

Chromasens GEN<i>CAM-SDK | Manual

Table of Contents

Chromasens GEN<i>CAM-SDK Manual	1
R11 / 2024-06-04	1
1 General information	4
1.1 About Chromasens	4
1.1.1 Contact information	4
1.1.2 Support	4
1.2 Conventions used in this manual	5
1.2.1 Styles	5
1.2.2 Symbols	6
1.2.3 List of abbreviations	6
2 General aspects of the API	8
3 Getting started	9
3.1 Initialization of the SDK	9
3.2 Connecting to a camera	9
3.3 Getting and setting features	10
3.4 Acquiring images	11
3.5 Examples	11
3.5.1 Visual Studio Example Projects	12
3.5.2 Build examples	12
4 List of SDK-functions	14
4.1 Init/Deinit-functions	14
4.2 Connecting and closing a device	14
4.3 Discover, open and close an interface	17
4.4 Getting and setting module parameters	18
4.5 Functions related to image acquisition	25
4.6 File transfer functions	29
4.7 Calibration Data Generator functions	30
4.8 Memory transfer functions	31
4.9 Helper functions	33
4.10 Enumerations	35
4.11 Structures	40
5 Installation	44
5.1 Windows installation	44
5.1.1 Installer Contents	44

5.2	Linux installation	44
5.2.1	Preparation	44
5.2.2	Step by Step Installation Ubuntu 20.04	44
5.2.3	Installer Contents	45

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: <http://www.chromasens.de/en/support>

Visit our website at <http://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	<pre>typedef enum csiEventType { CSI_EVT_NEWIMAGEDATA = 0x00, CSI_EVT_ERROR = 0x01, CSI_EVT_MODULE = 0x02, CSI_EVT_CUSTOM = 0x1000 } csiEventType;</pre>
Elements	<p>CSI_EVT_NEWIMAGEDATA: New image data received CSI_EVT_ERROR: Error occurred in the SDK CSI_EVT_MODULE: General event notification CSI_EVT_CUSTOM: A custom event was triggered</p>

Structures:

For example:

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

Functions:

For example:

csiDiscoverDevices Searches for the devices currently connected to the system	
Syntax	<pre>csiErr csiDiscoverDevices(csiDiscoveryInfo* discoveryInfoOut, uint64_t timeoutMilliseconds, csiDiscoveryInfoCallbackFunc discCallbackFunc = NULL, const char* additionalSearchPaths = NULL, bool overrideSearchPath CSI_DEFAULT_PARAM_FALSE);</pre>
<u>Parameters:</u>	<p>In:</p> <p>timeoutMilliseconds: The amount of time to search on a specific transport layer for a device discCallbackFunc: pointer to a callback function which gets called when a result was received</p> <p>AdditionalSearchPaths: as default only the paths given in the system variable "GENICAM_GENTL64_PATH" are being searched for the used transport layers</p> <p>overrideSearchPath: If set, only the given path is searched for transport layers to use.</p> <p>Out:</p> <p>discoveryInfoOut: The structure will be filled with the available devices</p>
<u>Return value:</u>	Returns csiSuccess or an error defined in the csiErr-Enum.
<u>Comment:</u>	

1.2.2 Symbols



CAUTION

Indicates a potentially hazardous situation or task, which, if not avoided, may result in minor or moderate injury.

NOTICE

Indicates a potentially hazardous situation or task, which, if not avoided, could result in damage to the product or the surrounding environment.



Indicates a helpful tip.



More detailed information can be retrieved online.

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 GenICam-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 >= 20.04

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 4.1 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`. Detailed information regarding the functions can be found here: 3.2 Connecting to a camera

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 “`csiGetFeatureParameter`” 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 `csiFeatureParameter`-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.

