# Chromasens GEN<i>CAM-SDK | Manual

R04

# 1 General Information

## 1.1 About Chromasens

The name of our company, Chromasens, is a combination of 'Chroma' which means color, and 'Sens' which stands for sensor technology.

Chromasens designs, develops, and produces high-quality and user-friendly products:

- Line scan cameras
- Camera systems
- Camera illumination systems
- Image acquisition systems
- Image processing solutions

Today, Chromasens GmbH is experiencing steady growth and is continually penetrating new sales markets around the globe. The company's technologies are used, for example, in products and for applications such as book and document scanners, sorting systems, and inspection systems for quality assurance monitoring.

Customers from all over the world of a wide range of industrial sectors have placed their trust in the experience of Chromasens in the field of industrial image processing.

### 1.1.1 Contact Information

**Chromasens GmbH**
Max-Stromeyer-Str. 116
78467 Konstanz
Germany

Phone: +49 (0) 7531 / 876-0
Fax: +49 (0) 7531 / 876-303
Email: info@chromasens.de
HP: www.chromasens.de

### 1.1.2 Support

**Chromasens GmbH**
Max-Stromeyer-Str. 116
D-78467 Konstanz
Germany

Phone: +49 (0) 7531 / 876-500
Fax: +49 (0) 7531 / 876-303
Email: support@chromasens.de
HP: Support Page

Visit our website at www.chromasens.de which features detailed information on our company and products.

## 1.2 Conventions used in this manual

### 1.2.1 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 {<br>CSI_EVT_NEWIMAGEDATA = 0x00,<br>CSI_EVT_ERROR = 0x01,<br>CSI_EVT_MODULE = 0x02,<br>CSI_EVT_CUSTOM = 0x1000<br>} csiEventType; |
| **Elements** | CSI_EVT_NEWIMAGEDATA: New image data received<br>CSI_EVT_ERROR: Error occurred in the SDK<br>CSI_EVT_MODULE: General event notification<br>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,<br>uint64_t timeoutMilliseconds,<br>csiDiscoveryInfoCallbackFunc discCallbackFunc = NULL,<br>const char* additionalSearchPaths = NULL,<br>bool overrideSearchPath = CSI_DEFAULT_PARAM_FALSE); |
| **Parameters** | timeoutMilliSeconds: The amount of time to search in a specific transport layer for a device<br>discoverCallbackFunc: Pointer to a callback function which gets called when a result was received<br>AdditionalSearchPaths: As default only the paths given in the system variable "GENICAM_GENTL64_PATH" are being searched for the used transport layers<br>overrideSearchPath: If set, only the given path is searched for transport layers to use<br>discoveryInfoOut: The structure will be filled with the available devices |
| **Return value** | Returns csiSuccess or an error defined in the csiErr-Enum. |
| **Comments** | - |

### 1.2.2 Symbols

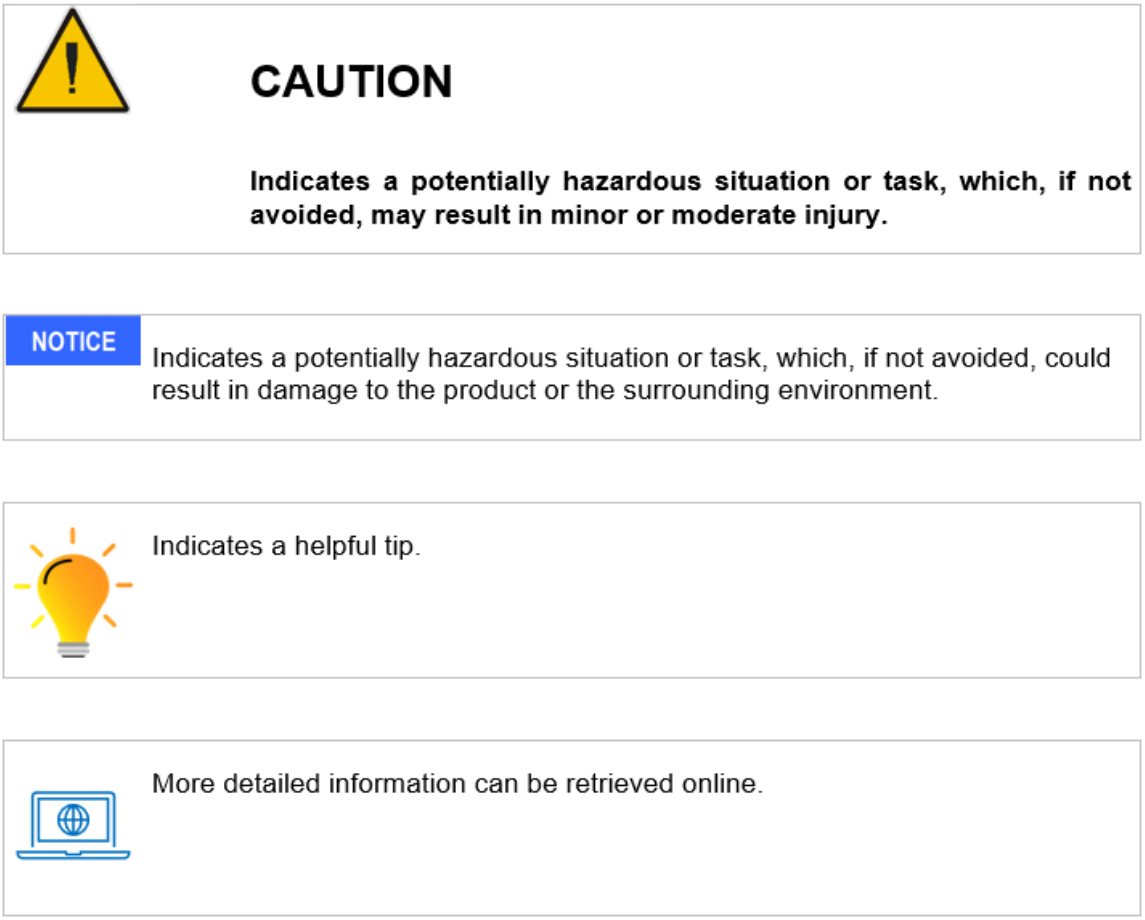



**Figure 1 Symbols**

### 1.2.3 List of abbreviations

| Abbreviation | Meaning | Explanation |
|---|---|---|
| CCM | Color conversion matrix | The CCM supports the conversion from for example RGB to sRGB or any user-defined conversion |
| Corona II | LED illumination | Chromasens product |
| DSNU | Dark signal non-uniformity | Irregularity in the dark image |
| GenICam | Generic interface for cameras | Generic programming interface for industrial cameras administered by the European Machine Vision Association www.emva.org |
| CTI | Common Transport Interface | A GenTL Producer implementation as dynamic loadable platform dependent library |
| GCT | GenICam Control Tool | Graphical user interface using the SDK. Provides a graphical way to configure devices using different TLs. |
| GenApi | GenICam Module | - |
| GenTL | Generic Transport Layer | - |
| GenTL Consumer | A library or application using an implementation of a Transport Layer Interface | - |
| GenTL Producer | Transport Layer Interface implementation | - |
| LED | Light emitting diode | - |
| PRNU | Photo response non-uniformity | Difference in sensitivity of the individual pixels |
| ROI | Region of interest | - |
| RS485 | | ANSI standard defining the electrical characteristics of drivers and receivers for use in serial communications systems. |
| SFNC | Standard Feature Naming Convention | Document of the GenICam standard, which provides feature names for common camera features. |
| VSync | Vertical synchronization | Frame signal for an image (corresponds to FVAL: frame valid) |

# 2 General aspects of the API

The purpose of the Chromasens Gen<I>Cam-SDK is to provide a user friendly and easy way to handle all Chromasens cameras regardless of the physical interface.

Requirements

Supported operating systems:

- Windows: Windows 10 Version 2
- Linux: Ubuntu >= 18.X

Supported compiler:

- Visual Studio >= 2015
- GCC

# 3 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.

## 3.1 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.

## 3.2 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.

## 3.3 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.

## 3.4 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

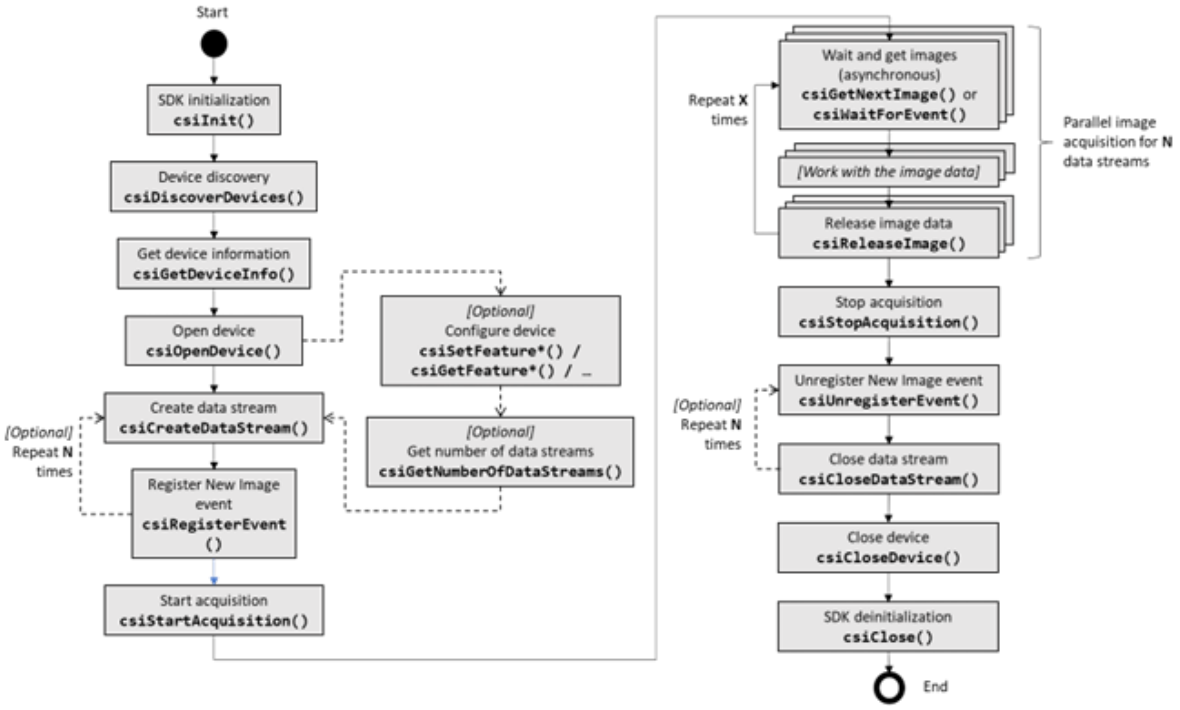



**Figure 2 overview_acquiring_process**

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.

