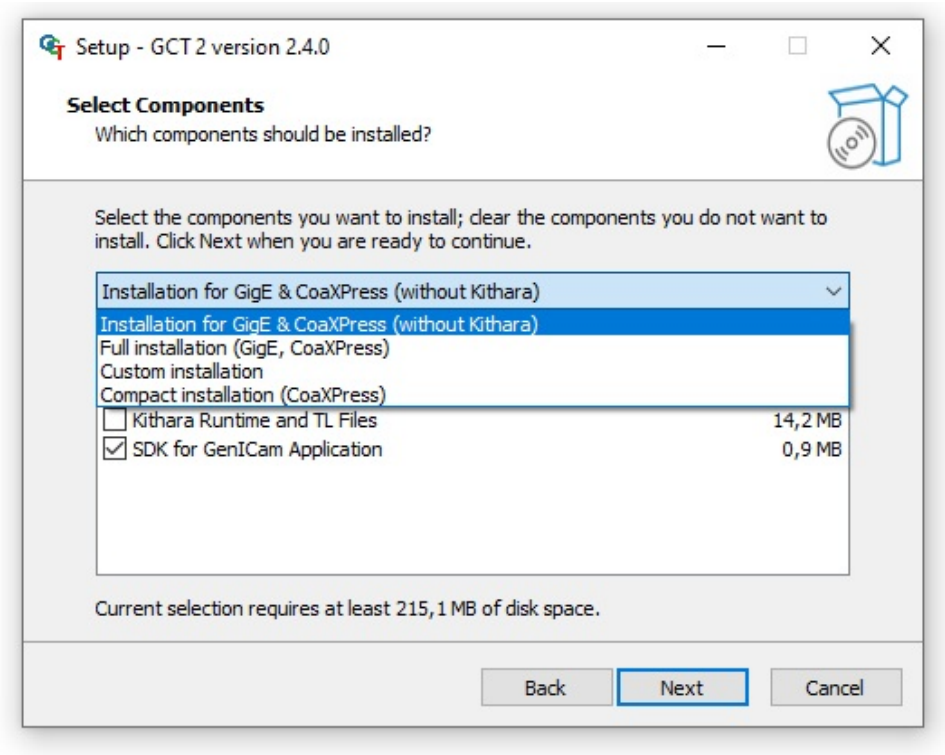
Installation CXP

Note

The connection between the CXP camera and GCT is done with a transport layer from the frame grabber vendor. For the driver of the CoaXPress frame grabber as well as the transport layer, please follow the instructions of the grabber's manufacturer.

To install the GCT software for Windows:

1. Download the installer from our homepage.
2. Install your *Framegrabber* software
3. Start the GCT installer and follow the installation instructions.
4. At the window *Select Components:* Select the **Compact installation** for a CXP camera.

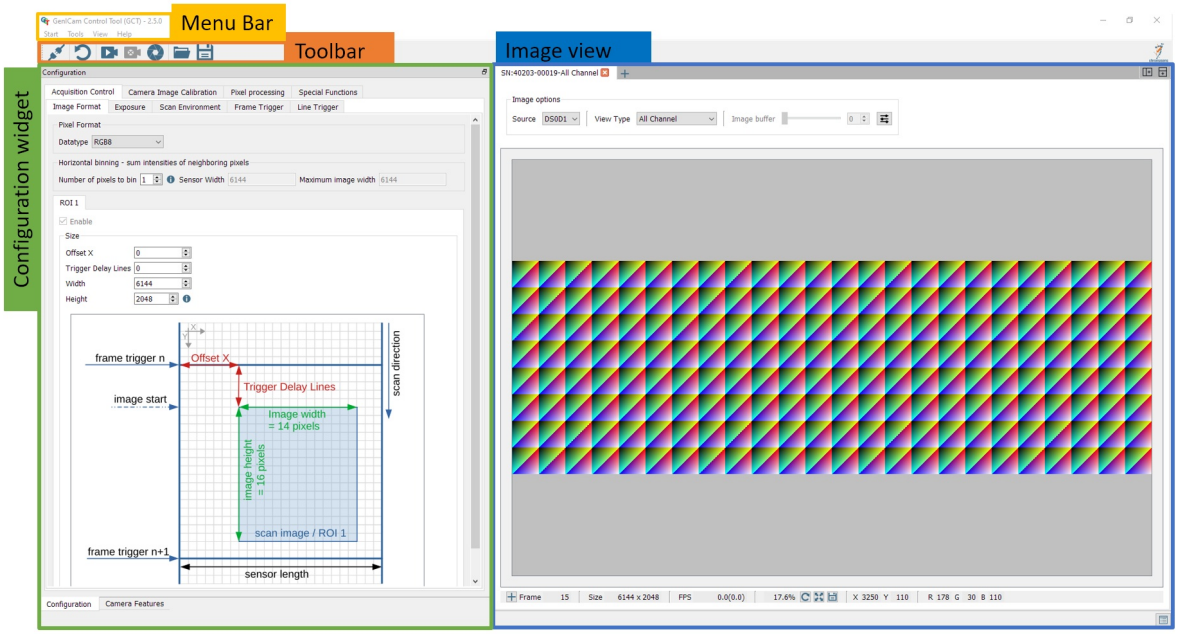


5. After the installation please *restart* your PC.

Overview

The software provides different panel elements, they can be placed, resized, and docked freely within GCT window. The GCT user interface starts after a successful connection with your camera. The window contains four main areas. The Menu bar widget is for general access, the Toolbar widget is for fast access to the most common functions, the image view widget is where the image and all relevant information are displayed, and the configuration widget is for the camera configuration.

Main window



Menu Bar

The Start Menu offers the following functions:

- Connect/Disconnect: Opens device discovery widget or disconnect the camera,Connection and disconnection of Camera
- Start Grabbing: Starts continues image grabbing
- Device description: Load and save XML file from the camera
- Settings: GCT setting options, GCT Options
- Exit: Close the GCT window

The Tools Menu offers the following functions

- Calibration: Opens a camera calibration dialog, Create a black-reference (DSNU) and Create a shading-reference (PRNU)

- Maintenance Mode: Only for experts
- Up/Download: Opens an Up/Download dialog, Up-Download
- Register Editor: Only for experts
- Perform Adjustment

The View Menu offers the following functions

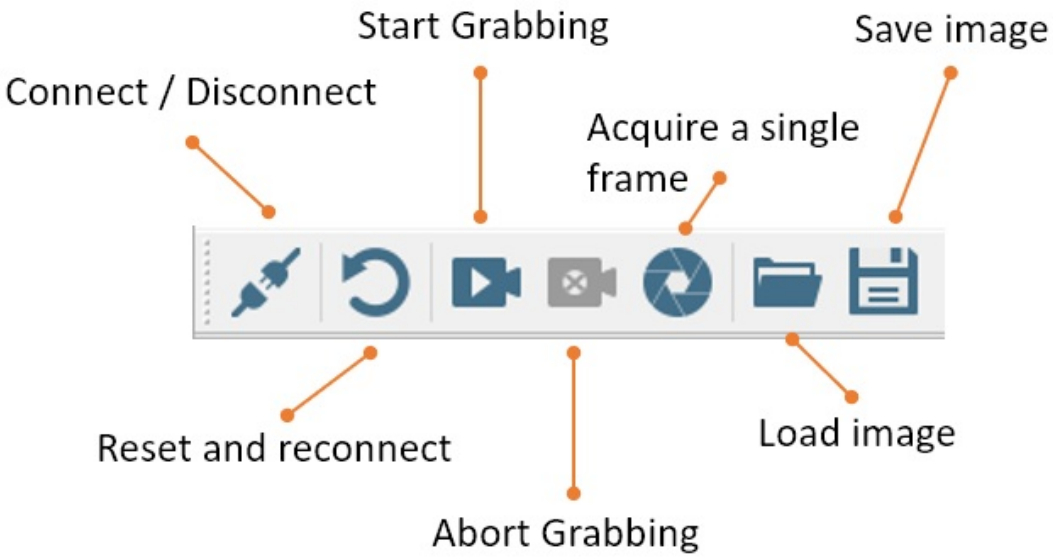
- Default view: Set the GCT view to default
- Configuration Widget: Shows the configuration panel in the configuration widget
- Feature Widget: Shows the Feature tree panel in the configuration widget
- Image View: Shows this Image view window
- Message Log: Display the Message Log under the Image view
- Refresh Feature Tree: Refreshes the feature tree

The Help Menu offers the following functions

- About: General Information about GCT
- Open GCT Manual: Opens the GCT Manual
- System Information: Generates a System information report
- Search for GCT Update: Search for GCT updates

Toolbar

The Toolbar contains the following icons and their functions.



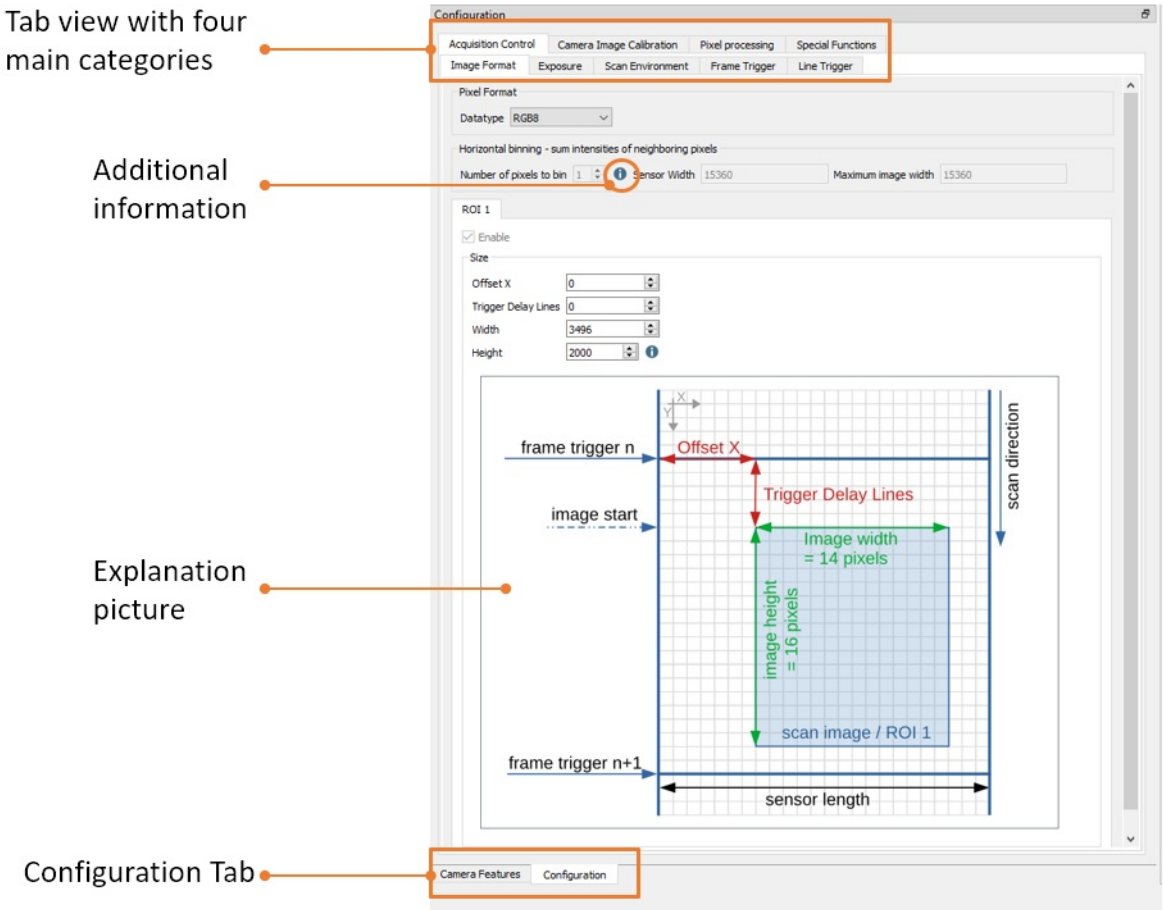
The Toolbar offers the following button functions:

- Connect/Disconnect: Opens device discovery widget or disconnect the camera, Connection and disconnection of Camera

- Reset and reconnect: Resets the camera and reconnect
- Start Grabbing: Starts continues image grabbing
- Abort Grabbing: Stops the image acquisition
- Acquire a single frame: Acquires a single frame
- Load image: Load an image from the disk
- Save image: Save image to disk

Configuration widget

The Configuration widget provides two panels, Camera features which is the GenIcam parameter tree, and the configuration panel which is a graphical representation of the GenIcam parameter tree.
The following image shows the configuration panel of the configuration widget. The panel contains a tab view, from which four main categories can be selected. Each tab contains another tab view with subcategories. The content in each sub-tab view is scrollable.
An Info box or tooltip is shown when hovering over a parameter input field or the blue information icon.



The Camera Features tab contains the GenIcam parameter tree, the user setup level, and an extra panel to set up the filter driver.

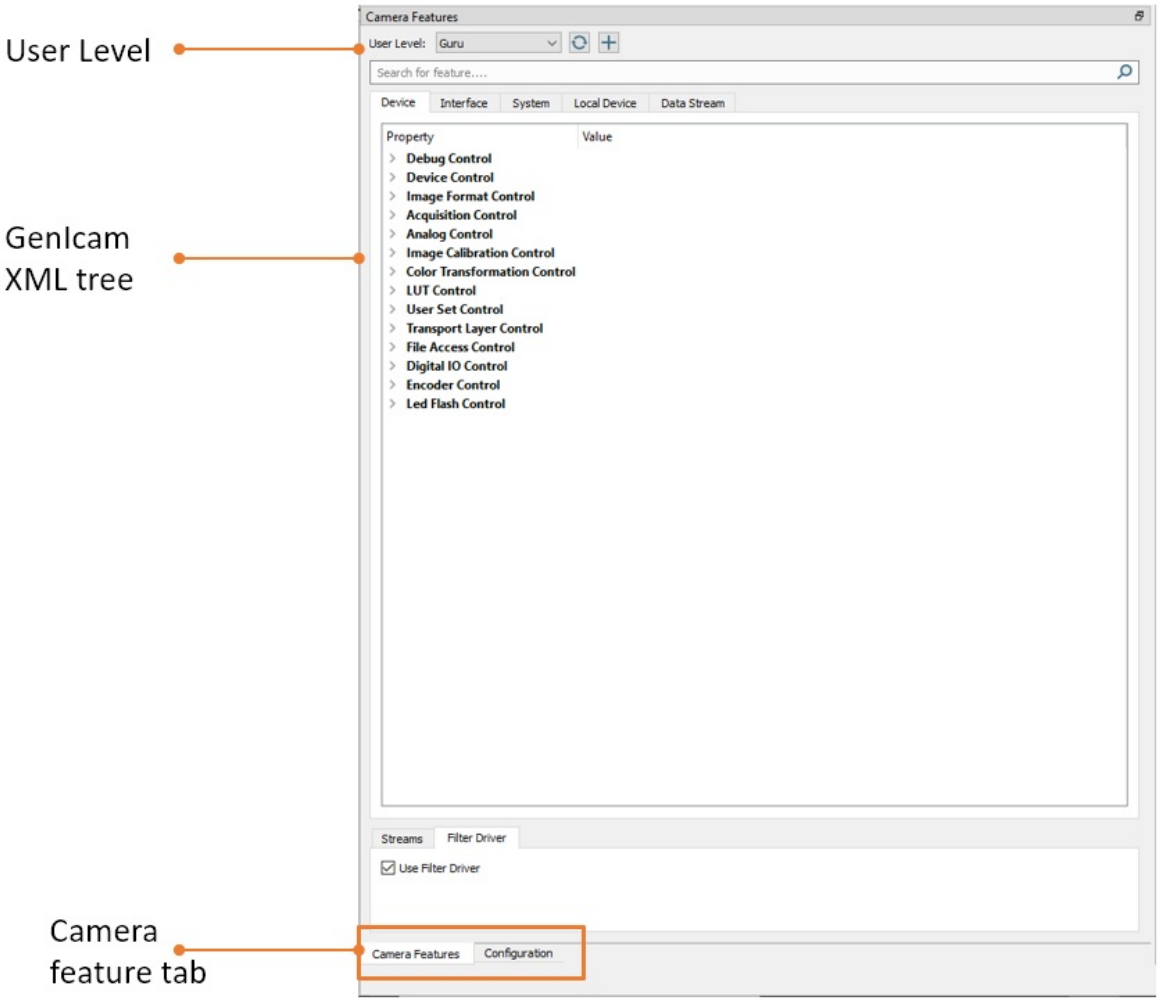
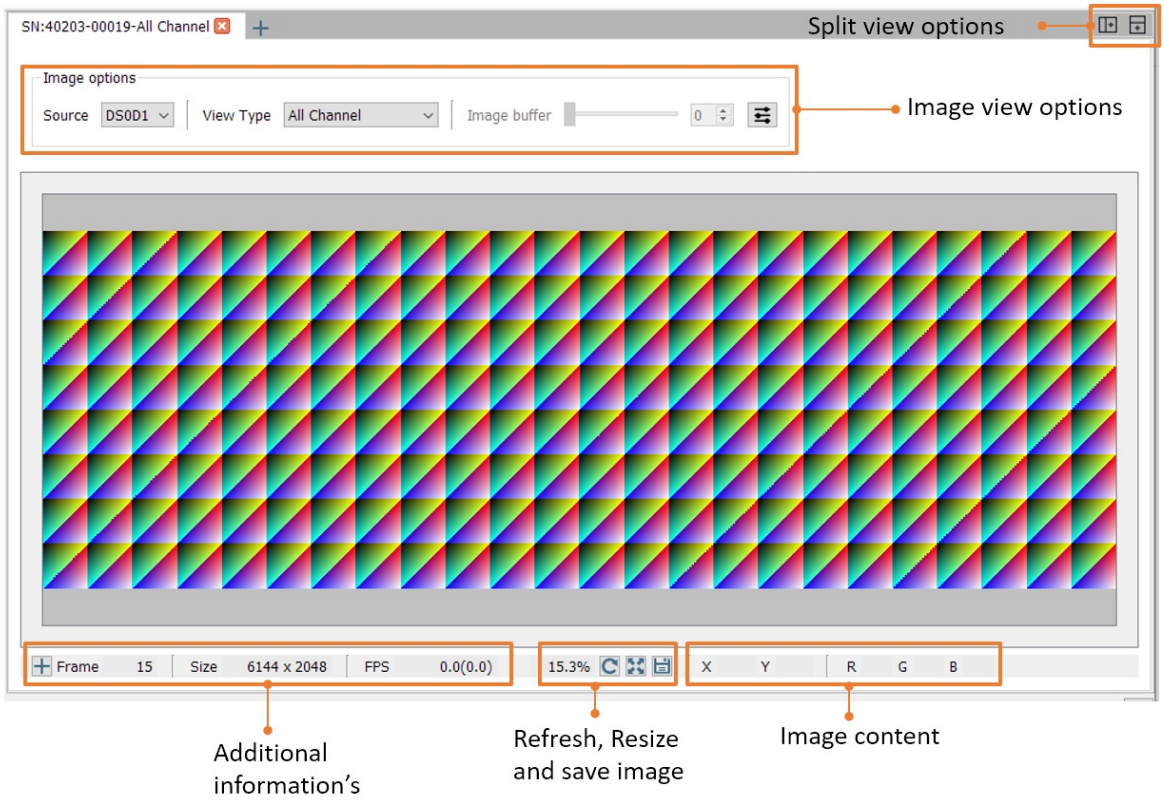


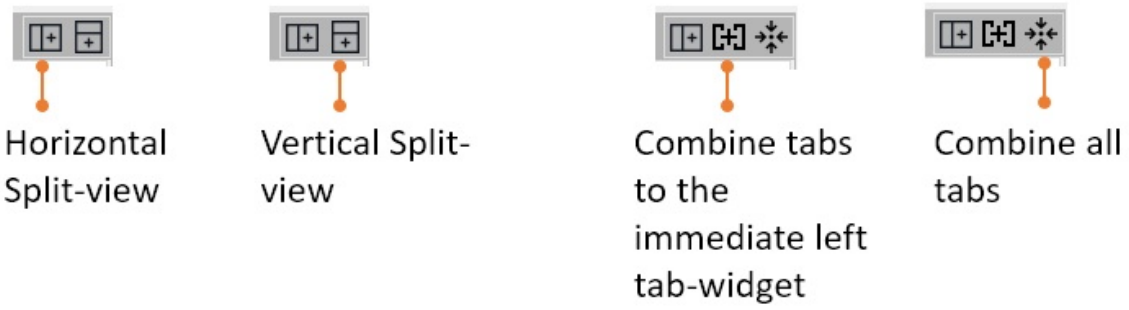
Image view widget

The image view widget shows the current image view and additional image and stream relevant information.



Split-view

The Split View option allows you to display multiple tabs side by side or on top of each other. As soon as two or more tabs are open you can select between the **Horizontal Split-view** and the **Vertical Split view** option.



To end the Split-view you can choose between **Combine tabs to the immediate left tab-widget** or **Combine all tabs**.
The following image shows an example of the Horizontal Split-view.

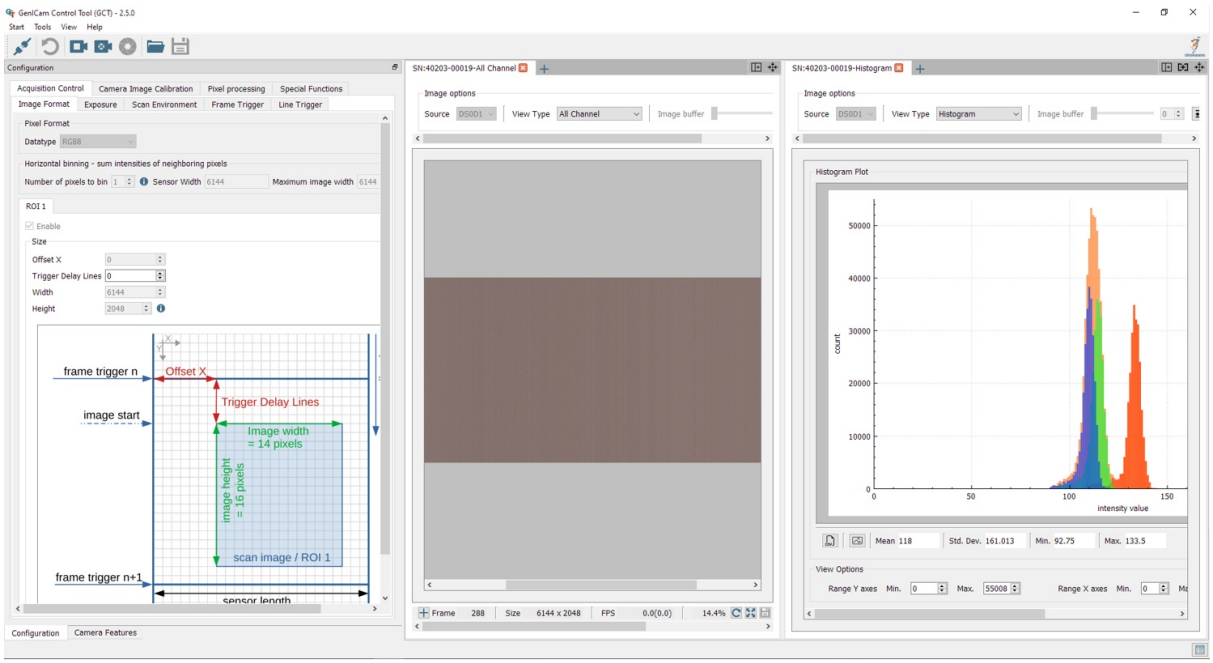
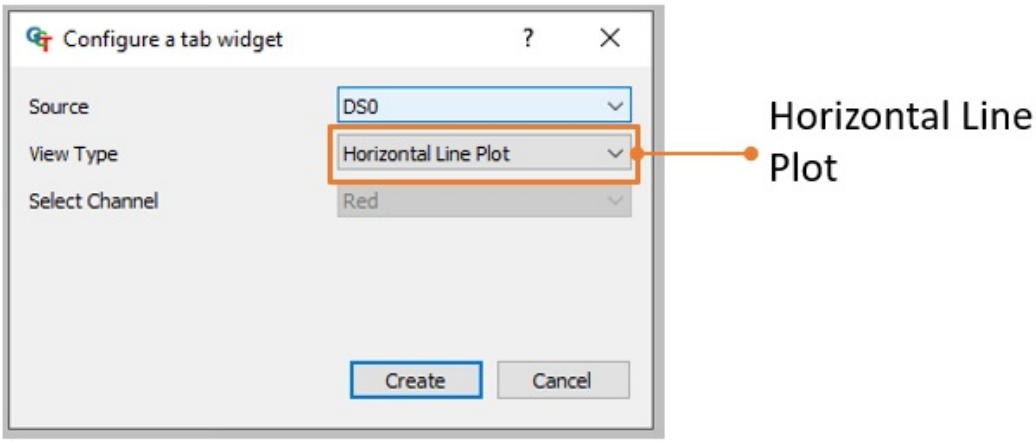


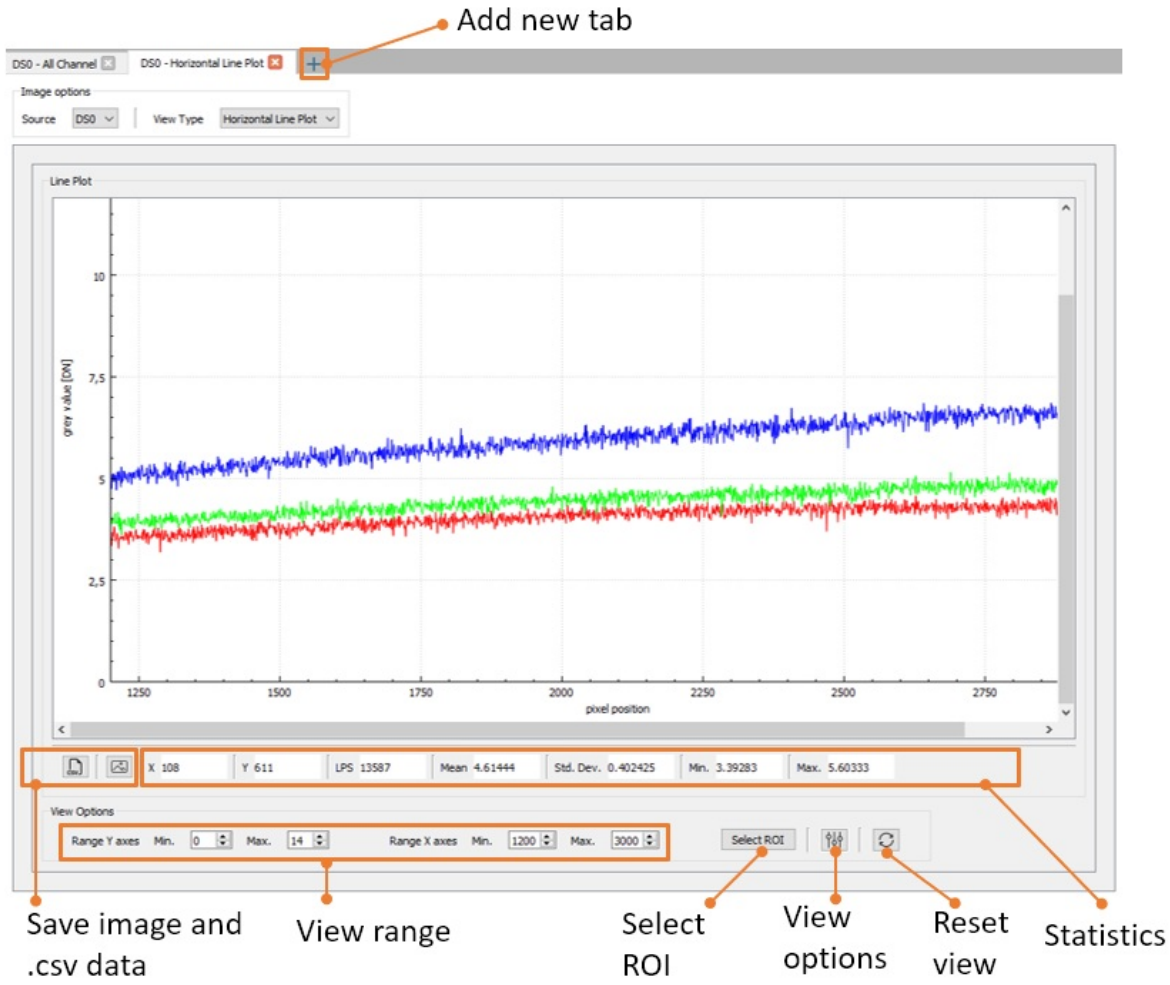
Image view options

Line plot view

GCT can display a horizontal line plot in the image view widget. To add this feature, add a new tab, therefore press the plus on the top. A popup window will be open, please select under the “View Type” the “Horizontal Line plot.

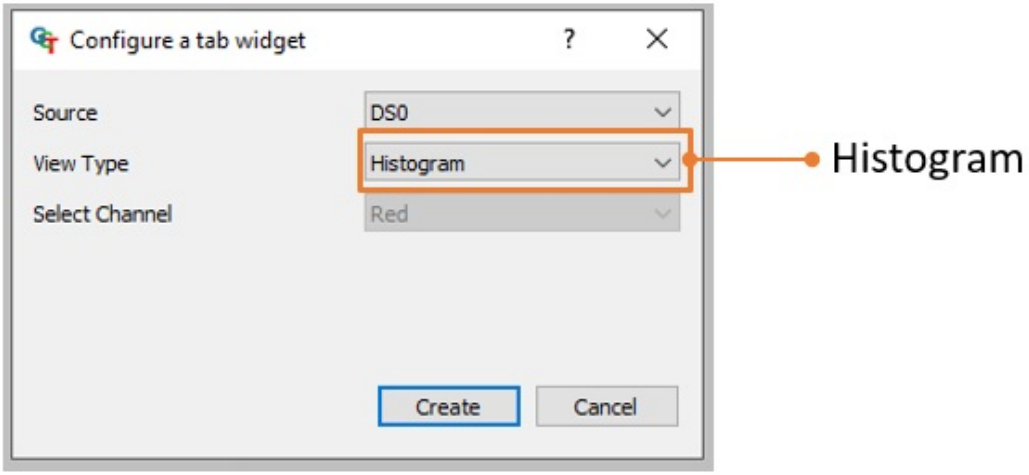


After this configuration, a new window will open as shown in the following figure.

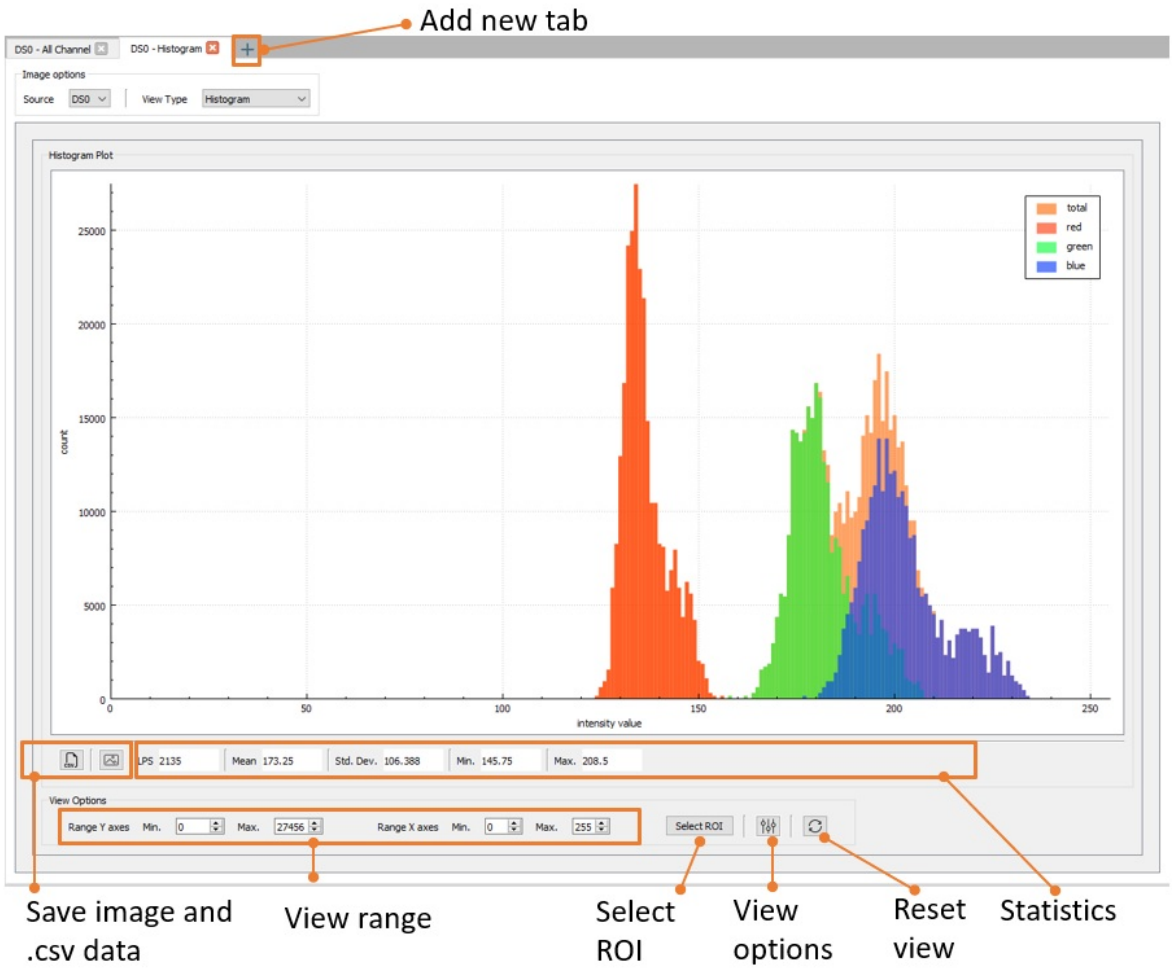


Histogram view

GCT can display a histogram of the image content in the image view widget. To add this feature, add a new tab, therefore press the plus on the top. A popup window will be open, please select under the “View Type” the “Horizontal Line plot.

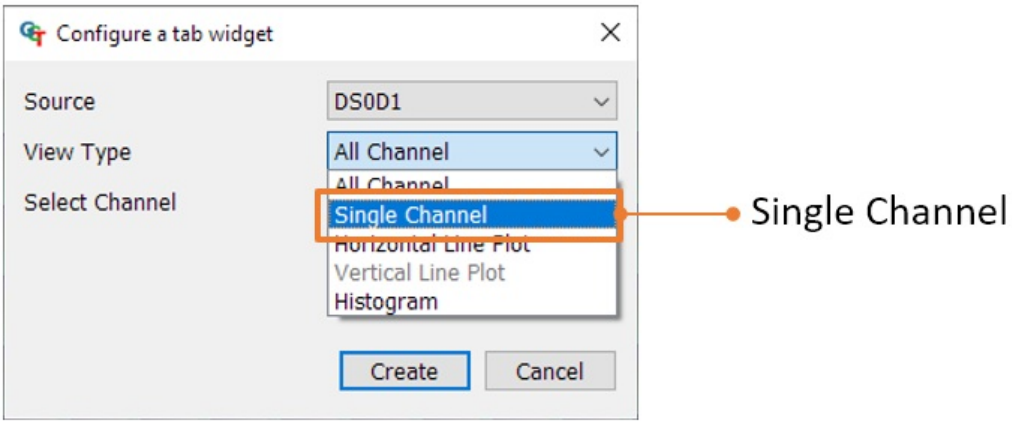


After this configuration, a new window will open as shown in the following figure.



Single Channel

GCT can display only a Singel Channel of the image content in the image view widget. To add this feature, add a new tab, therefore press the plus on the top. A popup window will be open, please select under the “View Type” the Single Channel.



After this configuration, a new window will open as shown in the following figure.