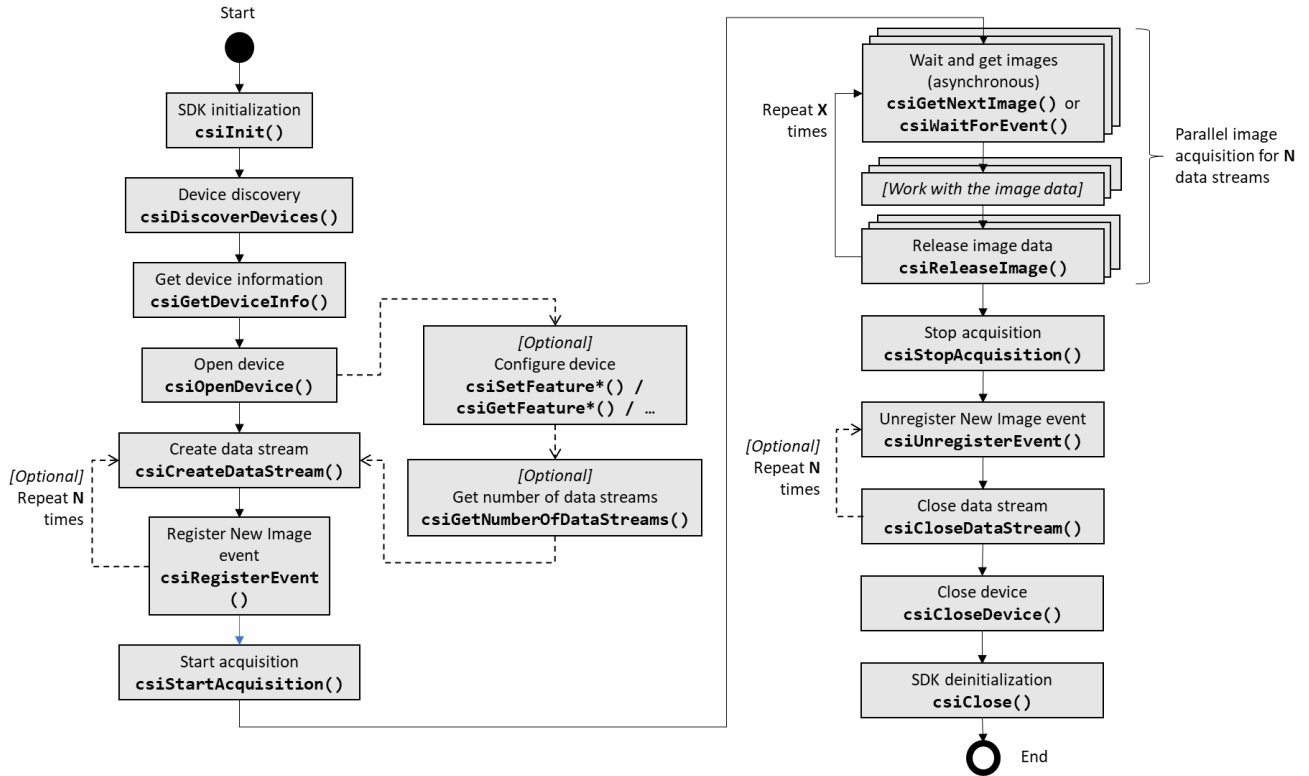
To set the values please use the type-specific set-functions. For example, use “`csiSetFeatureParameter`” for an integer value.

These functions are described in detail in the chapter “3.3 Getting and setting features”.

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.



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 which is described in the chapter 4.5 Functions related to image acquisition.

In general, two different ways in 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`.

By failing to do so the user 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:

acquisition_basics	<p>Demonstrates how to discover and open a device and how to acquire images.</p> <p>Locations:</p> <p>Windows: C:\Users\Public\Documents\Chromasens\GCT2\examples\basic</p> <p>Linux: /usr/CSGenicam-SDK/share/csgenicam/examples/basic</p>
feature_iteration	<p>Demonstrates how to iterate through the feature tree of a device and how to set / get features.</p> <p>Locations:</p> <p>Windows: C:\Users\Public\Documents\Chromasens\GCT2\examples\feature_iteration</p> <p>Linux: /usr/CSGenicam-SDK/share/csgenicam/examples/feature_iteration</p>
calibration_generate	<p>Demonstrates how to generate PRNU and DSNU calibration files and upload it to camera or to save it in local PC memory.</p> <p>Locations:</p> <p>Windows: C:\Users\Public\Documents\Chromasens\GCT2\examples\calibration_generate</p> <p>Linux: /usr/CSGenicam-SDK/share/csgenicam/examples/feature_iteration</p>
save_rgb10and12	<p>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)</p> <p>Locations:</p> <p>Windows: C:\Users\Public\Documents\Chromasens\GCT2\examples\save_rgb10and12</p> <p>Linux: /usr/CSGenicam-SDK/share/csgenicam/examples/save_rgb10and12</p>
file_download	<p>Demonstrates on how to download the device(camera) firmware files into the device(camera).</p> <p>Locations:</p> <p>Windows: C:\Users\Public\Documents\Chromasens\GCT2\examples\file_download</p> <p>Linux: /usr/CSGenicam-SDK/share/csgenicam/examples/file_download</p>