## 3.5 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.<br>Locations:<br>Windows: C:\Users\Public\Documents\Chromasens\GCT2\examples\basic<br>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.<br>Locations:<br>Windows: C:\Users\Public\Documents\Chromasens\GCT2\examples\feature_iteration<br>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.<br>Locations:<br>Windows: C:\Users\Public\Documents\Chromasens\GCT2\examples\calibration_↩ generate<br>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)<br>Locations:<br>Windows: C:\Users\Public\Documents\Chromasens\GCT2\examples\save_↩ rgb10and12<br>Linux: /usr/CSGenicam-SDK/share/csgenicam/examples/save_rgb10and12 |

### 3.5.1 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.

### 3.5.2 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")

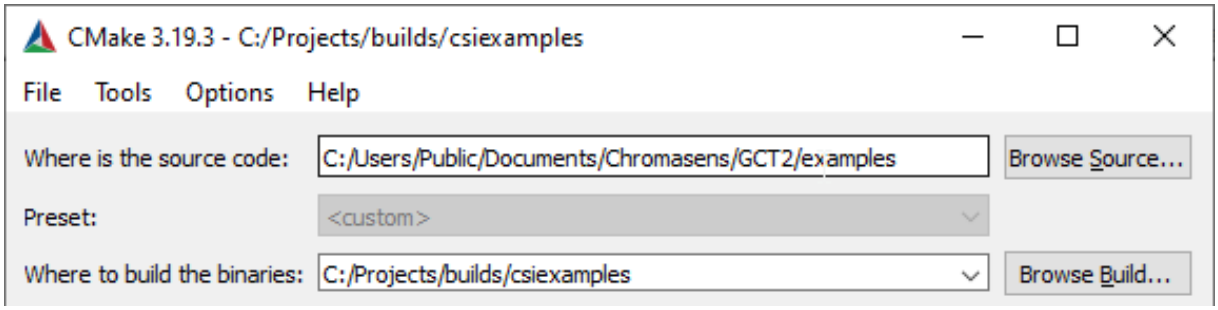



**Figure 3 cmake build project**

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

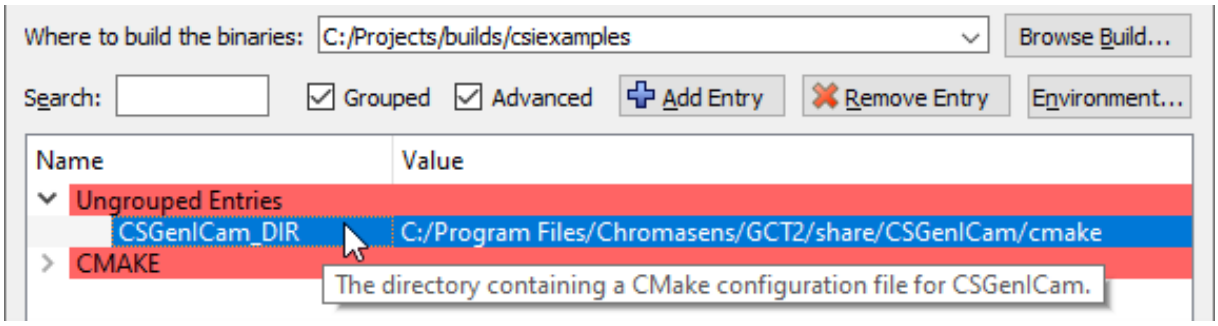



**Figure 4 cmake project configuration**

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.

# 4 Installation

## 4.1 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.

### 4.1.1 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`

## 4.2 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

### 4.2.1 Preparation

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

### 4.2.2 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.

### 4.2.3 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`

Chromasens GmbH Max-Stromeyer-Straße 116 Phone: +49 7531 876-0 www.chromasens.de 78467 Konstanz Fax: +49 7531 876-303 info@chromasens.de Germany

# 5 Data Structure Index

## 5.1 Data Structures

Here are the data structures with brief descriptions:



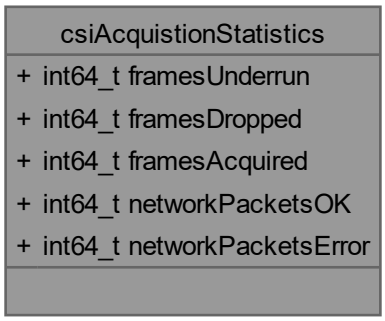
# 6 Data Structure Documentation

## 6.1 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

#### 6.1.1 Detailed Description

Structure containing acquisition statistics.

**Note**

Negative statistic values indicate that they are not available.

#### 6.1.2 Field Documentation

**framesAcquired**

`int64_t framesAcquired`

Total number of frames acquired in current acquisition.

**framesDropped**

`int64_t framesDropped`

Number of frames dropped during acquisition.

#### framesUnderrun

```
int64_t framesUnderrun
```

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

#### networkPacketsError

```
int64_t networkPacketsError
```

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

#### networkPacketsOK

```
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:

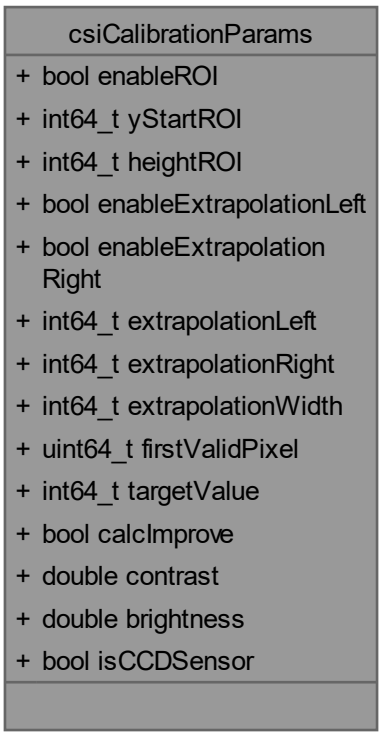
- csi.h

## 6.2 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

### 6.2.1 Detailed Description

Structure containing calibration parameters.

### 6.2.2 Field Documentation

#### brightness

```
double brightness
```

Brightness value for calibration.

#### calcImprove

```
bool calcImprove
```

To enable outlier suppression

#### contrast

```
double contrast
```

Contrast value for calibration.

#### enableExtrapolationLeft

```
bool enableExtrapolationLeft
```

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

### enableExtrapolationRight

```
bool enableExtrapolationRight
```

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

### enableROI

```
bool enableROI
```

To enable ROI based PRNU calibration data generation.

### extrapolationLeft

```
int64_t extrapolationLeft
```

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

### extrapolationRight

```
int64_t extrapolationRight
```

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

### extrapolationWidth

```
int64_t extrapolationWidth
```

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

### firstValidPixel

```
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.

### heightROI

```
int64_t heightROI
```

Height of the ROI used for PRNU calibration generation.

#### isCCDSensor

```
bool isCCDSensor = false
```

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

#### targetValue

```
int64_t targetValue
```

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

#### yStartROI

```
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:

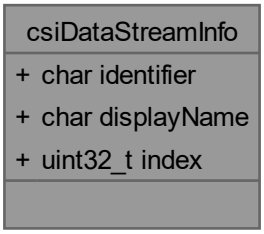
- csi.h

## 6.3 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

### 6.3.1 Detailed Description

Structure containing data stream infomation.

### 6.3.2 Field Documentation

#### displayName

```
char displayName[CSI_INFO_STRING_BUFFER_SIZE]
```

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

#### identifier

```
char identifier[CSI_INFO_STRING_BUFFER_SIZE]
```

Unique identifier of a data stream

#### index

```
uint32_t index
```

Internal index of the data stream

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

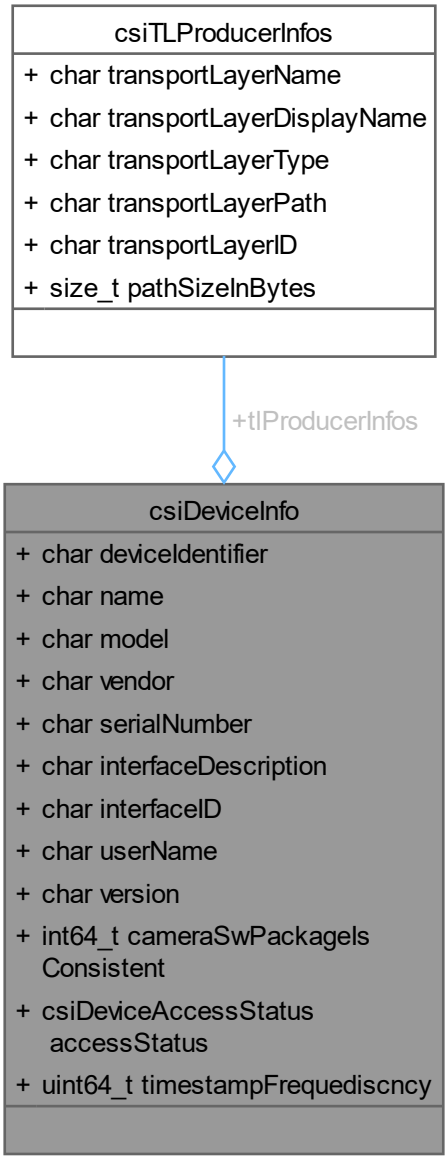
- csi.h

## 6.4 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

### 6.4.1 Detailed Description

Structure containing detailed device information.

### 6.4.2 Field Documentation

#### accessStatus

```
csiDeviceAccessStatus accessStatus
```

The current access status of the device, see csiDeviceAccessStatus

#### cameraSwPackageIsConsistent

```
int64_t cameraSwPackageIsConsistent
```

#### deviceIdentifier

```
char deviceIdentifier[CSI_INFO_STRING_BUFFER_SIZE]
```

Unique identifier of the device

#### interfaceDescription

```
char interfaceDescription[CSI_INFO_STRING_BUFFER_SIZE]
```

Name or description of the interface the device is connected to

#### interfaceID

```
char interfaceID[CSI_INFO_STRING_BUFFER_SIZE]
```

Unique identifier of the interface the device is connected to

#### model

```
char model[CSI_INFO_STRING_BUFFER_SIZE]
```

Model name of the device

### name

```
char name[CSI_INFO_STRING_BUFFER_SIZE]
```

Name of the device

### serialNumber

```
char serialNumber[CSI_INFO_STRING_BUFFER_SIZE]
```

Serial number of the device

### timestampFrequediscncy

```
uint64_t timestampFrequediscncy
```

Frequency of the timestamps coming from the device

### tlProducerInfos

```
csiTLProducerInfos tlProducerInfos
```

Information about the transport layer the device is connected to

### userName

```
char userName[CSI_INFO_STRING_BUFFER_SIZE]
```

Username when opening the device

### vendor

```
char vendor[CSI_INFO_STRING_BUFFER_SIZE]
```

Vendor of the device

### version

```
char version[CSI_INFO_STRING_BUFFER_SIZE]
```

Version of the device

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

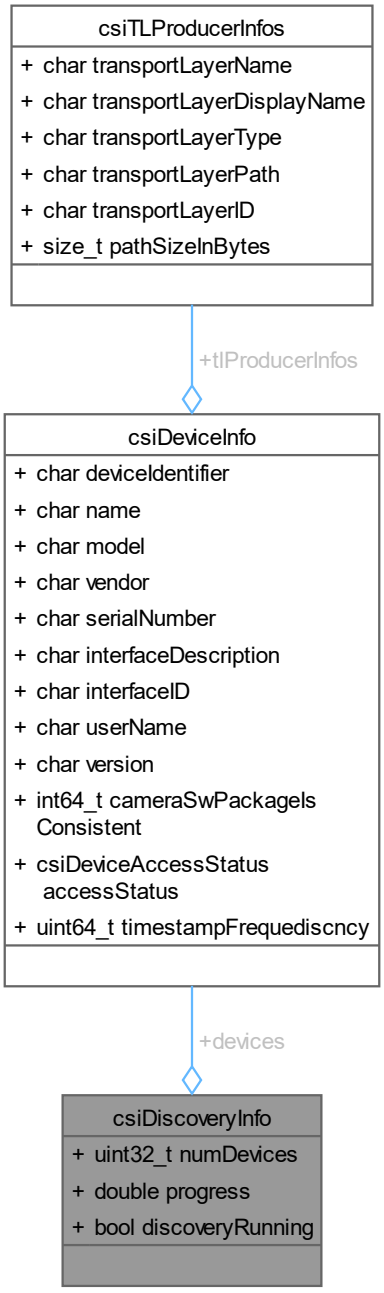
- csi.h

## 6.5 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]

### 6.5.1 Detailed Description

Structure containing information about device discovery.

### 6.5.2 Field Documentation

#### devices

```
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.