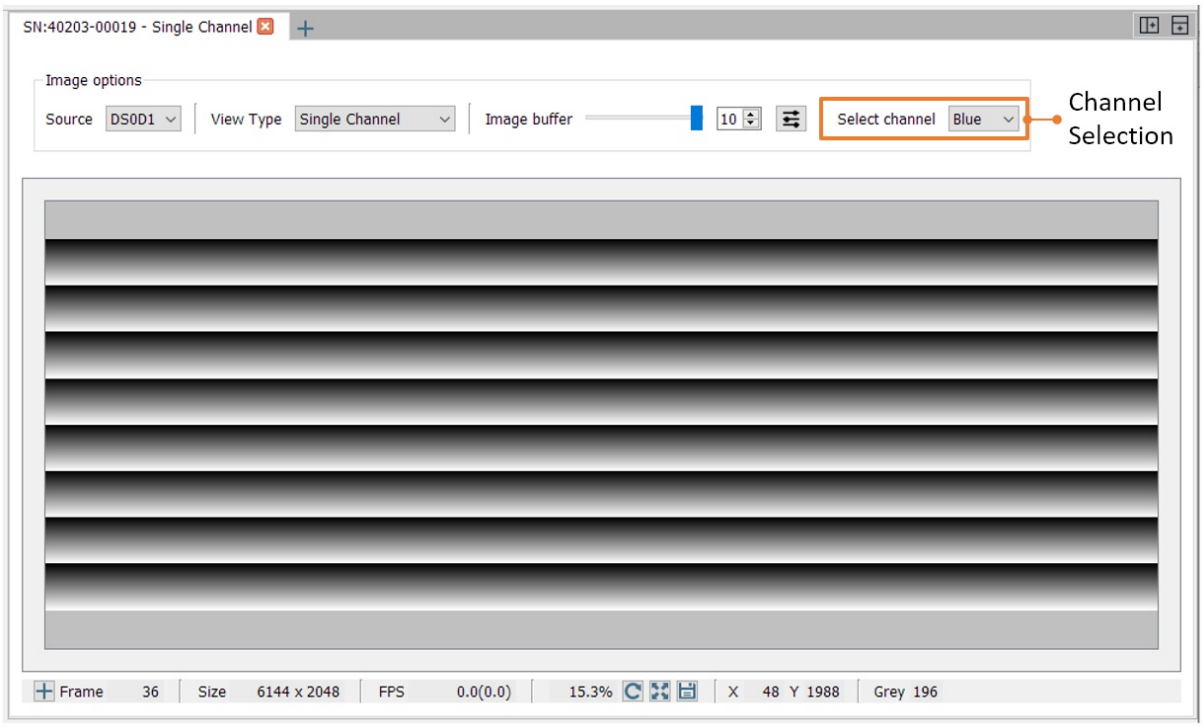
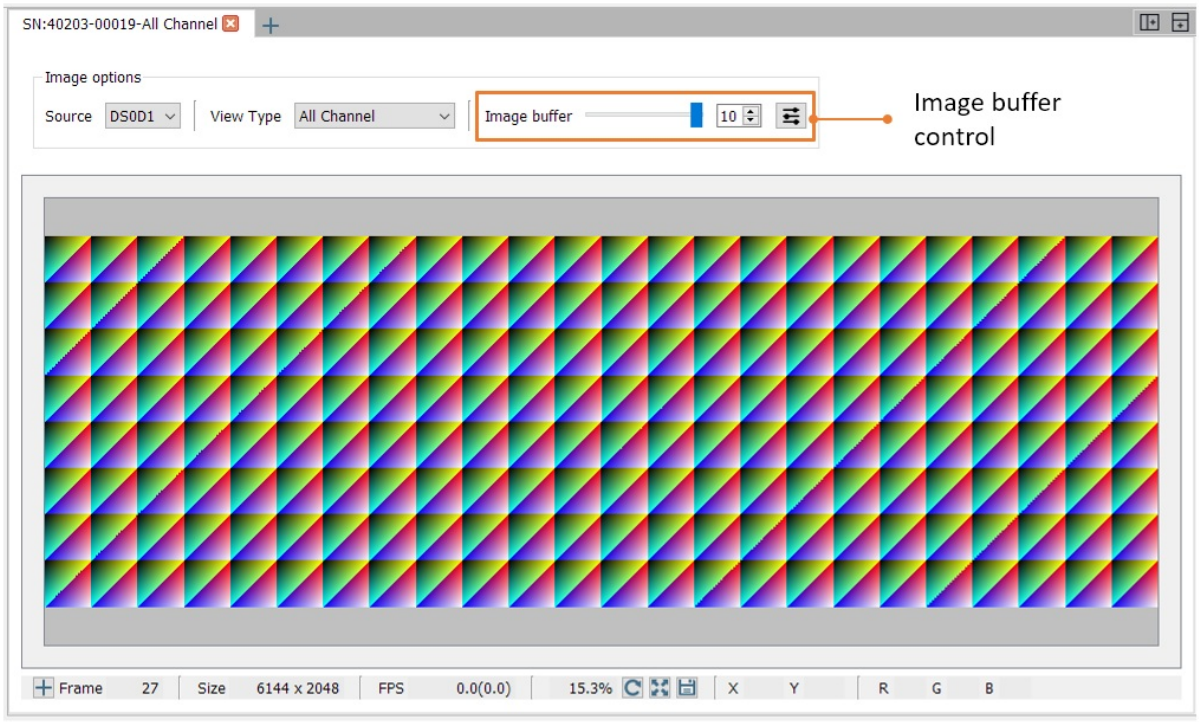


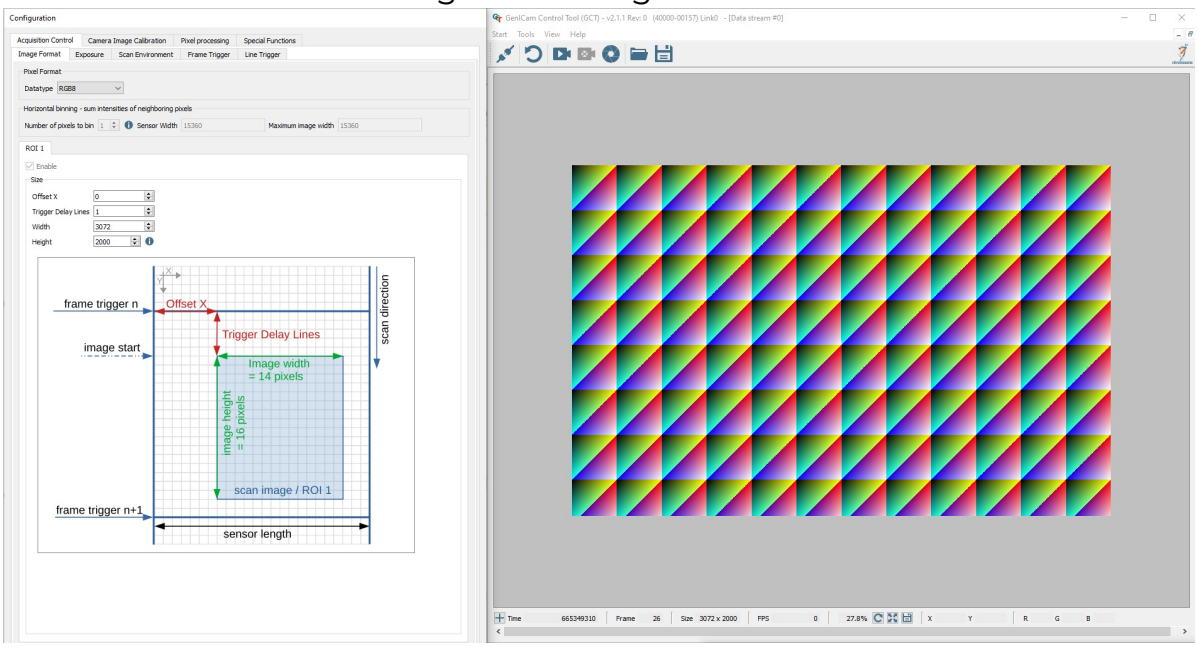
Image buffer

Multiple images can be buffered in GCT and displayed individually after stopping the acquisition. To enable this option, *click* on the right icon with the tree sliders. A new window will be opened *check* the **Enable image buffer** option.

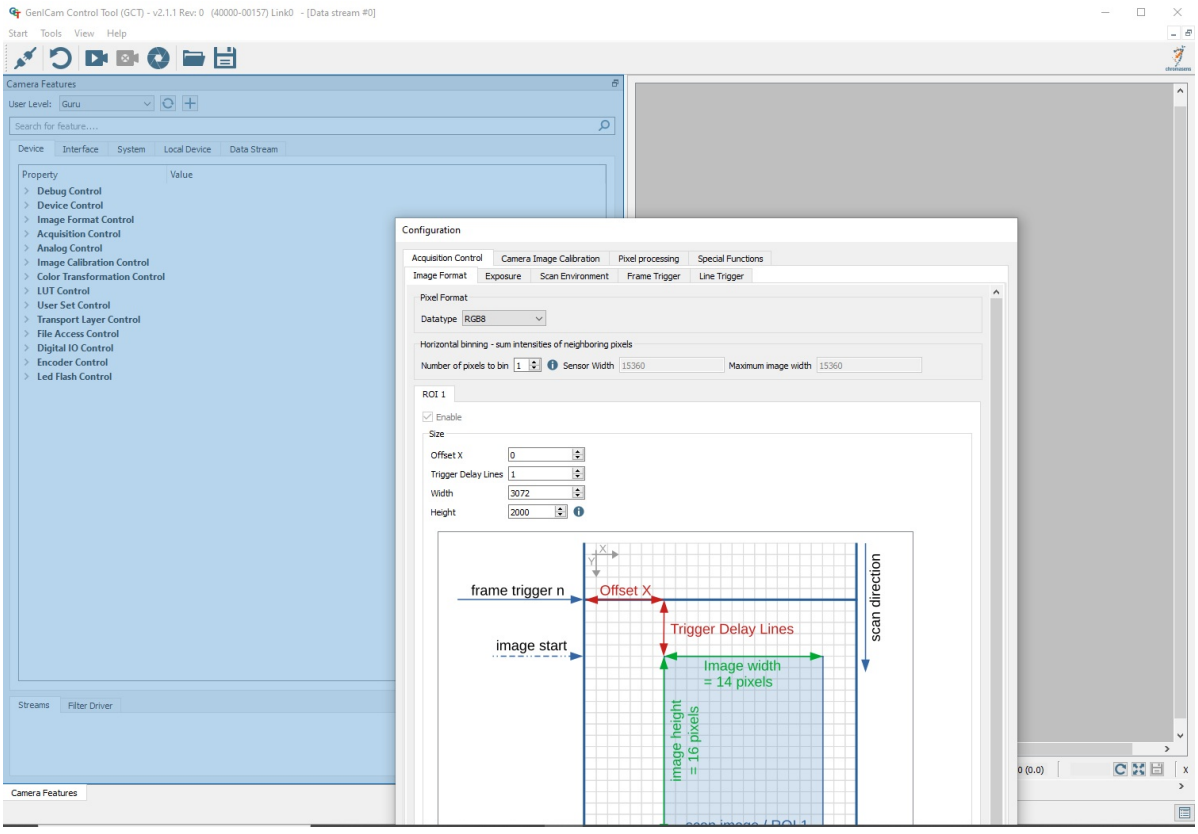


Arrange / Rearrange widgets

The software provides different widget elements, they can be placed, resized, and docked freely within GCT window. If you are using two monitors you can place, for example, the configuration widget on the second screen and the image view widget on the first screen.

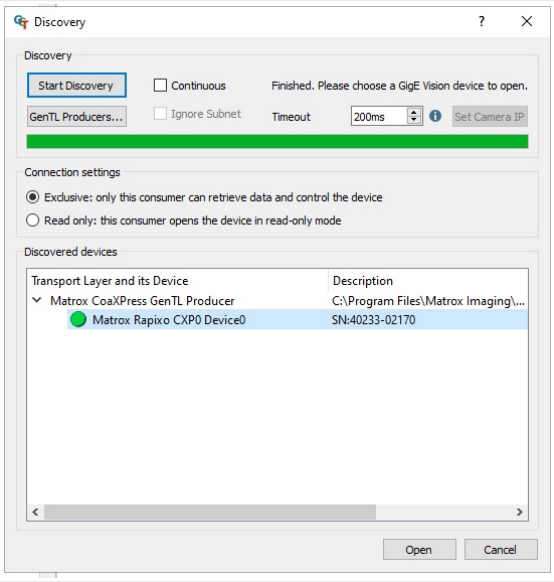


The software provides different widget elements, they can be placed, resized, and docked freely within GCT window. If you are using two monitors you can place, for example, the configuration widget on the second screen and the image view widget on the first screen.



Connection and disconnection of Camera

Connecting the camera

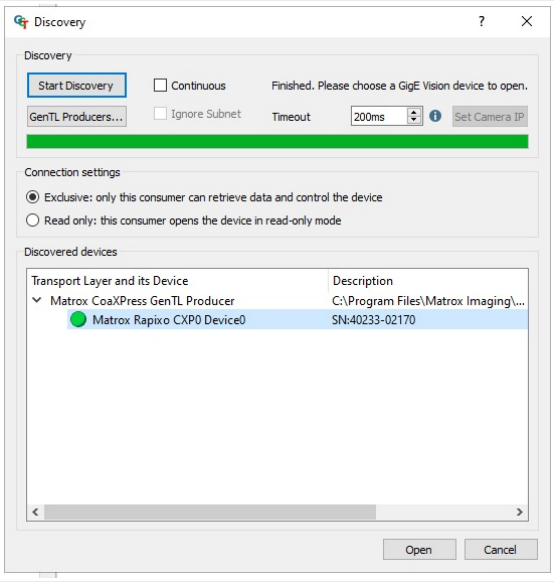
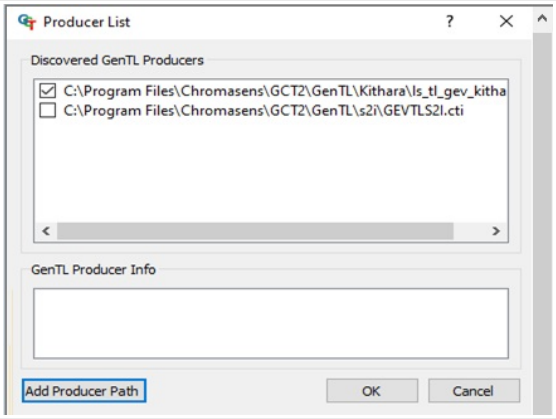
1. Switch on the camera. Initialization can take up to about 40 seconds.	
2. Start **GCT** from your *desktop*.	
3. On the **toolbar** *click* **Run device discovery**.	
4. *Click* **Start Discovery** in the *Discovery* dialog. The Dialog window shows all discovered cameras.	
5. Select the camera and *click* **Open.**	

Connection two cameras with one PC

The latest GCT2 version does not have support for multiple cameras in one GCT2 instance. To connect more than one camera to the PC, you need to open two instances (Windows) to connect the camera to a PC.
If you connect two cameras to the PC and open a new instance of GCT2, you can select one camera and connect it to the GCT2 instance. Both cameras are visible in the Discovery window. With a second GCT2 instance, you can connect the second camera.

Edit Transport Layer

The Transport Layer (TL) is used to refer to the interface such as GigE (Kithara or s2i) or CoaXPress.
To change the Transport Layer, please follow the instructions below.

1. Switch on the camera. Initialization can take up to about 40 seconds.	
2. Start **GCT** from your *desktop*.	
3. On the **toolbar** *click* **Run device discovery**.	
4. *Click* **Start Discovery** in the *Discovery* dialog. The Dialog window shows all discovered cameras.	
5. *Click* **GenTL Producers.**	
	
6. Select the certain file, for example, **ls_tL_gev_kithara.cti** or **GEVTLS2I.cti**, and *click* or *clear* the bock	
7. Confirm your configuration with **OK**.	

Note

During the first discovery process after installation, GCT fetches the paths from the system environment variable GENICAM_GENTL64_PATH and searches for .cti files in each folder. If the transport layer files are found and parsed, the camera can be detected with the corresponding transport layer. This may take several minutes during the first discovery process.

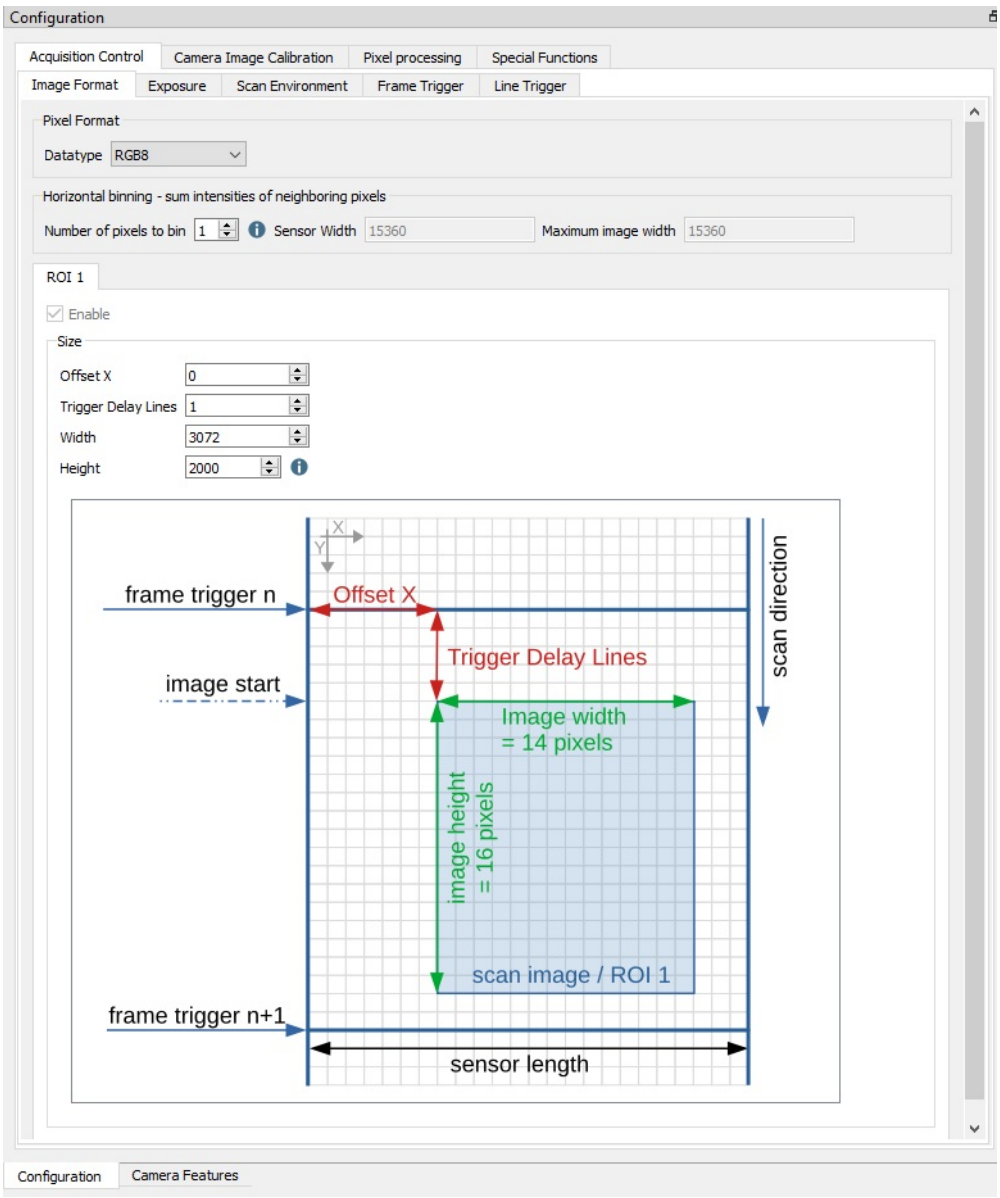
Modifying and showing features

Modifying features

GCT allowed to modify the features in two different ways. On the one hand, the configuration panel makes setting functions more convenient, special views show a set of related functions on which you can easily change these parameters. On the other hand, the camera feature panel allowed to edit the parameter in the XML-Tree.

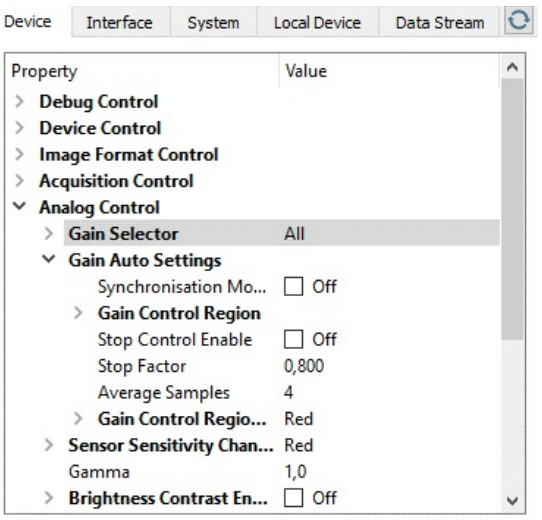
In the Configuration panel

The view consists of four main tabs Acquisition Control, Camera Image Calibration, Pixel Processing and Special functions. Each tab shows several subtabs.
The Acquisition Control tab, for example, show five subtabs, which permit to specify many features such as image format and trigger settings. Some features also have additional explanation pictures.



In the camera feature panel (XML Tree)

On the left side of the GCT window, features and their values are shown in a tree structure organized in feature groups. To show the features of a feature group, click on the arrow preceding the group name.



Modifying features

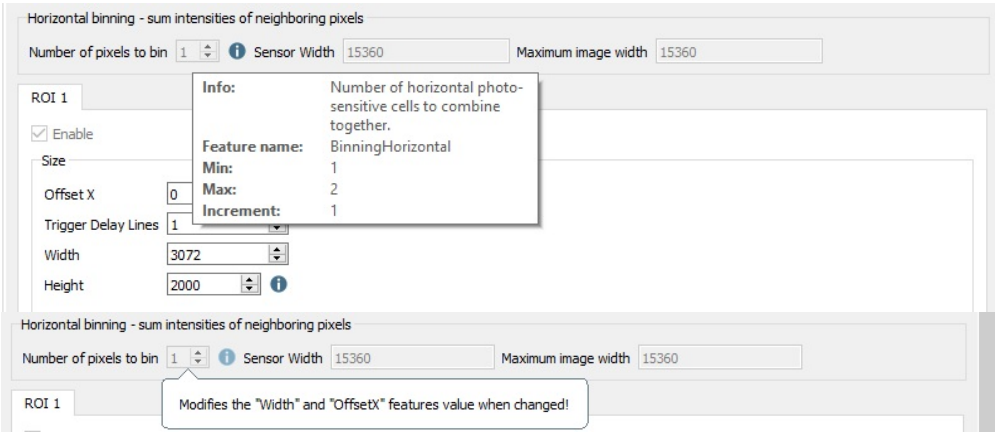
In the configuration panel, the information is displayed when you move the mouse pointer over the parameter input field or the blue information icon.

Showing feature details

Feature details can be displayed in the configuration panel and in the camera feature panel.

In the Configuration panel

In the configuration panel, the information is displayed when you move the mouse pointer over the parameter input field or the blue information icon.



Up-Download

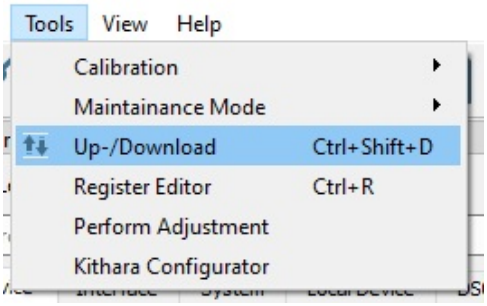
	NOTICE
	Irreparable damage to the camera
	If the camera ist powered down during firmware update it may get into a non-functional state. Recovery may not be possible.

The Up-/Download dialog allows you to Upload files from the PC to the Camera, for example, Camera Package files, as well as user sets. You can also download files from the camera to the PC. Supported file types and extensions depend on the camera type.

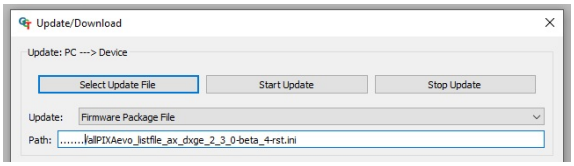
Uploading files to the camera

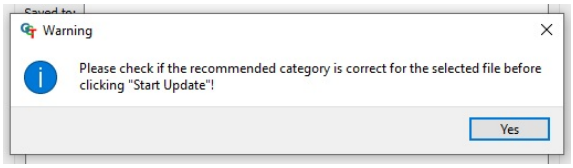
You can restore user settings, calibration files, and other file types which are previously downloaded from the camera.

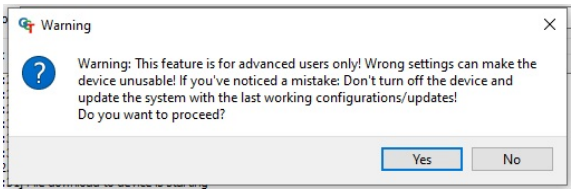
1. In the *menu bar* navigate to *Tools.*
2. Click **Up-/Download** or use the hotkey **Ctrl+D.**

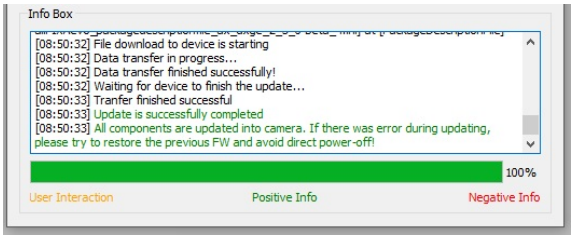


The *Update/Download* wizard opens.
3. Click **Select Update File** and select the file you want to upload and click **Open.**
4. GCT shows a warning message.
5. Check if the Type of your file corresponds to the selected category in *Update.*
6. Click **Start Update.**
7. GCT shows a warning message.
8. Click **Yes** to start the Upload.
9. Check the text in the Info Box: If the update was successful, it contains a green confirmation message “Update is successfully completed”.



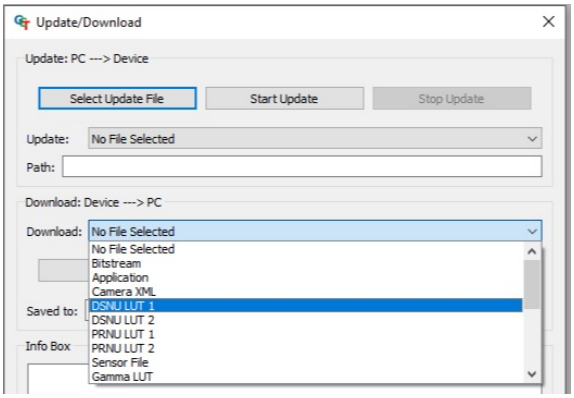




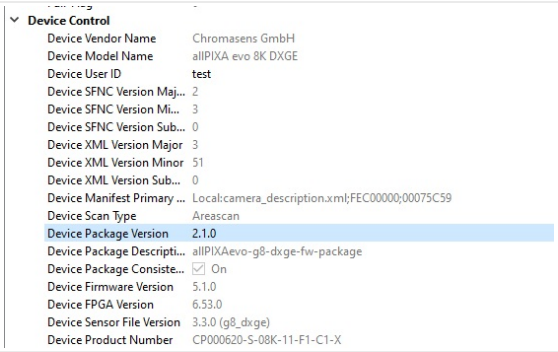
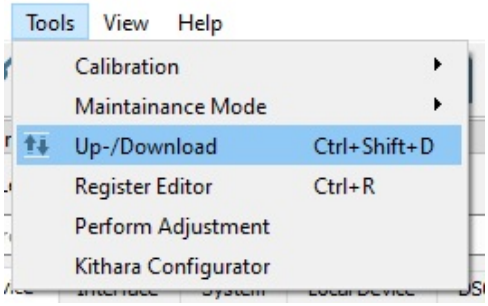


Downloading files from the camera to the PC

Firmware files, as well as user sets and other file types, can be downloaded from the camera to the PC and saved to a file.

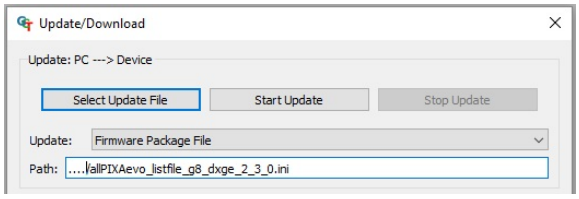
1. In the *menu bar* navigate to *Tools*. 2. Click **Up-/Download** or use the hotkey **Ctrl+D.**	
The *Update/Download* wizard opens. 1. Click **Download** and select your file in the drop-down menu. 2. Click **Download to**. 3. Select a folder, enter a file name, and then click **Save**.	

Updating the firmware

1. Download firmware from the Chromasens website or use the firmware file provided by Chromasens.	
2. Note the **Device Package Version** of the currently installed firmware displayed in the *Device Control feature* group.	
3. In the *menu bar* navigate to *Tools*. 4. Click **Up-/Download**or use the hotkey **Ctrl+Shift+D**.	
The *Update/Download* wizard opens. 5. Click **Select Update File** and select the *Firmware Package file* to	

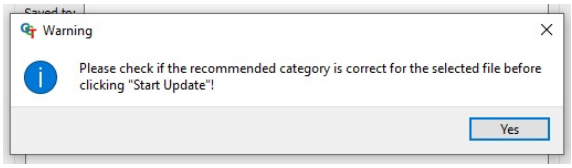
upload and click **Open.**

> **NOTE: Firmware Package file**
>
> For allPIXA evo select the allPIXAevo_listfile_.....ini file.
> For allPIXA neo select the allPIXAneo_listfile_.....ini file.



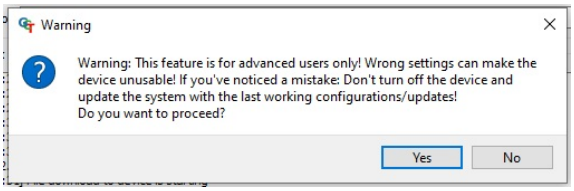
GCT shows a warning message.

6. Check if the *Update* field shows the *Firmware Package file* type.
7. Click **Start Update**.



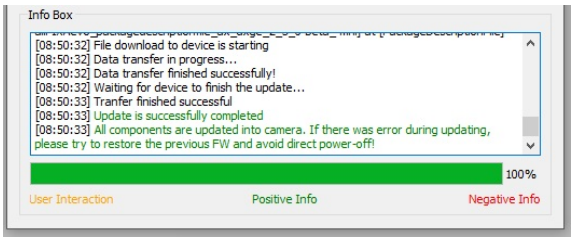
GCT shows a warning message.

8. Click **Yes** to start the Upload.



> **NOTE: Time for Update**
>
> Depending on the file size, firmware upload may take up to several minutes.

9. Check the text in the Info Box: If the update was successful, it contains a green confirmation message “Update is successfully completed”.



> **NOTE: Update Status**
>
> If the update was unsuccessful, do not switch off the camera, try to restore the previous state by uploading the correct file for the previously selected file type.

10. Power Cycle and Reconnect the camera **NOTE: Power Cycle** It is recommended to disconnect the power supply for 30 seconds.	
11. Check the **Device Package Version** in the *Device Control feature* group to make sure that the camera successfully booted with the new firmware.	

Video description

Click here to download a video

Create a black-reference (DSNU)

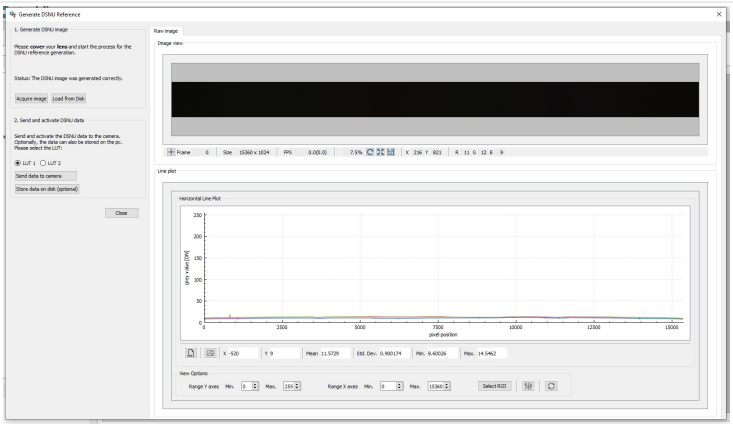
Create a black-reference with DSNU.

1. Switch off the illumination.
2. Cover the lens with a black or dark piece of cardboard or plastic.
   No light may reach the sensor.

3. In the *menu bar* navigate to *Tools* → *Calibration*.
4. Click **Generate DSNU Reference**.

The *Generate DSNU Reference* wizard opens.

5. Click **Acquire image** to generate the DSNU directly from the camera **or** click **Load from Disk** to load an image from the hard drive.

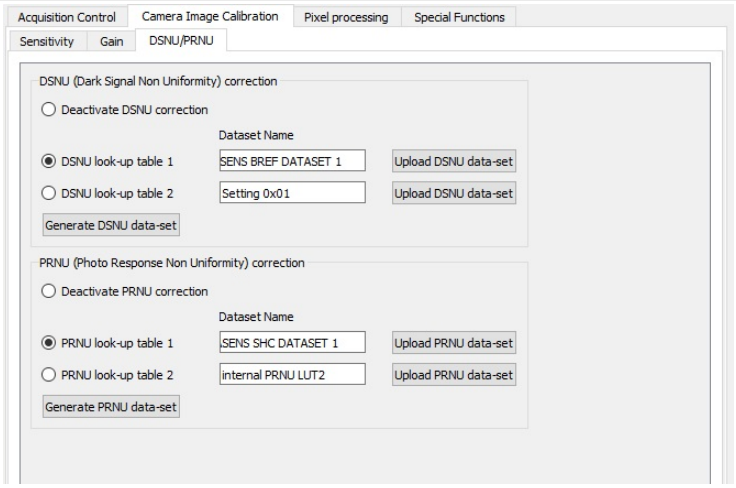


> **NOTE: Load from Disk**
>
> Make sure that the image has been taken with active image calibration mode by using the wizard.

The raw image and the line plot of the image is displayed.
Send the calculated DSNU to the camera:

6. Select **LUT 1** or **LUT 2**.
7. Click **Send data to camera**.

8. In the Configuration window navigate to *Camera Image Calibration → DSNU/PRNU*.
9. Make sure that DSNU is enabled.



Create a shading-reference (PRNU)

Calculation of PRNU

The following equation describes the calculation of the PRNU

$Calibrated_{Image} = (Raw_{Image} - DSNU) / PRNU$

$PRNU = (PRNU_{Image}-DSNU)/Target_{Value}$

$Calibrated_{Image}$ = Camera output with applied DSNU and PRNU

Raw_{Image} = Camera output image without any correction

$Target_{Value}$ = Target Value of PRNU, default value is 255

$PRNU_{Image}$ = Acquired image of the white-reference

PRNU = Photo response non-uniformity

DSNU = Dark signal non-uniformity

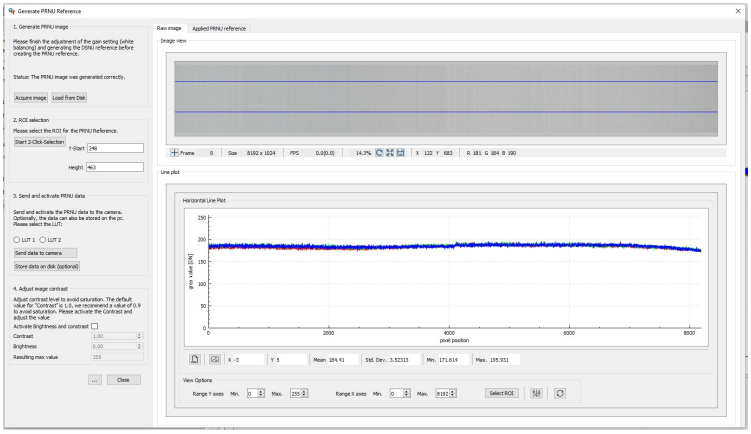
Standard PRNU reference generating

Create a shading-reference with PRNU.

1. Place a moving white target. If using a stationary target, place it slightly out of focus. Acquire an image: 2. In the toolbar click **Acquire a single frame or** click **Start grabbing**, wait until an image ist displayed, click **Stop grabbing**.	

3. In the *menu bar* navigate to *Tools → Calibration*.
4. Click **Generate PRNU Reference**.

The *Generate PRNU Reference* wizard opens.



5. Click **Acquire image** to generate the PRNU directly from the camera
or
click **Load from Disk** to load an image from the hard drive.

> **NOTE: Load from Disk**
>
> Make sure that the image has been taken with active image calibration mode by using the wizard.

The raw image and the line plot of the image is displayed.