3.5.1 Visual Studio Example Projects

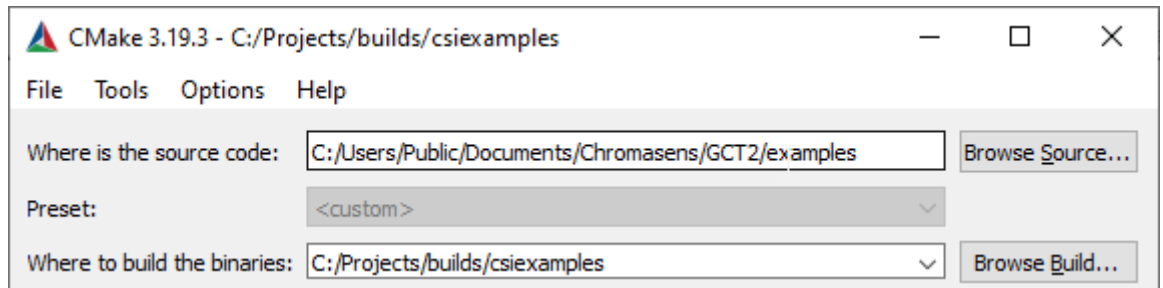
The Visual Studio projects for the two examples are also included in the SDK. These projects could be found in the same location as stated above. These example projects could also be built with CMake. The following section explains how to build a project with CMake.

3.5.2 Build examples

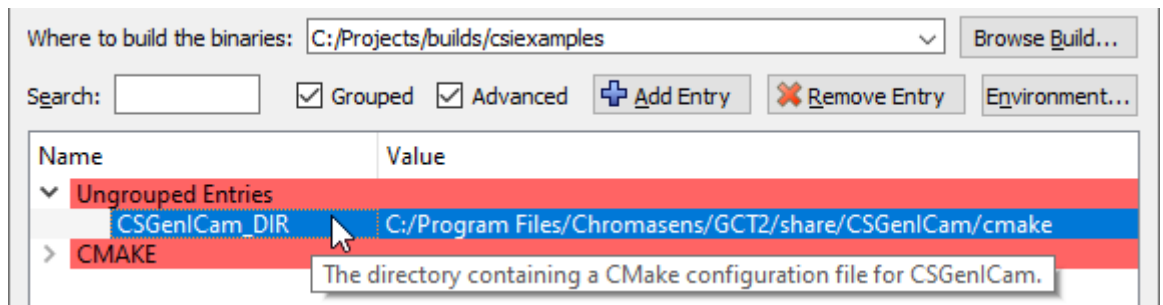
To build the examples requires CMake version > v3.14 and a build environment. The steps to build the examples are the same for both **Windows and Linux**:

- 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")



- 3) Press the "Configure" button. After the first configuration, if the CSGenIcam_DIR is not found, it is required to manually set the path to the CSGenIcam CMake configuration files:



- 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 section 3.5.1, to add the DLL search path for the application.

4 List of SDK-functions

4.1 Init/Deinit-functions

csiInit		Initializes the SDK. Needs to be called first before any other function of the SDK is called!
Syntax	<i>csiErr csiInit(csiLogLevel logLvl = CSI_LOGLEVEL_WARN, , csiLogSinkCallbackFunc logCallbackFunc = NULL, csiLogUserData* userdata = NULL)</i>	
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 logCallbackFunc: An optional callback function for log messages coming from the SDK. userData: Optional user data that will be passed as parameter when the log callback function is called. Out: Nothing	
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.	
Comment:	After the usage of the SDK make sure to call csiClose in order to free all memory again and not leaving any interfaces open.	

csiClose		Closes the SDK and frees all allocated memory and interfaces.
Syntax	<i>csiErr csiClose();</i>	
Parameters:	In: Nothing Out: Nothing	
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.	
Comment:		

4.2 Connecting and closing a device

csiDiscoverDevices		This function will look for attached GenICAM-devices on the available transport layers.
Syntax	<i>csiErr csiDiscoverDevices(csiDiscoveryInfo* discoveryInfoOut, uint64_t timeoutMilliseconds, csiDiscoveryInfoCallbackFunc discCallbackFunc = NULL, const char* additionalSearchPaths = NULL, bool overrideSearchPath = false)</i>	
Parameters:	In: timeoutMilliseconds The time until when a response from a device needs to be received when doing a discovery 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. 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 overrideSearchPath If this flag is set, only the path(s) provided in "additionalSearchPaths" will be searched for the cti-files to load 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	
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.	
Comment:	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).	