#### discoveryRunning

```
bool discoveryRunning
```

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

#### numDevices

```
uint32_t numDevices
```

Current number of devices found during discovery

#### progress

```
double progress
```

Discovery progress

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

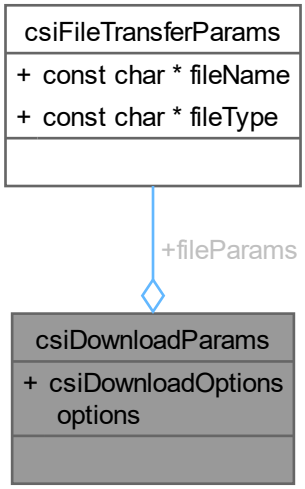
- csi.h

## 6.6 csiDownloadParams Struct Reference

Structure containing Download parameters and options.

`#include <csi/csi.h>`

Collaboration diagram for csiDownloadParams:




### Data Fields

- csiFileTransferParams fileParams
- csiDownloadOptions options

### 6.6.1 Detailed Description

Structure containing Download parameters and options.

### 6.6.2 Field Documentation

#### fileParams

```
csiFileTransferParams fileParams
```

#### options

```
csiDownloadOptions options
```

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

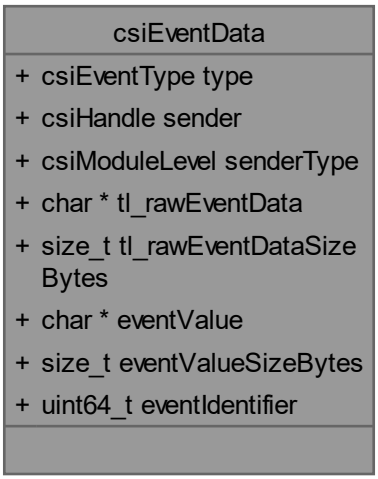
- csi.h

## 6.7 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

### 6.7.1 Detailed Description

Structure containing information about event.

### 6.7.2 Field Documentation

#### eventIdentifier

`uint64_t eventIdentifier`

Event identifier

### eventValue

```
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.

### eventValueSizeBytes

```
size_t eventValueSizeBytes
```

Size of the eventValue member in bytes

### sender

```
csiHandle sender
```

Handle to the sender of the event

### senderType

```
csiModuleLevel senderType
```

Module level of the sender handle, see csiModuleLevel

### tl_rawEventData

```
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.

### tl_rawEventDataSizeBytes

```
size_t tl_rawEventDataSizeBytes
```

Size of the eventData member in bytes

### type

```
csiEventType type
```

Type of the event, see csiEventType

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

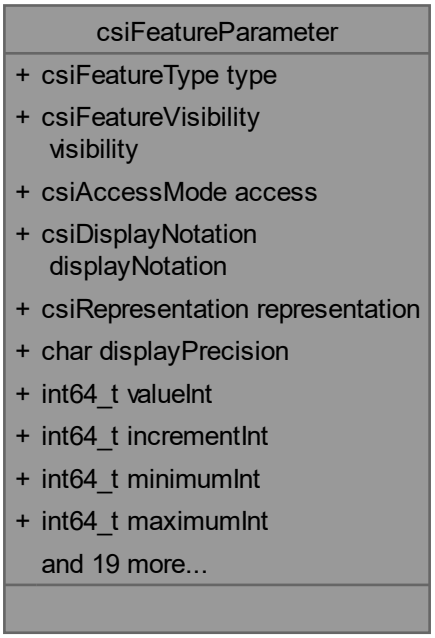
- csi.h

## 6.8 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

### 6.8.1 Detailed Description

Structure containing information about raw feature parameter.

### 6.8.2 Field Documentation

#### access

```
csiAccessMode access
```

How a feature can be accessed, see csiAccessMode

#### displayName

```
char displayName[CSI_INFO_STRING_BUFFER_SIZE]
```

Display name of the feature for UI display

#### displayNotation

```
csiDisplayNotation displayNotation
```

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

#### displayPrecision

```
char displayPrecision
```

Precision of floating-point value representation

#### enumCounter

```
uint32_t enumCounter
```

Number of elements in the enumeration feature

### enumIndex

```
int32_t enumIndex
```

The index of an enumeration entry

### featureRegAddress

```
int64_t featureRegAddress
```

Address of a register feature

### featureRegLength

```
size_t featureRegLength
```

Length of a register feature

### incrementFlt

```
double incrementFlt
```

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

### incrementInt

```
int64_t incrementInt
```

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

### isFeature

```
bool isFeature
```

Requested node is a feature

### isLittleEndian

```
bool isLittleEndian
```

### level

```
int64_t level
```

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

### maximumFlt

```
double maximumFlt
```

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

### maximumInt

```
int64_t maximumInt
```

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

### maximumStringLength

```
size_t maximumStringLength
```

Maximum length of the string feature value

### minimumFlt

```
double minimumFlt
```

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

### minimumInt

```
int64_t minimumInt
```

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

### name

```
char name[CSI_INFO_STRING_BUFFER_SIZE]
```

The name that identifies a feature

### representation

```
csiRepresentation representation
```

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

### tooltip

```
char tooltip[CSI_INFO_STRING_BUFFER_SIZE]
```

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

### type

```
csiFeatureType type
```

Data type of the feature, see csiFeatureType

### validValueSetInt

```
int64_t validValueSetInt[CSI_INFO_INT_BUFFER_SIZE]
```

### validValueSetSizeInt

```
size_t validValueSetSizeInt
```

### valueFlt

```
double valueFlt
```

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

### valueInt

```
int64_t valueInt
```

Value of the feature, in case of integer feature type

### valueStr

```
char valueStr[CSI_INFO_STRING_BUFFER_SIZE]
```

Value of the feature, in case of string feature type

### valueUnit

```
char valueUnit[CSI_INFO_STRING_BUFFER_SIZE]
```

String unit to append to the value representation in a GUI

### visibility

```
csiFeatureVisibility visibility
```

The visibility of a feature, see csiFeatureVisibility

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

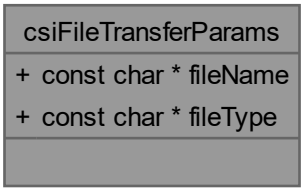
- csi.h

## 6.9 csiFileTransferParams Struct Reference

Structure containing file transfer parameter.

`#include <csi/csi.h>`

Collaboration diagram for csiFileTransferParams:




### Data Fields

- const char ∗ fileName
- const char ∗ fileType

### 6.9.1 Detailed Description

Structure containing file transfer parameter.

### 6.9.2 Field Documentation

#### fileName

```
const char* fileName
```

#### fileType

```
const char* fileType
```

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

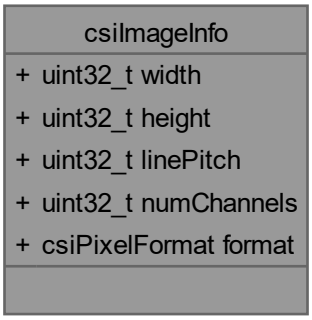
- csi.h

## 6.10 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

### 6.10.1 Detailed Description

Structure containing information about captured image.

### 6.10.2 Field Documentation

#### format

`csiPixelFormat format`

Pixel format of the image data, see csiPixelFormat

#### height

`uint32_t height`

Height of the image

#### linePitch

`uint32_t linePitch`

Line pitch of the image data in bytes

#### numChannels

`uint32_t numChannels`

Number of channels

#### width

`uint32_t width`

Width of the image

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

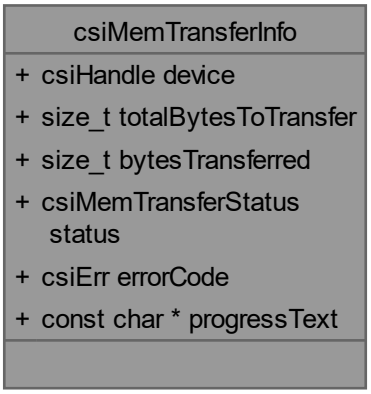
- csi.h

### 6.11 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

### 6.11.1 Detailed Description

Structure containing information about memory transfer.

### 6.11.2 Field Documentation

#### bytesTransferred

```
size_t bytesTransferred
```

Current number of bytes already transferred.

#### device

```
csiHandle device
```

Handle to the device where the transfer is running on.

#### errorCode

```
csiErr errorCode
```

Error code in case an error occurred.

#### progressText

```
const char* progressText
```

Progress information text.

#### status

```
csiMemTransferStatus status
```

Status of the memory transfer, see csiMemTransferStatus.

#### totalBytesToTransfer

```
size_t totalBytesToTransfer
```

Total number of bytes to be transferred.

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

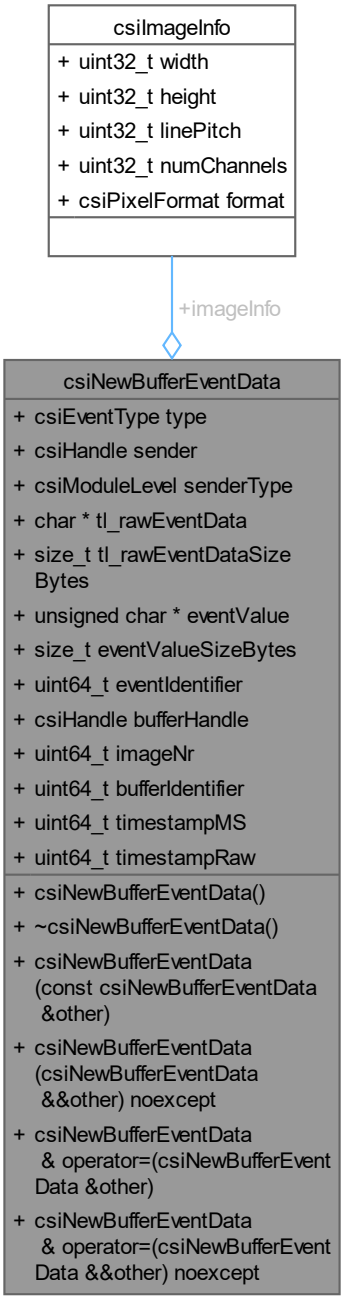
- csi.h

## 6.12 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 {}

### 6.12.1 Detailed Description

Structure containing data of an incoming buffer event.