6. Click **Start 2-Click-Selection**.
7. Click **on the image** to select the ROI.

Send the calculated PRNU to the camera:

8. Select **LUT 1** or **LUT 2**.
9. Click **Send data to camera**.

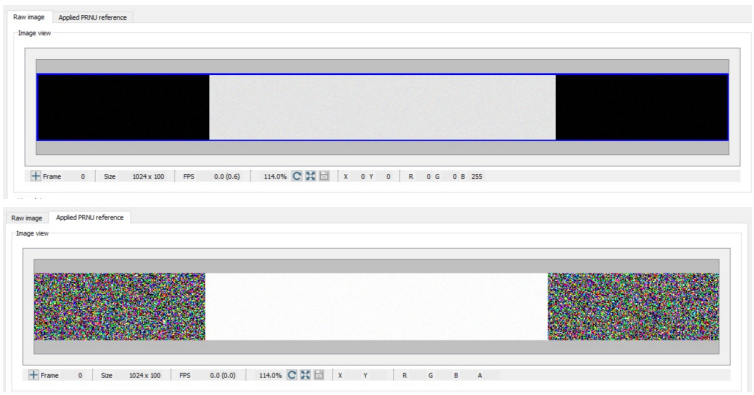
Activate brightness and contrast:

10. Select the **Activate Brightness and contrast** checkbox.
11. Set the contrast to 0.9.

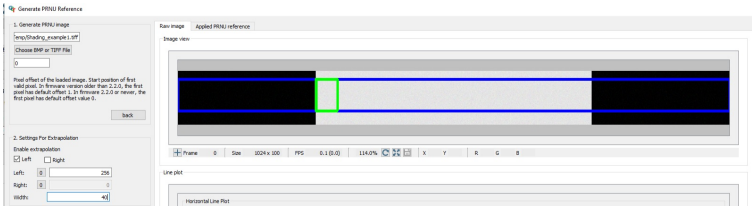
Extrapolation function

If the white reference does not cover the entire FOV, the extrapolation function can be used to generate it. In this case, a straight line is fitted to the gradient. Therefore follow the description below.

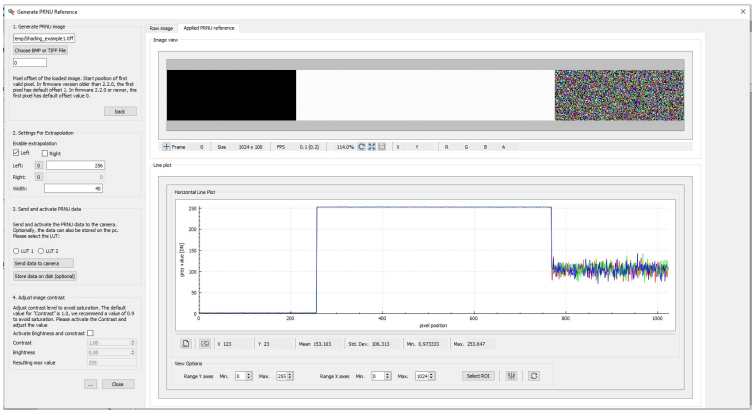
On the right side, you can see the *raw image* and the *Applied PRNU reference* without the extrapolation function. In the area with low image content, the *Applied PRNU reference* shows some artifacts.



1. Select the ROI, by using the **Start-2-Click-Selection**.
2. Press the button with the **three dots** on the bottom.
3. **Enable** your option, in this example the left extrapolation.
4. Select the **start position** of your extrapolation. The width defines the area where the extrapolation is created. From the start position to column 0, the extrapolation is applied.



5. Check the applied PRNU in the ***Applied PRNU*** *reference* tab.

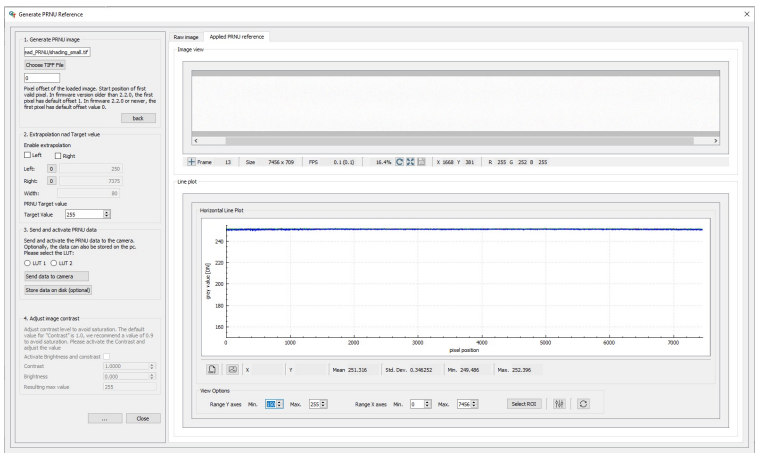
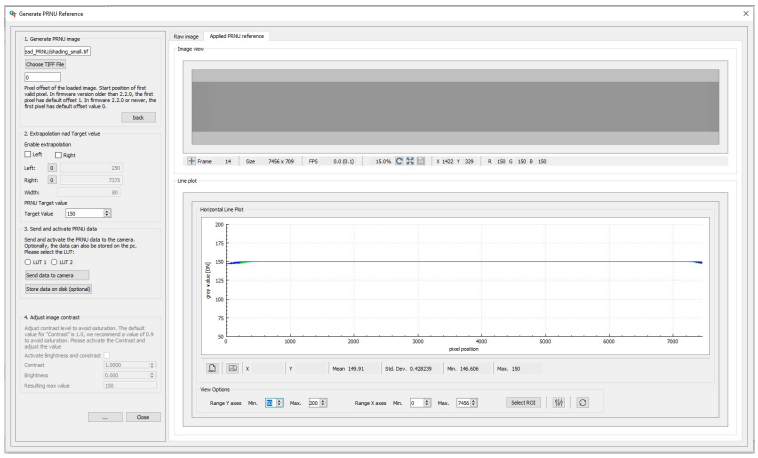


Target Value

The Target Value limits the maximum intensity of your Calibrated$_{Image}$.

Note

Make sure that the Values of your $PRNU_{Image}$ are smaller than your $Target_{Value}$

1. Press the button with the **three dots** on the bottom.	
2. Change the **Target Value**. 3. Check the applied PRNU in the ***Applied PRNU*** *reference* tab.	

Performance Test

The GCT installer provides a performance test tool. The tool allowed you to analyze the performance of your camera connection.

Performance Test tool folder

C:\Users\Public\Documents\Chromasens\GCT2\performanceTest

On your desktop, you can find a direct link to the performance test, therefore **double-click** the desktop link *"Testperformance GCT2".*
The folder provides three different *.bat* files. These files are predefined examples to run the performance test.

Name	Description
performanceTestCompatHelp.bat	Opens the Help dialog of the *performance test tool*
runPerformanceTestCSIGeneric.bat	Opens a generic configuration of the *performance test tool, which* can be used for CXP and Gige Cameras
runPerformanceTestCSIwithKithara.bat	Opens the *performance test tool* for *Kithara* configured cameras
runPerformanceTestCSIwiths2i.bat	Opens the *performance test tool for S2I configured cameras*

Parameter description

The program has the following adjustable parameters:

Parameter	Entries	Description
--checkImageConsistency	*GreyHorizontalRamp* or *GreyHorizontalRampMoving*	Method for checking error proune images
--checkImageConsistencyOption	*STSISDComplete* or *STSISD*	Option for checking error proune images

--deviceSelection	*auto* or *manual*	"auto" selects the first visible device, "manual" you can select the device
--height		Height in pixel
--help		Performance tool options
--line_time_step_width		Number of microseconds to increase from one test run to the next
--pixelFormat	*RGB8*, *Mono8*....	GeniCam SFNC Pixel Format
--reportFileDir		Option for setting directory of the written report file
--ring_buffer_count		Number of used frame buffers
--single_test_duration		Duration in seconds for a single integration time
--start_line_time		Line time to start with in us.
--stop_line_time		When this line time is reached, the test will be stopped.
--use_filter_driver	*1* and *0*	1 for on or 0 for off, default off
--width		Width in pixel

Example of a command line:

> PerformanceTest.exe --use_filter_driver 1 --single_test_duration 3 --start_line_time 60 --line_time_step_width 3 --stop_line_time 65 --pixelFormat RGB8 --checkImageConsistency GreyHorizontalRamp

This means the camera uses RGB8 as image format, starts with an acquisition line time of 60 microseconds (us), stops after 65 us, and the step is 3 us. With each acquisition line time, the test of each different line time will take 3 seconds. The filter driver for the S2I transportlayer is set to "on". An image consistency check is done with the *"GreyHorizontalRamp"*. The running program can be stopped by pressing any key.

Results and Output

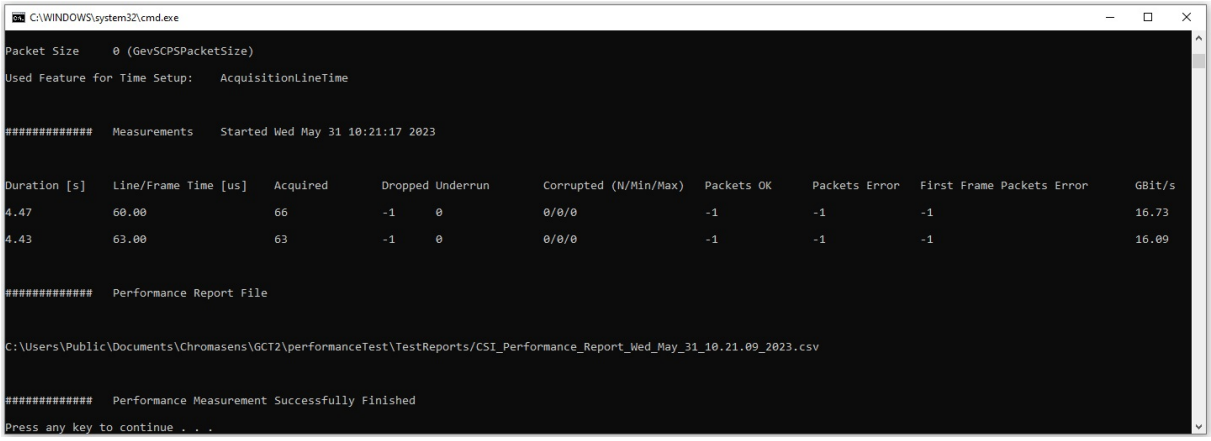
The output of the *Performance Test Tool* is shown in the command line window and in the Test Reports folder.

Test Reports Folder

C:\Users\Public\Documents\Chromasens\GCT2\performanceTest\TestReports

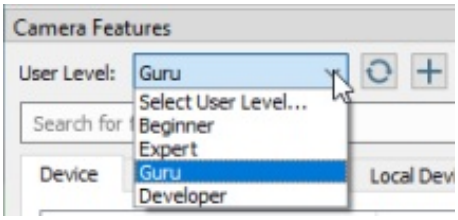
Parameter	Description
Duration [s]	Acquiring time in seconds
Line/Frame Time [us]	Set line time in microseconds
Acquired	Acquired frames
Dropped	Dropped frames
Corrupted (N/Min/Max)	Number of corrupted pixels N = corrupted images, Min = Minimum number of corrupted pixels in one image, Max = Maximum number of corrupted pixels in one image
Packets OK	Number of transmitted packets without an error
Packets Error	Number of transmitted packets with an error
First Frame Packets Error	Packet error in the first frame
GBit/s	Transmission rate

*(-1 means that this method was not used for this test)



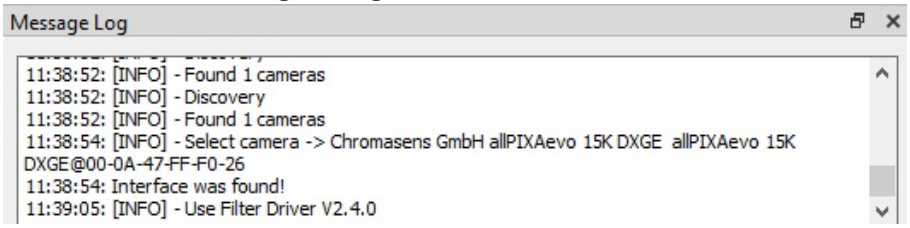
User level

Each camera contains an XML file specifying the available camera features. When GCT connects to the camera, it loads the XML file and shows the features as a tree structure.
Each feature has a visibility level. While all features are displayed for user-level D**evelopers**, many advanced features are hidden for user-level **Beginner**.
The User Level is shown above the feature area. To modify the user level, click the displayed User Level, and then click the desired level:



Show message log

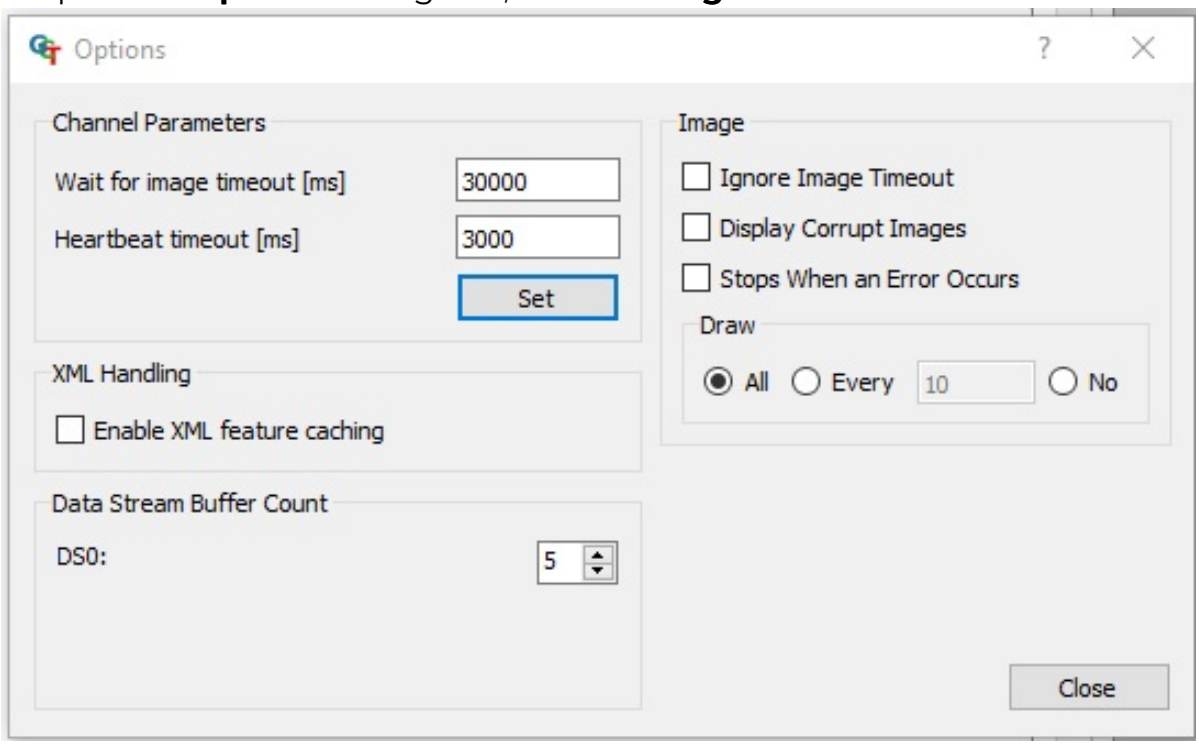
To show the message log, click **Message Log** on the **View** menu or click **Show message log window** in the bottom right corner of the GCT window. A **Message Log** area is then shown below the image area:



To copy messages to the clipboard, mark the messages, right-click on the selected messages, and then click **Copy** on the appearing context menu. To copy all present messages, right-click on the **Message Log** area, click **Select All** on the appearing context menu, and then press **Ctrl+C**.

GCT Options

To open the **Options** dialog box, click **Settings** on the **Start** menu.



Option name	Default value	Description
Wait for image timeout	30000	If the next image is not received after the specified time, then it will send a timeout. The value can be modified. When GCT is restarted, it is set back to 30000.
Heartbeat timeout	3000	Read the value of **GevHeartbeatTimeout** from the camera (control channel).
Data Stream Buffer Count	5	Specifies the number of buffers used for each stream. The value should be at least 2 or 3.
Ignore Image Timeout	Not selected	Can be selected if the acquisition process should continue despite timeout error.
Display Corrupt Image	Not selected	Can be selected if an image should be displayed despite missing data
Stops If an Error Occurs	Not selected	
Draw	All	Can be set to display every x frames, or to shut off display to reduce CPU usage.
Enable XML feature caching	Not selected	So that the actual value is retrieved directly from the camera, instead of from a cached register.

The options (except **Wait for image timeout** and **Heartbeat timeout**) are saved in GCC.ini

GCC.ini file

The GCC.ini file, which is located under *C:\Users\Public\Documents\Chromasens\GCT2,* including the content of producer path, producer filename, and selected producer for the discovery are already adjusted automatically in the file GCC.ini. file. Therefore, if the discovery has already been executed, the file GCC.ini could look as follows in the block of [DefaultProducers].

```
[DefaultProducers]
ProducerList=C:\\Program Files\\Chromasens\\GCT2\\GenTL\\Kithara\\ls_tl_gev_kithara.cti, C:\\Program Files\\Chromasens\\GCT2\\GenTL\\s2i\\GEVTLS2I.cti
AdditionalSearchPath=C:\\Program Files\\Chromasens\\GCT2\\GenTL\\s2i
SelectedProducerList=C:\\Program Files\\Chromasens\\GCT2\\GenTL\\Kithara\\ls_tl_gev_kithara.cti
```

- ProducerList contains the found cti transport layer files.
- AdditionalSearchPath contains the additional search paths for cti files, which can be added by clicking **Add Producer Path**.
- SelectedProducerList contains the cti transport layer files which are selected to be applied for the discovery process.

10 GigE transport layer

GCT offers two different configurations of the GigE interface Transport Layer to connect and configure a TKH vision group camera. The 10 GigE real-time solution with the Kithara Transport Layer (TKH *GenTL, "C:\Program Files\Chromasens\GCT2\GenTL\Kithara\tkh_tl_gev_kithara.cti"*) and the 10 GigE with the s2i Transport Layer (S2I GEV TL Interface, *"C:\Program Files\Chromasens\GCT2\GenTL\s2i\GEVTLS2I.cti"*).

allPIXA evo DXGE

For a allPIXA evo and a dual link configuration we recommend using the Kithara real-time solution. In some cases, it is also possible to use the s2i Transport Layer, but only for single link connections.

NOTE

Packets lost can occur without the Kithara real-time kernel solution even with large receive buffers in combination with different components

allPIXA neo

With the appropriates hardware, the allPIXA neo can be operated with the s2i Transport Layer.

10 GigE with Kithara transport layer

Kithara real-time can provide fast image capturing with the GigE Vision standard and can achieve high performance. Running Kithara real-time requires dedicated CPU cores. On those reserved cores, the real-time system is booted, which, from here on, functions just like a separate RTOS while Windows retains its full operability on the remaining CPU cores. From this point on, Windows and the real-time system run simultaneously and parallel to each other on the same computer, without one restricting the other.
The Kithara transport layer enables communication with GigE cameras based on the Kithara software.

Note

Make sure that your PC offers enough 8-lane PCIe 3.1 slots to cope with the data transmission of two cameras! For more information see the manual of our PC (Mainboard)

General Kithara Informations

Requirements

Description	Value
System logic cores	up to 48 cores
Computer memory	up to 125 GB RAM
Maximum usage of dedicated CPU's	8 (one CPU per link)
Maximum usage of network ports	8 • A Single Link consumes 1 port • A Dual Link consumes 3 ports (2 physical, 1 virtual port) • This results in a maximum of 8 Single Link or 2 Dual Link cameras and 2 Single Link cameras
Number of Dongles per system	One per system, regardless of the number of connected cameras
IP-Adress Configuration	• DHCP • Static IP-Address in the range of 169.254.X.X (169.254.1.202 using a subnet mask of 255.255.0.0)

Licensing

A valid software license including a USB dongle is required to use the Kithara transport layer. Don't hesitate to get in touch with Chromasens about the information on licensing options. The license dongle is part of the delivery for allPIXA evo DXGE cameras.

Camera connection with Kithara Transport Layer

After the configuration of Kithara, the camera connection is working with the Kithara Transport Layer. To connect the camera, proceed as follows:

1. Make sure that the license USB dongle is plugged in before using the Kithara Transport Layer.
2. Make sure that you configure the Kithara Transport Layer properly,

see 10 GigE with Kithara transport layer.

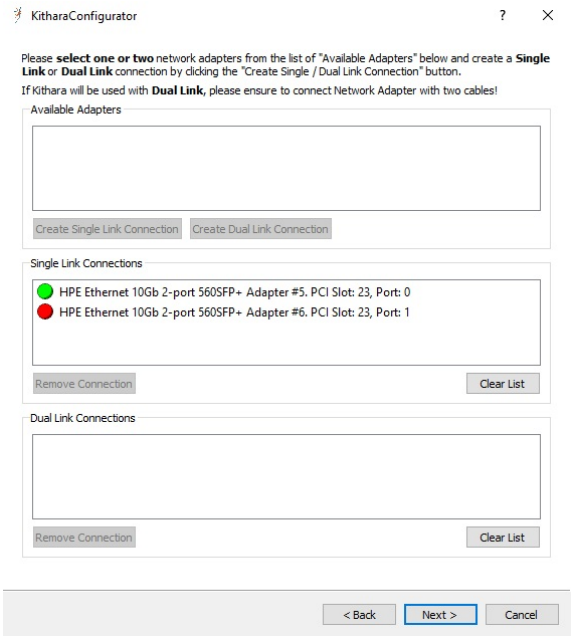
3. When you start GCT, the first step is to detect the camera. The Kithara TL can be used like any other producer.

Note

If the Kithara transport layer file ls_tl_gev_kithara.cti has been selected in the Select TL Producer dialog box, and the file kithara_config.txt in the folder C:\Users\Public\Documents\Chromasens\GCT2 has been configured, a blue Kithara window may appear for some seconds during the first camera discovery with Kithara TL.

Setting up the Kithara Transport Layer

You can Configure the Kithara Transport Layer with the Kithara Configurator on your Desktop.

1. Plug in the license USB dongle before using the Kithara Transport Layer.	
2. Open the **Generate Config for Kithara** from your desktop.	
3. Read the instructions and *confirm* by *clicking* **Next.**	
4. Select the adapter. • If Kithara is going to be used with a single cable connection, select the corresponding adapter port in the adapter list and *click***Create Single Link Connection**. • If Kithara is going to be used with two cables (which connect two dual ports of the same network adapter), select the two adapter ports in the adapter list and *click* **Create Dual Link Connection.**	
5. *Click* **Next** to confirm the configuration.	

The configurator adds a file kithara_config.txt to the folder C:\Users\Public\Documents\Chromasens\GCT2, which contains the hardware IDs of the network adapters on which Kithara will be executed. Without this file, the Kithara transport layer is not able to detect the devices in GCT. The Kithara Configurator also adds the dedicated CPU cores for Kithara and disables Hyper-Threading. The Kithara Configurator will prompt the system to reboot to update the changes.

Warning

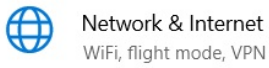
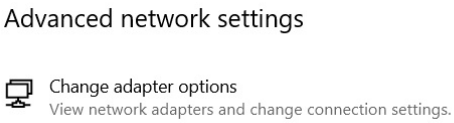
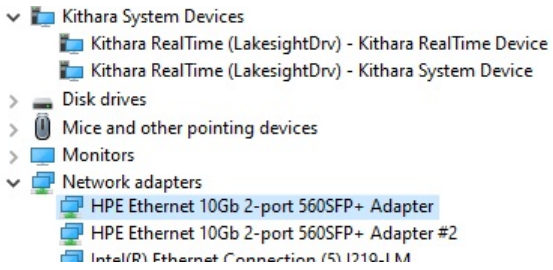
The Kithara configuration can currently be executed once before the first use of Kithara. After Kithara has been used at least once, the configured adapter port(s) are dedicated to Kithara and cannot be used from within Windows unless the driver is set back manually. In the Windows Device Manager, the network adapters can be found in the Kithara System Device section. If the cable connection for Kithara is changed, it is required to reset the network adapter driver to the original Windows network adapter driver (so that the device can be found under Windows) and rerun the Kithara Configurator.

Resetting network adapter driver back to Windows driver

After discovering the camera with the Kithara transport layer file, the corresponding network adapter connections are assigned to the Kithara driver and can no longer be found in the list of normal network adapters in the Windows Device Manager.
As shown in the following screenshot of the Windows Device Manager, the two entries of the dual port HP network adapter (HPE Ethernet 10Gb 2-port 560SFP) are not available in the Network adapters list. Instead, they are present in the Kithara System Devices list with the name Kithara RealTime -PCI Intel 10 GbE Network Controller.

To reset a Kithara RealTime PCI network controller:

1. Open the **Generate Config for Kithara** from your desktop.	
2. Read the instructions and *confirm* by *clicking* **Next.**	

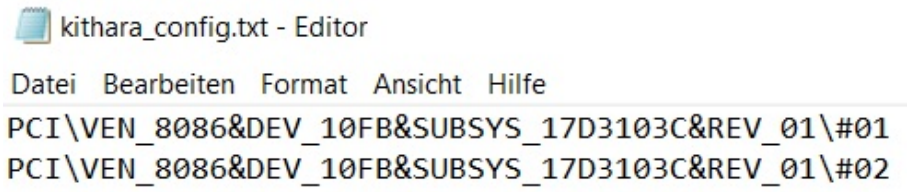
3. Remove the single or dual link connection from the list, therefore, select the connection and press the **Remove** connection button.	
4. *Click* **Next** to confirm the configuration.	
5. Navigate to *Windows Settings* and click **Network & Internet.**	
6. Click **Change adapter options.**	
7. **Right-click** one of the Kithara RealTime PCI network controllers, and then click **Properties**.	
8. Click **Configure** and on the next dialog box click the **Drivers** tab.	
7. Click **Update Driver**.	
8. Click **Search driver** on my PC and then click **Select available drivers from a list of my PC**.	
9. Select the suitable adapter driver and then click **Continue**.	
10. After the resetting of Kithara you can find the Network adapter in the Device manager under Network adapters.	
11. After the reset of the Network adapter it is required to set up the parameter of the network adapter, therefore please refer to Configure the network adapter.	

Kithara_config.txt file

The kithara_config.txt contains the hardware ID of the network adapters and the configuration whether it is a single or dual connection. The file is created by executing the Kithara Configurator. The file is stored under:

> C:\Users\Public\Documents\Chromasens\GCT2

- If only a single link is used with Kithara, this file contains the hardware ID of the chosen connection as a single entry.
- If the dual link is used with Kithara, this file contains two items from the dual connections.

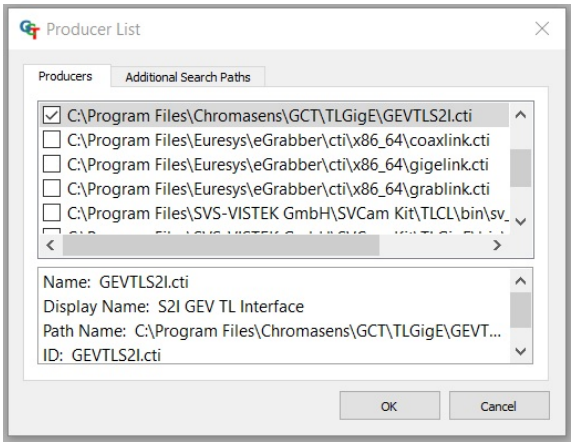


10 GigE with s2i transport layer

The s2i transport layer does not require an additional license dongle.

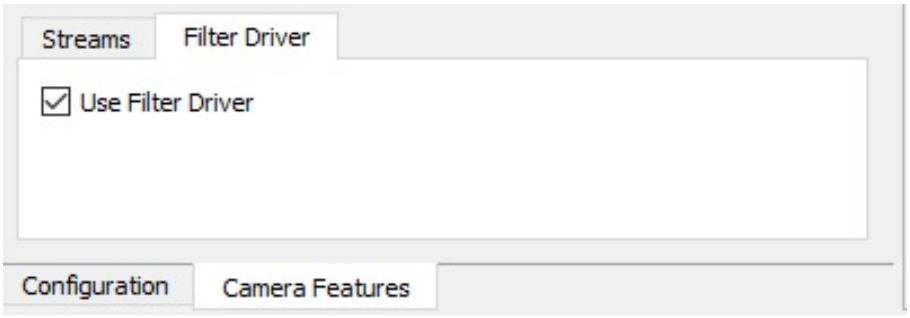
> **Note**
>
> Make sure that the **parameters** of your **network adapter** are set correctly. For more information, see **Configure the network adapter**.

Connect the camera with s2i transport layer

1. *Stat* **Device Discovery**	
2. *Press* **GenTL Producers** and *check* the **GEVTLS2I.cti** file.	
3. *Press* **OK.**	
4. *Press* **Start Discovery.**	

Filter driver

The s2i transport layer offers the usage of a filter driver. A filter driver is used to get the UDP-image-data packets before the operating system accesses them. This leads to a drastically improved error rate and decreases the number of lost packets drastically. To configure the filter driver, go to the Configuration widget and select the Camera Features tab, here you can check and uncheck the filter driver.


Streams
Filter Driver
Use Filter Driver
Configuration
Camera Features

Configure the network adapter

Set network adapter parameter

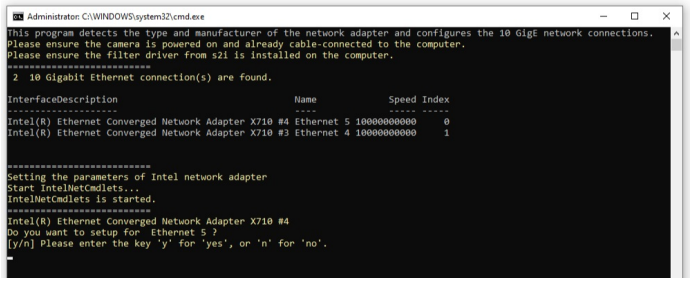
Automatic setting of the parameters

> **Note**
>
> This configuration of your **GigE network adapters** is done during the installation of GCT. You will also need it if you want to remove the *Kithara TL*.
> You can run this script if you use a network adapter with an Intel chipset.

In the Public folder of GCT installation, a configuration script is also stored. For the configuration please:

1. Turn on your camera and connect it to the PC.	
2. Open the Windows command line in the *Administrator* mode, therefore press **Start** and type **CMD** into the search box. Now you can open the **CMD** in the *Administrator* mode.	
3. Copy the following command into the **CMD** and press **enter.**	`cd C:\Users\Public\Documents\Chromasens\GCT2`
4. Copy the following command into the **CMD** and press **enter.**	`runConfig10GigE.bat`

5. The command line window lists all available ethernet adapters and you can configure the network adapter by entering a **Y** into the script.

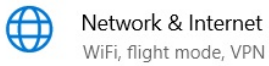
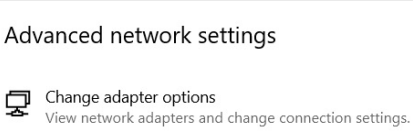


> **Note**
>
> If an error occurs during the execution of the script you have to configure the network adapter manually.

6. After the configuration of the network parameter, the script can help you to set a static IP Address of the network adapter. *We recommend disagreeing* by entering **N** into the window. If you want to set a static IP Address for your network, you can answer with **Y**.

Manual setting of the parameters

In contrast to the section above, this section is now oriented to the network adapters, which are not supported by Intel, or the operating system is Win7 x64. The setup depends on the adapter manufacturer.

1. Navigate to *Windows Settings* and click **Network & Internet.**	
2. Click **Change adapter options.**	
3. Right-click the Ethernet adapter that will be used to connect the camera, and then click **Properties.**	
4. In the network tab only enable **Sphinx GigE Filter Driver** and **Internet Protocol version 4 (TCP/IPv4)**.	
4. *Click* **Configure**, and then *click* the **Advanced tab**. The available parameters may vary and are depending on the adapter.	
5. Set the following parameter.	

For HP or X710 network adapter

Attribute name (English)	Attribute name (German)	Value
Interrupt Moderation	Interrupt-Drosselung	Enabled
Jumbo packet	Jumbo Packet	9014 Byte
Large-Send-Offload V2(IPv4)	Large-Send-Offload V2(IPv4)	Enabled
Large-Send-Offload V2(IPv6)	Large-Send-Offload V2(IPv6)	Enabled
Direct Cache Access	Direct Cache Access	Enabled
Receive Buffers	Empfangsbuffer	Max (e.g. 4096)
Flow control	Flusssteuerung	Disabled
Interrupt Moderation Rate	Interrupt-Drosselungsrate	Adaptive
Low Latency Interrupts	Low Latency Interrupts(LLIs)	Disabled
Transmit Buffers	Übertragungspuffer	Max (e.g.16384)
Rx and Tx from Offloading Options	Rx und Tx von Offload-Optionen	Enabled
Receive Side Scaling	RSS (Empfangsseitige Skalierung)	Enabled
RSS queues	RSS-Warteschlangen	2
Log Link State Event	Verbindungsereignis protokollieren	disabled

For Broadcom network adapter

Attribute name	Value
Energy Efficient Ethernet	disabled
Flow Control	Rx and Tx enabled
Forward Error Correction	disabled
Interrupt Moderation	enabled

Interrupt Moderation Configuration	Medium
Jumbo Packet	9174 Byte
Large-Send-Offload V2 (IPv4)	enabled
Large-Send-Offload V2 (IPv6)	enabled
Locally Administered Address	Not present
Maximum Number of MSI-X Messages	Not present
Maximum Number of RSS Processors	Not present
Maximum Number of RSS Queues	16
Maximum RSS Processor Number	Not present
NetworkDirect Functionality	enabled
NetworkDirect Technology	RoCEv2
Preffered Numa Node	Not present
Priority & VLAN	Priority & VLAN disabled
Quality of Service	Enabled
Receive Buffers (0= Auto)	0
Receive Side Scaling	enabled
Recv Segment Coalescing IPv4	disabled
Recv Segment Coalescing IPv6	disabled
RoCE MTU	1024
RSS Base Processor Group	Not present
RSS Base Processor Number	Not present
RSS Load Balancing Profile	Not present
RSS Max Processor Group	Not present
Speed & Duplex	10 Gbit/s Full Duplex
TCP/UDP Checksum Offload IPv4	Rx and Tx enabled

TCP/UDP Checksum Offload IPv6	Rx and Tx enabled
Transmit Buffers (0= Auto)	0
Virtual Machine Queues	enabled
VLAN ID	0

Set IP address

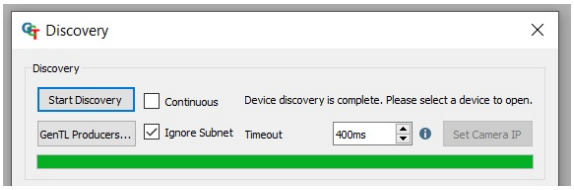
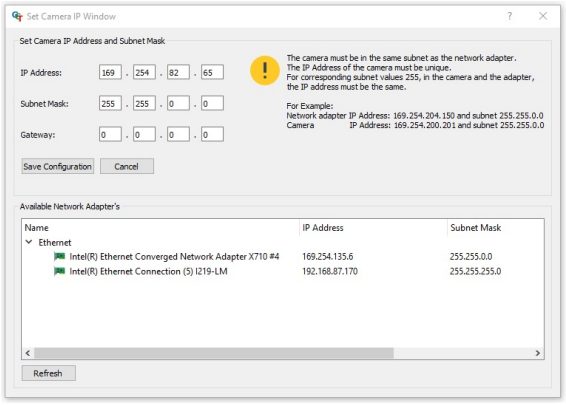
To connect GCT to the camera, you may need to adjust the properties of the network adapter. There are basically three methods to configure the IP address and subnet mask of your PC and camera:

- Use LLA (link-local address) autoconfiguration, this is by default always enabled.
- Use a DHCP server to configure the network settings of the PC and camera, in the default factory setting it is enabled.
- Use manually assigned persistent IP addresses and subnet masks for the camera and network adapter, in the default factory setting it is disabled.

In typical installations, the camera is connected directly to the network adapter of the PC with the DHCP server configuration.

Set the camera IP address with GCT

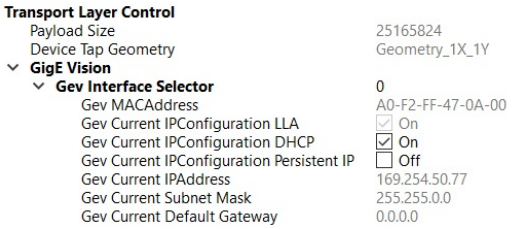
If the camera does not show up in the GCT discovery window you can set the IP-Address in GCT.