csiGetDeviceInfo A function to get information about the found devices in the system	
Syntax	<code>csiErr csiGetDeviceInfo(uint32_t deviceIndex, csiDeviceInfo* deviceInfoOut)</code>
Parameters:	<p>In: deviceIndex Index of the found device from the <i>csiDiscoverDevices</i>-function</p> <p>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.</p>
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	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

csiGetNumberOfTLProducers Returns the number of available transport layers in the system	
Syntax	<code>csiErr csiGetNumberOfTLProducers(int32_t *numTLProducers);</code>
Parameters:	<p>In: Nothing</p> <p>Out: <i>numTLProducers</i> The number of transport layers detected in the environment.</p>
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	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.

csiGetTLProducerInfoByIndex Returns additional information about a transport layer.	
Syntax	<code>CSI_DLL_EXPORT csiErr csiGetTLProducerInfoByIndex(csiTLProducerInfos *tlProducerInfos, uint32_t indexTL);</code>
Parameters:	<p>In: indexTL: The index of the transport layer in the list.</p> <p>Out: <i>tlProdcrInfos</i>: The structure holding additional information about the transport layer.</p>
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	The index must be within 0 and the number of transport layer returned by csiGetNumberOfTLProducers.

csiGetTLProducerInfoByFilePath Returns additional information about a transport layer.	
Syntax	<code>csiErr csiGetTLProducerInfoByFilePath(csiTLProducerInfos *tlProdcrInfos, const char* producerName)</code>
Parameters:	<p>In: <i>producerName</i>: The name of the producer (usually the file path to the producer *.cti file)</p> <p>Out: <i>tlProdcrInfos</i>: The structure holding additional information about the transport layer.</p>
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	Similar to csiGetTLProducerInfoByIndex but using the name (file path) of the producer.

csiOpenDevice		Open the device given by the index. The TL of this index is used							
Syntax	<i>csiErr csiOpenDevice(const char* deviceIdentifier, const char* interfaceID, csiHandle* deviceHandleOut, uint64_t timeoutMilliseconds, csiDeviceAccessMode openMode)</i>								
Parameters:	In: <table border="0" style="margin-left: 20px;"> <tr> <td style="padding-right: 20px;">deviceIdentifier</td> <td>Index of the found device from the <i>csiDiscoverDevices</i>-function</td> </tr> <tr> <td>interfaceID</td> <td>Interface ID of the device. This information can be found in device-info struct</td> </tr> <tr> <td>timeoutMilliseconds</td> <td>Timeout in milliseconds until the device needs to be opened successfully</td> </tr> <tr> <td>openMode</td> <td>The device can be opened in different modes to enable/hinder concurrent access to the device.</td> </tr> </table> <p>The following modes might be used: CSI_DEV_MODE_EXCLUSIVE: Only this process can communicate with the camera CSI_DEV_MODE_READ: Camera-parameters can be read and images can be acquired CSI_DEV_MODE_CONTROL: Camera-parameters can be read and written. Read-access by another process to the device is still possible</p> Out: deviceHandleOut: Handle to the device. This handle needs to be used to any successive call.	deviceIdentifier	Index of the found device from the <i>csiDiscoverDevices</i> -function	interfaceID	Interface ID of the device. This information can be found in device-info struct	timeoutMilliseconds	Timeout in milliseconds until the device needs to be opened successfully	openMode	The device can be opened in different modes to enable/hinder concurrent access to the device.
deviceIdentifier	Index of the found device from the <i>csiDiscoverDevices</i> -function								
interfaceID	Interface ID of the device. This information can be found in device-info struct								
timeoutMilliseconds	Timeout in milliseconds until the device needs to be opened successfully								
openMode	The device can be opened in different modes to enable/hinder concurrent access to the device.								
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.								
Comment:	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.								

csiCloseDevice		Close the connection to the specific device
Syntax	<i>csiErr csiCloseDevice(csiHandle device)</i>	
Parameters:	In: device: Handle provided by the <i>csiOpenDevice</i> -function Out:	
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.	
Comment:	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.	