### 6.12.2 Constructor & Destructor Documentation

**csiNewBufferEventData() [1/3]**

```
csiNewBufferEventData ( )
```

**~csiNewBufferEventData()**

```
~csiNewBufferEventData ( )
```

**csiNewBufferEventData() [2/3]**

```
csiNewBufferEventData (
            const csiNewBufferEventData & other )
```

**csiNewBufferEventData() [3/3]**

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

### 6.12.3 Member Function Documentation

**operator=() [1/2]**

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

### operator=() [2/2]

```
csiNewBufferEventData & operator= (
            csiNewBufferEventData & other )
```

### 6.12.4 Field Documentation

#### bufferHandle

```
csiHandle bufferHandle { 0 }
```

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

#### bufferIdentifier

```
uint64_t bufferIdentifier { 0 }
```

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

#### eventIdentifier

```
uint64_t eventIdentifier { 0 }
```

Unique identifier of this event

#### eventValue

```
unsigned char* eventValue { nullptr }
```

Pointer to the image data

#### eventValueSizeBytes

```
size_t eventValueSizeBytes { 0 }
```

Size of the image data in bytes

#### imageInfo

```
csiImageInfo imageInfo {}
```

Further image information, see csiImageInfo

### imageNr

```
uint64_t imageNr { 0 }
```

Number of the recorded image

### sender

```
csiHandle sender { CSI_EMPTY_HANDLE }
```

Sender of the event, a stream handle

### senderType

```
csiModuleLevel senderType { CSI::csiModuleLevel::CSI_UNKNOWN_MODULE }
```

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

### timestampMS

```
uint64_t timestampMS { 0 }
```

Timestamp of the image in milliseconds

### timestampRaw

```
uint64_t timestampRaw { 0 }
```

Raw timestamp of the image

### tl_rawEventData

```
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.

### tl_rawEventDataSizeBytes

```
size_t tl_rawEventDataSizeBytes { 0 }
```

Size of the eventData member in bytes

### type

```
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:

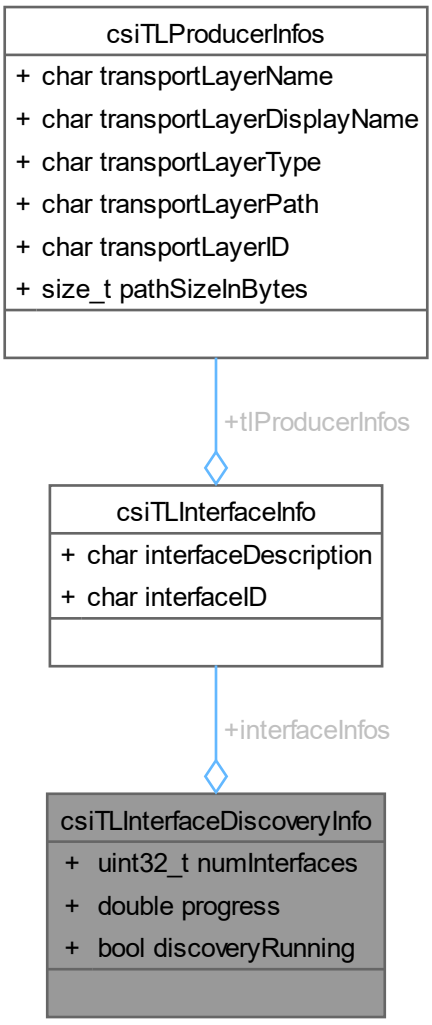
- csi.h

## 6.13 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]

### 6.13.1 Detailed Description

Structure containing information about transport layer interface discovery.

### 6.13.2 Field Documentation

**discoveryRunning**

```
bool discoveryRunning
```

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

**interfaceInfos**

```
csiTLInterfaceInfo interfaceInfos[CSI_TL_INTERFACE_COUNT]
```

A list of TL interfaces found so far.

**numInterfaces**

```
uint32_t numInterfaces
```

number of interfaces

**progress**

```
double progress
```

Discovery progress

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

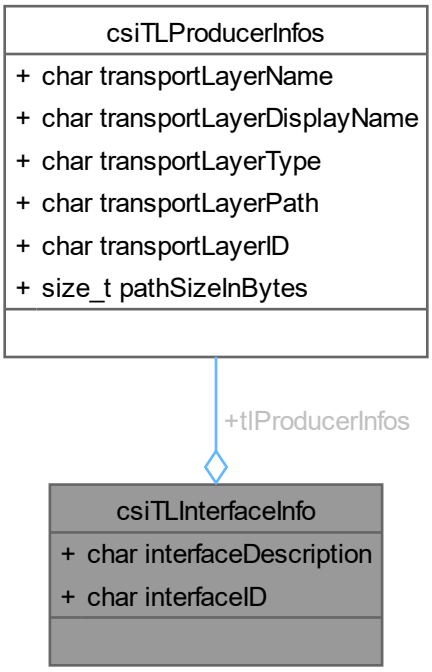
- csi.h

## 6.14 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

### 6.14.1 Detailed Description

Structure containing information about transport layer interface.

### 6.14.2 Field Documentation

#### interfaceDescription

`char interfaceDescription[CSI_INFO_STRING_BUFFER_SIZE]`

interface description

#### interfaceID

```
char interfaceID[CSI_INFO_STRING_BUFFER_SIZE]
```

interface id

#### tlProducerInfos

```
csiTLProducerInfos tlProducerInfos
```

information of TL producers

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

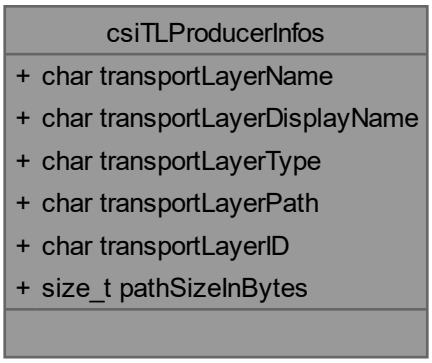
- csi.h

## 6.15 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

### 6.15.1 Detailed Description

Structure containing infomation about transport layer producer.

### 6.15.2 Field Documentation

#### pathSizeInBytes

```
size_t pathSizeInBytes
```

Length of the transport layer path.

#### transportLayerDisplayName

```
char transportLayerDisplayName[CSI_INFO_STRING_BUFFER_SIZE]
```

Display name of a transport layer for GUI representation.

#### transportLayerID

```
char transportLayerID[CSI_INFO_STRING_BUFFER_SIZE]
```

Unique identifier of the transport layer as string.

#### transportLayerName

```
char transportLayerName[CSI_INFO_STRING_BUFFER_SIZE]
```

Name of a transport layer.

#### transportLayerPath

```
char transportLayerPath[CSI_INFO_STRING_BUFFER_SIZE]
```

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

#### transportLayerType

```
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:

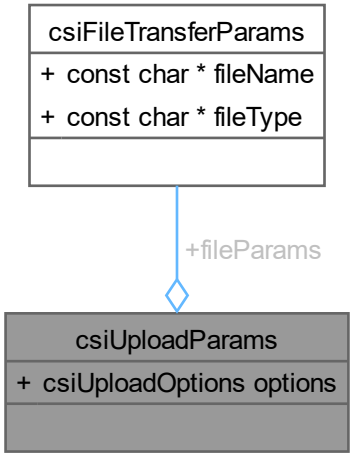
- csi.h

## 6.16 csiUploadParams Struct Reference

Structure containing Upload parameters and options.

`#include <csi/csi.h>`

Collaboration diagram for csiUploadParams:




### Data Fields

- csiFileTransferParams fileParams
- csiUploadOptions options

### 6.16.1 Detailed Description

Structure containing Upload parameters and options.

### 6.16.2 Field Documentation

#### fileParams

```
csiFileTransferParams fileParams
```

#### options

```
csiUploadOptions options
```

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

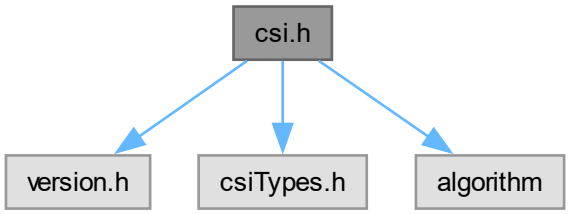
- csi.h

# 7 File Documentation

## 7.1 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 csiMemTransferUser↩ Data *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_PACKE = 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_UN
  }

  *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 *numberOf↩ StreamsOut)

  *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 **bufferInfo↩ Out, uint64_t timeoutMilliseconds)

  *Get the next image from the data stream.*
- CSI_DLL_EXPORT csiErr csiReleaseImage (csiHandle dataStream, const csiNewBufferEventData *buffer↩ Info)

  *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 *interface↩ HandleOut, 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.*

### 7.1.1 Macro Definition Documentation

#### CSI_COLOR_FORMAT

```
#define CSI_COLOR_FORMAT 0x02000000
```

#### CSI_DISCOVERY_INFO_DEVICE_COUNT

```
#define CSI_DISCOVERY_INFO_DEVICE_COUNT 16
```

#### CSI_INFITIE_TIME

```
#define CSI_INFITIE_TIME 0xffffffff
```

#### CSI_INFO_INT_BUFFER_SIZE

```
#define CSI_INFO_INT_BUFFER_SIZE 512
```

#### CSI_INFO_STRING_BUFFER_SIZE

```
#define CSI_INFO_STRING_BUFFER_SIZE 512
```

#### CSI_MONO_FORMAT

```
#define CSI_MONO_FORMAT 0x01000000
```

#### CSI_OCCUPY_10BIT

```
#define CSI_OCCUPY_10BIT 0x000A0000
```

### CSI_OCCUPY_12BIT

```
#define CSI_OCCUPY_12BIT 0x000C0000
```

### CSI_OCCUPY_14BIT

```
#define CSI_OCCUPY_14BIT 0x000E0000
```

### CSI_OCCUPY_16BIT

```
#define CSI_OCCUPY_16BIT 0x00100000
```

### CSI_OCCUPY_24BIT

```
#define CSI_OCCUPY_24BIT 0x00180000
```

### CSI_OCCUPY_30BIT

```
#define CSI_OCCUPY_30BIT 0x001E0000
```

### CSI_OCCUPY_32BIT

```
#define CSI_OCCUPY_32BIT 0x00200000
```

### CSI_OCCUPY_48BIT

```
#define CSI_OCCUPY_48BIT 0x00300000
```

### CSI_OCCUPY_64BIT

```
#define CSI_OCCUPY_64BIT 0x00400000
```

### CSI_OCCUPY_8BIT

```
#define CSI_OCCUPY_8BIT 0x00080000
```

### CSI_TL_INTERFACE_COUNT

```
#define CSI_TL_INTERFACE_COUNT 16
```

### 7.1.2 Typedef Documentation

#### CalibrationMode

```
typedef enum CalibrationMode CalibrationMode
```

#### CB_FEATURE_INVALIDATED_PFN

```
typedef void(* CB_FEATURE_INVALIDATED_PFN) (const char *featureName, void *userdata)
```

#### CB_OBJECT

```
typedef void* CB_OBJECT
```

#### csiAccessMode

```
typedef enum csiAccessMode csiAccessMode
```

#### csiAcquisitionMode

```
typedef enum csiAcquisitionMode csiAcquisitionMode
```

#### csiAcquistionStatistics

```
typedef struct csiAcquistionStatistics csiAcquistionStatistics
```

#### csiCalibrationParams

```
typedef struct csiCalibrationParams csiCalibrationParams
```

#### csiDataStreamInfo

```
typedef struct csiDataStreamInfo csiDataStreamInfo
```

#### csiDeviceAccessMode

```
typedef enum csiDeviceAccessMode csiDeviceAccessMode
```

#### csiDeviceAccessStatus

```
typedef enum csiDeviceAccessStatus csiDeviceAccessStatus
```

### csiDeviceInfo