1. Start **Device Discovery** in *GCT.*	
2. *Enable* the **Ignore Subnet** check box. If the camera now shows up but cannot be connected (Open button is disabled), you can set the camera IP address manually.	
3. *Click* the **Set Camera IP** button	
4. *Select* the network adapter the camera is connected. The program will assign an IP address to the camera	
5. *Click*Save Configuration	

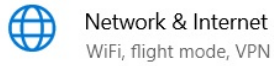
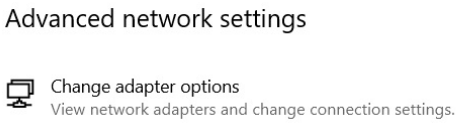
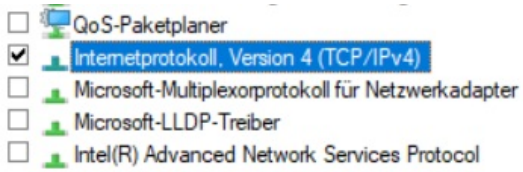
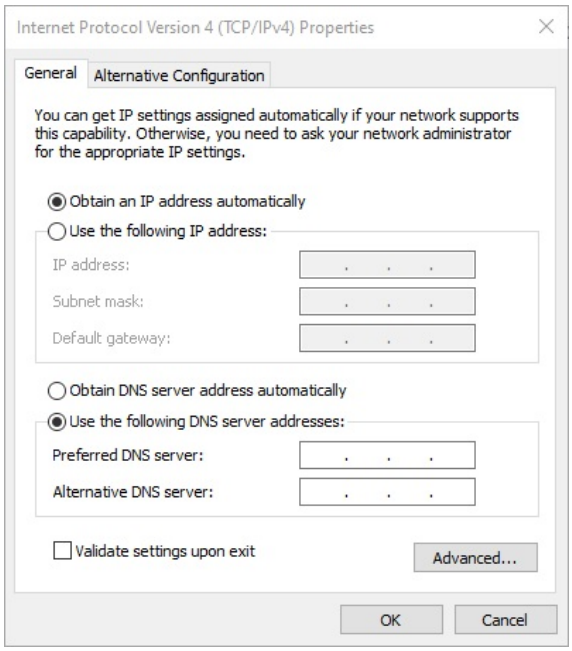
DHCP server configuration

The DHCP server configuration tries to assign different IP addresses from the Link-Local-Block (169.254.x.x) to both the camera and network adapter automatically and set the subnet mask of both devices to 255.255.0.0.

Configure the DHCP Mode in the Camera:

1. Switch On and connect the camera with GCT.	
2. In the *Camera Feature* tab navigate to --> *Transport Layer Control --> GigE Vision --> Gev Interface Selector.* *3. Switch the* **Gev Current IPConfiguration DCHP** *--> On.* *4. Switch the* **Gev Current IPConfiguration Persistent IP** *--> Off.*	

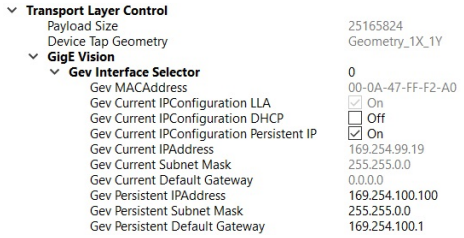
Configure the DHCP Mode in the Network adapter:

1. Navigate to *Windows Settings* and click **Network & Internet.**	
2. Click **Change adapter options.**	
3. Right-click the Ethernet adapter that will be used to connect the camera, and then click **Properties.**	
4. On the Networking tab select the *Internet Protocol Version 4* item and then click **Properties.**	
5. Make sure that the **Obtain anIP address automatically** check box is selected.	

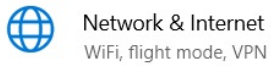
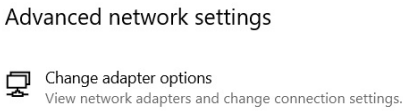
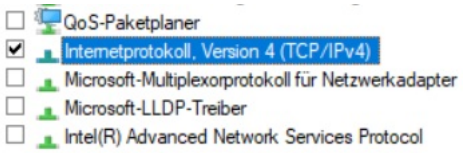
Persistent IP adress configuration

The persistent IP address implies that the feature Current IP Configuration Persistent IP is switched on in the camera.

Configure the Persistent IP Mode in the Camera:

1. Switch On and connect the camera with GCT.	
2. In the *Camera Feature* tab navigate to --> *Transport Layer Control --> GigE Vision --> Gev Interface Selector.* *3. Switch the* **Gev Current IPConfiguration DCHP** --> Off*.* *4. Switch the* **Gev Current IPConfiguration Persistent IP** --> *On.*	

Configure the Persistent IP MOde in the Network adapter:

<table>
<tr><td>1. Navigate to Windows Settings and click Network & Internet.</td><td> 
</td></tr>
<tr><td>2. Click Change adapter options.</td><td> 
</td></tr>
<tr><td>3. Right-click the Ethernet adapter that will be used to connect the camera, and then click Properties.</td><td></td></tr>
<tr><td>4. On the Networking tab select the Internet Protocol Version 4 item and then click Properties.</td><td> 
</td></tr>
<tr><td>5. Click Use the following IP address.
6. In the IP address field of the Network adapter specify an address that differs from the IP address of the camera only in the last field. When the camera IP address shown in the GCT discovery window is 169.254.100.100, you can set the IP address of the network adapter e.g. to 169.254.100.99.
7. At the Subnet mask enter 255.255.0.0.
8. Close the properties dialog boxes by clicking OK.
Note
For Kithara use a static IP address in the range of 169.254.X.X (169.254.1.202 using a subnet mask of 255.255.0.0)</td><td></td></tr>
</table>

Network adapters and transceivers

SFP+ connection

Note

If you are using the s2i Transport Layer it is required to configure the network adapter properly. The **HPE Ethernet 10Gb 2-port 560SFP+ Adapter** and **Intel (X710)** are configured during installation.
* Some manufacturers encode their cards. Those cards can only be used with the appropriate transceivers.
Example: Intel X710 chipsets only work with Intel or Intel-configured transceivers

Network adapter

The following network adapters are tested with GCT2. The results are shown in the following table.

Vendor Name	Type	Result	S2i TL	Kithara TL
HPE	HPE Ethernet 10Gb 2-port 560SFP+ Adapter	Fully functional	X	X
HPE *	HPE Ethernet 10Gb 2-port 562SFP+ Adapter	Not supported yet		
Intel *	Intel® Ethernet-Converged-Network-Adapter X710	Fully functional	X	X
Myricom	Myricom 2x 10GbE SFP+ low-profile NIC	Only with s2i	X	
Myricom	Myricom 1x 10GbE SFP+ low-profile NIC	Only with s2i	X	
Mellanox	NVIDIA Mellanox MCX512A-ACAT ConnectX®-5 EN Network adapter	Only with s2i	X	

Installation

Network adapter	
HPE Ethernet 10Gb 2-port 560SFP+ Adapter	**Download and install the following files** Download the *Wired_dirver_xx* and the *Wired_ PROSet_xx from the* ***Intel Homepage***.
Network adapter with X550 Intel chipset	**Download and install the following files** Download the *Wired_dirver_xx* and the *Wired_ PROSet_xx from the* ***Intel Homepage***.
Network adapter with X710 Intel chipset	**Download and install the following files** Download the Release_xx *from the* ***Intel Homepage***.

Transceiver

The following transceivers are tested with the allPIXA evo camera and the network adapters.

Vendor Name	Type	Comment	Camera side	Network adapter side
Finisar	FTLX8574D3BCL Class 1 21CFR1040:10		X	X
FS	SFP-10GSR-85		X	X
FS	Intel E10GSFPSRX Compatible 10GBASE-SR SFP+ 850nm 300m DOM Duplex LC MMF Transceiver Module	Recommended for Network adapters that are marked with *		X

RJ45 connection

Network adapter

The following network adapters are tested with GCT2. The results are shown in the following table.

Vendor name	Type	Result	s2i TL	Kithara TL
Broadcom	P210TP - 2 x 10GBASE-T PCIe NIC	Fully functional	X	

Installation

1. Download the driver file and follow the installation instructions.	**Download** For the Broadcom Network adapter "P210TP - 2 x 10GBASE-T PCIe NIC", you can find the download files **here**.

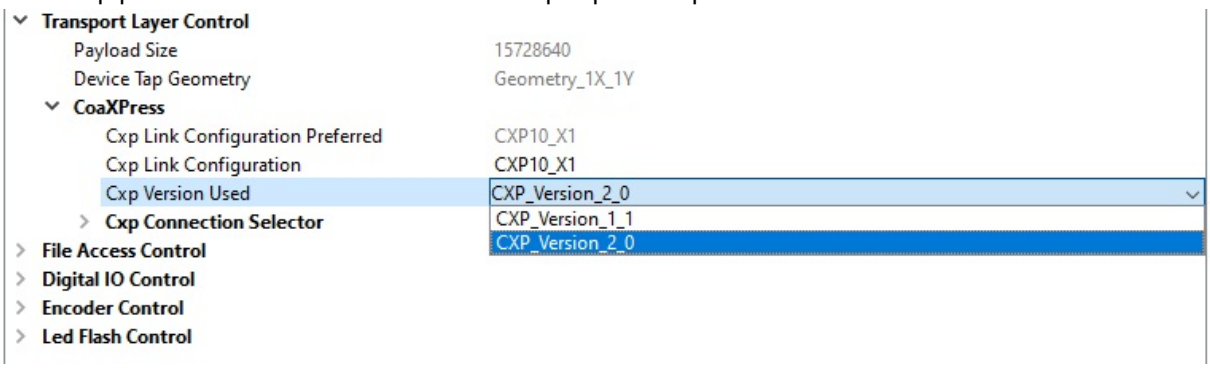
Connection two cameras with one frame grabber

If you want to use two cameras with one frame grabber, you may have to configure the frame grabber (see frame grabber manual). Once it is configured correctly, you can connect the cameras as described in chapter Connection and disconnection of Camera

CXP link configuration

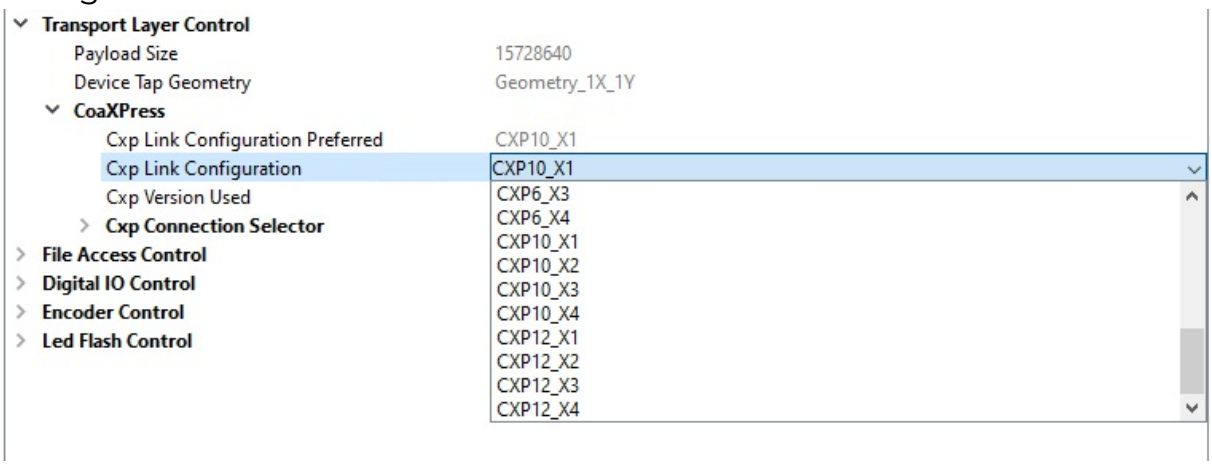
Configure CXP version

The data transfer rate depends on the configured CXP standard version. This option can be set with the parameter "Cxp version used" as shown in the following figure. We recommend using the version CSP_Version_2_0. The CSP standard 1.0 supports a data rate of 6.26 Gbps, the CXP standard 2.0 supports a data rate of 12.5 Gbps per cxp cable.



Link configuration

The link configuration defines how many CoaXPress cables are connected to the camera and the data transfer rate. The number of connected CoaXPress cables are assigned with the Xn. For example, a X2 means that two cables are connected to the camera. The data transfer rate per connected cable is defined with the first number in the selectable cxp link configuration CXPt_Xn.



The following table shows the link configuration parameters and there corresponding data rate (for CXP Version 2.0). The data rate refers to one connected CXP cable X1. The data rate increases with additional connected CXP cable. CXP12_X2, for example, has a data rate of 25 Gbps.

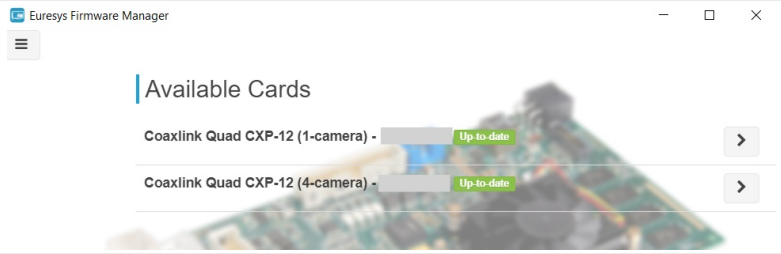
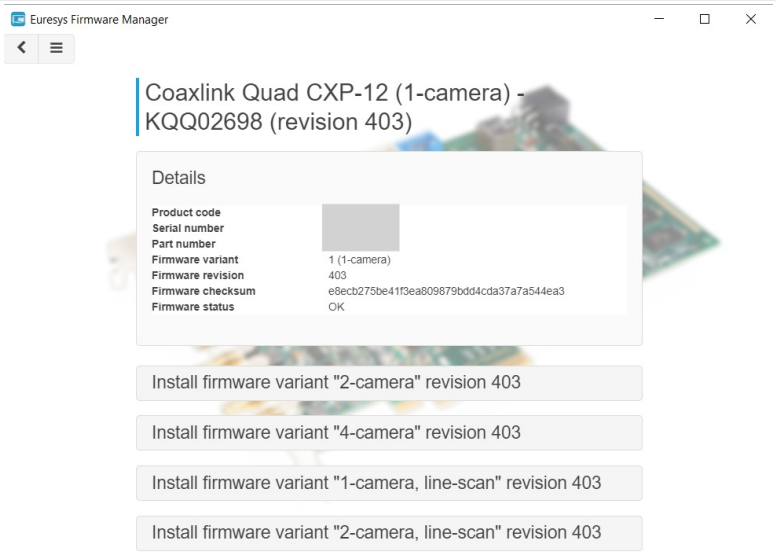
Link configuration	Data rate per link Xn in Gbps
CXP1_Xn	1.25
CXP2_Xn	2.50
CXP3_Xn	3.125
CXP5_Xn	5.00
CXP6_Xn	6.25
CXP10_Xn	10.00
CXP12_Xn	12.5

Configure the frame grabber

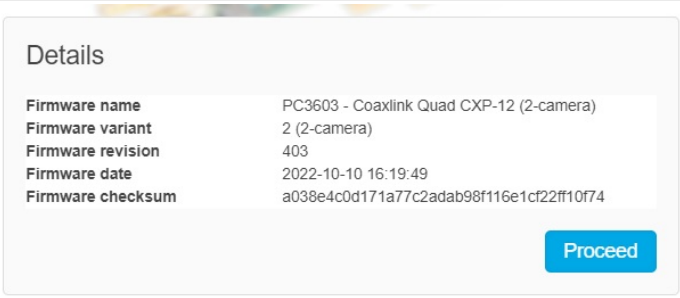
Configure an Euresys Coaxlink

> **Use the correct firmware**
>
> In order to operate a Chromasens CXP camera with an euresys frame grabber, the correct firmware must be loaded on the frame grabber.

To check the firmware version and change it if necessary, please follow the steps below:

<table>
<tr><td>1. Open the Euresys Firmware Manager</td><td> 
</td></tr>
<tr><td>2. Select the frame grabber</td><td></td></tr>
<tr><td>3. Chech the firmware version</td><td> 

it is required to use the following variants:
• Variante “1-camera” revision X
• Variante “2-camera” revision X
• Variante “4-camera” revision X

Please do not use the line-scan firmware</td></tr>
<tr><td>4. Change the firmware version</td><td> 

To change the firmware:
• Select the correct firmware version
• Press “Proceed” and wait until the download is finished
• Power-cycle the PC</td></tr>
</table>

Introduction

Conventions used in this manual

Styles

Notification

To ease the use of the document and to clearly indicate the type of the used data different colors for the different elements are used. Three different colors are used when displaying elements in tables:

Enumerations:

For example:

csiEventType	Defines events which can be received from the SDK
Definition	typedef enum csiEventType { CSI_EVT_NEWIMAGEDATA = 0x00, CSI_EVT_ERROR = 0x01, CSI_EVT_MODULE = 0x02, CSI_EVT_CUSTOM = 0x1000 } csiEventType;
Elements	CSI_EVT_NEWIMAGEDATA: New image data received CSI_EVT_ERROR: Error occurred in the SDK CSI_EVT_MODULE: General event notification CSI_EVT_CUSTOM: A custom event was triggered

Structures:

For example:

Struct-name	**csiDiscoveryInfo**	-
Variable type	**Element name**	**Description**
uint32_t	numDevices	-
double	progress	-
bool	discoveryRunning	-

Functions:

For example:

Function-name	Description
csiDiscoverDevices	Searches for the devices currently connected to the system
Syntax	csiErr csiDiscoverDevices(csiDiscoveryInfoOut, uint64_t timeoutMilliseconds, csiDiscoveryInfoCallbackFunc discCallbackFunc = NULL, const char* additionalSearchPaths = NULL, bool overrideSearchPath = CSI_DEFAULT_PARAM_FALSE);
Parameters	timeoutMilliSeconds: The amount of time to search in a specific transport layer for a device discoverCallbackFunc: Pointer to a callback function which gets called when a result was received AdditionalSearchPaths: As default only the paths given in the system variable "GENICAM_GENTL64_PATH" are being searched for the used transport layers overrideSearchPath: If set, only the given path is searched for transport layers to use discoveryInfoOut: The structure will be filled with the available devices
Return value	Returns csiSuccess or an error defined in the csiErr-Enum.
Comments	-

Getting started

This chapter will describe the basic functions/sequences needed to handle the basic functionality of the camera.
Ready to use-Examples are also shipped with the SDK in order to demonstrate the usage of the SDK regarding getting/setting features and acquiring images.

Initialization of the SDK

Before accessing any other functions of the SDK, an initialization needs to be done. Please refer to *Init/Deinit* -functions for the detailed description of the function **csiInit**.
After finishing the work with the SDK make sure to call the **csiClose** function. This makes sure that all memory is freed again, and all connections/interfaces are properly closed again.

Connecting to a camera

The use of the Chromasens Gen<I>CAM-SDK enables the user to use different transport layers and interfaces for the available devices.
Depending on the requirements for your application these transport layers can be selected during the device discovery process.
It is possible to use the standard search paths for the already installed transport layers.
These paths are set in the environmental variable
"GENICAM_GENTL64_PATH" or for 32Bit-applications:
"GENICAM_GENTL32_PATH" .
This is the default behavior. To reduce the time needed for the discovery process a specific path can be given. The search can also be limited to this single path when the *overrideSearchPath* is set.
To establish a connection, you will need to call 2 functions:
csiDiscoverDevices and **csiOpenDevice**.

Getting and setting features

To configure the camera, so called features can be set and read by using the feature names provided by the device-xml-file. All features are of a specific type. The following different types exist:
Boolean

Integer
Floating point
String
Command
Register
Enumeration
For each type, a “Get”- and “Set”-function does exist in the API. For example, use **csiGetFeatureFloat** to get a float parameter. To retrieve additional information, the function **csiGetFeatureParameter** exists. This function will fill a **csiFeatureParameter** structure which provides information about the display name, minimum and maximum values, etc. This function is especially useful if you do not know the valid thresholds of a parameter.
Please be careful when treating string features. You must not exceed the maximum length! This can also be retrieved with the function **csiGetFeatureParameter**. The parameter *maximumStringLength* of the **csiGetFeatureParameter** structure will indicate the maximum string length to set in the feature.
If the complete list of the device features needs to be retrieved, it is recommended to use the function *csiIterateFeatureTree* . An example is shipped with the SDK to demonstrate the usage of it.
To set the values, please use the type-specific set-functions. For example, use **csiSetFeatureInt** for an integer value.

Acquiring images

To get images from the device, it must be opened first by calling the appropriate functions. The diagram below provides an overview of the functions which should be called during an acquisition process

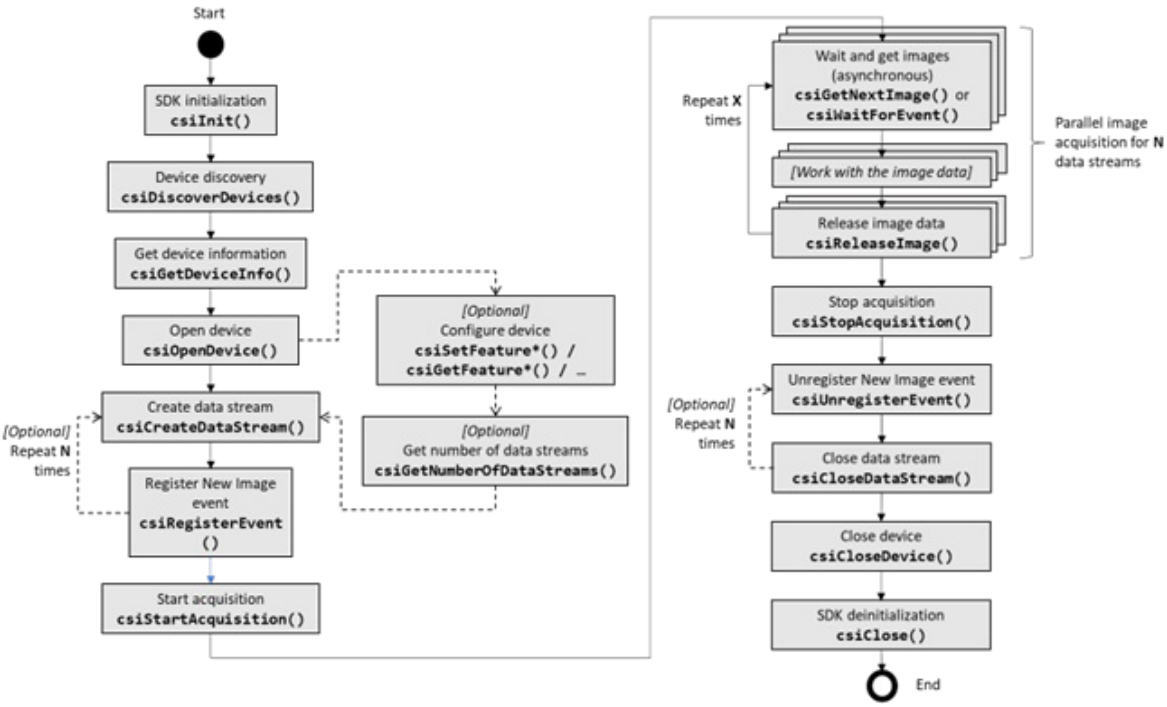


Image 1 overview_acquiring_process

Depending on the type of the device, it is possible to retrieve multiple data streams in parallel from the same device. This capability can be checked by using the **csiGetNumberOfDataStreams** function.

In general, two different ways of acquiring the images can be used:

1. Using Events (Events must be registered by the **csiRegisterEvent** function prior to the usage of the event.)
2. Directly calling the **csiGetNextImage** function

Independent of these two ways, the acquisition from the device must be started first by calling **csiStartAcquisition**. If enough images have been processed, this needs to be stopped again by calling **csiStopAcquisition**.

After a received image is processed, it must be released back into the receive buffer of the acquisition engine by calling **csiReleaseImage**.

Failing to do so will cause an error as soon as all receive buffers have been filled by the incoming data.

To grab images continuously, the processing part needs to keep up with the speed of the camera. Otherwise, images might be lost.

Examples

The SDK software package comes with a set of programming examples for C++. Currently there are four examples included:

Example	Description
acquisition_basics	Demonstrates how to discover and open a device and how to acquire images. Locations: Windows: C:\Users\Public\Documents\Chromasens\GCT2\examples\basic Linux: /usr/CSGenicam-SDK/share/csgenicam/examples/basic
feature_iteration	Demonstrates how to iterate through the feature tree of a device and how to set / get features. Locations: Windows: C:\Users\Public\Documents\Chromasens\GCT2\examples\feature_iteration Linux: /usr/CSGenicam-SDK/share/csgenicam/examples/feature_iteration
calibration_generate	Demonstrates how to generate PRNU and DSNU calibration files and upload it to camera or to save it in local PC memory. Locations: Windows: C:\Users\Public\Documents\Chromasens\GCT2\examples\calibration_generate Linux: /usr/CSGenicam-SDK/share/csgenicam/examples/calibration_generate
save_rgb10and12	Demonstrates on how to convert RGB10 and RGB12 pixelformat images into RGB8 or RGB16 bit images and save them in required image format. Supported image formats are (.png, .bmp. .tiff, .jpeg) Locations: Windows: C:\Users\Public\Documents\Chromasens\GCT2\examples\save_rgb10and12 Linux: /usr/CSGenicam-SDK/share/csgenicam/examples/save_rgb10and12

Visual Studio Example Projects

The Visual Studio projects for the two examples are also included in the SDK. These projects could the found the same location as stated above. These example projects could also be built with CMake. The following section explains how to build a project with CMake.

Build examples

To build the examples requires CMake version > v3.14 and a build environment. The steps to build the examples are the same for both Windows and Linux:
1) Open the CMake GUI and select the examples root directory as the source folder of your project. ("Where is the source code") 2) Next, select a directory where to generate the project files, should be somewhere outside the source tree. ("Where to build the binaries")

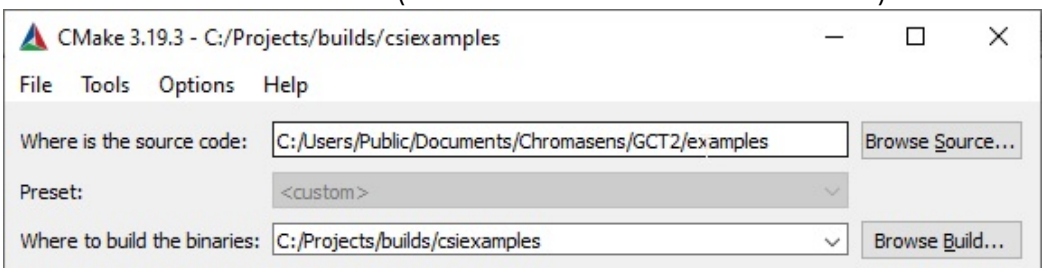


Image 1 cmake build project

3) Press the “Configure” button. After the first configuration it is required to manually set the path to the CSGenICam CMake configuration files:

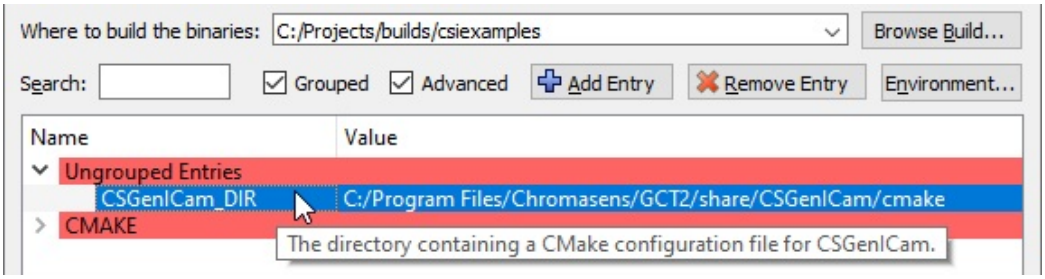


Image 1 cmake project configuration

4) Press “Configure” again and “Generate” afterwards. The project is now configured and can be opened and built from the directory selected in “Where to build the binaries”. 5) If the generated project is to be opened in Visual Studio, please follow the step mentioned in previousection, to add the DLL search path for the application.

Installation

Windows installation

On Windows platforms, the SDK can be installed together with the GCT software package. The SDK is not part of the default installation and must be selected during the installation phase of GCT. During the installation all required software will be placed in the installation folder.Please refer to the GCT documentation for a step by step installation of the full package.

Installer Contents

The default installation location of the SDK on Windows is C:\Program Files\Chromasens\GCT2
SDK The programming interface and library for customer applications
Locations: <installation root>\bin\csi.dll <installation root>\include\csi**csi.h** (and others)
<installation root>\lib\csi.lib
CMake Config CMake configuration files
Locations: <installation root>\share\CsiGenICam\cmake
GCT The camera configuration and acquisition application with graphical interface
Locations: <installation root>\bin\gct.exe
SDK Examples Example source code (C++) that shows the basic usage of the SDK
Locations: C:\Users\Public\Documents\chromasens\GCT2\examples
Documentation Documentation of the SDK
Location: <installation root>\doc
GenTL Producers (Optional) GenTL producers for Windows systems, if available
Locations: <installation root>\GenTL

Linux installation

This chapter covers the installation procedure of Chromasens Gen<i>Cam SDK on Linux. The SDK is distributed in an installation package and can be installed using the package manager of your distribution.
Note: Please note the list of currently supported Linux distributions:
Ubuntu 18.04 LTS

Preparation

Download the software package from the Chromasens website chromasens.de. Please note that the installation requires administrative rights on the system.

Step by Step Installation Ubuntu 18.04

1.) Open a new terminal window
2.) Navigate to the directory where the SDK software package is located. In this example it will be in the Downloads folder: **cd ~/Downloads**
3.) Update the package manager: **sudo apt update**
4.) Install the package using the package manager, replace the <version> part by the version of the downloaded package. The package manager might ask to install
additional required dependencies if they are not yet present in the system: **sudo apt install ./csgenicam-<version>.deb**
5.) After the installation, a system reboot is required to apply changes to the system environment.

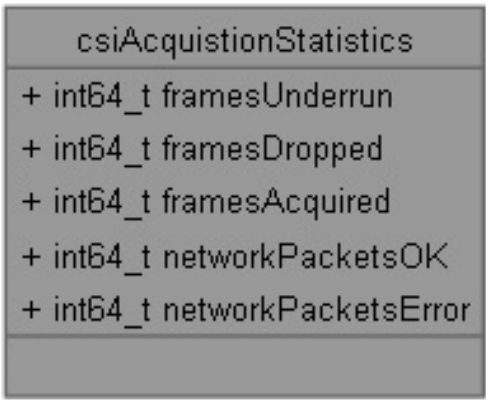
Installer Contents

The software package is grouped into the following components:
SDK The programming interface and library for customer applications
Locations: /usr/lib/libcsi.so /usr/include/csi/csi.h
/usr/share/CSGenICam/cmake/*
GCT The camera configuration and acquisition application with graphical interface
Locations: /usr/bin/gct
SDK Examples Example source code (C++) that shows the basic usage of the SDK
Locations: /usr/share/CSGenICam/examples
Documentation Documentation of the SDK
Location: /usr/share/CSGenICam/doc
GenTL Producers (Optional) GenTL producers for Linux systems, if available
Locations: /usr/lib
CCU Argus driver The driver for communication with the CCU hardware.
Locations: /usr/share/argus/driver

Data Structure Documentation

csiAcquistionStatistics Struct Reference

Structure containing acquisition statistics.
#include <csi/csi.h>
Collaboration diagram for csiAcquistionStatistics:



Data Fields

int64_t **framesUnderrun**
int64_t **framesDropped**
int64_t **framesAcquired**
int64_t **networkPacketsOK**
int64_t **networkPacketsError**

Detailed Description

Structure containing acquisition statistics.

Note

Negative statistic values indicate that they are not available.

Field Documentation

int64_t framesAcquired

Total number of frames acquired in current acquisition.

int64_t framesDropped

Number of frames dropped during acquisition.

int64_t framesUnderrun

The number of frames that were received in the TL but not send to the application because of missing buffers.

int64_t networkPacketsError

For GigE Vision: The number of network packets sent with an error.

int64_t networkPacketsOK

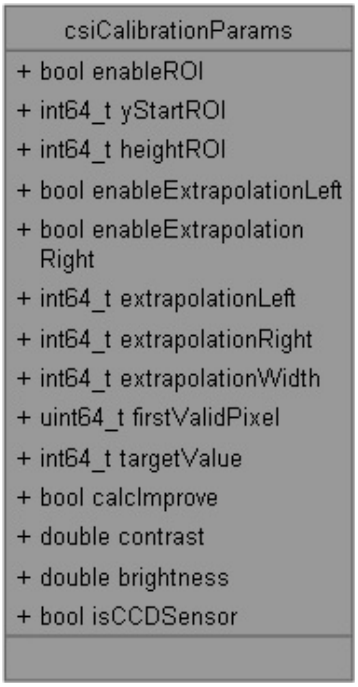
For GigE Vision: The number of network packets received without errors..

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

csi.h

csiCalibrationParams Struct Reference

Structure containing calibration parameters.
#include <csi/csi.h>
Collaboration diagram for csiCalibrationParams:



Data Fields

bool **enableROI**

int64_t **yStartROI**
int64_t **heightROI**
bool **enableExtrapolationLeft**
bool **enableExtrapolationRight**
int64_t **extrapolationLeft**
int64_t **extrapolationRight**
int64_t **extrapolationWidth**
uint64_t **firstValidPixel**
int64_t **targetValue**
bool **calcImprove**
double **contrast**
double **brightness**
bool **isCCDSensor** = false

Detailed Description

Structure containing calibration parameters.

Field Documentation

double brightness

Brightness value for calibration.

bool calcImprove

To enable outlier suppression

double contrast

Contrast value for calibration.

bool enableExtrapolationLeft

To enable/disable extrapolation based PRNU calibration generation on the left side of the image.

bool enableExtrapolationRight

To enable/disable extrapolation based PRNU calibration generation on the right side of the image.

bool enableROI

To enable ROI based PRNU calibration data generation.

int64_t extrapolationLeft

The first column on the left side of the image that starts the extrapolation area in left side.

int64_t extrapolationRight

The last column on the right side of the image that ends the extrapolation area in right side.

int64_t extrapolationWidth

The width of the extrapolation area on both left and right side.

uint64_t firstValidPixel

Pixel offset of the loaded image.Start position of first valid pixel.In firmware version older than 2.2.0, the first pixel has default offset 1. In firmware 2.2.0 or newer,the first pixelhas default offset value 0.

int64_t heightROI

Height of the ROI used for PRNU calibration generation.

bool isCCDSensor = false

For CCD Sensors, the extrapolation process is done differently than for CMOS sensors (All GigE and CXP-based Chromasens cameras).

int64_t targetValue

For PRNU, the target value of the resultant pixel value. Example, the max value of 8 bit pixel is 255.

int64_t yStartROI

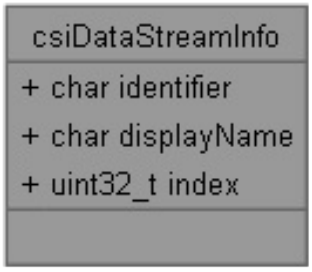
The row number of the source image which is the beginning of ROI.

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

csi.h

csiDataStreamInfo Struct Reference

Structure containing data stream infomation.
#include <csi/csi.h>
Collaboration diagram for csiDataStreamInfo:



Data Fields

char **identifier** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **displayName** [**CSI_INFO_STRING_BUFFER_SIZE**]
uint32_t **index**

Detailed Description

Structure containing data stream infomation.

Field Documentation

char displayName[CSI_INFO_STRING_BUFFER_SIZE]

Display name of a data stream that can be used for GUI representation

char identifier[CSI_INFO_STRING_BUFFER_SIZE]

Unique identifier of a data stream

uint32_t index

Internal index of the data stream

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

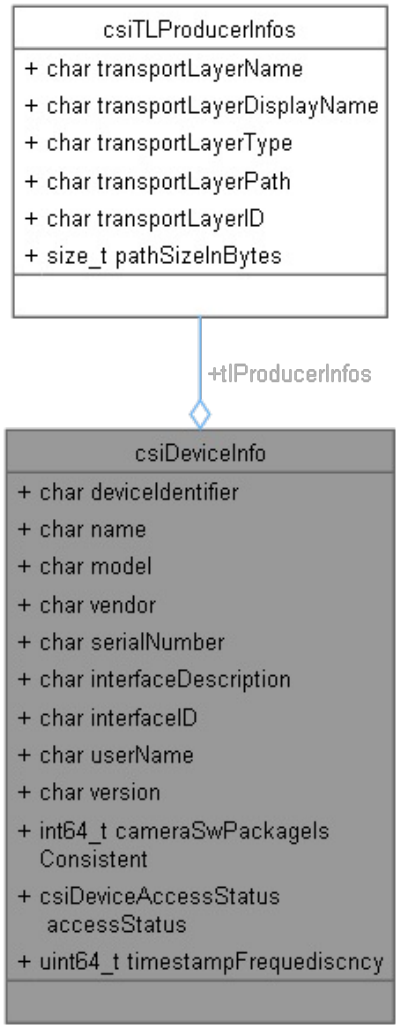
csi.h

csiDeviceInfo Struct Reference

Structure containing detailed device information.