4.3 Discover, open and close an interface

csiDiscoverTLInterfaces																
This function will look for TL interfaces provided by the available transport layers.																
Syntax	<i>csiErr csiDiscoverTLInterfaces(csiTLInterfaceDiscoveryInfo* discoveryInfoOut, uint64_t timeoutMilliseconds, csiDiscoveryTLInterfaceInfoCallbackFunc discCallbackFunc = NULL, const char* additionalSearchPaths = NULL, bool overrideSearchPath = false)</i>															
<u>Parameters:</u>	<table border="0"> <tr> <td style="vertical-align: top;">In:</td> <td style="vertical-align: top;">timeoutMilliseconds</td> <td style="vertical-align: top;">The time until when a response from a transport layer needs to be received when doing a discovery</td> </tr> <tr> <td></td> <td style="vertical-align: top;">discCallbackFunc</td> <td style="vertical-align: top;">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 interfaces Also a flag if the discovery is running is provided.</td> </tr> <tr> <td></td> <td style="vertical-align: top;">additionalSearchPaths</td> <td style="vertical-align: top;">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</td> </tr> <tr> <td></td> <td style="vertical-align: top;">overrideSearchPath</td> <td style="vertical-align: top;">If this flag is set, only the path(s) provided in "additionalSearchPaths" will be searched for the cti-files to load</td> </tr> <tr> <td style="vertical-align: top;">Out:</td> <td style="vertical-align: top;">discoveryInfoOut</td> <td style="vertical-align: top;">pointer to a structure which will contain the information about the found devices. The information is the same provided to the callback function</td> </tr> </table>	In:	timeoutMilliseconds	The time until when a response from a transport layer needs to be received when doing a discovery		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 interfaces Also a flag if the discovery is running is provided.		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		overrideSearchPath	If this flag is set, only the path(s) provided in "additionalSearchPaths" will be searched for the cti-files to load	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 transport layer needs to be received when doing a discovery														
	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 interfaces Also a flag if the discovery is running is provided.														
	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														
	overrideSearchPath	If this flag is set, only the path(s) provided in "additionalSearchPaths" will be searched for the cti-files to load														
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														
<u>Return value:</u>	Returns csiSuccess or an error defined in the csiErr-Enum.															
<u>Comment:</u>	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).															

csiOpenTLInterface										
Opens the interface given by the interface information										
Syntax	<i>csiErr csiOpenTLInterface(csiTLInterfaceInfo interfaceInfo, csiHandle* interfaceHandleOut, uint64_t timeoutMilliseconds)</i>									
<u>Parameters:</u>	<table border="0"> <tr> <td style="vertical-align: top;">In:</td> <td style="vertical-align: top;">interfaceInfo</td> <td style="vertical-align: top;">Interface information of the found TL interface from the <i>csiDiscoverTLInterfaces</i> function</td> </tr> <tr> <td></td> <td style="vertical-align: top;">timeoutMilliseconds</td> <td style="vertical-align: top;">Timeout in milliseconds until the interface needs to be opened successfully</td> </tr> <tr> <td style="vertical-align: top;">Out:</td> <td style="vertical-align: top;">interfaceHandleOut</td> <td style="vertical-align: top;">Handle to the interface. This handle needs to be used to any successive call for feature access.</td> </tr> </table>	In:	interfaceInfo	Interface information of the found TL interface from the <i>csiDiscoverTLInterfaces</i> function		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 feature access.
In:	interfaceInfo	Interface information of the found TL interface from the <i>csiDiscoverTLInterfaces</i> function								
	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 feature access.								
<u>Return value:</u>	Returns csiSuccess or an error defined in the csiErr-Enum.									
<u>Comment:</u>	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.									

csiCloseTLInterface							
Closes the connection to the specific interface							
Syntax	<i>csiErr csiCloseTLInterface(csiHandle interfaceHandle)</i>						
<u>Parameters:</u>	<table border="0"> <tr> <td style="vertical-align: top;">In:</td> <td style="vertical-align: top;">interfaceHandle</td> <td style="vertical-align: top;">Handle provided by the <i>csiOpenTLInterface</i> function.</td> </tr> <tr> <td style="vertical-align: top;">Out:</td> <td></td> <td></td> </tr> </table>	In:	interfaceHandle	Handle provided by the <i>csiOpenTLInterface</i> function.	Out:		
In:	interfaceHandle	Handle provided by the <i>csiOpenTLInterface</i> function.					
Out:							
<u>Return value:</u>	Returns csiSuccess or an error defined in the csiErr-Enum.						
<u>Comment:</u>	<p>To grant access to the underlying interface for other applications, the connection should be closed when it is not needed anymore.</p> <p>The API will cleanup no longer needed memory when this command is executed.</p>						

4.4 Getting and setting module parameters



It is possible to retrieve and set parameters on the device/camera. The SDK additionally provides the possibility to set parameters for other involved components such as the transport layer module.

Therefore, it is possible to indicate this by changing the module parameter from the default setting (CSI_DEVICE_MODULE) to the other components such as transport layer, stream, or buffer module.