```
typedef struct csiDeviceInfo csiDeviceInfo
```

### csiDiscoveryInfo

```
typedef struct csiDiscoveryInfo csiDiscoveryInfo
```

### csiDiscoveryInfoCallbackFunc

```
typedef void(* csiDiscoveryInfoCallbackFunc) (const csiDiscoveryInfo *discoveryInfo)
```

### csiDiscoveryTLInterfaceInfoCallbackFunc

```
typedef void(* csiDiscoveryTLInterfaceInfoCallbackFunc) (const csiTLInterfaceDiscoveryInfo
*discoveryInfo)
```

### csiDisplayNotation

```
typedef enum csiDisplayNotation csiDisplayNotation
```

### csiDownloadOptions

```
typedef enum csiDownloadOptions csiDownloadOptions
```

### csiErr

```
typedef enum csiErr csiErr
```

### csiEventData

```
typedef struct csiEventData csiEventData
```

### csiEventType

```
typedef enum csiEventType csiEventType
```

### csiFeatureParameter

```
typedef struct csiFeatureParameter csiFeatureParameter
```

### csiFeatureType

```
typedef enum csiFeatureType csiFeatureType
```

### csiFeatureVisibility

```
typedef enum csiFeatureVisibility csiFeatureVisibility
```

### csiFileTransferParams

```
typedef struct csiFileTransferParams csiFileTransferParams
```

### csiHandle

```
typedef uint64_t csiHandle
```

### csiImageInfo

```
typedef struct csiImageInfo csiImageInfo
```

### csiLogLevel

```
typedef enum csiLogLevel csiLogLevel
```

### csiLogSinkCallbackFunc

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

### csiMemTransferCallbackFunc

```
typedef bool(* csiMemTransferCallbackFunc) (const csiMemTransferInfo *info, struct csiMem↩
TransferUserData *userdata)
```

### csiMemTransferInfo

```
typedef struct csiMemTransferInfo csiMemTransferInfo
```

### csiMemTransferStatus

```
typedef enum csiMemTransferStatus csiMemTransferStatus
```

#### csiModuleLevel

```
typedef enum csiModuleLevel csiModuleLevel
```

#### csiNewBufferEventData

```
typedef struct CSI_DLL_EXPORT csiNewBufferEventData csiNewBufferEventData
```

#### csiPixelFormat

```
typedef enum csiPixelFormat csiPixelFormat
```

#### csiReferenceImgLoadMode

```
typedef enum csiReferenceImgLoadMode csiReferenceImgLoadMode
```

#### csiRepresentation

```
typedef enum csiRepresentation csiRepresentation
```

#### csiTLInterfaceDiscoveryInfo

```
typedef struct csiTLInterfaceDiscoveryInfo csiTLInterfaceDiscoveryInfo
```

#### csiTLInterfaceInfo

```
typedef struct csiTLInterfaceInfo csiTLInterfaceInfo
```

#### csiTLProducerInfos

```
typedef struct csiTLProducerInfos csiTLProducerInfos
```

#### csiUploadDownloadUploadParams

```
typedef struct csiDownloadParams csiUploadDownloadUploadParams
```

#### csiUploadOptions

```
typedef enum csiUploadOptions csiUploadOptions
```

#### csiUploadParams

```
typedef struct csiUploadParams csiUploadParams
```

### 7.1.3 Enumeration Type Documentation

#### CalibrationMode

```
enum CalibrationMode
```

Defines mode for calibration.

**Enumerator**

| | |
|---|---|
| DSNU_CALIBRATION | |
| PRNU_CALIBRATION | |

### csiAccessMode

`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. |

### csiAcquisitionMode

`enum csiAcquisitionMode`

Defines acquisition mode.

**Enumerator**

| | |
|---|---|
| CSI_ACQUISITION_SINGLE_FRAME | Acquire a single frame only |
| CSI_ACQUISITION_CONTINUOUS | Perform continuous frame acquisition |

### csiCalibrationLUT

`enum csiCalibrationLUT`

Defines calibration LUT options.

**Enumerator**

| | |
|---|---|
| CSI_DSNU_LUT1 | |
| CSI_DSNU_LUT2 | |
| CSI_PRNU_LUT1 | |
| CSI_PRNU_LUT2 | |

### csiDeviceAccessMode

`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. |

### csiDeviceAccessStatus

`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. |

### csiDisplayNotation

`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 |

### csiDownloadOptions

`enum csiDownloadOptions`

Defines download options.

**Enumerator**

| |
|---|
| CSI_DOWNLOAD_NO_OPTIONS |

### csiErr

`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 |
| csiFileOperationFailure | There was an error during file operation, see log for more information |
| csiFileOperationFatalError | 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) |
| csiWrongBufferSize | 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 |
| csiNotImplemented | 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 |
| csiResourceExhausted | 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 |

### csiEventType

`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 |

### csiFeatureType

`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 |

### csiFeatureVisibility

`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 |

**Enumerator**

| | |
|---|---|
| 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 |

### csiLogLevel

`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 | |

### csiMemTransferStatus

`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 |

### csiModuleLevel

`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) |

**Enumerator**

| | |
|---|---|
| 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 |

### csiPixelFormat

`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/genic

**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) |

### csiReferenceImgLoadMode

```
enum csiReferenceImgLoadMode
```

Defines mode of loading reference imgage.

**Enumerator**

| | |
|---|---|
| CSI_LOAD_REF_IMG_FROM_DISC | |
| CSI_ACQUIRE_REF_IMG_FROM_CAMERA | |

### csiRepresentation

```
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 |
| CSI_REPRESENTATION_IP | IP address representation |
| CSI_REPRESENTATION_MAC | Mac address representation |
| CSI_REPRESENTATION_UNDEFINED | Not defined, use default |

### csiUploadOptions

```
enum csiUploadOptions
```

Defines upload options.

**Enumerator**

| | |
|---|---|
| CSI_UPLOAD_NO_OPTIONS | |
| CSI_UPLOAD_IGNORE_CHECKSUM | |

## 7.1.4 Function Documentation

### csiAbortAcquisition()

```
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.

### csiBitsPerPixelFromFormat()

```
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.

### csiCheckAndReallocBuffers()

```
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.

### csiClose()

```
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.

### csiCloseDataStream()

```
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().

### csiCloseDevice()

```
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.

### csiCloseTLInterface()

```
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.

### csiCreateDataStream()

```
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.

### csiDiscoverDevices()

```
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)

### csiDiscoverTLInterfaces()

```
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).

### csiEventKill()

```
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.

### csiExecuteCommand()

```
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".

### csiFileDownloadToDevice()

```
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.

### csiFileDownloadToDeviceEx()

```
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.

### csiFileUploadFromDevice()

```
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.

### csiFileUploadFromDeviceEx()

```
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.

### csiGenerateAndSaveCalibrationDataToFile()

```
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 image↩ FilePath, the argument "imgLoadMode" of type "csiReferencingLoadMode" determines which input image data should be used for calibration generation.

### csiGenerateAndUploadCalibrationData()

```
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 image↩ FilePath, the argument "imgLoadMode" determines which input image data should be used for calibration generation.

### csiGenerateReferenceImage()

```
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 image↩ FilePath, the argument "imgLoadMode" of type "csiReferencingLoadMode" determines which input image data should be used for calibration generation.

### csiGetAcquisitionStatistics()

```
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.

### csiGetDataStreamInfo()

```
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.

### csiGetDeviceDataStreamInfo()

```
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.

### csiGetDeviceInfo()

```
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

### csiGetErrorDescription()

```
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.

### csiGetFeatureAccessMode()

```
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.

### csiGetFeatureBool()

```
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.

### csiGetFeatureEnum()

```
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. |

**Parameters**

| | | |
|---|---|---|
| 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:<br>• `enumEntries`: The number of enumeration entries which are available for this enumeration feature.<br>• `currentEntry`: Currently selected enumeration index.<br>• `values[]`: Name of the enumeration entries. |

**Returns**

csiSuccess or an error defined in the csiErr-Enum.

### csiGetFeatureEnumEntryByIndex()

```
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.

### csiGetFeatureEnumEntryByName()

```
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.

### csiGetFeatureEnumEntryCount()

```
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.

### csiGetFeatureFloat()

```
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.

### csiGetFeatureInt()

```
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.

### csiGetFeatureParameter()

```
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.

### csiGetFeatureReg()

```
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.

### csiGetFeatureString()

```
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.

### csiGetLibraryVersion()

```
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.

### csiGetNextImage()

```
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().

### csiGetNumberOfDataStreams()

```
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.

### csiGetNumberOfTLProducers()

```
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.

### csiGetTLProducerInfobyFilePath()

```
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.

### csiGetTLProducerPathByIndex()

```
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.

### csiGetUpdateFileType()

```
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.

### csiInit()

```
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.

### csiIsCommandActive()

```
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.

### csiIsFeatureCachingDisabled()

```
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.

### csiIsModuleLittleEndian()

```
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.

### csiIterateFeatureTree()

```
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.

### csiOpenDevice()

```
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:<br>• CSI_DEV_MODE_EXCLUSIVE: Only this process can communicate with the camera.<br>• CSI_DEV_MODE_READ: Camera-parameters can be read and images can be acquired.<br>• 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.

### csiOpenTLInterface()

```
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.

### csiReadMemory()

```
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.

### csiRegisterEvent()

```
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.

### csiRegisterInvalidateCB()

```
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)

### csiReleaseImage()

```
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.

### csiReset()

```
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.

### csiSetFeatureBool()

```
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.

### csiSetFeatureCachingDisabled()

```
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.

### csiSetFeatureEnum()

```
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.

### csiSetFeatureFloat()

```
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.

### csiSetFeatureInt()

```
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.

### csiSetFeatureReg()

```
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.

### csiSetFeatureString()

```
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 |
| GenICam_SDK | | 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.

### csiStartAcquisition()

```
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.

### csiStopAcquisition()

```
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.

### csiUnregisterEvent()

```
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().

### csiUnRegisterInvalidateCB()

```
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.

### csiWaitForEvent()

```
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.

### csiWriteMemory()

```
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.

## 7.2 csi.h

Go to the documentation of this file.