#include <csi/csi.h>

Collaboration diagram for csiDeviceInfo:



Data Fields

char **deviceIdentifier** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **name** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **model** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **vendor** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **serialNumber** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **interfaceDescription** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **interfaceID** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **userName** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **version** [**CSI_INFO_STRING_BUFFER_SIZE**]
int64_t **cameraSwPackageIsConsistent**
csiTLProducerInfos tlProducerInfos
csiDeviceAccessStatus accessStatus
uint64_t **timestampFrequediscncy**

Detailed Description

Structure containing detailed device information.

Field Documentation

csiDeviceAccessStatus accessStatus

The current access status of the device, see csiDeviceAccessStatus

int64_t cameraSwPackageIsConsistent

char deviceIdentifier[CSI_INFO_STRING_BUFFER_SIZE]

Unique identifier of the device

char interfaceDescription[CSI_INFO_STRING_BUFFER_SIZE]

Name or description of the interface the device is connected to

char interfaceID[CSI_INFO_STRING_BUFFER_SIZE]

Unique identifier of the interface the device is connected to

char model[CSI_INFO_STRING_BUFFER_SIZE]

Model name of the device

char name[CSI_INFO_STRING_BUFFER_SIZE]

Name of the device

char serialNumber[CSI_INFO_STRING_BUFFER_SIZE]

Serial number of the device

uint64_t timestampFrequediscncy

Frequency of the timestamps coming from the device

csiTLProducerInfos tlProducerInfos

Information about the transport layer the device is connected to

char userName[CSI_INFO_STRING_BUFFER_SIZE]

Username when opening the device

char vendor[CSI_INFO_STRING_BUFFER_SIZE]

Vendor of the device

char version[CSI_INFO_STRING_BUFFER_SIZE]

Version of the device

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

csi.h

csiDiscoveryInfo Struct Reference

Structure containing information about device discovery.
#include <csi/csi.h>
Collaboration diagram for csiDiscoveryInfo:

Data Fields

uint32_t **numDevices**
double **progress**
bool **discoveryRunning**
csiDeviceInfo devices [**CSI_DISCOVERY_INFO_DEVICE_COUNT**]

Detailed Description

Structure containing information about device discovery.

Field Documentation

csiDeviceInfo devices[CSI_DISCOVERY_INFO_DEVICE_COUNT]

A list of devices found so far. The number of the devices found might exceed the size of this list, in which case the information must be acquired using the **csiGetDeviceInfo()** function.

bool discoveryRunning

Indicates if the discovery is still ongoing (true) or finished (false)

uint32_t numDevices

Current number of devices found during discovery

double progress

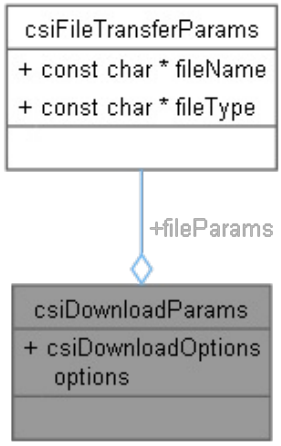
Discovery progress

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

csi.h

csiDownloadParams Struct Reference

Structure containing Download parameters and options.
#include <csi/csi.h>
Collaboration diagram for csiDownloadParams:



Data Fields

csiFileTransferParams fileParams
csiDownloadOptions options

Detailed Description

Structure containing Download parameters and options.

Field Documentation

csiFileTransferParams fileParams

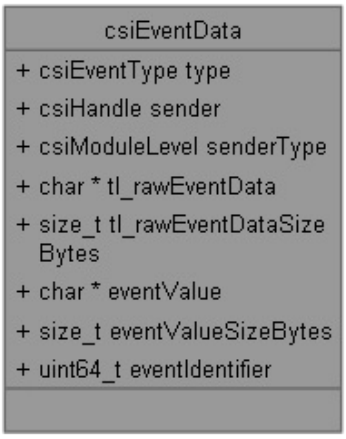
csiDownloadOptions options

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

csi.h

csiEventData Struct Reference

Structure containing information about event.
#include <csi/csi.h>
Collaboration diagram for csiEventData:



Data Fields

csiEventType type
csiHandle sender
csiModuleLevel senderType
char * **tl_rawEventData**
size_t **tl_rawEventDataSizeBytes**

char * **eventValue**
size_t **eventValueSizeBytes**
uint64_t **eventIdentifier**

Detailed Description

Structure containing information about event.

Field Documentation

uint64_t eventIdentifier

Event identifier

char* eventValue

The received value that was shipped together with the event. This can be for example the image data or a error description text in case of an error event. How to interpret the value depends on the type of event.

size_t eventValueSizeBytes

Size of the eventValue member in bytes

csiHandle sender

Handle to the sender of the event

csiModuleLevel senderType

Module level of the sender handle, see csiModuleLevel

char* tl_rawEventData

Raw data pointer to the event data as it was sent by the producer. This is just the raw data of the event which contains information about the type of event itself and not the value behind the event. See eventValue to get the actual value (e.g., image data) behind the event, if any.

size_t tl_rawEventDataSizeBytes

Size of the eventData member in bytes

csiEventType type

Type of the event, see csiEventType

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

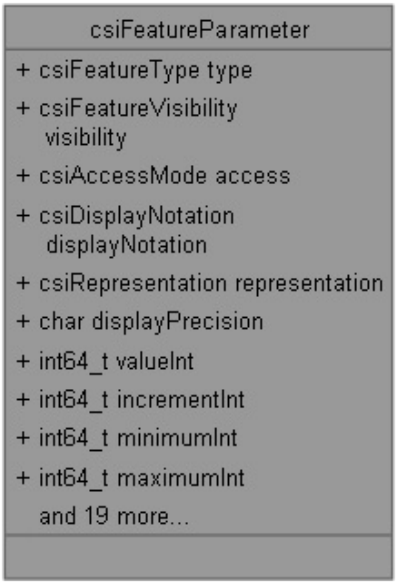
csi.h

csiFeatureParameter Struct Reference

Structure containing information about raw feature parameter.

#include <csi/csi.h>

Collaboration diagram for csiFeatureParameter:



Data Fields

csiFeatureType type
csiFeatureVisibility visibility
csiAccessMode access
csiDisplayNotation displayNotation
csiRepresentation representation
char **displayPrecision**
int64_t **valueInt**
int64_t **incrementInt**
int64_t **minimumInt**
int64_t **maximumInt**
int64_t **validValueSetInt** [**CSI_INFO_INT_BUFFER_SIZE**]
size_t **validValueSetSizeInt**
double **valueFlt**
double **incrementFlt**

double **minimumFlt**
double **maximumFlt**
char **valueStr** [**CSI_INFO_STRING_BUFFER_SIZE**]
size_t **maximumStringLength**
int64_t **level**
uint32_t **enumCounter**
int32_t **enumIndex**
char **displayName** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **name** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **tooltip** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **valueUnit** [**CSI_INFO_STRING_BUFFER_SIZE**]
size_t **featureRegLength**
int64_t **featureRegAddress**
bool **isFeature**
bool **isLittleEndian**

Detailed Description

Structure containing information about raw feature parameter.

Field Documentation

csiAccessMode access

How a feature can be accessed, see csiAccessMode

char displayName[CSI_INFO_STRING_BUFFER_SIZE]

Display name of the feature for UI display

csiDisplayNotation displayNotation

How to display floating point features, see csiDisplayNotation. (Optional)

char displayPrecision

Precision of floating-point value representation

uint32_t enumCounter

Number of elements in the enumeration feature

int32_t enumIndex

The index of an enumeration entry

int64_t featureRegAddress

Address of a register feature

size_t featureRegLength

Length of a register feature

double incrementFlt

Possible increment for the feature value, in case of floating-point feature type

int64_t incrementInt

Possible increment for the feature value, in case of integer feature type

bool isFeature

Requested node is a feature

bool isLittleEndian

int64_t level

The level of a feature in the tree (for graphical representation)

double maximumFlt

Maximum for the feature value, in case of floating-point feature type

int64_t maximumInt

Maximum for the feature value, in case of integer feature type

size_t maximumStringLength

Maximum length of the string feature value

double minimumFlt

Minimum for the feature value, in case of floating-point feature type

int64_t minimumInt

Minimum for the feature value, in case of integer feature type

char name[CSI_INFO_STRING_BUFFER_SIZE]

The name that identifies a feature

csiRepresentation representation

How feature data should be represented, see csiRepresentation (Optional)

char tooltip[CSI_INFO_STRING_BUFFER_SIZE]

Additional information about the feature that can be shown as tooltip in a GUI

csiFeatureType type

Data type of the feature, see csiFeatureType

int64_t validValueSetInt[CSI_INFO_INT_BUFFER_SIZE]

size_t validValueSetSizeInt

double valueFlt

Value of the feature, in case of floating-point feature type

int64_t valueInt

Value of the feature, in case of integer feature type

char valueStr[CSI_INFO_STRING_BUFFER_SIZE]

Value of the feature, in case of string feature type

char valueUnit[CSI_INFO_STRING_BUFFER_SIZE]

String unit to append to the value representation in a GUI

csiFeatureVisibility visibility

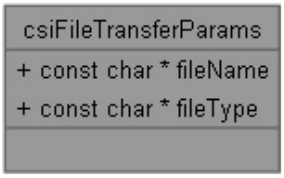
The visibility of a feature, see csiFeatureVisibility

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

csi.h

csiFileTransferParams Struct Reference

Structure containing file transfer parameter.
#include <csi/csi.h>
Collaboration diagram for csiFileTransferParams:



Data Fields

const char * **fileName**
const char * **fileType**

Detailed Description

Structure containing file transfer parameter.

Field Documentation

const char* fileName

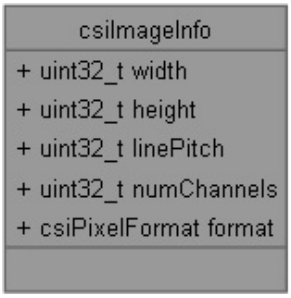
const char* fileType

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

csi.h

csiImageInfo Struct Reference

Structure containing information about captured image.
#include <csi/csi.h>
Collaboration diagram for csiImageInfo:



Data Fields

uint32_t **width**
uint32_t **height**
uint32_t **linePitch**
uint32_t **numChannels**
csiPixelFormat format

Detailed Description

Structure containing information about captured image.

Field Documentation

csiPixelFormat format

Pixel format of the image data, see csiPixelFormat

uint32_t height

Height of the image

uint32_t linePitch

Line pitch of the image data in bytes

uint32_t numChannels

Number of channels

uint32_t width

Width of the image

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

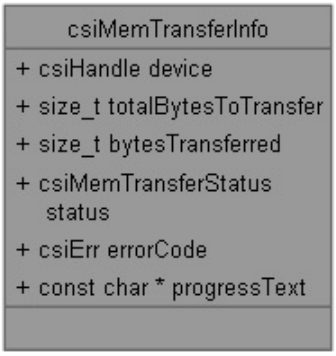
csi.h

csiMemTransferInfo Struct Reference

Structure containing information about memory transfer.

#include <csi/csi.h>

Collaboration diagram for csiMemTransferInfo:



Data Fields

csiHandle device
size_t **totalBytesToTransfer**
size_t **bytesTransferred**
csiMemTransferStatus status
csiErr errorCode
const char * **progressText**

Detailed Description

Structure containing information about memory transfer.

Field Documentation

size_t bytesTransferred

Current number of bytes already transferred.

csiHandle device

Handle to the device where the transfer is running on.

csiErr errorCode

Error code in case an error occurred.

const char* progressText

Progress information text.

csiMemTransferStatus status

Status of the memory transfer, see csiMemTransferStatus.

size_t totalBytesToTransfer

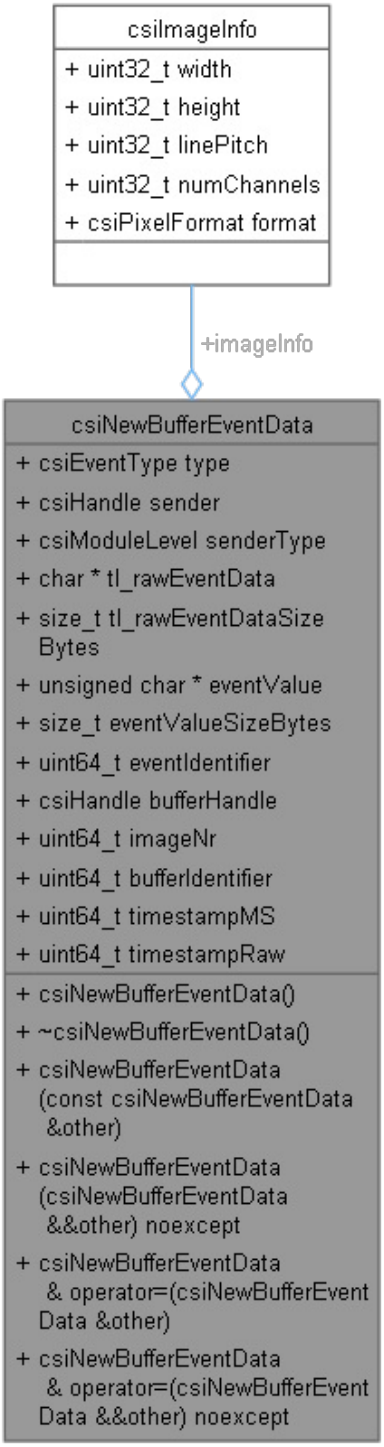
Total number of bytes to be transferred.

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

csi.h

csiNewBufferEventData Struct Reference

Structure containing data of an incoming buffer event.
#include <csi/csi.h>
Collaboration diagram for csiNewBufferEventData:



Public Member Functions

csiNewBufferEventData ()
~csiNewBufferEventData ()
csiNewBufferEventData (const **csiNewBufferEventData** &other)
csiNewBufferEventData (**csiNewBufferEventData** &&other) noexcept
csiNewBufferEventData & **operator=** (**csiNewBufferEventData** &other)
csiNewBufferEventData & **operator=** (**csiNewBufferEventData** &&other) noexcept

Data Fields

csiEventType type { CSI::csiEventType::CSI_EVT_NEWIMAGEDATA }
csiHandle sender { CSI_EMPTY_HANDLE }
csiModuleLevel senderType {
CSI::csiModuleLevel::CSI_UNKNOWN_MODULE }
char * **tl_rawEventData** { nullptr }
size_t **tl_rawEventDataSizeBytes** { 0 }
unsigned char * **eventValue** { nullptr }
size_t **eventValueSizeBytes** { 0 }
uint64_t **eventIdentifier** { 0 }
csiHandle bufferHandle { 0 }
uint64_t **imageNr** { 0 }
uint64_t **bufferIdentifier** { 0 }
uint64_t **timestampMS** { 0 }
uint64_t **timestampRaw** { 0 }
csiImageInfo imageInfo {}

Detailed Description

Structure containing data of an incoming buffer event.

Constructor & Destructor Documentation

csiNewBufferEventData ()

~csiNewBufferEventData ()

csiNewBufferEventData (const csiNewBufferEventData & *other*)

csiNewBufferEventData (csiNewBufferEventData && *other*) [noexcept]

Member Function Documentation

csiNewBufferEventData & operator= (csiNewBufferEventData && *other*)[noexcept]

csiNewBufferEventData & operator= (csiNewBufferEventData & *other*)

Field Documentation

csiHandle bufferHandle { 0 }

Handle to the buffer holding the image (for internal use)

uint64_t bufferIdentifier { 0 }

Unique identifier of the image, usually the pointer as integer representation

uint64_t eventIdentifier { 0 }

Unique identifier of this event

unsigned char* eventValue { nullptr }

Pointer to the image data

size_t eventValueSizeBytes { 0 }

Size of the image data in bytes

csiImageInfo imageInfo {}

Further image information, see **csiImageInfo**

uint64_t imageNr { 0 }

Number of the recorded image

csiHandle sender { CSI_EMPTY_HANDLE }

Sender of the event, a stream handle

csiModuleLevel senderType { CSI::csiModuleLevel::CSI_UNKNOWN_MODULE }

Type of the sender, this is always CSI_STREAM_MODULE for this type of event

uint64_t timestampMS { 0 }

Timestamp of the image in milliseconds

uint64_t timestampRaw { 0 }

Raw timestamp of the image

char* tl_rawEventData { nullptr }

Raw data pointer to the event data as it was sent by the producer. This is just the raw data of the event which contains information about the type of event itself and not the value behind the event. See eventValue to get the actual value (e.g., image data) behind the event, if any.

size_t tl_rawEventDataSizeBytes { 0 }

Size of the eventData member in bytes

csiEventType type { CSI::csiEventType::CSI_EVT_NEWIMAGEDATA }

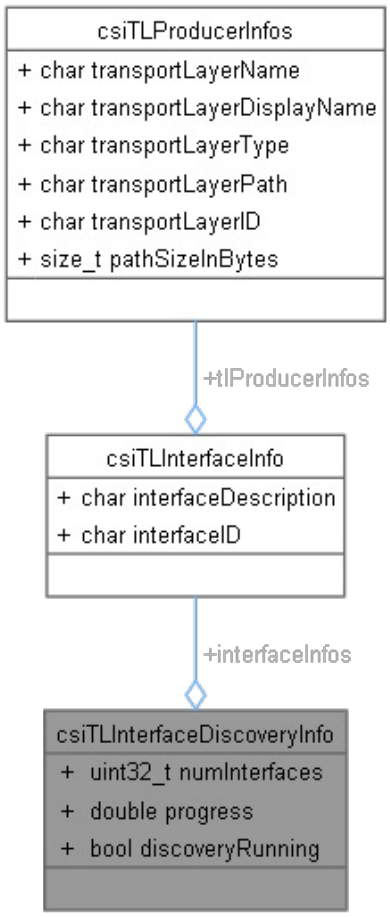
Type of the event, this is always CSI_EVT_NEWIMAGEDATA for this type of event

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

csi.h

csiTLInterfaceDiscoveryInfo Struct Reference

Structure containing information about transport layer interface discovery.
#include <csi/csi.h>
Collaboration diagram for csiTLInterfaceDiscoveryInfo:



Data Fields

uint32_t **numInterfaces**
double **progress**
bool **discoveryRunning**
csiTLInterfaceInfo interfaceInfos [**CSI_TL_INTERFACE_COUNT**]

Detailed Description

Structure containing information about transport layer interface discovery.

Field Documentation

bool discoveryRunning

Indicates if the discovery is still ongoing (true) or finished (false)

csiTLInterfaceInfo interfaceInfos[CSI_TL_INTERFACE_COUNT]

A list of TL interfaces found so far.

uint32_t numInterfaces

number of interfaces

double progress

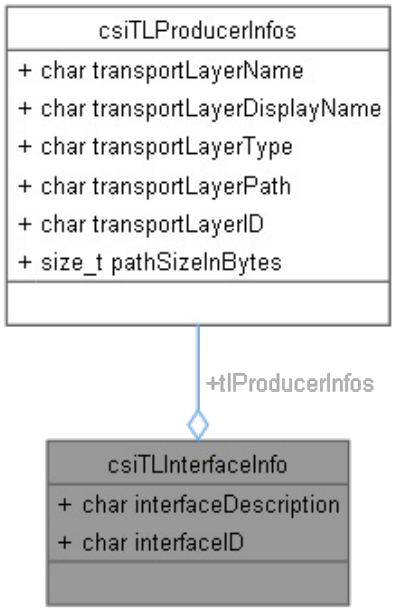
Discovery progress

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

csi.h

csiTLInterfaceInfo Struct Reference

Structure containing information about transport layer interface.
#include <csi/csi.h>
Collaboration diagram for csiTLInterfaceInfo:



Data Fields

char **interfaceDescription** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **interfaceID** [**CSI_INFO_STRING_BUFFER_SIZE**]
csiTLProducerInfos tlProducerInfos

Detailed Description

Structure containing information about transport layer interface.

Field Documentation

char interfaceDescription[CSI_INFO_STRING_BUFFER_SIZE]

interface description

char interfaceID[CSI_INFO_STRING_BUFFER_SIZE]

interface id

csiTLProducerInfos tlProducerInfos

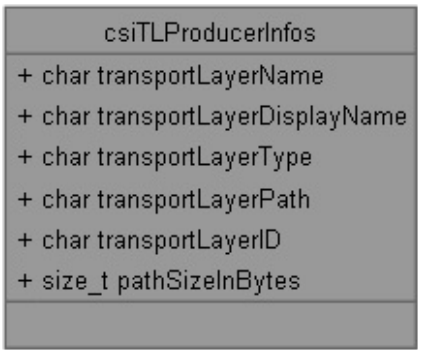
information of TL producers

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

csi.h

csiTLProducerInfos Struct Reference

Structure containing infomation about transport layer producer.
#include <csi/csi.h>
Collaboration diagram for csiTLProducerInfos:



Data Fields

char **transportLayerName** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **transportLayerDisplayName** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **transportLayerType** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **transportLayerPath** [**CSI_INFO_STRING_BUFFER_SIZE**]
char **transportLayerID** [**CSI_INFO_STRING_BUFFER_SIZE**]
size_t **pathSizeInBytes**

Detailed Description

Structure containing infomation about transport layer producer.

Field Documentation

size_t pathSizeInBytes

Length of the transport layer path.

char transportLayerDisplayName[CSI_INFO_STRING_BUFFER_SIZE]

Display name of a transport layer for GUI representation.

char transportLayerID[CSI_INFO_STRING_BUFFER_SIZE]

Unique identifier of the transport layer as string.

char transportLayerName[CSI_INFO_STRING_BUFFER_SIZE]

Name of a transport layer.

char transportLayerPath[CSI_INFO_STRING_BUFFER_SIZE]

Full path to the transport layer library file (.cti file).

char transportLayerType[CSI_INFO_STRING_BUFFER_SIZE]

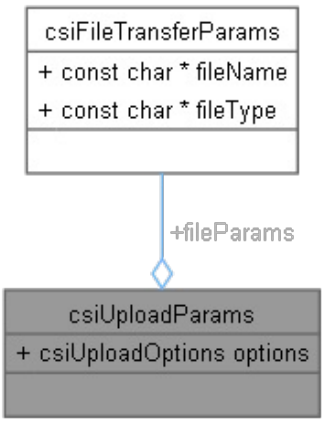
Type of the transport layer as string.

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

csi.h

csiUploadParams Struct Reference

Structure containing Upload parameters and options.
#include <csi/csi.h>
Collaboration diagram for csiUploadParams:



Data Fields

csiFileTransferParams fileParams
csiUploadOptions options

Detailed Description

Structure containing Upload parameters and options.

Field Documentation

csiFileTransferParams fileParams

csiUploadOptions options

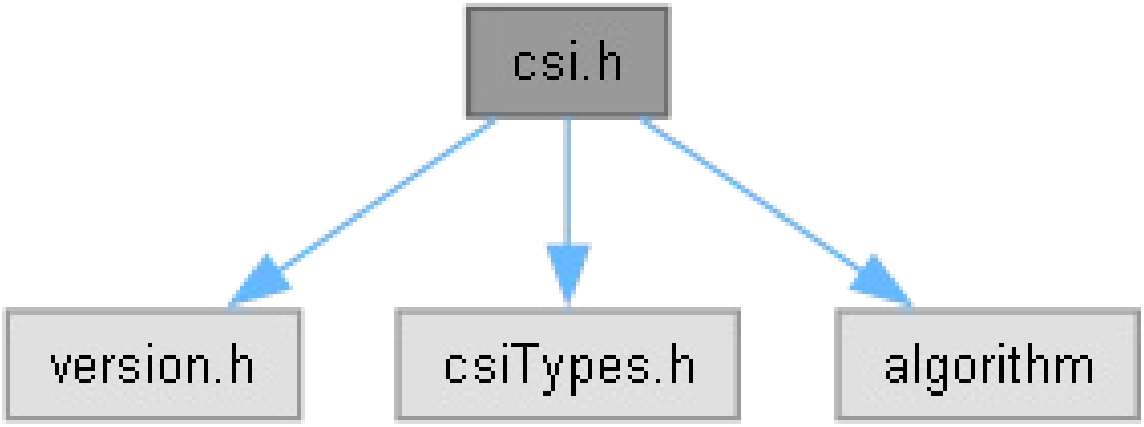
The documentation for this struct was generated from the following file:

csi.h

File Documentation

csi.h File Reference

#include "version.h"
#include "csiTypes.h"
#include <algorithm>
Include dependency graph for csi.h:



Data Structures

struct **csiFeatureParameter** *Structure containing information about raw feature parameter.*
struct **csiMemTransferInfo** *Structure containing information about memory transfer.*
struct **csiTLProducerInfos** *Structure containing infomation about transport layer producer.*
struct **csiDeviceInfo** *Structure containing detailed device information.*
struct **csiDiscoveryInfo** *Structure containing information about device discovery.*
struct **csiTLInterfaceInfo** *Structure containing information about transport layer interface.*
struct **csiTLInterfaceDiscoveryInfo** *Structure containing information about transport layer interface discovery.*
struct **csiDataStreamInfo** *Structure containing data stream infomation.*
struct **csiImageInfo** *Structure containing information about captured image.*
struct **csiEventData** *Structure containing information about event.*

struct **csiNewBufferEventData** *Structure containing data of an incoming buffer event.*
struct **csiAcquistionStatistics** *Structure containing acquisition statistics.*
struct **csiFileTransferParams** *Structure containing file transfer parameter.*
struct **csiDownloadParams** *Structure containing Download parameters and options.*
struct **csiUploadParams** *Structure containing Upload parameters and options.*
struct **csiCalibrationParams** *Structure containing calibration parameters.*

Macros

#define **CSI_INFO_STRING_BUFFER_SIZE** 512
#define **CSI_INFO_INT_BUFFER_SIZE** 512
#define **CSI_DISCOVERY_INFO_DEVICE_COUNT** 16
#define **CSI_TL_INTERFACE_COUNT** 16
#define **CSI_INFITIE_TIME** 0xffffffff
#define **CSI_MONO_FORMAT** 0x01000000
#define **CSI_COLOR_FORMAT** 0x02000000
#define **CSI_OCCUPY_8BIT** 0x00080000
#define **CSI_OCCUPY_10BIT** 0x000A0000
#define **CSI_OCCUPY_12BIT** 0x000C0000
#define **CSI_OCCUPY_14BIT** 0x000E0000
#define **CSI_OCCUPY_16BIT** 0x00100000
#define **CSI_OCCUPY_24BIT** 0x00180000
#define **CSI_OCCUPY_30BIT** 0x001E0000
#define **CSI_OCCUPY_32BIT** 0x00200000
#define **CSI_OCCUPY_48BIT** 0x00300000
#define **CSI_OCCUPY_64BIT** 0x00400000

Typedefs

typedef uint64_t **csiHandle**
typedef enum **csiPixelFormat csiPixelFormat**
typedef enum **csiDeviceAccessMode csiDeviceAccessMode**
typedef enum **csiDeviceAccessStatus csiDeviceAccessStatus**
typedef enum **csiFeatureType csiFeatureType**
typedef enum **csiAccessMode csiAccessMode**
typedef enum **csiFeatureVisibility csiFeatureVisibility**
typedef enum **csiModuleLevel csiModuleLevel**
typedef enum **csiDisplayNotation csiDisplayNotation**

typedef enum **csiRepresentation csiRepresentation**
typedef struct csiFeatureParameter **csiFeatureParameter**
typedef enum **csiLogLevel csiLogLevel**
typedef enum **csiErr csiErr**
typedef enum **csiEventType csiEventType**
typedef enum **csiAcquisitionMode csiAcquisitionMode**
typedef enum **csiMemTransferStatus csiMemTransferStatus**
typedef struct csiMemTransferInfo **csiMemTransferInfo**
typedef struct csiTLProducerInfos **csiTLProducerInfos**
typedef struct csiDeviceInfo **csiDeviceInfo**
typedef struct csiDiscoveryInfo **csiDiscoveryInfo**
typedef struct csiTLInterfaceInfo **csiTLInterfaceInfo**
typedef struct csiTLInterfaceDiscoveryInfo **csiTLInterfaceDiscoveryInfo**
typedef struct csiDataStreamInfo **csiDataStreamInfo**
typedef struct csiImageInfo **csiImageInfo**
typedef struct csiEventData **csiEventData**
typedef struct CSI_DLL_EXPORT csiNewBufferEventData **csiNewBufferEventData**
typedef struct csiAcquistionStatistics **csiAcquistionStatistics**
typedef enum **csiDownloadOptions csiDownloadOptions**
typedef enum **csiUploadOptions csiUploadOptions**
typedef struct csiFileTransferParams **csiFileTransferParams**
typedef struct **csiDownloadParams csiUploadDownloadUploadParams**
typedef struct csiUploadParams **csiUploadParams**
typedef enum **csiReferenceImgLoadMode csiReferenceImgLoadMode**
typedef struct csiCalibrationParams **csiCalibrationParams**
typedef enum **CalibrationMode CalibrationMode**
typedef void * **CB_OBJECT**
typedef void(* **CB_FEATURE_INVALIDATED_PFN**) (const char *featureName, void *userdata)
typedef void(* **csiDiscoveryInfoCallbackFunc**) (const **csiDiscoveryInfo** *discoveryInfo)
typedef void(* **csiDiscoveryTLInterfaceInfoCallbackFunc**) (const **csiTLInterfaceDiscoveryInfo** *discoveryInfo)
typedef bool(* **csiMemTransferCallbackFunc**) (const **csiMemTransferInfo** *info, struct csiMemTransferUserData *userdata)
typedef void(* **csiLogSinkCallbackFunc**) (**csiLogLevel**, const char *message, struct csiLogUserData *userdata)

Enumerations

enum **csiPixelFormat** { **CSI_PIX_FORMAT_UNKNOWN** = 0x00000000, **CSI_PIX_FORMAT_MONO8** = CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x001, **CSI_PIX_FORMAT_MONO10** = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x003, **CSI_PIX_FORMAT_MONO10_PACKED** = CSI_MONO_FORMAT | CSI_OCCUPY_10BIT | 0x046, **CSI_PIX_FORMAT_MONO12** = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x005, **CSI_PIX_FORMAT_MONO12_PACKED** = CSI_MONO_FORMAT | CSI_OCCUPY_12BIT | 0x047, **CSI_PIX_FORMAT_MONO16** = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x007, **CSI_PIX_FORMAT_RGB8** = CSI_COLOR_FORMAT | CSI_OCCUPY_24BIT | 0x014, **CSI_PIX_FORMAT_RGB10_PACKED** = CSI_COLOR_FORMAT | CSI_OCCUPY_32BIT | 0x01D, **CSI_PIX_FORMAT_RGB10** = CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x018, **CSI_PIX_FORMAT_BGR10** = CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x019, **CSI_PIX_FORMAT_RGB12** = CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x01A, **CSI_PIX_FORMAT_BGR12** = CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x01B, **CSI_PIX_FORMAT_RGBA8** = CSI_COLOR_FORMAT | CSI_OCCUPY_32BIT | 0x016, **CSI_PIX_FORMAT_BGRA8** = CSI_COLOR_FORMAT | CSI_OCCUPY_32BIT | 0x017, **CSI_PIX_FORMAT_BGR8** = CSI_COLOR_FORMAT | CSI_OCCUPY_24BIT | 0x015, **CSI_PIX_FORMAT_RGB16** = CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x033, **CSI_PIX_FORMAT_RGBA10** = CSI_COLOR_FORMAT | CSI_OCCUPY_64BIT | 0x05F, **CSI_PIX_FORMAT_RGBA12** = CSI_COLOR_FORMAT | CSI_OCCUPY_64BIT | 0x061, **CSI_PIX_FORMAT_RGBA16** = CSI_COLOR_FORMAT | CSI_OCCUPY_64BIT | 0x064, **CSI_PIX_FORMAT_BayerGR8** = CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x008, **CSI_PIX_FORMAT_BayerRG8** = CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x009, **CSI_PIX_FORMAT_BayerGB8** = CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x00A, **CSI_PIX_FORMAT_BayerBG8** = CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x00B, **CSI_PIX_FORMAT_BayerGR12** = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x010, **CSI_PIX_FORMAT_BayerRG12** = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x011, **CSI_PIX_FORMAT_BayerGB12** = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x012, **CSI_PIX_FORMAT_BayerBG12** = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x013 }

Defines the currently supported pixel data formats.

enum **csiDeviceAccessMode** { **CSI_DEV_MODE_UNKNOWN** = 0x00, **CSI_DEV_MODE_NONE** = 0x01, **CSI_DEV_MODE_EXCLUSIVE**, **CSI_DEV_MODE_READ**, **CSI_DEV_MODE_CONTROL** }

Defines the mode in which a device will be opened.

enum **csiDeviceAccessStatus** { **CSI_DEV_ACCESS_STATUS_UNKNOWN** = 0x00, **CSI_DEV_ACCESS_STATUS_READWRITE** = 0x01, **CSI_DEV_ACCESS_STATUS_READONLY** = 0x02, **CSI_DEV_ACCESS_STATUS_NOACCESS** = 0x03, **CSI_DEV_ACCESS_STATUS_BUSY** = 0x04, **CSI_DEV_ACCESS_STATUS_OPEN_READWRITE** = 0x05, **CSI_DEV_ACCESS_STATUS_OPEN_READ** = 0x06 }

Defines the current access status of a device as returned from device discovery.

enum **csiFeatureType** { **CSI_UNKNOWN_TYPE**, **CSI_BOOLEAN_TYPE**, **CSI_INT_TYPE**, **CSI_FLOAT_TYPE**, **CSI_STRING_TYPE**, **CSI_ENUMERATION_TYPE**, **CSI_ENUMENTRY_TYPE**, **CSI_CATEGORY**, **CSI_COMMAND**, **CSI_REGISTER**, **CSI_PORT** }

Defines the data type of a feature.

enum **csiAccessMode** { **CSI_ACCESS_UNKNOWN**, **CSI_ACCESS_NOT_AVAILABLE**, **CSI_ACCESS_READ_ONLY**, **CSI_ACCESS_READ_WRITE**, **CSI_ACCESS_WRITE_ONLY** }

Defines the access mode of a feature.

enum **csiFeatureVisibility** { **CSI_VISIBILITY_BEGINNER** = 1, **CSI_VISIBILITY_EXPERT**, **CSI_VISIBILITY_GURU**, **CSI_VISIBILITY_DEVELOPER**, **CSI_VISIBILITY_INVISIBLE** }

Defines the visibility of a feature depending on the role of a user.

enum **csiModuleLevel** { **CSI_UNKNOWN_MODULE**, **CSI_TRANSPORTLAYER_MODULE**, **CSI_INTERFACE_MODULE**, **CSI_DEVICE_MODULE**, **CSI_LOCAL_DEVICE_MODULE**, **CSI_STREAM_MODULE**, **CSI_BUFFER_MODULE** }

Defines the module level on which a specific action should be performed.

enum **csiDisplayNotation** { **CSI_NOTATION_AUTOMATIC**, **CSI_NOTATION_FIXED**, **CSI_NOTATION_SCIENTIFIC** }

Defines the display notation for a floating-point feature.

enum **csiRepresentation** { **CSI_REPRESENTATION_LINEAR**, **CSI_REPRESENTATION_LOGARITHMIC**, **CSI_REPRESENTATION_BOOLEAN**, **CSI_REPRESENTATION_PURENUMBER**, **CSI_REPRESENTATION_HEX**, **CSI_REPRESENTATION_IP**, **CSI_REPRESENTATION_MAC**, **CSI_REPRESENTATION_UNDEFINED** }

Defines how a feature value should be represented when printed in UI.

enum **csiLogLevel** { **CSI_LOGLEVEL_NONE** = 0, **CSI_LOGLEVEL_ERROR** = 1, **CSI_LOGLEVEL_WARN** = 2, **CSI_LOGLEVEL_INFO** = 4, **CSI_LOGLEVEL_DEBUG** = 8, **CSI_LOGLEVEL_TRACE** = 16 }

Defines the severity of log messages coming from the SDK.

enum **csiErr** { **csiSuccess** = 0, **csiNotInitialized** = -100, **csiInvalidState** = -101, **csiNotOpened** = -102, **csiNoImageDataAvailable** = -103, **csiNotFound** = -104, **csiInvalidParameter** = -105, **csiNotAvailable** = -106, **csiFunctionNotAvailable** = -107, **csiTimeout** = -108, **csiAborted** = -109, **csiFileOperationFailure** = -110, **csiFileOperationFatalError** = -111, **csiNoAccess** = -112, **csiWrongBufferSize** = -113, **csiInvalidBuffer** = -114, **csiResourceInUse** = -115, **csiNotImplemented** = -116, **csiInvalidHandle** = -117, **csiIOError** = -118, **csiParsingError** = -119, **csiInvalidValue** = -120, **csiResourceExhausted** = -121, **csiOutOfMemory** = -122, **csiBusy** = -123, **csiUnknown** = -200, **csiCustomErr** = -0x0f000000 }

Defines possible error values.

enum **csiEventType** { **CSI_EVT_NEWIMAGEDATA** = 0x00, **CSI_EVT_ERROR** = 0x01, **CSI_EVT_MODULE** = 0x02, **CSI_EVT_FEATURE_INVALIDATE** = 0x03, **CSI_EVT_FEATURE_CHANGE** = 0x04, **CSI_EVT_REMOTE_DEVICE** = 0x05, **CSI_EVT_CUSTOM** = 0x1000 }

Defines event types that the user application can listen for.

enum **csiAcquisitionMode** { **CSI_ACQUISITION_SINGLE_FRAME** = 0x00000001, **CSI_ACQUISITION_CONTINUOUS** = 0xFFFFFFFF }

Defines acquisition mode.

enum **csiMemTransferStatus** { **csiTransferStatusInit**, **csiTransferStatusInProgress**, **csiTransferStatusInProgressWaiting**, **csiTransferStatusFinishSucess**, **csiTransferStatusFinishError**, **csiTransferStatusCancelOnError** }

Defines the status of memory transfer functions as it is provided in the transfer callback.

enum **csiDownloadOptions** { **CSI_DOWNLOAD_NO_OPTIONS** = 0 }

Defines download options.

enum **csiUploadOptions** { **CSI_UPLOAD_NO_OPTIONS** = 0, **CSI_UPLOAD_IGNORE_CHECKSUM** = 1 }
Defines upload options.
enum **csiCalibrationLUT** { **CSI_DSNU_LUT1** = 1, **CSI_DSNU_LUT2**, **CSI_PRNU_LUT1**, **CSI_PRNU_LUT2** }
Defines calibration LUT options.
enum **csiReferenceImgLoadMode** { **CSI_LOAD_REF_IMG_FROM_DISC**, **CSI_ACQUIRE_REF_IMG_FROM_CAMERA** }
Defines mode of loading reference imgage.
enum **CalibrationMode** { **DSNU_CALIBRATION** = 1, **PRNU_CALIBRATION** }
Defines mode for calibration.