Please note that the parameters available for the different modules will differ significantly!

csiGetFeatureBool		Retrieve a boolean feature from the device
Syntax	<i>csiErr csiGetFeatureBool (csiHandle moduleHandle, const char* parameterName, bool* valueOut, csiModuleLevel module = CSI_DEVICE_MODULE)</i>	
<u>Parameters:</u>	In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name of the feature to get (Not the display name!) module: Module for which the parameter should be get. Please use the enum <i>csiModuleLevel</i> 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
<u>Return value:</u>	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .	
<u>Comment:</u>		

csiSetFeatureBool		Set a boolean feature on the device
Syntax	<i>csiErr csiSetFeatureBool(csiHandle moduleHandle, const char* parameterName, bool value, csiModuleLevel module = CSI_DEVICE_MODULE)</i>	
<u>Parameters:</u>	In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name (Not the display name!) of the feature to set value: Boolean value to set the feature to (true or false) module: Module for which the parameter should be set. Please use the enum <i>csiModuleLevel</i> to select. Determines if the parameter should be set on the device-, transport layer-, interface- stream- or buffer-module	Out:
<u>Return value:</u>	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .	
<u>Comment:</u>		


csiGetFeatureInt		Retrieve an integer feature from the device
Syntax	<i>csiErr csiGetFeatureInt(csiHandle moduleHandle, const char* parameterName, int64_t* valueOut, csiModuleLevel module = CSI_DEVICE_MODULE)</i>	
<u>Parameters:</u>	In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name (Not the display name!) of the feature to get module: Module for which the parameter should be get. Please use the enum <i>csiModuleLevel</i> to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module	Out: valueOut: Pointer to an <i>int64_t</i> -value where the current value of the feature will be written to
<u>Return value:</u>	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .	


<u>Comment:</u>	
-----------------	--

csiSetFeatureInt Set an integer feature on the device	
<u>Syntax</u>	<i>csiErr csiSetFeatureInt(csiHandle moduleHandle, const char* parameterName, int64_t value, csiModuleLevel module = CSI_DEVICE_MODULE)</i>
<u>Parameters:</u>	<p>In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name (Not the display name!) of the feature to set value: integer value to set the feature to module: Module for which the parameter should be set. Please use the enum <i>csiModuleLevel</i> to select. Determines if the parameter should be set on the device-, transport layer-, interface- stream- or buffer-module</p> <p>Out:</p>
<u>Return value:</u>	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .
<u>Comment:</u>	

csiGetFeatureFloat Retrieve a floating point feature from the device	
<u>Syntax</u>	<i>csiErr csiGetFeatureFloat(csiHandle moduleHandle, const char* parameterName, double* valueOut, csiModuleLevel module = CSI_DEVICE_MODULE)</i>
<u>Parameters:</u>	<p>In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name (Not the display name!) of the feature to get module: Module for which the parameter should be get. Please use the enum <i>csiModuleLevel</i> to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module</p> <p>Out: valueOut: Pointer to a double-value where the current value of the feature will be written to</p>
<u>Return value:</u>	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .
<u>Comment:</u>	

csiSetFeatureFloat Set a floating point value feature on the device	
<u>Syntax</u>	<i>csiErr csiSetFeatureFloat(csiHandle moduleHandle, const char* parameterName, double value, csiModuleLevel module = CSI_DEVICE_MODULE)</i>
<u>Parameters:</u>	<p>In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name (Not the display name!) of the feature to set value: floating point value to set module: Module for which the parameter should be set. Please use the enum <i>csiModuleLevel</i> to select. Determines if the parameter should be set on the device-, transport layer-, interface- stream- or buffer-module</p> <p>Out:</p>
<u>Return value:</u>	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .
<u>Comment:</u>	

csiGetFeatureString	
Retrieve a string feature from the device	
Syntax	<i>csiErr</i> csiGetFeatureString(<i>csiHandle</i> moduleHandle, const char* parameterName, char* valueOut, size_t* sizeOut, <i>csiModuleLevel</i> module = CSI_DEVICE_MODULE)
Parameters:	In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name (Not the display name!) of the feature to get module: Module for which the parameter should be set. Please use the enum <i>csiModuleLevel</i> 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 sizeOut: size of the read string
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .
Comment:	 <p>To avoid unexpected behavior, you should first retrieve the length of the string to be received!</p> <ol style="list-style-type: none"> 1. Call the function with valueOut set to NULL. The function will return the current size of the string parameter. This enables the user to provide sufficient space to return the desired string. Alternative: Call the function "csiGetFeatureParameter". This function will provide all necessary information about the parameter (including min and max values). The maximum string length to be retrieved can be read from from the "maximumStringLength"-parameter 2. Call the function as described by providing a pointer to the string buffer with the sufficient length