```
00001
00002 #ifndef CSI_CSIF_H_
00003 #define CSI_CSIF_H_
```

```
00004
00005 #include "version.h"
00006 #include "csiTypes.h"
00007
00008 #include <algorithm>
00009
00010 #ifdef __cplusplus
00011 #define CSI_DEFAULT_PARAM_MODULE = CSI_DEVICE_MODULE
00012 #define CSI_DEFAULT_PARAM_NULL = NULL
00013 #define CSI_DEFAULT_PARAM_ZERO = 0
00014 #define CSI_DEFAULT_PARAM_FALSE = false
00015 #endif
00016
00017 #ifdef __cplusplus
00018 extern "C"
00019 {
00020     namespace CSI {
00021 #endif // __cplusplus
00022
00023 #define CSI_INFO_STRING_BUFFER_SIZE 512
00024 #define CSI_INFO_INT_BUFFER_SIZE 512
00025 #define CSI_DISCOVERY_INFO_DEVICE_COUNT 16
00026 #define CSI_TL_INTERFACE_COUNT 16
00027 #define CSI_INFITIE_TIME 0xffffffff
00028
00029 typedef uint64_t csiHandle;
00030 static const csiHandle CSI_EMPTY_HANDLE = 0;
00031
00032 // Default discovery time which should enable a safe discovery of the camera
00033 static const int CSI_DEFAULT_DISCOVERY_TIME_MS = 400;
00034
00035 #define CSI_MONO_FORMAT     0x01000000
00036 #define CSI_COLOR_FORMAT    0x02000000
00037 #define CSI_OCCUPY_8BIT     0x00080000
00038 #define CSI_OCCUPY_10BIT    0x000A0000
00039 #define CSI_OCCUPY_12BIT    0x000C0000
00040 #define CSI_OCCUPY_14BIT    0x000E0000
00041 #define CSI_OCCUPY_16BIT    0x00100000
00042 #define CSI_OCCUPY_24BIT    0x00180000
00043 #define CSI_OCCUPY_30BIT    0x001E0000
00044 #define CSI_OCCUPY_32BIT    0x00200000
00045 #define CSI_OCCUPY_48BIT    0x00300000
00046 #define CSI_OCCUPY_64BIT    0x00400000
00047
00055         typedef enum csiPixelFormat
00056         {
00057             CSI_PIX_FORMAT_UNKNOWN = 0x00000000,
00058
00060             CSI_PIX_FORMAT_MONO8 =          CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x001,
00061             CSI_PIX_FORMAT_MONO10 =            CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x003,
00062             CSI_PIX_FORMAT_MONO10_PACKED =  CSI_MONO_FORMAT | CSI_OCCUPY_10BIT | 0x046,
00063             CSI_PIX_FORMAT_MONO12 =            CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x005,
00064             CSI_PIX_FORMAT_MONO12_PACKED =  CSI_MONO_FORMAT | CSI_OCCUPY_12BIT | 0x047,
00065             CSI_PIX_FORMAT_MONO16 =            CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x007,
00068             CSI_PIX_FORMAT_RGB8 =           CSI_COLOR_FORMAT | CSI_OCCUPY_24BIT | 0x014,
00069             CSI_PIX_FORMAT_RGB10_PACKED =    CSI_COLOR_FORMAT | CSI_OCCUPY_32BIT | 0x01D,
00070             CSI_PIX_FORMAT_RGB10    =       CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x018,
00071             CSI_PIX_FORMAT_BGR10 =          CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x019,
00072             CSI_PIX_FORMAT_RGB12 =          CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x01A,
00073             CSI_PIX_FORMAT_BGR12 =          CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x01B,
00074             CSI_PIX_FORMAT_RGBA8 =          CSI_COLOR_FORMAT | CSI_OCCUPY_32BIT | 0x016,
00075             CSI_PIX_FORMAT_BGRA8 =          CSI_COLOR_FORMAT | CSI_OCCUPY_32BIT | 0x017,
00076             CSI_PIX_FORMAT_BGR8 =           CSI_COLOR_FORMAT | CSI_OCCUPY_24BIT | 0x015,
00077             CSI_PIX_FORMAT_RGB16 =          CSI_COLOR_FORMAT | CSI_OCCUPY_48BIT | 0x033,
00078             CSI_PIX_FORMAT_RGBA10 =            CSI_COLOR_FORMAT | CSI_OCCUPY_64BIT | 0x05F,
00079             CSI_PIX_FORMAT_RGBA12 =            CSI_COLOR_FORMAT | CSI_OCCUPY_64BIT | 0x061,
00080             CSI_PIX_FORMAT_RGBA16 =            CSI_COLOR_FORMAT | CSI_OCCUPY_64BIT | 0x064,
00082             // Bayer
00083             CSI_PIX_FORMAT_BayerGR8 =       CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x008,
00084             CSI_PIX_FORMAT_BayerRG8 =       CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x009,
00086             CSI_PIX_FORMAT_BayerGB8 =       CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x00A,
00087             CSI_PIX_FORMAT_BayerBG8 =       CSI_MONO_FORMAT | CSI_OCCUPY_8BIT | 0x00B,
00089             CSI_PIX_FORMAT_BayerGR12 =      CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x010,
00090             CSI_PIX_FORMAT_BayerRG12 =      CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x011,
00092             CSI_PIX_FORMAT_BayerGB12 =      CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x012,
00093             CSI_PIX_FORMAT_BayerBG12 =      CSI_MONO_FORMAT | CSI_OCCUPY_16BIT | 0x013
00094         } csiPixelFormat;
00095
00100         typedef enum csiDeviceAccessMode {
00101             CSI_DEV_MODE_UNKNOWN = 0x00,
00102             CSI_DEV_MODE_NONE = 0x01,
00103             CSI_DEV_MODE_EXCLUSIVE,
00104             CSI_DEV_MODE_READ,
00105             CSI_DEV_MODE_CONTROL
00106         } csiDeviceAccessMode;
00107
00108
```

```
00113         typedef enum csiDeviceAccessStatus {
00114             CSI_DEV_ACCESS_STATUS_UNKNOWN = 0x00,
00115             CSI_DEV_ACCESS_STATUS_READWRITE = 0x01,
00116             CSI_DEV_ACCESS_STATUS_READONLY = 0x02,
00117             CSI_DEV_ACCESS_STATUS_NOACCESS = 0x03,
00118             CSI_DEV_ACCESS_STATUS_BUSY = 0x04,
00119             CSI_DEV_ACCESS_STATUS_OPEN_READWRITE = 0x05,
00120             CSI_DEV_ACCESS_STATUS_OPEN_READ = 0x06
00121         } csiDeviceAccessStatus;
00122
00123
00128         typedef enum csiFeatureType {
00129             CSI_UNKNOWN_TYPE,
00130             CSI_BOOLEAN_TYPE,
00131             CSI_INT_TYPE,
00132             CSI_FLOAT_TYPE,
00133             CSI_STRING_TYPE,
00134             CSI_ENUMERATION_TYPE,
00135             CSI_ENUMENTRY_TYPE,
00136             CSI_CATEGORY,
00137             CSI_COMMAND,
00138             CSI_REGISTER,
00139             CSI_PORT
00140         } csiFeatureType;
00141
00142
00147         typedef enum csiAccessMode {
00148             CSI_ACCESS_UNKNOWN,
00149             CSI_ACCESS_NOT_AVAILABLE,
00150             CSI_ACCESS_READ_ONLY,
00151             CSI_ACCESS_READ_WRITE,
00152             CSI_ACCESS_WRITE_ONLY
00153         } csiAccessMode;
00154
00155
00160         typedef enum csiFeatureVisibility {
00161             CSI_VISIBILITY_BEGINNER = 1,
00162             CSI_VISIBILITY_EXPERT,
00163             CSI_VISIBILITY_GURU,
00164             CSI_VISIBILITY_DEVELOPER,
00165             CSI_VISIBILITY_INVISIBLE
00166         } csiFeatureVisibility;
00167
00168
00173         typedef enum csiModuleLevel {
00174             CSI_UNKNOWN_MODULE,
00175             CSI_TRANSPORTLAYER_MODULE,
00176             CSI_INTERFACE_MODULE,
00177             CSI_DEVICE_MODULE,
00178             CSI_LOCAL_DEVICE_MODULE,
00179             CSI_STREAM_MODULE,
00180             CSI_BUFFER_MODULE
00181         } csiModuleLevel;
00182
00183
00188         typedef enum csiDisplayNotation {
00189             CSI_NOTATION_AUTOMATIC,
00190             CSI_NOTATION_FIXED,
00191             CSI_NOTATION_SCIENTIFIC
00192         } csiDisplayNotation;
00193
00194
00199         typedef enum csiRepresentation {
00200             CSI_REPRESENTATION_LINEAR,
00201             CSI_REPRESENTATION_LOGARITHMIC,
00202             CSI_REPRESENTATION_BOOLEAN,
00203             CSI_REPRESENTATION_PURENUMBER,
00204             CSI_REPRESENTATION_HEX,
00205             CSI_REPRESENTATION_IP,
00206             CSI_REPRESENTATION_MAC,
00207             CSI_REPRESENTATION_UNDEFINED
00208         } csiRepresentation;
00209
00210
00215         typedef struct csiFeatureParameter {
00216             csiFeatureType type;
00217             csiFeatureVisibility visibility;
00218             csiAccessMode access;
00219             csiDisplayNotation displayNotation;
00220             csiRepresentation representation;
00221             char displayPrecision;
00223             int64_t valueInt;
00224             int64_t incrementInt;
00225             int64_t minimumInt;
00226             int64_t maximumInt;
00227             int64_t validValueSetInt[CSI_INFO_INT_BUFFER_SIZE];
00228             size_t  validValueSetSizeInt;
```

```
00229
00230             double valueFlt;
00231             double incrementFlt;
00232             double minimumFlt;
00233             double maximumFlt;
00235             char valueStr[CSI_INFO_STRING_BUFFER_SIZE];
00236             size_t maximumStringLength;
00238             int64_t level;
00239             uint32_t enumCounter;
00240             int32_t enumIndex;
00242             char displayName[CSI_INFO_STRING_BUFFER_SIZE];
00243             char name[CSI_INFO_STRING_BUFFER_SIZE];
00244             char tooltip[CSI_INFO_STRING_BUFFER_SIZE];
00246             char valueUnit[CSI_INFO_STRING_BUFFER_SIZE];
00248             size_t featureRegLength;
00249             int64_t featureRegAddress;
00250             bool isFeature;
00251             bool isLittleEndian;
00252
00253         } csiFeatureParameter;
00254
00255
00260         typedef enum csiLogLevel
00261         {
00262             CSI_LOGLEVEL_NONE = 0,
00263             CSI_LOGLEVEL_ERROR = 1,
00264             CSI_LOGLEVEL_WARN = 2,
00265             CSI_LOGLEVEL_INFO = 4,
00266             CSI_LOGLEVEL_DEBUG = 8,
00267             CSI_LOGLEVEL_TRACE = 16,
00268         } csiLogLevel;
00269
00270
00275         typedef enum csiErr {
00276             csiSuccess = 0,
00277             csiNotInitialized = -100,
00278             csiInvalidState = -101,
00279             csiNotOpened = -102,
00280             csiNoImageDataAvailable = -103,
00281             csiNotFound = -104,
00282             csiInvalidParameter = -105,
00283             csiNotAvailable = -106,
00284             csiFunctionNotAvailable = -107,
00285             csiTimeout = -108,
00286             csiAborted = -109,
00287             csiFileOperationFailure = -110,
00288             csiFileOperationFatalError = -111,
00289             csiNoAccess = -112,
00290             csiWrongBufferSize = -113,
00291             csiInvalidBuffer = -114,
00292             csiResourceInUse = -115,
00293             csiNotImplemented = -116,
00294             csiInvalidHandle = -117,
00295             csiIOError = -118,
00296             csiParsingError = -119,
00297             csiInvalidValue = -120,
00298             csiResourceExhausted = -121,
00299             csiOutOfMemory = -122,
00300             csiBusy = -123,
00301             csiUnknown = -200,
00302             csiCustomErr = -0x0f000000
00303         } csiErr;
00304
00305
00310         typedef enum csiEventType
00311         {
00312             CSI_EVT_NEWIMAGEDATA = 0x00,
00313             CSI_EVT_ERROR = 0x01,
00314             CSI_EVT_MODULE = 0x02,
00315             CSI_EVT_FEATURE_INVALIDATE = 0x03,
00316             CSI_EVT_FEATURE_CHANGE = 0x04,
00317             CSI_EVT_REMOTE_DEVICE = 0x05,
00318             CSI_EVT_CUSTOM = 0x1000
00319         } csiEventType;
00320
00321
00326         typedef enum csiAcquisitionMode {
00327             CSI_ACQUISITION_SINGLE_FRAME = 0x00000001,
00328             CSI_ACQUISITION_CONTINUOUS = 0xFFFFFFFF
00329         } csiAcquisitionMode;
00330
00331
00336         typedef enum csiMemTransferStatus
00337         {
00338             csiTransferStatusInit,
00339             csiTransferStatusInProgress,
00340             csiTransferStatusInProgressWaiting,
```