Functions

CSI_DLL_EXPORT **csiErr csiInit** (**csiLogLevel** logLvl, **csiLogSinkCallbackFunc** logCallbackFunc CSI_DEFAULT_PARAM_NULL, csiLogUserData *userdata CSI_DEFAULT_PARAM_NULL)
Initializes the SDK. Needs to be called first before any other function of the SDK is called!
CSI_DLL_EXPORT **csiErr csiClose** ()
Closes the SDK and frees all allocated memory and interfaces.
CSI_DLL_EXPORT **csiErr csiReset** ()
Reset the SDK and frees all allocated memory and interfaces.
CSI_DLL_EXPORT **csiErr csiDiscoverDevices** (**csiDiscoveryInfo** *discoveryInfoOut, uint64_t timeoutMilliseconds, **csiDiscoveryInfoCallbackFunc** discCallbackFunc CSI_DEFAULT_PARAM_NULL, const char *additionalSearchPaths CSI_DEFAULT_PARAM_NULL, bool overrideSearchPath CSI_DEFAULT_PARAM_FALSE)
This function will look for attached GenICAM-devices on the available transport layers.
CSI_DLL_EXPORT **csiErr csiGetDeviceInfo** (uint32_t deviceIndex, **csiDeviceInfo** *deviceInfoOut)
A function to get information about the found devices in the system.
CSI_DLL_EXPORT **csiErr csiDiscoverTLInterfaces** (**csiTLInterfaceDiscoveryInfo** *discoveryInfoOut, uint64_t timeoutMilliseconds, **csiDiscoveryTLInterfaceInfoCallbackFunc** discCallbackFunc CSI_DEFAULT_PARAM_NULL, const char *additionalSearchPaths CSI_DEFAULT_PARAM_NULL, bool overrideSearchPath CSI_DEFAULT_PARAM_FALSE)

This function will look for TL interfaces provided by the available transport layers.

CSI_DLL_EXPORT **csiErr csiGetNumberOfTLProducers** (int32_t *numTLProducers)

Returns the number of available transport layers in the system.

CSI_DLL_EXPORT **csiErr csiGetTLProducerPathByIndex** (char *transportLayerPath, size_t bufferSize, uint32_t index)

Returns a path of transport layer producer specified by index.

CSI_DLL_EXPORT **csiErr csiGetTLProducerInfobyFilePath** (**csiTLProducerInfos** *tlProducerInfos, const char *producerName)

Returns additional information about a transport layer.

CSI_DLL_EXPORT **csiErr csiOpenDevice** (const char *deviceIdentifier, const char *interfaceID, **csiHandle** *deviceHandleOut, uint64_t timeoutMilliseconds, **csiDeviceAccessMode** openMode)

Open the device given by the index. The TL of this index is used.

CSI_DLL_EXPORT **csiErr csiCloseDevice** (**csiHandle** deviceHandle)

Close the connection to the specific device.

CSI_DLL_EXPORT **csiErr csiCheckAndReallocBuffers** (**csiHandle** deviceHandle)

Reallocating buffers if the size of allocated buffers is diffrent from the require size.

CSI_DLL_EXPORT **csiErr csiStartAcquisition** (**csiHandle** deviceHandle, **csiAcquisitionMode** mode)

Start the acquisition on the device and created data streams.

CSI_DLL_EXPORT **csiErr csiStopAcquisition** (**csiHandle** deviceHandle)

Stop the acquisition on the device.

CSI_DLL_EXPORT **csiErr csiAbortAcquisition** (**csiHandle** deviceHandle)

Aborts the Acquisition on the device immediately.

CSI_DLL_EXPORT **csiErr csiGetNumberOfDataStreams** (**csiHandle** deviceHandle, uint32_t *numberOfStreamsOut)

Return the available number of data streams of the device.

CSI_DLL_EXPORT **csiErr csiGetDataStreamInfo** (**csiHandle** deviceHandle, uint32_t dsIndex, **csiDataStreamInfo** *dataStreamInfoOut)

Return information about the desired data stream.

CSI_DLL_EXPORT **csiErr csiGetDeviceDataStreamInfo** (**csiHandle** moduleHandle, uint32_t dsIndex, **csiDataStreamInfo** *dataStreamInfoOut)

Return information about the desired data stream and device.

CSI_DLL_EXPORT **csiErr csiCreateDataStream** (**csiHandle** deviceHandle, uint32_t dsIndex, **csiHandle** *dataStreamOut, uint32_t numberOfBuffers, size_t bufferSize CSI_DEFAULT_PARAM_ZERO)
Create a data stream to receive images.
CSI_DLL_EXPORT **csiErr csiCloseDataStream** (**csiHandle** dataStream)
Close the data stream.
CSI_DLL_EXPORT **csiErr csiRegisterEvent** (**csiHandle** moduleHandle, **csiEventType** evtType, **csiHandle** *eventOut, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)
Register an event which will be signaled in the case the desired event is triggered.
CSI_DLL_EXPORT **csiErr csiWaitForEvent** (**csiHandle** evt, uint64_t timeoutMilliseconds, **csiEventData** **evtDataOut)
Wait for a desired event to happen.
CSI_DLL_EXPORT **csiErr csiUnregisterEvent** (**csiHandle** evt)
Unregister a specific event from the event handler.
CSI_DLL_EXPORT **csiErr csiEventKill** (**csiHandle** evt)
Cancel all waiting functions related to this event.
CSI_DLL_EXPORT **csiErr csiGetNextImage** (**csiHandle** eventHandle, **csiNewBufferEventData** **bufferInfoOut, uint64_t timeoutMilliseconds)
Get the next image from the data stream.
CSI_DLL_EXPORT **csiErr csiReleaseImage** (**csiHandle** dataStream, const **csiNewBufferEventData** *bufferInfo)
Release the image back into the processing buffer from the device.
CSI_DLL_EXPORT **csiErr csiGetAcquisitionStatistics** (**csiHandle** dataStream, **csiAcquistionStatistics** *stats)
Retrieve the statistical buffer regarding the data stream (e.g. transmitted frames, etc.).
CSI_DLL_EXPORT **csiErr csiOpenTLInterface** (**csiTLInterfaceInfo** interfaceInfo, **csiHandle** *interfaceHandleOut, uint64_t timeoutMilliseconds)
Open the interface given by the interface information.
CSI_DLL_EXPORT **csiErr csiCloseTLInterface** (**csiHandle** interfaceHandle)
Close the connection to the specific interface.
CSI_DLL_EXPORT **csiErr csiGetFeatureBool** (**csiHandle** moduleHandle, const char *featureName, bool *valueOut, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)
Retrieve a boolean feature from the device.

CSI_DLL_EXPORT **csiErr csiSetFeatureBool** (**csiHandle** moduleHandle, const char *featureName, bool value, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)

Set a boolean feature on the device.

CSI_DLL_EXPORT **csiErr csiGetFeatureInt** (**csiHandle** moduleHandle, const char *featureName, int64_t *valueOut, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)

Retrieve an integer feature from the device.

CSI_DLL_EXPORT **csiErr csiSetFeatureInt** (**csiHandle** moduleHandle, const char *featureName, int64_t value, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)

Set an integer feature on the device.

CSI_DLL_EXPORT **csiErr csiGetFeatureFloat** (**csiHandle** moduleHandle, const char *featureName, double *valueOut, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)

Retrieve a floating point feature from the device.

CSI_DLL_EXPORT **csiErr csiSetFeatureFloat** (**csiHandle** moduleHandle, const char *featureName, double value, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)

Set a floating point value feature on the device.

CSI_DLL_EXPORT **csiErr csiGetFeatureString** (**csiHandle** moduleHandle, const char *featureName, char *valueOut, size_t *sizeOut, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)

Retrieve a string feature from the device.

CSI_DLL_EXPORT **csiErr csiSetFeatureString** (**csiHandle** moduleHandle, const char *featureName, const char *value, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)

Set a string feature on the device.

CSI_DLL_EXPORT **csiErr csiExecuteCommand** (**csiHandle** moduleHandle, const char *featureName, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)

Execute a command on the device.

CSI_DLL_EXPORT **csiErr csiIsCommandActive** (**csiHandle** moduleHandle, const char *featureName, bool *isActive, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)

Check if a command is still active.

CSI_DLL_EXPORT **csiErr csiGetFeatureReg** (**csiHandle** moduleHandle, const char *featureName, char *buffer, size_t *length, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)

Retrieve a register value from the device.
CSI_DLL_EXPORT **csiErr csiSetFeatureReg** (**csiHandle** moduleHandle, const char *featureName, const char *buffer, size_t length, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)
Set a register value on the device.
CSI_DLL_EXPORT **csiErr csiGetFeatureEnum** (**csiHandle** moduleHandle, const char *featureName, **csiFeatureParameter** *featureParamOut, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)
Retrieve an enumeration feature from the device.
CSI_DLL_EXPORT **csiErr csiSetFeatureEnum** (**csiHandle** moduleHandle, const char *featureName, const char *value, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)
Set an enumeration feature on the device.
CSI_DLL_EXPORT **csiErr csiGetFeatureParameter** (**csiHandle** moduleHandle, const char *featureName, **csiFeatureParameter** *featureParamOut, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)
Retrieve a specific feature from the device. Detailed information about this feature will be returned.
CSI_DLL_EXPORT **csiErr csiGetFeatureAccessMode** (**csiHandle** moduleH, const char *featureName, **csiAccessMode** *access, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)
Retrieve a specific feature from the device. The access mode of this feature will be returned.
CSI_DLL_EXPORT **csiErr csiIterateFeatureTree** (**csiHandle** moduleHandle, const char *rootFeatureName, uint32_t index, char *featureNameOut, size_t nameBuffSize, **csiFeatureType** *type, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)
Provides a possibility to iterate through all available features on the camera.
CSI_DLL_EXPORT **csiErr csiGetFeatureEnumEntryCount** (**csiHandle** moduleHandle, const char *featureName, uint32_t *count, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)
Retrieve the entries size of an enumeration feature from the device by feature name.
CSI_DLL_EXPORT **csiErr csiGetFeatureEnumEntryByIndex** (**csiHandle** moduleHandle, const char *featureName, int32_t enumIndex, **csiFeatureParameter** *featureParamOut, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)

Retrieve an enumeration feature from the device by using its index.
CSI_DLL_EXPORT **csiErr csiGetFeatureEnumEntryByName** (**csiHandle** moduleHandle, const char *featureName, const char *enumValue, **csiFeatureParameter** *featureParamOut, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)
Retrieve an enumeration feature from the device by using its name.
CSI_DLL_EXPORT **csiErr csiGetUpdateFileType** (**csiHandle** moduleHandle, const char *fileName, char *fileTypeOut, size_t bufferSize)
Request the update file type of a file (if available).
CSI_DLL_EXPORT **csiErr csiFileDownloadToDevice** (**csiHandle** moduleHandle, **csiHandle** localDevice, const char *fileName, const char *fileType, uint64_t timeoutMilliseconds, **csiMemTransferCallbackFunc** listener CSI_DEFAULT_PARAM_NULL, csiMemTransferUserData *userdata CSI_DEFAULT_PARAM_NULL)
Downloads a file located on the local PC to the camera.
CSI_DLL_EXPORT **csiErr csiFileUploadFromDevice** (**csiHandle** moduleHandle, **csiHandle** localDevice, const char *fileName, const char *fileType, uint64_t timeoutMilliseconds, **csiMemTransferCallbackFunc** listener CSI_DEFAULT_PARAM_NULL, csiMemTransferUserData *userdata CSI_DEFAULT_PARAM_NULL)
Uploads a file from device to the local PC.
CSI_DLL_EXPORT **csiErr csiFileDownloadToDeviceEx** (**csiHandle** moduleHandle, **csiHandle** localDevice, const **csiDownloadParams** params, uint64_t timeoutMilliseconds, **csiMemTransferCallbackFunc** listener CSI_DEFAULT_PARAM_NULL, csiMemTransferUserData *userdata CSI_DEFAULT_PARAM_NULL)
Downloads a file located on the local PC to the camera providing custom options.
CSI_DLL_EXPORT **csiErr csiFileUploadFromDeviceEx** (**csiHandle** moduleHandle, **csiHandle** localDevice, const **csiUploadParams** params, uint64_t timeoutMilliseconds, **csiMemTransferCallbackFunc** listener CSI_DEFAULT_PARAM_NULL, csiMemTransferUserData *userdata CSI_DEFAULT_PARAM_NULL)
Uploads a file from device to the local PC providing custom options.
CSI_DLL_EXPORT **csiErr csiReadMemory** (**csiHandle** moduleHandle, uint64_t address, char *buffer, size_t sizeBytes)
Read memory from a register address on the device.
CSI_DLL_EXPORT **csiErr csiWriteMemory** (**csiHandle** moduleHandle, uint64_t address, const char *buffer, size_t sizeBytes)

Write memory to a register address on the device.
CSI_DLL_EXPORT **csiErr csiIsModuleLittleEndian** (**csiHandle** moduleHandle, **csiModuleLevel** lvl, bool *isLittleEndian)
Write memory to a register address on the device.
CSI_DLL_EXPORT **csiErr csiGetErrorDescription** (**csiErr** error, char *bufferOut, size_t bufferSize)
Returns a human readable description of an error code.
CSI_DLL_EXPORT unsigned char **csiBitsPerPixelFromFormat** (const **csiPixelFormat** format)
Returns the number of bits per pixels for the given pixel format.
CSI_DLL_EXPORT **csiErr csiRegisterInvalidateCB** (**csiHandle** moduleHandle, const char *featureName, **CB_OBJECT** objCB, **CB_FEATURE_INVALIDATED_PFN** pfnFeatureInvalidateCB, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)
Register an invalidation callback function to a specific feature by name.
CSI_DLL_EXPORT **csiErr csiUnRegisterInvalidateCB** (**csiHandle** moduleHandle, const char *featureName, **csiModuleLevel** module CSI_DEFAULT_PARAM_MODULE)
Unregister an invalidation callback function from a specific feature.
CSI_DLL_EXPORT **csiErr csiGetLibraryVersion** (uint32_t *major, uint32_t *minor, uint32_t *patch, uint32_t *revision, uint32_t *build)
Returns the current library version.
CSI_DLL_EXPORT **csiErr csiSetFeatureCachingDisabled** (bool disableCaching)
disable feature caching
CSI_DLL_EXPORT **csiErr csiIsFeatureCachingDisabled** (bool *isDisabled)
retrieve if feature caching is disabled or not
CSI_DLL_EXPORT **csiErr csiGenerateAndUploadCalibrationData** (**csiHandle** devHandle, **csiCalibrationLUT** LUTSelector, **csiReferenceImgLoadMode** imgLoadMode, **csiCalibrationParams** calibParam, **csiNewBufferEventData** *refImage, const char *imgFile)
Generate and upload calibration file to the camera.
CSI_DLL_EXPORT **csiErr csiGenerateAndSaveCalibrationDataToFile** (**csiHandle** devHandle, **CalibrationMode** calibMode, **csiReferenceImgLoadMode** imgLoadMode, **csiCalibrationParams** calibParam, **csiNewBufferEventData** *refImage, const char *imgFile, const char *calibrationFile)
Generate and save calibration file to the PC memory in provided file path.

CSI_DLL_EXPORT **csiErr csiGenerateReferenceImage** (**csiHandle** devHandle, const **CalibrationMode** calibMode, **csiReferenceImgLoadMode** imgLoadMode, **csiCalibrationParams** calibParam, const **csiNewBufferEventData** *refImage, const char *imgFile, int *referenceImage, int *resMaxValue)
Generates calibration data and applies it on the input image to generate calibrated output image.

Macro Definition Documentation

#define CSI_COLOR_FORMAT 0x02000000
#define CSI_DISCOVERY_INFO_DEVICE_COUNT 16
#define CSI_INFITIE_TIME 0xffffffff
#define CSI_INFO_INT_BUFFER_SIZE 512
#define CSI_INFO_STRING_BUFFER_SIZE 512
#define CSI_MONO_FORMAT 0x01000000
#define CSI_OCCUPY_10BIT 0x000A0000
#define CSI_OCCUPY_12BIT 0x000C0000
#define CSI_OCCUPY_14BIT 0x000E0000
#define CSI_OCCUPY_16BIT 0x00100000
#define CSI_OCCUPY_24BIT 0x00180000
#define CSI_OCCUPY_30BIT 0x001E0000
#define CSI_OCCUPY_32BIT 0x00200000
#define CSI_OCCUPY_48BIT 0x00300000
#define CSI_OCCUPY_64BIT 0x00400000
#define CSI_OCCUPY_8BIT 0x00080000
#define CSI_TL_INTERFACE_COUNT 16

Typedef Documentation

typedef enum CalibrationMode CalibrationMode
typedef void(* CB_FEATURE_INVALIDATED_PFN) (const char *featureName, void *userdata)
typedef void* CB_OBJECT
typedef enum csiAccessMode csiAccessMode
typedef enum csiAcquisitionMode csiAcquisitionMode
typedef struct csiAcquistionStatistics csiAcquistionStatistics
typedef struct csiCalibrationParams csiCalibrationParams
typedef struct csiDataStreamInfo csiDataStreamInfo
typedef enum csiDeviceAccessMode csiDeviceAccessMode
typedef enum csiDeviceAccessStatus csiDeviceAccessStatus

typedef struct csiDeviceInfo csiDeviceInfo
typedef struct csiDiscoveryInfo csiDiscoveryInfo
typedef void(* csiDiscoveryInfoCallbackFunc) (const csiDiscoveryInfo *discoveryInfo)
typedef void(* csiDiscoveryTLInterfaceInfoCallbackFunc) (const csiTLInterfaceDiscoveryInfo *discoveryInfo)
typedef enum csiDisplayNotation csiDisplayNotation
typedef enum csiDownloadOptions csiDownloadOptions
typedef enum csiErr csiErr
typedef struct csiEventData csiEventData
typedef enum csiEventType csiEventType
typedef struct csiFeatureParameter csiFeatureParameter
typedef enum csiFeatureType csiFeatureType
typedef enum csiFeatureVisibility csiFeatureVisibility
typedef struct csiFileTransferParams csiFileTransferParams
typedef uint64_t csiHandle
typedef struct csiImageInfo csiImageInfo
typedef enum csiLogLevel csiLogLevel
typedef void(* csiLogSinkCallbackFunc) (csiLogLevel, const char *message, struct csiLogUserData *userdata)
typedef bool(* csiMemTransferCallbackFunc) (const csiMemTransferInfo *info, struct csiMemTransferUserData *userdata)
typedef struct csiMemTransferInfo csiMemTransferInfo
typedef enum csiMemTransferStatus csiMemTransferStatus
typedef enum csiModuleLevel csiModuleLevel
typedef struct CSI_DLL_EXPORT csiNewBufferEventData csiNewBufferEventData
typedef enum csiPixelFormat csiPixelFormat
typedef enum csiReferenceImgLoadMode csiReferenceImgLoadMode
typedef enum csiRepresentation csiRepresentation
typedef struct csiTLInterfaceDiscoveryInfo csiTLInterfaceDiscoveryInfo
typedef struct csiTLInterfaceInfo csiTLInterfaceInfo
typedef struct csiTLProducerInfos csiTLProducerInfos
typedef struct csiDownloadParams csiUploadDownloadUploadParams
typedef enum csiUploadOptions csiUploadOptions
typedef struct csiUploadParams csiUploadParams

Enumeration Type Documentation

enum CalibrationMode

Defines mode for calibration.

Enumerator:

DSNU_CALIBRATION	
PRNU_CALIBRATION	

enum csiAccessMode

Defines the access mode of a feature.

Enumerator:

CSI_ACCESS_UNKNOWN	Unknown access mode, feature might not be accessible
CSI_ACCESS_NOT_AVAILABLE	Feature is flagged as Not Available (NA). There are multiple reasons for which a feature might become not available. For example, because the XML description defines it. It might also be temporarily not available because of the value of another node.
CSI_ACCESS_READ_ONLY	Feature is read only.
CSI_ACCESS_READ_WRITE	Feature can be accessed in read and write mode.
CSI_ACCESS_WRITE_ONLY	Feature can only be written.

enum csiAcquisitionMode

Defines acquisition mode.

Enumerator:

CSI_ACQUISITION_SINGLE_FRAME	Acquire a single frame only
CSI_ACQUISITION_CONTINUOUS	Perform continuous frame acquisition

enum csiCalibrationLUT

Defines calibration LUT options.

Enumerator:

CSI_DSNU_LUT1	
CSI_DSNU_LUT2	
CSI_PRNU_LUT1	
CSI_PRNU_LUT2	

enum csiDeviceAccessMode

Defines the mode in which a device will be opened.

Enumerator:

CSI_DEV_MODE_UNKNOWN	Undefined access mode
CSI_DEV_MODE_NONE	No device access mode specified
CSI_DEV_MODE_EXCLUSIVE	The device will be opened exclusively; no other application will be allowed to open the device.
CSI_DEV_MODE_READ	The device will be opened in read only mode, other application might open it in read only mode too.
CSI_DEV_MODE_CONTROL	The device will be opened in control mode (read/write), other application might still be able to open it in read mode.

enum csiDeviceAccessStatus

Defines the current access status of a device as returned from device discovery.

Enumerator:

CSI_DEV_ACCESS_STATUS_UNKNOWN	Device is not yet open and can be opened in read/write mode.
CSI_DEV_ACCESS_STATUS_READWRITE	Device is not yet open and can be opened in read/write mode.
CSI_DEV_ACCESS_STATUS_READONLY	Device is not yet open and can be opened in read only mode.
CSI_DEV_ACCESS_STATUS_NOACCESS	Device is listed but cannot be opened.
CSI_DEV_ACCESS_STATUS_BUSY	Device is open by another process thus cannot be opened again.
CSI_DEV_ACCESS_STATUS_OPEN_READWRITE	Device already owned by this producer in read write mode.
CSI_DEV_ACCESS_STATUS_OPEN_READ	Device already owned by this producer in read only mode.

enum csiDisplayNotation

Defines the display notation for a floating-point feature.

Enumerator:

CSI_NOTATION_AUTOMATIC	Notation not specified, can be decided by the application
CSI_NOTATION_FIXED	Fixed notation
CSI_NOTATION_SCIENTIFIC	Scientific notation

enum csiDownloadOptions

Defines download options.

Enumerator:

CSI_DOWNLOAD_NO_OPTIONS	

enum csiErr

Defines possible error values.

Enumerator:

csiSuccess	No error
csiNotInitialized	System is not initialized, call **csiInit()** first
csiInvalidState	An invalid state occurred, see log output for more information
csiNotOpened	There was an action that requires the device / network / stream to be opened
csiNoImageDataAvailable	There was no image data available
csiNotFound	General error that the requested information was not found, see log for more detailed info on this error.
csiInvalidParameter	A parameter had an invalid value
csiNotAvailable	An expected result or a resource was not available
csiFunctionNotAvailable	The called function or a sub-function is not available

csiTimeout	A timeout occurred
csiAborted	A pending operation was aborted
csiFileOperation Failure	There was an error during file operation, see log for more information
csiFileOperation FatalError	There was a fatal error during file operation, see log for more information
csiNoAccess	Access denied (e.g., when trying to write a read only feature)
csiWrongBufferS ize	A given buffer was too small to store the requested data
csiInvalidBuffer	The requested buffer is not valid
csiResourceInUse	The requested resource is already in use by the transport layer
csiNotImplemen ted	A function that was called is not yet implemented
csiInvalidHandle	A handle passed as parameter is not valid
csiIOError	There was an error during an I/O operation (e.g. file or network)
csiParsingError	An error occurred when parsing an XML node (map file)
csiInvalidValue	A value that was passed parameter is not valid
csiResourceExha usted	A requested resource is exhausted (e.g. hard disk space)
csiOutOfMemory	Memory allocation failed, there is no more memory available
csiBusy	The requested operation cannot be executed because the system is busy
csiUnknown	Generic error, see log for more information
csiCustomErr	Custom error codes defined by specific transport layers

enum csiEventType

Defines event types that the user application can listen for.

Enumerator:

CSI_EVT_NEWIMAGEDATA	New image data event, can be registered on data stream module only
CSI_EVT_ERROR	Error event, can be registered on all module levels
CSI_EVT_MODULE	Generic module event, can be registered on all module levels
CSI_EVT_FEATURE_INVALIDATE	
CSI_EVT_FEATURE_CHANGE	
CSI_EVT_REMOTE_DEVICE	
CSI_EVT_CUSTOM	Custom user defined event types

enum csiFeatureType

Defines the data type of a feature.

Enumerator:

CSI_UNKNOWN_TYPE	Unknown type
CSI_BOOLEAN_TYPE	Boolean data type
CSI_INT_TYPE	Integer data type
CSI_FLOAT_TYPE	Floating point data type
CSI_STRING_TYPE	String data type
CSI_ENUMERATION_TYPE	Enumeration feature type
CSI_ENUMENTRY_TYPE	

CSI_CATEGORY	Category feature type
CSI_COMMAND	Command feature type
CSI_REGISTER	Register feature type
CSI_PORT	Port of the feature note map

enum csiFeatureVisibility

Defines the visibility of a feature depending on the role of a user.

Enumerator:

CSI_VISIBILITY_BEGINNER	Feature is visible to beginner users and higher
CSI_VISIBILITY_EXPERT	Feature is visible to expert users and higher
CSI_VISIBILITY_GURU	Feature is visible to guru users and higher
CSI_VISIBILITY_DEVELOPER	Feature is visible to developer users only
CSI_VISIBILITY_INVISIBLE	Feature is invisible to any user

enum csiLogLevel

Defines the severity of log messages coming from the SDK.

Enumerator:

CSI_LOGLEVEL_NONE	
CSI_LOGLEVEL_ERROR	
CSI_LOGLEVEL_WARN	
CSI_LOGLEVEL_INFO	
CSI_LOGLEVEL_DEBUG	
CSI_LOGLEVEL_TRACE	

enum csiMemTransferStatus

Defines the status of memory transfer functions as it is provided in the transfer callback.

Enumerator:

csiTransferStatusInit	Transfer was initialized
csiTransferStatusInProgress	Transfer is in progress
csiTransferStatusInProgressWaiting	Transfer process is waiting for response from device
csiTransferStatusFinishSucess	Transfer finished successfully
csiTransferStatusFinishError	Transfer finished with an error
csiTransferStatusCancelOnError	Transfer was canceled after an error occurred

enum csiModuleLevel

Defines the module level on which a specific action should be performed.

Enumerator:

CSI_UNKNOWN_MODULE	Unknown module level
CSI_TRANSPORTLAYER_MODULE	Transport layer module (System module)
CSI_INTERFACE_MODULE	Interface module
CSI_DEVICE_MODULE	Device module
CSI_LOCAL_DEVICE_MODULE	Local device module
CSI_STREAM_MODULE	Data stream module
CSI_BUFFER_MODULE	Buffer module

enum csiPixelFormat

Defines the currently supported pixel data formats.

Note

The value of each entry corresponds to its value in PFNC standard. Please refer to the PFNC standard for more information on each specific format: https://www.emva.org/standards-technology/genicam/genicam-downloads/.

Enumerator:

CSI_PIX_FORMAT_UNKNOWN	
CSI_PIX_FORMAT_MONO8	0x01080001 (PFNC_Mono8)
CSI_PIX_FORMAT_MONO10	0x01100003 (PFNC_Mono10)
CSI_PIX_FORMAT_MONO10_PACKED	0x010A0046 (PFNC_Mono10p)
CSI_PIX_FORMAT_MONO12	0x01100005 (PFNC_Mono12)
CSI_PIX_FORMAT_MONO12_PACKED	0x010C0047 (PFNC_Mono12p)
CSI_PIX_FORMAT_MONO16	0x01100007 (PFNC_Mono16)
CSI_PIX_FORMAT_RGB8	0x02180014 (PFNC_RGB8)
CSI_PIX_FORMAT_RGB10_PACKED	0x0220001D (PFNC_RGB10p32
CSI_PIX_FORMAT_RGB10	0x02300018 (PFNC_RGB10)
CSI_PIX_FORMAT_BGR10	0x02300019 (PFNC_BGR10)
CSI_PIX_FORMAT_RGB12	0x0230001A (PFNC_RGB12)
CSI_PIX_FORMAT_BGR12	0x0230001B (PFNC_BGR12)
CSI_PIX_FORMAT_RGBA8	0x02200016 (PFNC_RGBa8)
CSI_PIX_FORMAT_BGRA8	0x02200017 (PFNC_BGRa8)
CSI_PIX_FORMAT_BGR8	0x02180015 (PFNC_BGR8)
CSI_PIX_FORMAT_RGB16	0x02300033 (PFNC_RGB16)
CSI_PIX_FORMAT_RGBA10	0x0240005F (PFNC_RGBa10)
CSI_PIX_FORMAT_RGBA12	0x02400061 (PFNC_RGBa12)
CSI_PIX_FORMAT_RGBA16	0x02400064 (PFNC_RGBa16)
CSI_PIX_FORMAT_BayerGR8	0x01080008 (PFNC_BayerGR8)
CSI_PIX_FORMAT_BayerRG8	0x01080009 (PFNC_BayerRG8)
CSI_PIX_FORMAT_BayerGB8	0x0108000A (PFNC_BayerGB8)
CSI_PIX_FORMAT_BayerBG8	0x0108000B (PFNC_BayerBG8)
CSI_PIX_FORMAT_BayerGR12	0x01100010 (PFNC_BayerGR12)

CSI_PIX_FORMAT_BayerRG12	0x01100011 (PFNC_BayerRG12)
CSI_PIX_FORMAT_BayerGB12	0x01100012 (PFNC_BayerGB12)
CSI_PIX_FORMAT_BayerBG12	0x01100013 (PFNC_BayerBG12)

enum csiReferenceImgLoadMode

Defines mode of loading reference imgage.

Enumerator:

CSI_LOAD_REF_IMG_FROM_DISC	
CSI_ACQUIRE_REF_IMG_FROM_CAMERA	

enum csiRepresentation

Defines how a feature value should be represented when printed in UI.

Enumerator:

CSI_REPRESENTATION_LINEAR	Linear representation (default)
CSI_REPRESENTATION_LOGARITHMIC	Logarithmic representation
CSI_REPRESENTATION_BOOLEAN	Boolean representation (true / false)
CSI_REPRESENTATION_PURENUMBER	Represent as pure number
CSI_REPRESENTATION_HEX	Hexadecimal representation (0x...)
CSI_REPRESENTATION_IP	IP address representation
CSI_REPRESENTATION_MAC	Mac address representation
CSI_REPRESENTATION_UNDEFINED	Not defined, use default

enum csiUploadOptions

Defines upload options.

Enumerator:

CSI_UPLOAD_NO_OPTIONS	
CSI_UPLOAD_IGNORE_CHECKSUM	

Function Documentation

CSI_DLL_EXPORT csiErr csiAbortAcquisition (csiHandle *deviceHandle*)

Aborts the Acquisition on the device immediately.

Parameters

in	*deviceHandle*	provided by the csiOpenDevice-function.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

When this function is executed, it aborts the current frame immediately without completing the current frame.

CSI_DLL_EXPORT unsigned char csiBitsPerPixelFromFormat (const csiPixelFormat *format*)

Returns the number of bits per pixels for the given pixel format.

Parameters

in	*format*	The pixel format to get the bits per pixel for.

Returns

The number of bits per pixels for the given pixel format or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiCheckAndReallocBuffers (csiHandle *deviceHandle*)

Reallocating buffers if the size of allocated buffers is diffrent from the require size.

Parameters

in	*deviceHandle*	Handle provided by the csiOpenDevice-function.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiClose ()

Closes the SDK and frees all allocated memory and interfaces.

Returns

Returns csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiCloseDataStream (csiHandle *dataStream*)

Close the data stream.

Parameters

in	*dataStream*	A handle to the data stream to be closed.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

Make sure to release all used buffers with **csiReleaseImage()** and unregister all events with **csiUnregisterEvent()** before closing the data stream. Any buffer or event will be invalid after a call to this function. In addition, acquisition must be stopped before calling this function. See **csiStopAcquisition()**.

CSI_DLL_EXPORT csiErr csiCloseDevice (csiHandle *deviceHandle*)

Close the connection to the specific device.

Parameters

in	*deviceHandle*	Handle provided by the csiOpenDevice-function.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

To grant access to the device for other applications, the connection should be closed when it is not needed anymore. The API will cleanup no longer needed memory when this command is executed.

CSI_DLL_EXPORT csiErr csiCloseTLInterface (csiHandle *interfaceHandle*)

Close the connection to the specific interface.

Parameters

in	*interfaceHandle*	Handle provided by the csiOpenTLInterface function.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

To grant access to the underlying interface for other applications, the connection should be closed when it is not needed anymore. The API will cleanup no longer needed memory when this command is executed.

CSI_DLL_EXPORT csiErr csiCreateDataStream (csiHandle *deviceHandle*, uint32_t *dsIndex*, csiHandle * *dataStreamOut*, uint32_t *numberOfBuffers*, size_t bufferSize *CSI_DEFAULT_PARAM_ZERO*)

Create a data stream to receive images.

Parameters

in	*deviceHandle*	provided by the csiOpenDevice-function.
in	*dsIndex*	An index to a data stream. Starting from 0 to the number of available data streams as returned by **csiGetNumberOfDataStreams()**.
out	*dataStreamOut*	A handle to the data stream that was created.
in	*numberOfBuffers*	The number of buffers to be allocated for the created data stream. This number must be at least 1, recommended is >= 3.
in	*bufferSize*	Size of one buffer in bytes. This parameter can be 0 in which case the size of a buffer will be determined by the standard 'PayloadSize' feature of a device.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

Use this function in combination with **csiGetNumberOfDataStreams()** to get the total number of available data streams in the camera.

CSI_DLL_EXPORT csiErr csiDiscoverDevices (csiDiscoveryInfo * *discoveryInfoOut*, uint64_t *timeoutMilliseconds*, csiDiscoveryInfoCallbackFunc discCallbackFunc *CSI_DEFAULT_PARAM_NULL*, const char *additionalSearchPaths *CSI_DEFAULT_PARAM_NULL*, bool overrideSearchPath *CSI_DEFAULT_PARAM_FALSE*)

This function will look for attached GenICAM-devices on the available transport layers.