csiSetFeatureString	
Set a string feature on the device	
Syntax	<i>csiErr</i> csiSetFeatureString(<i>csiHandle</i> moduleHandle, const char* parameterName, const char* value, <i>csiModuleLevel</i> module = CSI_DEVICE_MODULE)
Parameters:	In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name (Not the display name!) of the feature to set value: pointer to a character array which contains the string to set module: Module for which the parameter should be set. Please use the enum <i>csiModuleLevel</i> to select. Determines if the parameter should be set on the device-, transport layer-, interface- stream- or buffer-module Out:
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .
Comment:	 <p>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</p>

csiExecuteCommand	
Execute a command on the device	
Syntax	<i>csiErr</i> csiExecuteCommand(<i>csiHandle</i> moduleHandle, const char* parameterName, <i>csiModuleLevel</i> module = CSI_DEVICE_MODULE)
Parameters:	In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name (Not the display name!) of the feature to set. module: Module for which the parameter should be executed. Please use the enum <i>csiModuleLevel</i> to select. Determines if the parameter should be executed on the device-, transport layer-, interface- stream- or buffer-module Out:
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .
Comment:	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 " <i>csilsCommandActive</i> ".

csiIsCommandActive	
Check if a command is still active	
Syntax	<i>csiErr csiIsCommandActive(csiHandle moduleHandle, const char* parameterName, bool *isActive, csiModuleLevel module = CSI_DEVICE_MODULE)</i>
Parameters:	<p>In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name (Not the display name!) of the command to execute module: Module for which the parameter should be set. Please use the enum <i>csiModuleLevel</i> to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module</p> <p>Out: isActive: Pointer to a bool-value where the current state of the command is written to (true: active, false: inactive)</p>
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .
Comment:	If a lengthy operation is triggered, it is possible to check the current status by calling this command.

csiGetFeatureReg	
Retrieve a register value from the device	
Syntax	<i>csiErr csiGetFeatureReg(csiHandle moduleHandle, const char* parameterName, char* buffer, size_t* length, csiModuleLevel module = CSI_DEVICE_MODULE)</i>
Parameters:	<p>In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name (Not the display name!) of the feature to get value: module: Module for which the parameter should be set. Please use the enum <i>csiModuleLevel</i> to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module</p> <p>Out: buffer: Pointer to a char-array where the current value of the feature will be written to length: Current length of the retrieved data</p>
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .
Comment:	<p>To avoid unexpected behavior, it is recommended to retrieve the maximum buffer length before getting it from the device.</p> <p>This can be achieved by using the function "<i>csiGetFeatureParameter</i>". 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</p>

csiSetFeatureReg	
Set a register value on the device	
Syntax	<i>csiErr csiSetFeatureReg(csiHandle moduleHandle, const char* parameterName, const char* buffer, size_t length, csiModuleLevel module = CSI_DEVICE_MODULE)</i>
Parameters:	<p>In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name (Not the display name!) of the feature to set buffer: pointer to the data which will be written to the register length: number of bytes to write to the register module: Module for which the register should be set. Please use the enum <i>csiModuleLevel</i> to select. Determines if the register should be set on the device-, transport layer-, interface- stream- or buffer-module</p> <p>Out:</p>
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .
Comment:	<p>To avoid unexpected behavior, it is recommended to retrieve the maximum buffer length before setting it to the device.</p> <p>This can be achieved by using the function "<i>csiGetFeatureParameter</i>". 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</p>

csiGetFeatureParameter	
Retrieve a specific feature from the device. Detailed information about this feature will be returned	
<u>Syntax</u>	<i>csiErr csiGetFeatureParameter(csiHandle moduleHandle, const char* parameterName, csiFeatureParameter* featureParamOut, csiModuleLevel module = CSI_DEVICE_MODULE)</i>
<u>Parameters:</u>	<p>In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name (Not the display name!) of the feature to get module: Module for which the parameter should be set. Please use the enum <i>csiModuleLevel</i> to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module</p> <p>Out: featureParamOut</p>
<u>Return value:</u>	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .
<u>Comment:</u>	