```
00341             csiTransferStatusFinishSucess,
00342             csiTransferStatusFinishError,
00343             csiTransferStatusCancelOnError
00345         } csiMemTransferStatus;
00346
00347
00352         typedef struct csiMemTransferInfo {
00353             csiHandle device;
00354             size_t totalBytesToTransfer;
00355             size_t bytesTransferred;
00356             csiMemTransferStatus status;
00357             csiErr errorCode;
00358             const char* progressText;
00359         } csiMemTransferInfo;
00360
00361
00362         struct csiEventUserData;
00363         struct csiMemTransferUserData;
00364         struct csiLogUserData;
00365
00366
00371         typedef struct csiTLProducerInfos {
00372             char transportLayerName[CSI_INFO_STRING_BUFFER_SIZE];
00373             char transportLayerDisplayName[CSI_INFO_STRING_BUFFER_SIZE];
00374             char transportLayerType[CSI_INFO_STRING_BUFFER_SIZE];
00375             char transportLayerPath[CSI_INFO_STRING_BUFFER_SIZE];
00376             char transportLayerID[CSI_INFO_STRING_BUFFER_SIZE];
00377             size_t pathSizeInBytes;
00378         } csiTLProducerInfos;
00379
00380
00385         typedef struct csiDeviceInfo
00386         {
00387             char deviceIdentifier[CSI_INFO_STRING_BUFFER_SIZE];
00388             char name[CSI_INFO_STRING_BUFFER_SIZE];
00389             char model[CSI_INFO_STRING_BUFFER_SIZE];
00390             char vendor[CSI_INFO_STRING_BUFFER_SIZE];
00391             char serialNumber[CSI_INFO_STRING_BUFFER_SIZE];
00392             char interfaceDescription[CSI_INFO_STRING_BUFFER_SIZE];
00393             char interfaceID[CSI_INFO_STRING_BUFFER_SIZE];
00394             char userName[CSI_INFO_STRING_BUFFER_SIZE];
00395             char version[CSI_INFO_STRING_BUFFER_SIZE];
00396             int64_t cameraSwPackageIsConsistent;
00397             csiTLProducerInfos tlProducerInfos;
00398             csiDeviceAccessStatus accessStatus;
00399             uint64_t timestampFrequediscncy;
00400         } csiDeviceInfo;
00401
00402
00407         typedef struct csiDiscoveryInfo {
00408             uint32_t numDevices;
00409             double progress;
00410             bool discoveryRunning;
00411             csiDeviceInfo devices[CSI_DISCOVERY_INFO_DEVICE_COUNT];
00412         } csiDiscoveryInfo;
00413
00418         typedef struct csiTLInterfaceInfo
00419         {
00420             char interfaceDescription[CSI_INFO_STRING_BUFFER_SIZE];
00421             char interfaceID[CSI_INFO_STRING_BUFFER_SIZE];
00422             csiTLProducerInfos tlProducerInfos;
00423         } csiTLInterfaceInfo;
00424
00429         typedef struct csiTLInterfaceDiscoveryInfo
00430         {
00431             uint32_t numInterfaces;
00432             double progress;
00433             bool discoveryRunning;
00434             csiTLInterfaceInfo interfaceInfos[CSI_TL_INTERFACE_COUNT];
00435         } csiTLInterfaceDiscoveryInfo;
00436
00437
00442         typedef struct csiDataStreamInfo {
00443             char identifier[CSI_INFO_STRING_BUFFER_SIZE];
00444             char displayName[CSI_INFO_STRING_BUFFER_SIZE];
00445             uint32_t index;
00446         } csiDataStreamInfo;
00447
00448
00453         typedef struct csiImageInfo {
00454             uint32_t width;
00455             uint32_t height;
00456             uint32_t linePitch;
00457             uint32_t numChannels;
00458             csiPixelFormat format;
00459         } csiImageInfo;
00460
```

```
00461
00466         typedef struct csiEventData {
00467             csiEventType type;
00468             csiHandle sender;
00469             csiModuleLevel senderType;
00470             char* tl_rawEventData;
00471             size_t tl_rawEventDataSizeBytes;
00472             char* eventValue;
00473             size_t eventValueSizeBytes;
00474             uint64_t eventIdentifier;
00475         } csiEventData;
00476
00481         typedef struct CSI_DLL_EXPORT csiNewBufferEventData
00482         {
00483             csiNewBufferEventData();
00484             ~csiNewBufferEventData();
00485
00486             csiNewBufferEventData(const csiNewBufferEventData& other);
00487             csiNewBufferEventData(csiNewBufferEventData&& other) noexcept;
00488             csiNewBufferEventData& operator=(csiNewBufferEventData& other);
00489             csiNewBufferEventData& operator=(csiNewBufferEventData&& other) noexcept;
00490
00491             csiEventType type{ CSI::csiEventType::CSI_EVT_NEWIMAGEDATA };
00492             csiHandle sender{ CSI_EMPTY_HANDLE };
00493             csiModuleLevel senderType{ CSI::csiModuleLevel::CSI_UNKNOWN_MODULE };
00494             char* tl_rawEventData{ nullptr };
00495             size_t tl_rawEventDataSizeBytes{ 0 };
00496             unsigned char* eventValue{ nullptr };
00497             size_t eventValueSizeBytes{ 0 };
00498             uint64_t eventIdentifier{ 0 };
00499             csiHandle bufferHandle{ 0 };
00500             uint64_t imageNr{ 0 };
00501             uint64_t bufferIdentifier{ 0 };
00502             uint64_t timestampMS{ 0 };
00503             uint64_t timestampRaw{ 0 };
00504             csiImageInfo imageInfo{};
00505       } csiNewBufferEventData;
00506
00507
00515         typedef struct csiAcquistionStatistics
00516         {
00517             int64_t framesUnderrun;
00518             int64_t framesDropped;
00519             int64_t framesAcquired;
00520             int64_t networkPacketsOK;
00521             int64_t networkPacketsError;
00523         } csiAcquistionStatistics;
00524
00529         typedef enum csiDownloadOptions
00530         {
00531             CSI_DOWNLOAD_NO_OPTIONS = 0
00532         } csiDownloadOptions;
00533
00534
00539         typedef enum csiUploadOptions
00540         {
00541             CSI_UPLOAD_NO_OPTIONS = 0,
00542             CSI_UPLOAD_IGNORE_CHECKSUM = 1
00543         } csiUploadOptions;
00544
00550         typedef struct alignas(uint64_t) csiFileTransferParams
00551         {
00552             const char* fileName;
00553             const char* fileType;
00554         } csiFileTransferParams;
00555
00561         typedef struct alignas(uint64_t) csiDownloadParams
00562         {
00563             csiFileTransferParams fileParams;
00564             csiDownloadOptions options;
00565         } csiUploadDownloadUploadParams;
00566
00572         typedef struct alignas(uint64_t) csiUploadParams
00573         {
00574             csiFileTransferParams fileParams;
00575             csiUploadOptions options;
00576         } csiUploadParams;
00577
00582         enum csiCalibrationLUT
00583         {
00584             CSI_DSNU_LUT1 = 1,
00585             CSI_DSNU_LUT2,
00586             CSI_PRNU_LUT1,
00587             CSI_PRNU_LUT2
00588         };
00589
00594         typedef enum csiReferenceImgLoadMode
```

```
00595         {
00596             CSI_LOAD_REF_IMG_FROM_DISC,
00597             CSI_ACQUIRE_REF_IMG_FROM_CAMERA
00598         } csiReferenceImgLoadMode;
00599 
00600 
00605         typedef struct csiCalibrationParams
00606         {
00607             bool enableROI;
00608             int64_t yStartROI;
00609             int64_t heightROI;
00610             bool enableExtrapolationLeft;
00611             bool enableExtrapolationRight;
00612             int64_t extrapolationLeft;
00613             int64_t extrapolationRight;
00614             int64_t extrapolationWidth;
00615             uint64_t firstValidPixel;
00617             int64_t targetValue;
00618             bool calcImprove;
00619             double contrast;
00620             double brightness;
00621             bool isCCDSensor = false;
00622         } csiCalibrationParams;
00623 
00628         typedef enum CalibrationMode
00629          {
00630             DSNU_CALIBRATION = 1,
00631             PRNU_CALIBRATION
00632         } CalibrationMode;
00633 
00634         typedef void *CB_OBJECT;
00635         typedef void (*CB_FEATURE_INVALIDATED_PFN)(const char *featureName, void* userdata);
00636 
00637         typedef void(*csiDiscoveryInfoCallbackFunc)(const csiDiscoveryInfo* discoveryInfo);
00638         typedef void (*csiDiscoveryTLInterfaceInfoCallbackFunc)(const csiTLInterfaceDiscoveryInfo*
      discoveryInfo);
00639         typedef bool(*csiMemTransferCallbackFunc)(const csiMemTransferInfo* info, struct
      csiMemTransferUserData* userdata);
00640         typedef void(*csiLogSinkCallbackFunc)(csiLogLevel, const char* message, struct csiLogUserData*
      userdata);
00641 
00642         // Global general functions
00643 
00656         CSI_DLL_EXPORT csiErr csiInit(csiLogLevel logLvl, csiLogSinkCallbackFunc logCallbackFunc
      CSI_DEFAULT_PARAM_NULL, csiLogUserData* userdata CSI_DEFAULT_PARAM_NULL);
00657 
00663         CSI_DLL_EXPORT csiErr csiClose();
00664 
00672         CSI_DLL_EXPORT csiErr csiReset();
00673 
00691         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);
00692 
00705         CSI_DLL_EXPORT csiErr csiGetDeviceInfo(uint32_t deviceIndex, csiDeviceInfo* deviceInfoOut);
00706 
00731         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);
00732 
00733 
00747         CSI_DLL_EXPORT csiErr csiGetNumberOfTLProducers(int32_t *numTLProducers);
00748 
00757         CSI_DLL_EXPORT csiErr csiGetTLProducerPathByIndex(char *transportLayerPath, size_t bufferSize,
      uint32_t index);
00758 
00772         CSI_DLL_EXPORT csiErr csiGetTLProducerInfobyFilePath(csiTLProducerInfos* tlProducerInfos,
      const char* producerName);
00773 
00774         // Device functions
00775 
00804         CSI_DLL_EXPORT csiErr csiOpenDevice(const char* deviceIdentifier, const char* interfaceID,
      csiHandle* deviceHandleOut, uint64_t timeoutMilliseconds, csiDeviceAccessMode openMode);
00805 
00818         CSI_DLL_EXPORT csiErr csiCloseDevice(csiHandle deviceHandle);
00819 
00827         CSI_DLL_EXPORT csiErr csiCheckAndReallocBuffers(csiHandle deviceHandle);
00828 
00843         CSI_DLL_EXPORT csiErr csiStartAcquisition(csiHandle deviceHandle, csiAcquisitionMode mode);
00844 
00856         CSI_DLL_EXPORT csiErr csiStopAcquisition(csiHandle deviceHandle);
00857 
00858 
00868         CSI_DLL_EXPORT csiErr csiAbortAcquisition(csiHandle deviceHandle);
00869 
00870         // Data Stream functions
```