Parameters

out	*discoveryInfoOut*	pointer to a structure which will contain the information about the found devices. The information is the same provided to the callback function
in	*timeoutMilliseconds*	The time until when a response from a device needs to be received when doing a discovery
in	*discCallbackFunc*	Pointer to a callback function which receives information about the discovery progress. The callback function receives the current progress in %, number and names of the found devices Also a flag if the discovery is running is provided.
in	*additionalSearchPaths*	You can specify additional paths to search for transport layers. If you want to specify multiple paths, you need to divide the paths by using a ; *-sign*
in	*overrideSearchPath*	If this flag is set, only the path(s) provided in *additionalSearchPaths will* be searched for the cti - files to load

Returns

Returns csiSuccess or an error defined in the csiErr-Enum.

Note

By default, this function tries to use all available transport layers in the system. The search paths for the cti-files are set in the environmental variable GENICAM_GENTL64_PATH(64 bit) or GENICAM_GENTL32_PATH(32 bit applications)

CSI_DLL_EXPORT csiErr csiDiscoverTLInterfaces (csiTLInterfaceDiscoveryInfo * *discoveryInfoOut*, uint64_t *timeoutMilliseconds*, csiDiscoveryTLInterfaceInfoCallbackFunc discCallbackFunc *CSI_DEFAULT_PARAM_NULL*, const char *additionalSearchPaths *CSI_DEFAULT_PARAM_NULL*, bool overrideSearchPath *CSI_DEFAULT_PARAM_FALSE*)

This function will look for TL interfaces provided by the available transport layers.

Parameters

in	*timeoutMilliseconds*	The time until when a response from a transport layer needs to be received when doing a discovery call.
in	*additionalSearchPaths*	This flag indicates if only the paths given by the application or environment variable should be searched.
in	*overrideSearchPath*	This flag (if set) indicates that only the path(s) provided in 'additionalSearchPaths' will be searched for the cti-files to find.
out	*discoveryInfoOut*	Pointer to a structure which will contain the information about the found devices. The information is the same provided to the callback function.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

By default, this function tries to use all available transport layers in the system. The search paths for the cti-files are set in the environmental variable GENICAM_GENTL64_PATH (64 bit) or GENICAM_GENTL32_PATH (32 bit applications).

CSI_DLL_EXPORT csiErr csiEventKill (csiHandle *evt*)

Cancel all waiting functions related to this event.

Parameters

in	*evt*	The handle of the event to be canceled.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

Any pending call to **csiWaitForEvent()** or **csiGetNextImage()** on this event will be canceled.

CSI_DLL_EXPORT csiErr csiExecuteCommand (csiHandle *moduleHandle*, const char * *featureName*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Execute a command on the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the feature to execute.
in	*module*	Module for which the parameter should be executed. Please use the enum csiModuleLevel to select. Determines if the parameter should be executed on the device-, transport layer-, interface- stream- or buffer-module.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

The function will return immediately. Even if the triggered function is still active. To check if the command is still running, please use the function "csiIsCommandActive".

CSI_DLL_EXPORT csiErr csiFileDownloadToDevice (csiHandle *moduleHandle*, csiHandle *localDevice*, const char * *fileName*, const char * *fileType*, uint64_t *timeoutMilliseconds*, csiMemTransferCallbackFunc listener *CSI_DEFAULT_PARAM_NULL*, csiMemTransferUserData *userdata *CSI_DEFAULT_PARAM_NULL*)

Downloads a file located on the local PC to the camera.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice function.
in	*localDevice*	Handle of the local device module (usually the same handle value of device)
in	*fileName*	Path of the file to be uploaded.
in	*fileType*	The type of file to be uploaded (as returned from csiGetUpdateFileType).
in	*timeoutMilliseconds*	Maximum time allowed for the file upload operation.
in	*listener*	Optional callback function that can be called during the upload process to inform about the progress.
in	*userdata*	Optional user data that will be passed as parameter to the progress callback function.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

An update process might take several minutes depending on the type of file, please choose a timeout of at least a few minutes here.

CSI_DLL_EXPORT csiErr csiFileDownloadToDeviceEx (csiHandle *moduleHandle*, csiHandle *localDevice*, const csiDownloadParams *params*, uint64_t *timeoutMilliseconds*, csiMemTransferCallbackFunc listener *CSI_DEFAULT_PARAM_NULL*, csiMemTransferUserData *userdata *CSI_DEFAULT_PARAM_NULL*)

Downloads a file located on the local PC to the camera providing custom options.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice function.
in	*localDevice*	Handle of the local device module (usually the same handle value of device)
in	*params*	File related parameter and options, see **csiDownloadParams**.
in	*timeoutMilliseconds*	Timeout for the update process in milliseconds.
in	*listener*	Optional callback function that will be called during the update process to inform about the progress.
in	*userdata*	Optional user data that will be passed as parameter to the progress callback function.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

An update process might take several minutes depending on the type of file, please choose a timeout of at least a few minutes here.

CSI_DLL_EXPORT csiErr csiFileUploadFromDevice (csiHandle *moduleHandle*, csiHandle *localDevice*, const char * *fileName*, const char * *fileType*, uint64_t *timeoutMilliseconds*, csiMemTransferCallbackFunc listener *CSI_DEFAULT_PARAM_NULL*, csiMemTransferUserData *userdata *CSI_DEFAULT_PARAM_NULL*)

Uploads a file from device to the local PC.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice function.
in	*localDevice*	Handle of the local device module (usually the same handle value of device)
in	*fileName*	Name of the file on the local PC.
in	*fileType*	The type of file to be uploaded (as returned from csiGetUpdateFileType).
in	*timeoutMilliseconds*	Maximum time allowed for the file upload operation.
in	*listener*	Optional callback function that can be called during the transfer process to inform about the progress.
in	*userdata*	Optional user data that will be passed as parameter to the progress callback function.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

A file transfer process might take several minutes depending on the type of file, please choose a timeout of at least a few minutes here.

CSI_DLL_EXPORT csiErr csiFileUploadFromDeviceEx (csiHandle *moduleHandle*, csiHandle *localDevice*, const csiUploadParams *params*, uint64_t *timeoutMilliseconds*, csiMemTransferCallbackFunc listener *CSI_DEFAULT_PARAM_NULL*, csiMemTransferUserData *userdata *CSI_DEFAULT_PARAM_NULL*)

Uploads a file from device to the local PC providing custom options.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice function.
in	*localDevice*	Handle of the local device module (usually the same handle value of device)
in	*params*	File related parameter and options, see **csiUploadParams**.
in	*timeoutMilliseconds*	Timeout for the upload procedure in milliseconds.
in	*listener*	Optional callback function that will be called during the transfer process to inform about the progress.
in	*userdata*	Optional user data that will be passed as parameter to the progress callback function.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

A file transfer process might take several minutes depending on the type of file, please choose a timeout of at least a few minutes here.

CSI_DLL_EXPORT csiErr csiGenerateAndSaveCalibrationDataToFile (csiHandle *devHandle*, CalibrationMode *calibMode*, csiReferenceImgLoadMode *imgLoadMode*, csiCalibrationParams *calibParam*, csiNewBufferEventData * *refImage*, const char * *imgFile*, const char * *calibrationFile*)

Generate and save calibration file to the PC memory in provided file path.

Parameters

in	*devHandle*	Handle provided by the csiOpenDevice function.
in	*calibMode*	The desired calibration mode. Either DSNU or PRNU.
in	*imgLoadMode*	Specifying the source of the input image used to generate calibration data.
in	*calibParam*	The input parameters can be loaded from PC memory.
in	*refImage*	If the input image is captured from the camera, it can be loaded in this argument.
in	*imgFile*	If the input image is loaded from the PC, this can be a *null* .
in	*calibrationFile*	The absolute path of the output calibration file, where the file is to be saved. The appropriate file type of "*.dslr" or "*.prnu" should be mentioned.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

In this function, even if input data is provided to both arguments of type "csiNewBufferEventData" and imageFilePath, the argument "imgLoadMode" of type "csiReferencingLoadMode" determines which input image data should be used for calibration generation.

CSI_DLL_EXPORT csiErr csiGenerateAndUploadCalibrationData (csiHandle *devHandle*, csiCalibrationLUT *LUTSelector*, csiReferenceImgLoadMode *imgLoadMode*, csiCalibrationParams *calibParam*, csiNewBufferEventData * *refImage*, const char * *imgFile*)

Generate and upload calibration file to the camera.

Parameters

in	*devHandle*	Handle provided by the csiOpenDevice function.
in	*LUTSelector*	The desired LUT or an area in memory where the file is to be uploaded.
in	*imgLoadMode*	Specifying the source of the input image used to generate calibration data.
in	*calibParam*	The input parameters can be loaded from PC memory.
in	*refImage*	If the input image is captured from the camera, then it can be loaded in this argument.
in	*imgFile*	If the input image is loaded from the PC, this can be used for calibration.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

In this function, even if input data is provided to both arguments of type "csiNewBufferEventData" and imageFilePath, the argument "imgLoadMode" determines which input image data should be used for calibration generation.

CSI_DLL_EXPORT csiErr csiGenerateReferenceImage (csiHandle *devHandle*, const CalibrationMode *calibMode*, csiReferenceImgLoadMode *imgLoadMode*, csiCalibrationParams *calibParam*, const csiNewBufferEventData * *refImage*, const char * *imgFile*, int * *referenceImage*, int * *resMaxValue*)

Generates calibration data and applies it on the input image to generate calibrated output image.

Parameters

in	*devHandle*	Handle provided by the csiOpenDevice function.
in	*calibMode*	The desired calibration mode. Either DSNU or PRNU.
in	*imgLoadMode*	Specifying the source of the input image used to generate calibration data.
in	*calibParam*	The input parameters can be loaded from PC memory.
in	*refImage*	If the input image is captured from the camera, it can be loaded in this argument.
in	*imgFile*	If the input image is loaded from the PC, this can be a *null* .
out	*referenceImage*	The calibration data is applied on the input image and generates calibrated image.
out	*resMaxValue*	The resultant raw value of a pixel present in the calibrated output image.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

In this function, even if input data is provided to both arguments of type "csiNewBufferEventData" and imageFilePath, the argument "imgLoadMode" of type "csiReferencingLoadMode" determines which input image data should be used for calibration generation.

CSI_DLL_EXPORT csiErr csiGetAcquisitionStatistics (csiHandle *dataStream*, csiAcquistionStatistics * *stats*)

Retrieve the statistical buffer regarding the data stream (e.g. transmitted frames, etc.).

Parameters

in	*dataStream*	Handle to the data stream the acquisition statistics should be collected on.
out	*stats*	The acquisition statistics, see Fehler! Verweisquelle konnte nicht gefunden werden.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiGetDataStreamInfo (csiHandle *deviceHandle*, uint32_t *dsIndex*, csiDataStreamInfo * *dataStreamInfoOut*)

Return information about the desired data stream.

Parameters

in	*deviceHandle*	provided by the csiOpenDevice-function.
in	*dsIndex*	An index to a data stream. Starting from 0 to the number of available data streams as returned by **csiGetNumberOfDataStreams()**.
out	*dataStreamInfoOut*	Information about the selected data stream given by the dsIndex parameter or NULL if the data stream is not found. See documentation on **csiDataStreamInfo** for more information.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiGetDeviceDataStreamInfo (csiHandle *moduleHandle*, uint32_t *dsIndex*, csiDataStreamInfo * *dataStreamInfoOut*)

Return information about the desired data stream and device.

Parameters

in	*moduleHandle*	provided by the csiOpenDevice-function.
in	*dsIndex*	An index to a data stream. Starting from 0 to the number of available data streams as returned by **csiGetNumberOfDataStreams()**.
out	*dataStreamInfoOut*	Information about the selected data stream given by the dsIndex parameter and moduleHandle or NULL if the data stream is not found. See documentation on **csiDataStreamInfo** for more information.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiGetDeviceInfo (uint32_t *deviceIndex*, csiDeviceInfo * *deviceInfoOut*)

A function to get information about the found devices in the system.

Parameters

in	*deviceIndex*	Index of the found device from the csiDiscoverDevices-function
out	*deviceInfoOut*	Detailed information of the found device. The information will be provided in this structure. The structure must be allocated on the caller side.

Returns

Returns csiSuccess or an error defined in the csiErr-Enum

Note

This function provides in more detailed information about the found devices such as device identifier, name, model, vendor, serial number, interface description, interface-ID, username, version, consistency of camera package, TL-Producer-information, access status

CSI_DLL_EXPORT csiErr csiGetErrorDescription (csiErr *error*, char * *bufferOut*, size_t *bufferSize*)

Returns a human readable description of an error code.

Parameters

in	*error*	The error code to retrieve the text for.
in	*bufferSize*	The size of the provided text buffer.
out	*bufferOut*	char-buffer where the error text will be written to.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiGetFeatureAccessMode (csiHandle *moduleH*, const char * *featureName*, csiAccessMode * *access*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Retrieve a specific feature from the device. The access mode of this feature will be returned.

Parameters

in	*moduleH*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the feature to get.
in	*module*	Module for which the parameter should be set. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface-stream- or buffer-module.
out	*access*	Structure which contains the value of acess mode about the requested feature.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiGetFeatureBool (csiHandle *moduleHandle*, const char * *featureName*, bool * *valueOut*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Retrieve a boolean feature from the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name of the feature to get (Not the display name).
in	*module*	Module for which the parameter should be get. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface-stream- or buffer-module.
out	*valueOut*	Pointer to a bool-value where the current value of the feature will be written to.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiGetFeatureEnum (csiHandle *moduleHandle*, const char * *featureName*, csiFeatureParameter * *featureParamOut*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Retrieve an enumeration feature from the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the enumeration to get.
in	*module*	Module for which the parameter should be set. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface-stream- or buffer-module.
out	*featureParamOut*	Structure which contains all necessary information about the requested feature:

enumEntries: The number of enumeration entries which are available for this enumeration feature.

currentEntry: Currently selected enumeration index.
values []: Name of the enumeration entries.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiGetFeatureEnumEntryByIndex (csiHandle *moduleHandle*, const char * *featureName*, int32_t *enumIndex*, csiFeatureParameter * *featureParamOut*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Retrieve an enumeration feature from the device by using its index.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the enumeration to get.
in	*enumIndex*	Index of the enumeration entry to be retrieved.
in	*module*	Module for which the parameter should be retrieved. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module.
out	*featureParamOut*	Name of the enumeration of the requested index.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

The enumeration string will be given in the **csiFeatureParameter** structure: valueStr.

CSI_DLL_EXPORT csiErr csiGetFeatureEnumEntryByName (csiHandle *moduleHandle*, const char * *featureName*, const char * *enumValue*, csiFeatureParameter * *featureParamOut*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Retrieve an enumeration feature from the device by using its name.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*parameterName*	Name (Not the display name) of the enumeration to get.
in	*enumValue*	Name of the enumeration entry to be retrieved.
in	*module*	Module for which the parameter should be retrieved. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module.
out	*featureParamOut*	Structure which contains all necessary information about the requested feature.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiGetFeatureEnumEntryCount (csiHandle *moduleHandle*, const char * *featureName*, uint32_t * *count*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Retrieve the entries size of an enumeration feature from the device by feature name.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the enumeration to get.
in	*module*	Module for which the parameter should be retrieved. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module.
out	*count*	Entries size of the enumeration feature.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiGetFeatureFloat (csiHandle *moduleHandle*, const char * *featureName*, double * *valueOut*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Retrieve a floating point feature from the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name of the feature to get (Not the display name).
in	*module*	Module for which the parameter should be get. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module.
out	*valueOut*	Pointer to a double-value where the current value of the feature will be written to.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiGetFeatureInt (csiHandle *moduleHandle*, const char * *featureName*, int64_t * *valueOut*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Retrieve an integer feature from the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name of the feature to get (Not the display name).
in	*module*	Module for which the parameter should be get. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface-stream- or buffer-module.
out	*valueOut*	Pointer to an int64_t value where the current value of the feature will be written to.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiGetFeatureParameter (csiHandle *moduleHandle*, const char * *featureName*, csiFeatureParameter * *featureParamOut*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Retrieve a specific feature from the device. Detailed information about this feature will be returned.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the feature to get.
in	*module*	Module for which the parameter should be set. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface-stream- or buffer-module.
out	*featureParamOut*	Structure which contains all necessary information about the requested feature.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiGetFeatureReg (csiHandle *moduleHandle*, const char * *featureName*, char * *buffer*, size_t * *length*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Retrieve a register value from the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the feature to get.
in	*module*	Module for which the parameter should be get. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface-stream- or buffer-module.
out	*buffer*	Pointer to a char-array where the current value of the feature will be written to.
out	*length*	Current length of the retrieved data.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

To avoid unexpected behavior, it is recommended to retrieve the maximum buffer length before getting it from the device. This can be achieved by using the function "csiGetFeatureParameter". This function will provide all necessary information about the parameter (including min and max values). The register length must not exceed the length given in the "featureRegLength" parameter.

CSI_DLL_EXPORT csiErr csiGetFeatureString (csiHandle *moduleHandle*, const char * *featureName*, char * *valueOut*, size_t * *sizeOut*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Retrieve a string feature from the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the feature to get.
in	*module*	Module for which the parameter should be get. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface-stream- or buffer-module.
out	*valueOut*	Pointer to a char-value where the current value of the feature will be written to.
out	*sizeOut*	Size of the read string.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

To avoid unexpected behavior, you should first retrieve the length of the string to be received:
1. Call the function with valueOut set to NULL. The function will return the current size of the string parameter.
2. Allocate the required memory to store the string feature to be set.

3. Call the function as described by providing a pointer to a string buffer with the sufficient length.

CSI_DLL_EXPORT csiErr csiGetLibraryVersion (uint32_t * *major*, uint32_t * *minor*, uint32_t * *patch*, uint32_t * *revision*, uint32_t * *build*)

Returns the current library version.

Parameters

out	*major*	The major version number.
out	*minor*	The minor version number.
out	*patch*	The patch version number.
out	*revision*	The revision number.
out	*build*	The build number.

Returns

Always returns csiSuccess.

CSI_DLL_EXPORT csiErr csiGetNextImage (csiHandle *eventHandle*, csiNewBufferEventData ** *bufferInfoOut*, uint64_t *timeoutMilliseconds*)

Get the next image from the data stream.

Parameters

in	*eventHandle*	The handle of the data stream to wait for images on. Requires a previous call to **csiRegisterEvent()** to register for new image events on this stream.
in	*timeoutMilliseconds*	Timeout after the waiting stops if no event was received.
out	*bufferInfoOut*	The event data structure containing the image data and information.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

This is a convenience function that is recommended to use for image acquisition. It is an alternative to **csiWaitForEvent()** that returns a more general representation of the event data. Please note that a CSI_EVT_NEWIMAGEDATA event must be registered on the data stream to be able to wait for new images. The image buffer returned from this function will be valid and usable as long as it is not given back to the acquisition engine with **csiReleaseImage()**.

CSI_DLL_EXPORT csiErr csiGetNumberOfDataStreams (csiHandle *deviceHandle*, uint32_t * *numberOfStreamsOut*)

Return the available number of data streams of the device.

Parameters

in	*deviceHandle*	provided by the csiOpenDevice-function.
out	*numberOfStreamsOut*	Number of available data streams for the given device.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

The returned number of data streams can be 0 if the given device is not a streaming device. The actual number of available data streams depends on the capabilities of the device. Use this function in combination with **csiGetDataStreamInfo()** which receives an index to a data stream as parameter.

CSI_DLL_EXPORT csiErr csiGetNumberOfTLProducers (int32_t * *numTLProducers*)

Returns the number of available transport layers in the system.

Parameters

out	*numTLProducers*	The number of transport layers detected in the environment.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

The SDK uses the environment variable GENICAM_GENTL64_PATH to search for available transport layers. This function allows to request the number of transport layers available through that environment variable.

CSI_DLL_EXPORT csiErr csiGetTLProducerInfobyFilePath (csiTLProducerInfos * *tlProducerInfos*, const char * *producerName*)

Returns additional information about a transport layer.

Parameters

in	*producerName*	The name of the producer (usually the file path to the producer *.cti file).
out	*tlProducerInfos*	The structure holding additional information about the transport layer.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

Similar to csiGetTLProducerInfoByIndex but using the name (file path) of the producer.

CSI_DLL_EXPORT csiErr csiGetTLProducerPathByIndex (char * *transportLayerPath*, size_t *bufferSize*, uint32_t *index*)

Returns a path of transport layer producer specified by index.

Parameters

in	*bufferSize*	Size of one buffer in bytes
in	*index*	index of transport layer
out	*transportLayerPath*	path to the transport layer.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiGetUpdateFileType (csiHandle *moduleHandle*, const char * *fileName*, char * *fileTypeOut*, size_t *bufferSize*)

Request the update file type of a file (if available).

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice function.
in	*fileName*	The path to the file that should be checked.
in	*bufferSize*	The size of the output buffer.
out	*fileTypeOut*	If the file is a valid file that can be used to update on the device, this buffer should contain the type of the file as string representation.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

This function can be used to request the type of a specific file. There are multiple different types of files that can be uploaded to a device (e.g. firmware files, script files, etc.). To make sure that the file is a valid file type, this can be used to detect if a file can be used for the intended purpose. It is required first to get the type of a file before uploading it to the device to see if it is a valid file.

CSI_DLL_EXPORT csiErr csiInit (csiLogLevel *logLvl*, csiLogSinkCallbackFunc logCallbackFunc *CSI_DEFAULT_PARAM_NULL*, csiLogUserData *userdata *CSI_DEFAULT_PARAM_NULL*)

Initializes the SDK. Needs to be called first before any other function of the SDK is called!

Parameters

in	*logLvl*	Defines the loglevel for the SDK. This will enable a closer debugging of the SDK. Use the enum csiLogLevel for setting the desired loglevel
in	*logCallbackFunc*	An optional callback function for log messages coming from the SDK
in	*userdata*	Optional user data that will be passed as parameter when the log callback function is called.

Returns

Returns csiSuccess or an error defined in the csiErr-Enum

Note

After the usage of the SDK make sure to call csiClose in order to free all memory again and not leaving any interfaces open.

CSI_DLL_EXPORT csiErr csiIsCommandActive (csiHandle *moduleHandle*, const char * *featureName*, bool * *isActive*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Check if a command is still active.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the command to check.
in	*module*	Module for which the parameter should be set. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface-stream- or buffer-module.
out	*isActive*	Pointer to a bool-value where the current state of the command is written to (true: active, false: inactive).

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

If a lengthy operation is triggered, it is possible to check the current status by calling this command.

CSI_DLL_EXPORT csiErr csiIsFeatureCachingDisabled (bool * *isDisabled*)

retrieve if feature caching is disabled or not

Parameters

out	*isDisabled*	value indicates that if feature caching is disabled or not

Returns

Always returns csiSuccess.

CSI_DLL_EXPORT csiErr csiIsModuleLittleEndian (csiHandle *moduleHandle*, csiModuleLevel *lvl*, bool * *isLittleEndian*)

Write memory to a register address on the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice-function.
in	*lvl*	Module for which the parameter should be retrieved. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module.
out	*isLittleEndian*	indicates if module specified by moduleHandle is little endian or not.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiIterateFeatureTree (csiHandle *moduleHandle*, const char * *rootFeatureName*, uint32_t *index*, char * *featureNameOut*, size_t *nameBuffSize*, csiFeatureType * *type*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Provides a possibility to iterate through all available features on the camera.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*rootFeatureName*	Name of the feature to start from. To start from the very beginning use "root".
in	*index*	This will indicate the number of the element of RootFeatureName to retrieve.
in	*nameBuffSize*	size of the provided buffer for the featureNameOut
in	*module*	Module for which the feature tree should be iterated. Please use the enum csiModuleLevel to select. Determines if the feature tree on the device-, transport layer-, interface- stream- or buffer-module should be used.
out	*featureNameOut*	Name of the retrieved feature.
out	*type*	Type of the retrieved feature.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

If starting from the very beginning, use "Root" as RootFeatureName. From there call this function for each returned feature in order to get all features of the device. Please check the provided example "feature_iteration" for a template of usage.

CSI_DLL_EXPORT csiErr csiOpenDevice (const char * *deviceIdentifier*, const char * *interfaceID*, csiHandle * *deviceHandleOut*, uint64_t *timeoutMilliseconds*, csiDeviceAccessMode *openMode*)

Open the device given by the index. The TL of this index is used.

Parameters

in	*deviceIdentifier*	Index of the found device from the csiDiscoverDevices-function.
in	*interfaceID*	Interface ID of the device. This information can be found in device-info struct.
in	*timeoutMilliseconds*	Timeout in milliseconds until the device needs to be opened successfully.
in	*openMode*	The device can be opened in different modes to enable/hinder concurrent access to the device. The following modes might be used:

CSI_DEV_MODE_EXCLUSIVE: Only this process can communicate with the camera.
CSI_DEV_MODE_READ: Camera-parameters can be read and images can be acquired.
CSI_DEV_MODE_CONTROL: Camera-parameters can be read and written. Read-access by another process to the device is still possible.

out	*deviceHandleOut*	Handle to the device. This handle needs to be used to any successive call.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

Depending on the used transport layer it might be necessary to use a longer timeout. Please refer to the information provided with the specific TL.

CSI_DLL_EXPORT csiErr csiOpenTLInterface (csiTLInterfaceInfo *interfaceInfo*, csiHandle * *interfaceHandleOut*, uint64_t *timeoutMilliseconds*)

Open the interface given by the interface information.

Parameters

in	*interfaceInfo*	Interface information of the found TL interface from the discovery function.
in	*timeoutMilliseconds*	Timeout in milliseconds until the interface needs to be opened successfully.
out	*interfaceHandleOut*	Handle to the interface. This handle needs to be used to any successive call for the interface access.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

Depending on the used transport layer it might be necessary to use a longer timeout. Please refer to the information provided with the specific TL.

CSI_DLL_EXPORT csiErr csiReadMemory (csiHandle *moduleHandle*, uint64_t *address*, char * *buffer*, size_t *sizeBytes*)

Read memory from a register address on the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice function.
in	*address*	The register address to read from.
out	*buffer*	The buffer into which the data should be stored.
in	*sizeBytes*	The size of the buffer into which the buffer should be read and at the same time the number of bytes to read from the address.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiRegisterEvent (csiHandle *moduleHandle*, csiEventType *evtType*, csiHandle * *eventOut*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Register an event which will be signaled in the case the desired event is triggered.

Parameters

in	*moduleHandle*	Handle to the module that is used to register an event. This can be either a device handle or a data stream handle.
in	*evtType*	The type of event that should be registered.
in	*module*	The module level where the event should be registered on. Please use the enum csiModuleLevel to select.
out	*eventOut*	A handle to the event that was registered. Use this handle to wait for events using the **csiWaitForEvent()** or csiGetWaitImage() functions.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiRegisterInvalidateCB (csiHandle *moduleHandle*, const char * *featureName*, CB_OBJECT *objCB*, CB_FEATURE_INVALIDATED_PFN *pfnFeatureInvalidateCB*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Register an invalidation callback function to a specific feature by name.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name of the feature for which the invalidation callback should be registered.
in	*An*	user object that is passed as parameter to the callback function.
in	*pfnFeatureInvalidateCB*	The callback function.
in	*module*	Module for which the invalidation callback should be registered. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer_module

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

This function is useful to get informed about changes in the feature tree which lead to an invalidation of features. Whenever a feature changes its value or another attribute, the application should get informed about it. The callback function must be of the form: void featureInvalidated(const char *featureName, void* userObj)

CSI_DLL_EXPORT csiErr csiReleaseImage (csiHandle *dataStream*, const csiNewBufferEventData * *bufferInfo*)

Release the image back into the processing buffer from the device.

Parameters

in	*dataStream*	The handle to the data stream this buffer belongs to. This handle is also part of the bufferInfo structure.
in	*bufferInfo*	Pointer to the buffer event data that was previously received from **csiWaitForEvent()** or **csiGetNextImage()**.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

Releasing an image buffer from the user application back to the transport layer for acquisition. Releasing the image buffer passes the ownership of the buffer back to the transport layer and the user application is not allowed to use it anymore after the release. To ensure a fluent image acquisition it is recommended to keep the buffer ownership as short as possible and release the buffer as soon as it is not needed anymore.

CSI_DLL_EXPORT csiErr csiReset ()

Reset the SDK and frees all allocated memory and interfaces.

Returns

Returns csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiSetFeatureBool (csiHandle *moduleHandle*, const char * *featureName*, bool *value*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Set a boolean feature on the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the feature to set.
in	*value*	Boolean value to set the feature to (true or false).
in	*module*	Module for which the parameter should be set. Please use the enum csiModuleLevel to select. Determines if the parameter should be set on the device-, transport layer-, interface-stream- or buffer-module.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiSetFeatureCachingDisabled (bool *disableCaching*)

disable feature caching

Parameters

in	*disableCaching*	value indicates that if feature caching should be disabled or not

Returns

Always returns csiSuccess.

CSI_DLL_EXPORT csiErr csiSetFeatureEnum (csiHandle *moduleHandle*, const char * *featureName*, const char * *value*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Set an enumeration feature on the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the enumeration to get.
in	*value*	Pointer to a character array which contains the string to set.
in	*module*	Module for which the parameter should be set. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface-stream- or buffer-module.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

To retrieve the possible values for this enumeration, two functions need to be called:

1. Call "csiGetFeatureEnum". In the returned structure, the element "enumCounter" indicates the number of available entries.
2. The different entries can be retrieved by using the function "csiGetFeatureEnumEntryByIndex" simply by iterating from 0 until the "enumCounter"-1.

CSI_DLL_EXPORT csiErr csiSetFeatureFloat (csiHandle *moduleHandle*, const char * *featureName*, double *value*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Set a floating point value feature on the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the feature to set.
in	*value*	Floating point value to set.
in	*module*	Module for which the parameter should be set. Please use the enum csiModuleLevel to select. Determines if the parameter should be set on the device-, transport layer-, interface-stream- or buffer-module.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiSetFeatureInt (csiHandle *moduleHandle*, const char * *featureName*, int64_t *value*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Set an integer feature on the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the feature to set.
in	*value*	Integer value to set the feature to.
in	*module*	Module for which the parameter should be set. Please use the enum csiModuleLevel to select. Determines if the parameter should be set on the device-, transport layer-, interface-stream- or buffer-module.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiSetFeatureReg (csiHandle *moduleHandle*, const char * *featureName*, const char * *buffer*, size_t *length*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Set a register value on the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the feature to set.
in	*buffer*	Pointer to the data which will be written to the register.
in	*length*	Number of bytes to write to the register.
in	*module*	Module for which the register should be set. Please use the enum csiModuleLevel to select. Determines if the register should be set on the device-, transport layer-, interface- stream- or buffer-module.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

To avoid unexpected behavior, it is recommended to retrieve the maximum buffer length before setting it to the device. This can be achieved by using the function "csiGetFeatureParameter". This function will provide all necessary information about the parameter (including min and max values). The register length must not exceed the length given in the "featureRegLength" parameter.

CSI_DLL_EXPORT csiErr csiSetFeatureString (csiHandle *moduleHandle*, const char * *featureName*, const char * *value*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Set a string feature on the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name (Not the display name) of the feature to set.
in	*value*	Pointer to a character array which contains the string to set.
in	*module*	Module for which the parameter should be set. Please use the enum csiModuleLevel to select. Determines if the parameter should be set on the device-, transport layer-, interface- stream- or buffer-module.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

To avoid unexpected behavior, it is recommended to retrieve the maximum string length before setting it to the device. This can be achieved by using the function "csiGetFeatureParameter". This function will provide all necessary information about the parameter (including min and max values). The string length must not exceed the length given in the "maximumStringLength" parameter.

CSI_DLL_EXPORT csiErr csiStartAcquisition (csiHandle *deviceHandle*, csiAcquisitionMode *mode*)

Start the acquisition on the device and created data streams.

Parameters

in	*deviceHandle*	provided by the csiOpenDevice-function.
in	*mode*	Acquisition mode as defined in csiAcquisitionMode.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

This function will start the acquisition on the camera device passed with the device parameter and on all created data streams of that device.

CSI_DLL_EXPORT csiErr csiStopAcquisition (csiHandle *deviceHandle*)

Stop the acquisition on the device.

Parameters

in	*deviceHandle*	provided by the csiOpenDevice-function.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

This function will stop the acquisition on the camera device passed with the device parameter and on all created data streams of that device.

CSI_DLL_EXPORT csiErr csiUnregisterEvent (csiHandle *evt*)

Unregister a specific event from the event handler.

Parameters

in	*evt*	The handle to the event that should be unregistered.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

Unregistering the event will cancel all active pending calls to **csiWaitForEvent()** or **csiGetNextImage()**.

CSI_DLL_EXPORT csiErr csiUnRegisterInvalidateCB (csiHandle *moduleHandle*, const char * *featureName*, csiModuleLevel module *CSI_DEFAULT_PARAM_MODULE*)

Unregister an invalidation callback function from a specific feature.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice or csiOpenTLInterface function.
in	*featureName*	Name of the feature to unregister the callback from.
in	*module*	Module for which the feature invalidation callback should be registered. Please use the enum csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module.

Returns

csiSuccess or an error defined in the csiErr-Enum.

CSI_DLL_EXPORT csiErr csiWaitForEvent (csiHandle *evt*, uint64_t *timeoutMilliseconds*, csiEventData ** *evtDataOut*)

Wait for a desired event to happen.

Parameters

in	*evt*	The handle of the event to wait for.
in	*timeoutMilliseconds*	Timeout after the waiting stops if no event was received.
out	*evtDataOut*	The event data for further use. See **csiEventData** for more information.

Returns

csiSuccess or an error defined in the csiErr-Enum.

Note

After the event was registered with **csiRegisterEvent()** it is possible to actively wait for the event using this function. The waiting can be done asynchronously in a separate thread. This function must be called for each event separately. Please note that this function will return a more general representation of the event data in **csiEventData**. There exists also an event data structure for each event type.
Also see **csiGetNextImage()** which can be used as convenience function to wait for new image data events.

CSI_DLL_EXPORT csiErr csiWriteMemory (csiHandle *moduleHandle*, uint64_t *address*, const char * *buffer*, size_t *sizeBytes*)

Write memory to a register address on the device.

Parameters

in	*moduleHandle*	Handle provided by the csiOpenDevice-function.
in	*address*	Address of the register on the device to which the memory should be written.
in	*buffer*	Buffer holding the data to write.
in	*sizeBytes*	The number of bytes to write from buffer to the register address.

Returns

csiSuccess or an error defined in the csiErr-Enum.

csi.h

Go to the documentation of this file.

C++ Copy

```
1
2 #ifndef CSI_CSIF_H_
3 #define CSI_CSIF_H_
4
5 #include "version.h"
6 #include "csiTypes.h"
7
8 #include <algorithm>
9
10 #ifdef __cplusplus
11 #define CSI_DEFAULT_PARAM_MODULE = CSI_DEVICE_MODULE
12 #define CSI_DEFAULT_PARAM_NULL = NULL
13 #define CSI_DEFAULT_PARAM_ZERO = 0
14 #define CSI_DEFAULT_PARAM_FALSE = false
15 #endif
16
17 #ifdef __cplusplus
18 extern "C"
19 {
```

```
20 namespace CSI {
21 #endif // __cplusplus
22
23 #define CSI_INFO_STRING_BUFFER_SIZE 512
24 #define CSI_INFO_INT_BUFFER_SIZE 512
25 #define CSI_DISCOVERY_INFO_DEVICE_COUNT 16
26 #define CSI_TL_INTERFACE_COUNT 16
27 #define CSI_INFITIE_TIME 0xffffffff
28
29 typedef uint64_t csiHandle;
30 static const csiHandle CSI_EMPTY_HANDLE = 0;
31
32 // Default discovery time which should enable a safe discovery of
the camera
33 static const int CSI_DEFAULT_DISCOVERY_TIME_MS = 400;
34
35 #define CSI_MONO_FORMAT 0x01000000
36 #define CSI_COLOR_FORMAT 0x02000000
37 #define CSI_OCCUPY_8BIT 0x00080000
38 #define CSI_OCCUPY_10BIT 0x000A0000
39 #define CSI_OCCUPY_12BIT 0x000C0000
40 #define CSI_OCCUPY_14BIT 0x000E0000
41 #define CSI_OCCUPY_16BIT 0x00100000
```

```
42 #define CSI_OCCUPY_24BIT 0x00180000

43 #define CSI_OCCUPY_30BIT 0x001E0000

44 #define CSI_OCCUPY_32BIT 0x00200000

45 #define CSI_OCCUPY_48BIT 0x00300000

46 #define CSI_OCCUPY_64BIT 0x00400000

47

55 typedef enum csiPixelFormat

56 {

57 CSI_PIX_FORMAT_UNKNOWN = 0x00000000,

58

60 CSI_PIX_FORMAT_MONO8 = CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x001,

61 CSI_PIX_FORMAT_MONO10 = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x003,

62 CSI_PIX_FORMAT_MONO10_PACKED = CSI_MONO_FORMAT | CSI_OCCUPY_10BIT |
0x046,

63 CSI_PIX_FORMAT_MONO12 = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x005,

64 CSI_PIX_FORMAT_MONO12_PACKED = CSI_MONO_FORMAT | CSI_OCCUPY_12BIT |
0x047,

65 CSI_PIX_FORMAT_MONO16 = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x007,

68 CSI_PIX_FORMAT_RGB8 = CSI_COLOR_FORMAT | CSI_OCCUPY_24BIT | 0x014,

69 CSI_PIX_FORMAT_RGB10_PACKED = CSI_COLOR_FORMAT | CSI_OCCUPY_32BIT |
0x01D,

70 CSI_PIX_FORMAT_RGB10 = CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x018,

71 CSI_PIX_FORMAT_BGR10 = CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x019,
```

```
72 CSI_PIX_FORMAT_RGB12 = CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x01A,

73 CSI_PIX_FORMAT_BGR12 = CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x01B,

74 CSI_PIX_FORMAT_RGBA8 = CSI_COLOR_FORMAT | CSI_OCCUPY_32BIT | 0x016,

75 CSI_PIX_FORMAT_BGRA8 = CSI_COLOR_FORMAT | CSI_OCCUPY_32BIT | 0x017,

76 CSI_PIX_FORMAT_BGR8 = CSI_COLOR_FORMAT | CSI_OCCUPY_24BIT | 0x015,

77 CSI_PIX_FORMAT_RGB16 = CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x033,

78 CSI_PIX_FORMAT_RGBA10 = CSI_COLOR_FORMAT | CSI_OCCUPY_64BIT | 0x05F,

79 CSI_PIX_FORMAT_RGBA12 = CSI_COLOR_FORMAT | CSI_OCCUPY_64BIT | 0x061,

80 CSI_PIX_FORMAT_RGBA16 = CSI_COLOR_FORMAT | CSI_OCCUPY_64BIT | 0x064,

82 // Bayer

83 CSI_PIX_FORMAT_BayerGR8 = CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x008,

84 CSI_PIX_FORMAT_BayerRG8 = CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x009,

86 CSI_PIX_FORMAT_BayerGB8 = CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x00A,

87 CSI_PIX_FORMAT_BayerBG8 = CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x00B,

89 CSI_PIX_FORMAT_BayerGR12 = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT |
0x010,

90 CSI_PIX_FORMAT_BayerRG12 = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT |
0x011,

92 CSI_PIX_FORMAT_BayerGB12 = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT |
0x012,

93 CSI_PIX_FORMAT_BayerBG12 = CSI_MONO_FORMAT | CSI_OCCUPY_16BIT |
0x013

94 } csiPixelFormat;

95
```

```
100 typedef enum csiDeviceAccessMode {
101 CSI_DEV_MODE_UNKNOWN = 0x00,
102 CSI_DEV_MODE_NONE = 0x01,
103 CSI_DEV_MODE_EXCLUSIVE,
104 CSI_DEV_MODE_READ,
105 CSI_DEV_MODE_CONTROL
106 } csiDeviceAccessMode;
107
108
113 typedef enum csiDeviceAccessStatus {
114 CSI_DEV_ACCESS_STATUS_UNKNOWN = 0x00,
115 CSI_DEV_ACCESS_STATUS_READWRITE = 0x01,
116 CSI_DEV_ACCESS_STATUS_READONLY = 0x02,
117 CSI_DEV_ACCESS_STATUS_NOACCESS = 0x03,
118 CSI_DEV_ACCESS_STATUS_BUSY = 0x04,
119 CSI_DEV_ACCESS_STATUS_OPEN_READWRITE = 0x05,
120 CSI_DEV_ACCESS_STATUS_OPEN_READ = 0x06
121 } csiDeviceAccessStatus;
122
123
128 typedef enum csiFeatureType {
129 CSI_UNKNOWN_TYPE,
```

```
130 CSI_BOOLEAN_TYPE,
131 CSI_INT_TYPE,
132 CSI_FLOAT_TYPE,
133 CSI_STRING_TYPE,
134 CSI_ENUMERATION_TYPE,
135 CSI_ENUMENTRY_TYPE,
136 CSI_CATEGORY,
137 CSI_COMMAND,
138 CSI_REGISTER,
139 CSI_PORT
140 } csiFeatureType;
141
142
147 typedef enum csiAccessMode {
148 CSI_ACCESS_UNKNOWN,
149 CSI_ACCESS_NOT_AVAILABLE,
150 CSI_ACCESS_READ_ONLY,
151 CSI_ACCESS_READ_WRITE,
152 CSI_ACCESS_WRITE_ONLY
153 } csiAccessMode;
154
155
```

```
160 typedef enum csiFeatureVisibility {
161 CSI_VISIBILITY_BEGINNER = 1,
162 CSI_VISIBILITY_EXPERT,
163 CSI_VISIBILITY_GURU,
164 CSI_VISIBILITY_DEVELOPER,
165 CSI_VISIBILITY_INVISIBLE
166 } csiFeatureVisibility;
167
168
173 typedef enum csiModuleLevel {
174 CSI_UNKNOWN_MODULE,
175 CSI_TRANSPORTLAYER_MODULE,
176 CSI_INTERFACE_MODULE,
177 CSI_DEVICE_MODULE,
178 CSI_LOCAL_DEVICE_MODULE,
179 CSI_STREAM_MODULE,
180 CSI_BUFFER_MODULE
181 } csiModuleLevel;
182
183
188 typedef enum csiDisplayNotation {
189 CSI_NOTATION_AUTOMATIC,
```

```
190 CSI_NOTATION_FIXED,
191 CSI_NOTATION_SCIENTIFIC
192 } csiDisplayNotation;
193
194
199 typedef enum csiRepresentation {
200 CSI_REPRESENTATION_LINEAR,
201 CSI_REPRESENTATION_LOGARITHMIC,
202 CSI_REPRESENTATION_BOOLEAN,
203 CSI_REPRESENTATION_PURENUMBER,
204 CSI_REPRESENTATION_HEX,
205 CSI_REPRESENTATION_IP,
206 CSI_REPRESENTATION_MAC,
207 CSI_REPRESENTATION_UNDEFINED
208 } csiRepresentation;
209
210
215 typedef struct csiFeatureParameter {
216 csiFeatureType type;
217 csiFeatureVisibility visibility;
218 csiAccessMode access;
219 csiDisplayNotation displayNotation;
```

```
220 csiRepresentation representation;
221 char displayPrecision;
223 int64_t valueInt;
224 int64_t incrementInt;
225 int64_t minimumInt;
226 int64_t maximumInt;
227 int64_t validValueSetInt[CSI_INFO_INT_BUFFER_SIZE];
228 size_t validValueSetSizeInt;
229
230 double valueFlt;
231 double incrementFlt;
232 double minimumFlt;
233 double maximumFlt;
235 char valueStr[CSI_INFO_STRING_BUFFER_SIZE];
236 size_t maximumStringLength;
238 int64_t level;
239 uint32_t enumCounter;
240 int32_t enumIndex;
242 char displayName[CSI_INFO_STRING_BUFFER_SIZE];
243 char name[CSI_INFO_STRING_BUFFER_SIZE];
244 char tooltip[CSI_INFO_STRING_BUFFER_SIZE];
246 char valueUnit[CSI_INFO_STRING_BUFFER_SIZE];
```

```
248 size_t featureRegLength;
249 int64_t featureRegAddress;
250 bool isFeature;
251 bool isLittleEndian;
252
253 } csiFeatureParameter;
254
255
260 typedef enum csiLogLevel
261 {
262 CSI_LOGLEVEL_NONE = 0,
263 CSI_LOGLEVEL_ERROR = 1,
264 CSI_LOGLEVEL_WARN = 2,
265 CSI_LOGLEVEL_INFO = 4,
266 CSI_LOGLEVEL_DEBUG = 8,
267 CSI_LOGLEVEL_TRACE = 16,
268 } csiLogLevel;
269
270
275 typedef enum csiErr {
276 csiSuccess = 0,
277 csiNotInitialized = -100,
```

```
278 csiInvalidState = -101,
279 csiNotOpened = -102,
280 csiNoImageDataAvailable = -103,
281 csiNotFound = -104,
282 csiInvalidParameter = -105,
283 csiNotAvailable = -106,
284 csiFunctionNotAvailable = -107,
285 csiTimeout = -108,
286 csiAborted = -109,
287 csiFileOperationFailure = -110,
288 csiFileOperationFatalError = -111,
289 csiNoAccess = -112,
290 csiWrongBufferSize = -113,
291 csiInvalidBuffer = -114,
292 csiResourceInUse = -115,
293 csiNotImplemented = -116,
294 csiInvalidHandle = -117,
295 csiIOError = -118,
296 csiParsingError = -119,
297 csiInvalidValue = -120,
298 csiResourceExhausted = -121,
299 csiOutOfMemory = -122,
```

```
300 csiBusy = -123,
301 csiUnknown = -200,
302 csiCustomErr = -0x0f000000
303 } csiErr;
304
305
310 typedef enum csiEventType
311 {
312 CSI_EVT_NEWIMAGEDATA = 0x00,
313 CSI_EVT_ERROR = 0x01,
314 CSI_EVT_MODULE = 0x02,
315 CSI_EVT_FEATURE_INVALIDATE = 0x03,
316 CSI_EVT_FEATURE_CHANGE = 0x04,
317 CSI_EVT_REMOTE_DEVICE = 0x05,
318 CSI_EVT_CUSTOM = 0x1000
319 } csiEventType;
320
321
326 typedef enum csiAcquisitionMode {
327 CSI_ACQUISITION_SINGLE_FRAME = 0x00000001,
328 CSI_ACQUISITION_CONTINUOUS = 0xFFFFFFFF
329 } csiAcquisitionMode;
```

```
330
331
336 typedef enum csiMemTransferStatus
337 {
338 csiTransferStatusInit,
339 csiTransferStatusInProgress,
340 csiTransferStatusInProgressWaiting,
341 csiTransferStatusFinishSucess,
342 csiTransferStatusFinishError,
343 csiTransferStatusCancelOnError
345 } csiMemTransferStatus;
346
347
352 typedef struct csiMemTransferInfo {
353 csiHandle device;
354 size_t totalBytesToTransfer;
355 size_t bytesTransferred;
356 csiMemTransferStatus status;
357 csiErr errorCode;
358 const char* progressText;
359 } csiMemTransferInfo;
360
```

```
361
362 struct csiEventUserData;
363 struct csiMemTransferUserData;
364 struct csiLogUserData;
365
366
371 typedef struct csiTLProducerInfos {
372 char transportLayerName[CSI_INFO_STRING_BUFFER_SIZE];
373 char transportLayerDisplayName[CSI_INFO_STRING_BUFFER_SIZE];
374 char transportLayerType[CSI_INFO_STRING_BUFFER_SIZE];
375 char transportLayerPath[CSI_INFO_STRING_BUFFER_SIZE];
376 char transportLayerID[CSI_INFO_STRING_BUFFER_SIZE];
377 size_t pathSizeInBytes;
378 } csiTLProducerInfos;
379
380
385 typedef struct csiDeviceInfo
386 {
387 char deviceIdentifier[CSI_INFO_STRING_BUFFER_SIZE];
388 char name[CSI_INFO_STRING_BUFFER_SIZE];
389 char model[CSI_INFO_STRING_BUFFER_SIZE];
390 char vendor[CSI_INFO_STRING_BUFFER_SIZE];
```

```
391 char serialNumber[CSI_INFO_STRING_BUFFER_SIZE];
392 char interfaceDescription[CSI_INFO_STRING_BUFFER_SIZE];
393 char interfaceID[CSI_INFO_STRING_BUFFER_SIZE];
394 char userName[CSI_INFO_STRING_BUFFER_SIZE];
395 char version[CSI_INFO_STRING_BUFFER_SIZE];
396 int64_t cameraSwPackageIsConsistent;
397 csiTLProducerInfos tlProducerInfos;
398 csiDeviceAccessStatus accessStatus;
399 uint64_t timestampFrequediscncy;
400 } csiDeviceInfo;
401
402
407 typedef struct csiDiscoveryInfo {
408 uint32_t numDevices;
409 double progress;
410 bool discoveryRunning;
411 csiDeviceInfo devices[CSI_DISCOVERY_INFO_DEVICE_COUNT];
412 } csiDiscoveryInfo;
413
418 typedef struct csiTLInterfaceInfo
419 {
420 char interfaceDescription[CSI_INFO_STRING_BUFFER_SIZE];
```

```
421 char interfaceID[CSI_INFO_STRING_BUFFER_SIZE];
422 csiTLProducerInfos tlProducerInfos;
423 } csiTLInterfaceInfo;
424
429 typedef struct csiTLInterfaceDiscoveryInfo
430 {
431 uint32_t numInterfaces;
432 double progress;
433 bool discoveryRunning;
434 csiTLInterfaceInfo interfaceInfos[CSI_TL_INTERFACE_COUNT];
435 } csiTLInterfaceDiscoveryInfo;
436
437
442 typedef struct csiDataStreamInfo {
443 char identifier[CSI_INFO_STRING_BUFFER_SIZE];
444 char displayName[CSI_INFO_STRING_BUFFER_SIZE];
445 uint32_t index;
446 } csiDataStreamInfo;
447
448
453 typedef struct csiImageInfo {
454 uint32_t width;
```

```
455 uint32_t height;
456 uint32_t linePitch;
457 uint32_t numChannels;
458 csiPixelFormat format;
459 } csiImageInfo;
460
461
466 typedef struct csiEventData {
467 csiEventType type;
468 csiHandle sender;
469 csiModuleLevel senderType;
470 char* tl_rawEventData;
471 size_t tl_rawEventDataSizeBytes;
472 char* eventValue;
473 size_t eventValueSizeBytes;
474 uint64_t eventIdentifier;
475 } csiEventData;
476
481 typedef struct CSI_DLL_EXPORT csiNewBufferEventData
482 {
483 csiNewBufferEventData();
484 ~csiNewBufferEventData();
```

```
485
486 csiNewBufferEventData(const csiNewBufferEventData& other);
487 csiNewBufferEventData(csiNewBufferEventData&& other) noexcept;
488 csiNewBufferEventData& operator=(csiNewBufferEventData& other);
489 csiNewBufferEventData& operator=(csiNewBufferEventData&& other)
noexcept;
490
491 csiEventType type{ CSI::csiEventType::CSI_EVT_NEWIMAGEDATA };
492 csiHandle sender{ CSI_EMPTY_HANDLE };
493 csiModuleLevel senderType{ CSI::csiModuleLevel::CSI_UNKNOWN_MODULE
};
494 char* tl_rawEventData{ nullptr };
495 size_t tl_rawEventDataSizeBytes{ 0 };
496 unsigned char* eventValue{ nullptr };
497 size_t eventValueSizeBytes{ 0 };
498 uint64_t eventIdentifier{ 0 };
499 csiHandle bufferHandle{ 0 };
500 uint64_t imageNr{ 0 };
501 uint64_t bufferIdentifier{ 0 };
502 uint64_t timestampMS{ 0 };
503 uint64_t timestampRaw{ 0 };
504 csiImageInfo imageInfo{};
505 } csiNewBufferEventData;
```

```
506
507
515 typedef struct csiAcquistionStatistics
516 {
517 int64_t framesUnderrun;
518 int64_t framesDropped;
519 int64_t framesAcquired;
520 int64_t networkPacketsOK;
521 int64_t networkPacketsError;
523 } csiAcquistionStatistics;
524
529 typedef enum csiDownloadOptions
530 {
531 CSI_DOWNLOAD_NO_OPTIONS = 0
532 } csiDownloadOptions;
533
534
539 typedef enum csiUploadOptions
540 {
541 CSI_UPLOAD_NO_OPTIONS = 0,
542 CSI_UPLOAD_IGNORE_CHECKSUM = 1
543 } csiUploadOptions;
```

```
544

550 typedef struct alignas(uint64_t) csiFileTransferParams

551 {

552 const char* fileName;

553 const char* fileType;

554 } csiFileTransferParams;

555

561 typedef struct alignas(uint64_t) csiDownloadParams

562 {

563 csiFileTransferParams fileParams;

564 csiDownloadOptions options;

565 } csiUploadDownloadUploadParams;

566

572 typedef struct alignas(uint64_t) csiUploadParams

573 {

574 csiFileTransferParams fileParams;

575 csiUploadOptions options;

576 } csiUploadParams;

577

582 enum csiCalibrationLUT

583 {

584 CSI_DSNU_LUT1 = 1,
```

```
585 CSI_DSNU_LUT2,
586 CSI_PRNU_LUT1,
587 CSI_PRNU_LUT2
588 };
589
594 typedef enum csiReferenceImgLoadMode
595 {
596 CSI_LOAD_REF_IMG_FROM_DISC,
597 CSI_ACQUIRE_REF_IMG_FROM_CAMERA
598 } csiReferenceImgLoadMode;
599
600
605 typedef struct csiCalibrationParams
606 {
607 bool enableROI;
608 int64_t yStartROI;
609 int64_t heightROI;
610 bool enableExtrapolationLeft;
611 bool enableExtrapolationRight;
612 int64_t extrapolationLeft;
613 int64_t extrapolationRight;
614 int64_t extrapolationWidth;
```

```
615 uint64_t firstValidPixel;

617 int64_t targetValue;

618 bool calcImprove;

619 double contrast;

620 double brightness;

621 bool isCCDSensor = false;

622 } csiCalibrationParams;

623

628 typedef enum CalibrationMode

629 {

630 DSNU_CALIBRATION = 1,

631 PRNU_CALIBRATION

632 } CalibrationMode;

633

634 typedef void *CB_OBJECT;

635 typedef void (*CB_FEATURE_INVALIDATED_PFN)(const char *featureName,
void* userdata);

636

637 typedef void(*csiDiscoveryInfoCallbackFunc)(const csiDiscoveryInfo*
discoveryInfo);

638 typedef void (*csiDiscoveryTLInterfaceInfoCallbackFunc)(const
csiTLInterfaceDiscoveryInfo* discoveryInfo);

639 typedef bool(*csiMemTransferCallbackFunc)(const csiMemTransferInfo*
info, struct csiMemTransferUserData* userdata);
```

```
640 typedef void(*csiLogSinkCallbackFunc)(csiLogLevel, const char*
message, struct csiLogUserData* userdata);

641

642 // Global general functions

643

656 CSI_DLL_EXPORT csiErr csiInit(csiLogLevel logLvl,
csiLogSinkCallbackFunc logCallbackFunc CSI_DEFAULT_PARAM_NULL,
csiLogUserData* userdata CSI_DEFAULT_PARAM_NULL);

657

663 CSI_DLL_EXPORT csiErr csiClose();

664

672 CSI_DLL_EXPORT csiErr csiReset();

673

691 CSI_DLL_EXPORT csiErr csiDiscoverDevices(csiDiscoveryInfo*
discoveryInfoOut, uint64_t timeoutMilliseconds,
csiDiscoveryInfoCallbackFunc discCallbackFunc CSI_DEFAULT_PARAM_NULL,
const char* additionalSearchPaths CSI_DEFAULT_PARAM_NULL, bool
overrideSearchPath CSI_DEFAULT_PARAM_FALSE);

692

705 CSI_DLL_EXPORT csiErr csiGetDeviceInfo(uint32_t deviceIndex,
csiDeviceInfo* deviceInfoOut);

706

731 CSI_DLL_EXPORT csiErr
csiDiscoverTLInterfaces(csiTLInterfaceDiscoveryInfo* discoveryInfoOut,
uint64_t timeoutMilliseconds, csiDiscoveryTLInterfaceInfoCallbackFunc
discCallbackFunc CSI_DEFAULT_PARAM_NULL, const char*
additionalSearchPaths CSI_DEFAULT_PARAM_NULL, bool overrideSearchPath
CSI_DEFAULT_PARAM_FALSE);

732
```

```
733

747 CSI_DLL_EXPORT csiErr csiGetNumberOfTLProducers(int32_t
*numTLProducers);

748

757 CSI_DLL_EXPORT csiErr csiGetTLProducerPathByIndex(char
*transportLayerPath, size_t bufferSize, uint32_t index);

758

772 CSI_DLL_EXPORT csiErr
csiGetTLProducerInfobyFilePath(csiTLProducerInfos* tlProducerInfos,
const char* producerName);

773

774 // Device functions

775

804 CSI_DLL_EXPORT csiErr csiOpenDevice(const char* deviceIdentifier,
const char* interfaceID, csiHandle* deviceHandleOut, uint64_t
timeoutMilliseconds, csiDeviceAccessMode openMode);

805

818 CSI_DLL_EXPORT csiErr csiCloseDevice(csiHandle deviceHandle);

819

827 CSI_DLL_EXPORT csiErr csiCheckAndReallocBuffers(csiHandle
deviceHandle);

828

843 CSI_DLL_EXPORT csiErr csiStartAcquisition(csiHandle deviceHandle,
csiAcquisitionMode mode);

844

856 CSI_DLL_EXPORT csiErr csiStopAcquisition(csiHandle deviceHandle);
```

```
857

858

868 CSI_DLL_EXPORT csiErr csiAbortAcquisition(csiHandle deviceHandle);

869

870 // Data Stream functions

888 CSI_DLL_EXPORT csiErr csiGetNumberOfDataStreams(csiHandle
deviceHandle, uint32_t* numberOfStreamsOut);

889

904 CSI_DLL_EXPORT csiErr csiGetDataStreamInfo(csiHandle deviceHandle,
uint32_t dsIndex, csiDataStreamInfo* dataStreamInfoOut);

905

920 CSI_DLL_EXPORT csiErr csiGetDeviceDataStreamInfo(csiHandle
moduleHandle, uint32_t dsIndex, csiDataStreamInfo* dataStreamInfoOut);

921

944 CSI_DLL_EXPORT csiErr csiCreateDataStream(csiHandle deviceHandle,
uint32_t dsIndex, csiHandle* dataStreamOut, uint32_t numberOfBuffers,
size_t bufferSize CSI_DEFAULT_PARAM_ZERO);

945

960 CSI_DLL_EXPORT csiErr csiCloseDataStream(csiHandle dataStream);

961

978 CSI_DLL_EXPORT csiErr csiRegisterEvent(csiHandle moduleHandle,
csiEventType evtType, csiHandle* eventOut, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

979

1001 CSI_DLL_EXPORT csiErr csiWaitForEvent(csiHandle evt, uint64_t
timeoutMilliseconds, csiEventData** evtDataOut);
```

```
1002

1013 CSI_DLL_EXPORT csiErr csiUnregisterEvent(csiHandle evt);

1014

1024 CSI_DLL_EXPORT csiErr csiEventKill(csiHandle evt);

1025

1047 CSI_DLL_EXPORT csiErr csiGetNextImage(csiHandle eventHandle,
csiNewBufferEventData** bufferInfoOut, uint64_t timeoutMilliseconds);

1048

1070 CSI_DLL_EXPORT csiErr csiReleaseImage(csiHandle dataStream, const
csiNewBufferEventData* bufferInfo);

1071

1082 CSI_DLL_EXPORT csiErr csiGetAcquisitionStatistics(csiHandle
dataStream, csiAcquistionStatistics* stats);

1083

1084 // TL interface functions

1085

1102 CSI_DLL_EXPORT csiErr csiOpenTLInterface(csiTLInterfaceInfo
interfaceInfo, csiHandle* interfaceHandleOut, uint64_t
timeoutMilliseconds);

1103

1116 CSI_DLL_EXPORT csiErr csiCloseTLInterface(csiHandle
interfaceHandle);

1117

1118 // Module configuration functions

1134 CSI_DLL_EXPORT csiErr csiGetFeatureBool(csiHandle moduleHandle,
```

```
const char* featureName, bool* valueOut, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1135

1151 CSI_DLL_EXPORT csiErr csiSetFeatureBool(csiHandle moduleHandle,
const char* featureName, bool value, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1152

1153

1164 CSI_DLL_EXPORT csiErr csiGetFeatureInt(csiHandle moduleHandle,
const char* featureName, int64_t* valueOut, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1165

1180 CSI_DLL_EXPORT csiErr csiSetFeatureInt(csiHandle moduleHandle,
const char* featureName, int64_t value, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1181

1197 CSI_DLL_EXPORT csiErr csiGetFeatureFloat(csiHandle moduleHandle,
const char* featureName, double* valueOut, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1198

1213 CSI_DLL_EXPORT csiErr csiSetFeatureFloat(csiHandle moduleHandle,
const char* featureName, double value, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1214

1241 CSI_DLL_EXPORT csiErr csiGetFeatureString(csiHandle moduleHandle,
const char* featureName, char* valueOut, size_t* sizeOut,
csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);

1242

1266 CSI_DLL_EXPORT csiErr csiSetFeatureString(csiHandle moduleHandle,
```

```
const char* featureName, const char* value, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1267

1287 CSI_DLL_EXPORT csiErr csiExecuteCommand(csiHandle moduleHandle,
const char* featureName, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1288

1308 CSI_DLL_EXPORT csiErr csiIsCommandActive(csiHandle moduleHandle,
const char* featureName, bool *isActive, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1309

1334 CSI_DLL_EXPORT csiErr csiGetFeatureReg(csiHandle moduleHandle,
const char* featureName, char* buffer, size_t* length, csiModuleLevel
module CSI_DEFAULT_PARAM_MODULE);

1335

1360 CSI_DLL_EXPORT csiErr csiSetFeatureReg(csiHandle moduleHandle,
const char* featureName, const char* buffer, size_t length,
csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);

1361

1384 CSI_DLL_EXPORT csiErr csiGetFeatureEnum(csiHandle moduleHandle,
const char* featureName, csiFeatureParameter *featureParamOut,
csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);

1385

1410 CSI_DLL_EXPORT csiErr csiSetFeatureEnum(csiHandle moduleHandle,
const char* featureName, const char* value, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1411

1428 CSI_DLL_EXPORT csiErr csiGetFeatureParameter(csiHandle
moduleHandle, const char* featureName, csiFeatureParameter*
featureParamOut, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
```

```
1429

1444 CSI_DLL_EXPORT csiErr csiGetFeatureAccessMode(csiHandle moduleH,
const char* featureName, csiAccessMode* access, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1445

1471 CSI_DLL_EXPORT csiErr csiIterateFeatureTree(csiHandle
moduleHandle, const char* rootFeatureName, uint32_t index, char*
featureNameOut, size_t nameBuffSize, csiFeatureType* type,
csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);

1472

1488 CSI_DLL_EXPORT csiErr csiGetFeatureEnumEntryCount(csiHandle
moduleHandle, const char* featureName, uint32_t* count, csiModuleLevel
module CSI_DEFAULT_PARAM_MODULE);

1489

1512 CSI_DLL_EXPORT csiErr csiGetFeatureEnumEntryByIndex(csiHandle
moduleHandle, const char* featureName, int32_t enumIndex,
csiFeatureParameter *featureParamOut, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1513

1533 CSI_DLL_EXPORT csiErr csiGetFeatureEnumEntryByName(csiHandle
moduleHandle, const char* featureName, const char* enumValue,
csiFeatureParameter *featureParamOut, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1534

1535

1536 // File transfer functions

1559 CSI_DLL_EXPORT csiErr csiGetUpdateFileType(csiHandle moduleHandle,
const char* fileName, char* fileTypeOut, size_t bufferSize);

1560
```

```
1582 CSI_DLL_EXPORT csiErr csiFileDownloadToDevice(csiHandle
moduleHandle,

1583 csiHandle localDevice,

1584 const char* fileName,

1585 const char* fileType,

1586 uint64_t timeoutMilliseconds,

1587 csiMemTransferCallbackFunc listener CSI_DEFAULT_PARAM_NULL,

1588 csiMemTransferUserData* userdata CSI_DEFAULT_PARAM_NULL);

1589

1609 CSI_DLL_EXPORT csiErr csiFileUploadFromDevice(csiHandle
moduleHandle,

1610 csiHandle localDevice,

1611 const char* fileName,

1612 const char* fileType,

1613 uint64_t timeoutMilliseconds,

1614 csiMemTransferCallbackFunc listener CSI_DEFAULT_PARAM_NULL,

1615 csiMemTransferUserData* userdata CSI_DEFAULT_PARAM_NULL);

1616

1617

1637 CSI_DLL_EXPORT csiErr csiFileDownloadToDeviceEx

1638 (csiHandle moduleHandle,

1639 csiHandle localDevice,

1640 const csiDownloadParams params,
```

```
1641 uint64_t timeoutMilliseconds,

1642 csiMemTransferCallbackFunc listener CSI_DEFAULT_PARAM_NULL,

1643 csiMemTransferUserData* userdata CSI_DEFAULT_PARAM_NULL);

1644

1666 CSI_DLL_EXPORT csiErr csiFileUploadFromDeviceEx

1667 (csiHandle moduleHandle,

1668 csiHandle localDevice,

1669 const csiUploadParams params,

1670 uint64_t timeoutMilliseconds,

1671 csiMemTransferCallbackFunc listener CSI_DEFAULT_PARAM_NULL,

1672 csiMemTransferUserData* userdata CSI_DEFAULT_PARAM_NULL);

1673

1674 // Memory transfer functions

1688 CSI_DLL_EXPORT csiErr csiReadMemory(csiHandle moduleHandle,
uint64_t address, char* buffer, size_t sizeBytes);

1689

1702 CSI_DLL_EXPORT csiErr csiWriteMemory(csiHandle moduleHandle,
uint64_t address, const char* buffer, size_t sizeBytes);

1703

1716 CSI_DLL_EXPORT csiErr csiIsModuleLittleEndian(csiHandle
moduleHandle, csiModuleLevel lvl, bool *isLittleEndian);

1717

1727 CSI_DLL_EXPORT csiErr csiGetErrorDescription(csiErr error, char*
bufferOut, size_t bufferSize);
```

```
1728

1729 // Helper functions

1739 CSI_DLL_EXPORT unsigned char csiBitsPerPixelFromFormat(const
csiPixelFormat format);

1740

1761 CSI_DLL_EXPORT csiErr csiRegisterInvalidateCB(csiHandle
moduleHandle, const char *featureName, CB_OBJECT objCB,
CB_FEATURE_INVALIDATED_PFN pfnFeatureInvalidateCB, csiModuleLevel
module CSI_DEFAULT_PARAM_MODULE);

1762

1763 //unregistering the call-back function

1779 CSI_DLL_EXPORT csiErr csiUnRegisterInvalidateCB(csiHandle
moduleHandle, const char *featureName, csiModuleLevel module
CSI_DEFAULT_PARAM_MODULE);

1780

1791 CSI_DLL_EXPORT csiErr csiGetLibraryVersion(uint32_t* major,
uint32_t* minor, uint32_t* patch, uint32_t* revision, uint32_t* build);

1792

1799 CSI_DLL_EXPORT csiErr csiSetFeatureCachingDisabled(bool
disableCaching);

1800

1808 CSI_DLL_EXPORT csiErr csiIsFeatureCachingDisabled(bool
*isDisabled);

1809

1832 CSI_DLL_EXPORT csiErr
csiGenerateAndUploadCalibrationData(csiHandle devHandle,

1833 csiCalibrationLUT LUTSelector,
```

```
1834 csiReferenceImgLoadMode imgLoadMode,

1835 csiCalibrationParams calibParam,

1836 csiNewBufferEventData *refImage,

1837 const char* imgFile);

1866 CSI_DLL_EXPORT csiErr
csiGenerateAndSaveCalibrationDataToFile(csiHandle devHandle,

1867 CalibrationMode calibMode,

1868 csiReferenceImgLoadMode imgLoadMode,

1869 csiCalibrationParams calibParam,

1870 csiNewBufferEventData* refImage,

1871 const char* imgFile,

1872 const char* calibrationFile);

1873

1902 CSI_DLL_EXPORT csiErr csiGenerateReferenceImage(csiHandle
devHandle,

1903 const CalibrationMode calibMode,

1904 csiReferenceImgLoadMode imgLoadMode,

1905 csiCalibrationParams calibParam,

1906 const csiNewBufferEventData* refImage,

1907 const char* imgFile,

1908 int *referenceImage, int *resMaxValue);

1909 #ifdef __cplusplus

1910 }
```

```
1911 }
1912 #endif // __cplusplus
1913
1914 #endif // CSI_CSIF_H_
```

Troubleshooting with GigE Interface

During Installation

Error Description	Possible cause	Action
The PowerShell script for setting up the 10 Gige Network adapter replies with an error. "10 GigE network connection is not found. Please check the hardware device."	The Network adapter is not Supported	Setup the Network adapter manually, 10 GigE with s2i transport layer
	The Network adapter is not Installed	Install the Network adapter, Network adapters and transceivers
	The camera is not connected to the PC	Connect the camera to the PC
	The Camera is turned off	Switch the camera on
	The Camera interface is CXP	Skip the script
	The interface cables of the camera are not connected properly	Connect the cables again
	Transceiver which is not configured for the network adapter	Use a correctly configured transceiver, Network adapters and transceivers

During device discovery

Error Description	Possible cause	Action
No Camera was found during the device discovery	The correct GenTL Produces is not selected	Change GenTL Producer Connection and disconnection of Camera
	The IP address of the Network adapter is not correct	Change the IP address of the camera or adapter, Configure the network adapter
	The camera is not connected to the PC	Connect the camera to the PC
	The Camera is turned off	Switch the camera on
	The interface cables of the camera are not connected properly	Connect the cables again

During Steaming

Error Description	Possible cause	Action
GCT shows a back image with a image content in R = 0, G = 0 and B = 0	The Network adapter was not configured correctly	Setup the Network adapter manually, 10 GigE with s2i transport layer

Working with Kithara

During device discovery

Error Description	Possible cause	Action
No Camera was found during the device discovery	The correct GenTL Produces is not selected	Change GenTL Producer Connection and disconnection of Camera.
	No Dongle plugged	Plug in the Dongle.
	Kithara is not configured correctly	Configure the Kithara TL, please refer to 10 GigE with Kithara transport layer.
	Wrong IP Address of Camera and Network adapter	Check the IP-Configuration, please refer to 10 GigE with Kithara transport layer.
	Wrong SFP+ connectors	Some manufacturers encode their cards, please refer to Network adapters and transceivers.

Possible reasons for GEV_TIMEOUT_ERROR

Sometimes during image grabbing, there comes no image from the camera, and the message log shows GEV_TIMEOUT_ERROR in the bottom area. This can be caused by following reasons.

Trigger mode not set correctly

Trigger source: If there is no explicit trigger available and the camera should run in “free-running” mode, but the value of trigger source does not correspond to this mode, then GCT will return “GEV_TIMEOUT_ERROR” and cannot grab images.

Filter driver not installed properly (only for 10GiGE without Kithara)

GCT can use the filter driver developed by s2i. If the filter driver should be used for the grabbing process, it must be installed and enabled correctly beforehand in the adapter settings.
To check the adapter settings, go to Start menu -> Settings -> Network & Internet. On the Settings window, click Change adapter options. A new dialog box opens with a list of network connections. Right-click the corresponding GigE Ethernet connection for the camera and then click choose Properties. If a teaming group is used, then right-click the teaming group and choose its attributes. Ensure that s2i GigE-Vision Filter Driver is selected and that the version is correct.

“Secure Boot” not disabled in BIOS settings (only for 10GiGEwithout Kithara)

If the filter driver is installed properly and can also be detected, but “GEV_STATUS_DRIVER_READ_ERROR” appears during grabbing, it could be due to Secure Boot in the BIOS settings. In this case refer to section 3.4.2.

Firewall is not completely allowed for GCT

Data transfer between camera and PC requires firewall allowance. Normally, the firewall rule is exclusively allowed for GCT automatically after installation.
To verify the firewall rule:

1. On the Window Start menu, click Settings-> Update & Security -> Windows Security -> Firewall & network protection.
2. Click Advanced settings and highlight Inbound Rules in the left panel.
3. Find the rule name GCT2_x64, open its properties, and ensure that the firewall rule is allowed for all types: domain, private, and public.

Reboot

If GEV_TIMEOUT_ERROR occurs only if the filter driver is activated, you can reboot your PC to resolve unknown conflicts in the hardware.