```
00888         CSI_DLL_EXPORT csiErr csiGetNumberOfDataStreams(csiHandle deviceHandle, uint32_t*
      numberOfStreamsOut);
00889
00904         CSI_DLL_EXPORT csiErr csiGetDataStreamInfo(csiHandle deviceHandle, uint32_t dsIndex,
      csiDataStreamInfo* dataStreamInfoOut);
00905
00920         CSI_DLL_EXPORT csiErr csiGetDeviceDataStreamInfo(csiHandle moduleHandle, uint32_t dsIndex,
      csiDataStreamInfo* dataStreamInfoOut);
00921
00944         CSI_DLL_EXPORT csiErr csiCreateDataStream(csiHandle deviceHandle, uint32_t dsIndex, csiHandle*
      dataStreamOut, uint32_t numberOfBuffers, size_t bufferSize CSI_DEFAULT_PARAM_ZERO);
00945
00960         CSI_DLL_EXPORT csiErr csiCloseDataStream(csiHandle dataStream);
00961
00978         CSI_DLL_EXPORT csiErr csiRegisterEvent(csiHandle moduleHandle, csiEventType evtType,
      csiHandle* eventOut, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
00979
01001         CSI_DLL_EXPORT csiErr csiWaitForEvent(csiHandle evt, uint64_t timeoutMilliseconds,
      csiEventData** evtDataOut);
01002
01013         CSI_DLL_EXPORT csiErr csiUnregisterEvent(csiHandle evt);
01014
01024         CSI_DLL_EXPORT csiErr csiEventKill(csiHandle evt);
01025
01047         CSI_DLL_EXPORT csiErr csiGetNextImage(csiHandle eventHandle, csiNewBufferEventData**
      bufferInfoOut, uint64_t timeoutMilliseconds);
01048
01070         CSI_DLL_EXPORT csiErr csiReleaseImage(csiHandle dataStream, const csiNewBufferEventData*
      bufferInfo);
01071
01082         CSI_DLL_EXPORT csiErr csiGetAcquisitionStatistics(csiHandle dataStream,
      csiAcquistionStatistics* stats);
01083
01084         // TL interface functions
01085
01102         CSI_DLL_EXPORT csiErr csiOpenTLInterface(csiTLInterfaceInfo interfaceInfo, csiHandle*
      interfaceHandleOut, uint64_t timeoutMilliseconds);
01103
01116         CSI_DLL_EXPORT csiErr csiCloseTLInterface(csiHandle interfaceHandle);
01117
01118         // Module configuration functions
01134         CSI_DLL_EXPORT csiErr csiGetFeatureBool(csiHandle moduleHandle, const char* featureName, bool*
      valueOut, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01135
01151         CSI_DLL_EXPORT csiErr csiSetFeatureBool(csiHandle moduleHandle, const char* featureName, bool
      value, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01152
01153
01164         CSI_DLL_EXPORT csiErr csiGetFeatureInt(csiHandle moduleHandle, const char* featureName,
      int64_t* valueOut, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01165
01180         CSI_DLL_EXPORT csiErr csiSetFeatureInt(csiHandle moduleHandle, const char* featureName,
      int64_t value, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01181
01197         CSI_DLL_EXPORT csiErr csiGetFeatureFloat(csiHandle moduleHandle, const char* featureName,
      double* valueOut, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01198
01213         CSI_DLL_EXPORT csiErr csiSetFeatureFloat(csiHandle moduleHandle, const char* featureName,
      double value, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01214
01241         CSI_DLL_EXPORT csiErr csiGetFeatureString(csiHandle moduleHandle, const char* featureName,
      char* valueOut, size_t* sizeOut, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01242
01266         CSI_DLL_EXPORT csiErr csiSetFeatureString(csiHandle moduleHandle, const char* featureName,
      const char* value, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01267
01287         CSI_DLL_EXPORT csiErr csiExecuteCommand(csiHandle moduleHandle, const char* featureName,
      csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01288
01308         CSI_DLL_EXPORT csiErr csiIsCommandActive(csiHandle moduleHandle, const char* featureName, bool
      *isActive, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01309
01334         CSI_DLL_EXPORT csiErr csiGetFeatureReg(csiHandle moduleHandle, const char* featureName, char*
      buffer, size_t* length, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01335
01360         CSI_DLL_EXPORT csiErr csiSetFeatureReg(csiHandle moduleHandle, const char* featureName, const
      char* buffer, size_t length, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01361
01384         CSI_DLL_EXPORT csiErr csiGetFeatureEnum(csiHandle moduleHandle, const char* featureName,
      csiFeatureParameter *featureParamOut, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01385
01410         CSI_DLL_EXPORT csiErr csiSetFeatureEnum(csiHandle moduleHandle, const char* featureName, const
      char* value, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01411
01428         CSI_DLL_EXPORT csiErr csiGetFeatureParameter(csiHandle moduleHandle, const char* featureName,
      csiFeatureParameter* featureParamOut, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01429
```

```
01444         CSI_DLL_EXPORT csiErr csiGetFeatureAccessMode(csiHandle moduleH, const char* featureName,
      csiAccessMode* access, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01445
01471         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);
01472
01488         CSI_DLL_EXPORT csiErr csiGetFeatureEnumEntryCount(csiHandle moduleHandle, const char*
      featureName, uint32_t* count, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01489
01512         CSI_DLL_EXPORT csiErr csiGetFeatureEnumEntryByIndex(csiHandle moduleHandle, const char*
      featureName, int32_t enumIndex, csiFeatureParameter *featureParamOut, csiModuleLevel module
      CSI_DEFAULT_PARAM_MODULE);
01513
01533         CSI_DLL_EXPORT csiErr csiGetFeatureEnumEntryByName(csiHandle moduleHandle, const char*
      featureName, const char* enumValue, csiFeatureParameter *featureParamOut, csiModuleLevel module
      CSI_DEFAULT_PARAM_MODULE);
01534
01535
01536         // File transfer functions
01559         CSI_DLL_EXPORT csiErr csiGetUpdateFileType(csiHandle moduleHandle, const char* fileName, char*
      fileTypeOut, size_t bufferSize);
01560
01582         CSI_DLL_EXPORT csiErr csiFileDownloadToDevice(csiHandle moduleHandle,
01583                                                       csiHandle localDevice,
01584                                                       const char* fileName,
01585                                                       const char* fileType,
01586                                                       uint64_t timeoutMilliseconds,
01587                                                       csiMemTransferCallbackFunc listener
      CSI_DEFAULT_PARAM_NULL,
01588                                                       csiMemTransferUserData* userdata
      CSI_DEFAULT_PARAM_NULL);
01589
01609         CSI_DLL_EXPORT csiErr csiFileUploadFromDevice(csiHandle moduleHandle,
01610                                                       csiHandle localDevice,
01611                                                       const char* fileName,
01612                                                       const char* fileType,
01613                                                       uint64_t timeoutMilliseconds,
01614                                                       csiMemTransferCallbackFunc listener
      CSI_DEFAULT_PARAM_NULL,
01615                                                       csiMemTransferUserData* userdata
      CSI_DEFAULT_PARAM_NULL);
01616
01617
01637         CSI_DLL_EXPORT csiErr csiFileDownloadToDeviceEx
01638             (csiHandle moduleHandle,
01639             csiHandle localDevice,
01640             const csiDownloadParams params,
01641             uint64_t timeoutMilliseconds,
01642             csiMemTransferCallbackFunc listener CSI_DEFAULT_PARAM_NULL,
01643             csiMemTransferUserData* userdata CSI_DEFAULT_PARAM_NULL);
01644
01666         CSI_DLL_EXPORT csiErr csiFileUploadFromDeviceEx
01667             (csiHandle moduleHandle,
01668             csiHandle localDevice,
01669             const csiUploadParams params,
01670             uint64_t timeoutMilliseconds,
01671             csiMemTransferCallbackFunc listener CSI_DEFAULT_PARAM_NULL,
01672             csiMemTransferUserData* userdata CSI_DEFAULT_PARAM_NULL);
01673
01674         // Memory transfer functions
01688         CSI_DLL_EXPORT csiErr csiReadMemory(csiHandle moduleHandle, uint64_t address, char* buffer,
      size_t sizeBytes);
01689
01702         CSI_DLL_EXPORT csiErr csiWriteMemory(csiHandle moduleHandle, uint64_t address, const char*
      buffer, size_t sizeBytes);
01703
01716         CSI_DLL_EXPORT csiErr csiIsModuleLittleEndian(csiHandle moduleHandle, csiModuleLevel lvl, bool
      *isLittleEndian);
01717
01727         CSI_DLL_EXPORT csiErr csiGetErrorDescription(csiErr error, char* bufferOut, size_t
      bufferSize);
01728
01729         // Helper functions
01739         CSI_DLL_EXPORT unsigned char csiBitsPerPixelFromFormat(const csiPixelFormat format);
01740
01761         CSI_DLL_EXPORT csiErr csiRegisterInvalidateCB(csiHandle moduleHandle,  const char
      *featureName, CB_OBJECT objCB, CB_FEATURE_INVALIDATED_PFN pfnFeatureInvalidateCB, csiModuleLevel
      module CSI_DEFAULT_PARAM_MODULE);
01762
01763         //unregistering the call-back function
01779         CSI_DLL_EXPORT csiErr csiUnRegisterInvalidateCB(csiHandle moduleHandle, const char
      *featureName, csiModuleLevel module CSI_DEFAULT_PARAM_MODULE);
01780
01791         CSI_DLL_EXPORT csiErr csiGetLibraryVersion(uint32_t* major, uint32_t* minor, uint32_t* patch,
      uint32_t* revision, uint32_t* build);
01792
```

```
01799         CSI_DLL_EXPORT csiErr csiSetFeatureCachingDisabled(bool disableCaching);
01800
01808         CSI_DLL_EXPORT csiErr csiIsFeatureCachingDisabled(bool *isDisabled);
01809
01832         CSI_DLL_EXPORT csiErr csiGenerateAndUploadCalibrationData(csiHandle devHandle,
01833                                                                   csiCalibrationLUT LUTSelector,
01834                                                                   csiReferenceImgLoadMode imgLoadMode,
01835                                                                   csiCalibrationParams calibParam,
01836                                                                   csiNewBufferEventData *refImage,
01837                                                                   const char* imgFile);
01866         CSI_DLL_EXPORT csiErr csiGenerateAndSaveCalibrationDataToFile(csiHandle devHandle,
01867                                                                   CalibrationMode calibMode,
01868                                                                   csiReferenceImgLoadMode imgLoadMode,
01869                                                                   csiCalibrationParams calibParam,
01870                                                                   csiNewBufferEventData* refImage,
01871                                                                   const char* imgFile,
01872                                                                   const char* calibrationFile);
01873
01902         CSI_DLL_EXPORT csiErr csiGenerateReferenceImage(csiHandle devHandle,
01903                                                         const CalibrationMode calibMode,
01904                                                         csiReferenceImgLoadMode imgLoadMode,
01905                                                         csiCalibrationParams calibParam,
01906                                                         const csiNewBufferEventData* refImage,
01907                                                         const char* imgFile,
01908                                                         int *referenceImage, int *resMaxValue);
01909 #ifdef __cplusplus
01910     }
01911 }
01912 #endif // __cplusplus
01913
01914 #endif // CSI_CSIF_H_
```

# Index