csiIterateFeatureTree	
Provides a possibility to iterate through all available features on the camera	
<u>Syntax</u>	<i>csiErr csiIterateFeatureTree(csiHandle moduleHandle, const char* rootFeatureName, uint32_t index, char* featureNameOut, size_t nameBuffSize, csiFeatureType* type, csiModuleLevel module = CSI_DEVICE_MODULE)</i>
<u>Parameters:</u>	<p>In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function rootFeatureName: name of the feature to start from. To start from the very beginning use "root" index: This will indicate the number of the child element of the rootFeature to retrieve nameBuffSize: size of the provided buffer for the featureNameOut module: Module for which the feature tree should be iterated. Please use the enum <i>csiModuleLevel</i> to select. Determines if the feature tree on the device-, transport layer-, interface- stream- or buffer-module should be used</p> <p>Out: featureNameOut: name of the retrieved feature type: type of the retrieved feature</p>
<u>Return value:</u>	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .
<u>Comment:</u>	<p>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.</p> <p>Please check the provided example "feature_iteration" for a template of usage.</p>

csiGetFeatureEnum	
Retrieve an enumeration feature from the device	
<u>Syntax</u>	<i>csiErr csiGetFeatureEnum(csiHandle moduleHandle, const char* parameterName, csiFeatureParameter *featureParamOut, csiModuleLevel module = CSI_DEVICE_MODULE)</i>
<u>Parameters:</u>	<p>In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function parameterName: name (Not the display name!) of the enumeration to get module: Module for which the parameter should be set. Please use the enum <i>csiModuleLevel</i> to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module</p> <p>Out: featureParamOut: Structure which contains all necessary information about the requested feature: the relevant entries of the <i>csiFeatureParameter</i>-structure: enumCounter: Number of different enum-entries for the enumeration enumIndex: Currently selected enumeration index valueStr: name of the enum-entry</p>
<u>Return value:</u>	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .
<u>Comment:</u>	

csiSetFeatureEnum	
Set an enumeration feature on the device	
Syntax	<code>csiSetFeatureEnum(csiHandle moduleHandle, const char* parameterName, const char* value, csiModuleLevel module = CSI_DEVICE_MODULE)</code>
Parameters:	In: moduleHandle: Handle provided by the <code>csiOpenDevice</code> or <code>csiOpenTLInterface</code> function parameterName: name (Not the display name!) of the enumeration to get module: Module for which the parameter should be set. Please use the enum <code>csiModuleLevel</code> to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module Out:
Return value:	Returns <code>csiSuccess</code> or an error defined in the <code>csiErr-Enum</code> .
Comment:	To retrieve the possible values for this enumeration, two functions need to be called: <ol style="list-style-type: none"> <code>csiGetFeatureEnum</code>: in the returned structure, the element "enumCounter" indicates the number of available entries The different entries can be retrieved by using the function "<code>csiGetFeatureEnumEntryByIndex</code>" simply by iterating from 0 until the "enumCounter"-1

csiGetFeatureEnumEntryByIndex	
Retrieve an enumeration feature from the device by using its index	
Syntax	<code>csiErr csiGetFeatureEnumEntryByIndex(csiHandle moduleHandle, const char* parameterName, int32_t enumIndex, csiFeatureParameter *featureParamOut, csiModuleLevel module = CSI_DEVICE_MODULE)</code>
Parameters:	In: moduleHandle: Handle provided by the <code>csiOpenDevice</code> or <code>csiOpenTLInterface</code> function parameterName: name (Not the display name!) of the enumeration to get enumIndex: the index of the enumeration to get module: Module for which the parameter should be retrieved. Please use the enum <code>csiModuleLevel</code> 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
Return value:	Returns <code>csiSuccess</code> or an error defined in the <code>csiErr-Enum</code> .
Comment:	The enumeration string will be given in the <code>csiFeatureParameter</code> -structure: valueStr

csiGetFeatureEnumEntryByName	
Retrieve an enumeration feature from the device by using its name	
Syntax	<code>csiErr csiGetFeatureEnumEntryByName(csiHandle moduleHandle, const char* parameterName, const char* enumValue, csiFeatureParameter *featureParamOut, csiModuleLevel module = CSI_DEVICE_MODULE)</code>
Parameters:	In: moduleHandle: Handle provided by the <code>csiOpenDevice</code> or <code>csiOpenTLInterface</code> function parameterName: name (Not the display name!) of the enumeration to get enumValue: module: Module for which the parameter should be set. Please use the enum <code>csiModuleLevel</code> to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module Out: featureParamOut
Return value:	Returns <code>csiSuccess</code> or an error defined in the <code>csiErr-Enum</code> .
Comment:	

csiRegisterInvalidateCB	
Register an invalidation callback function to a specific feature by name	
Syntax	<code>csiErr csiRegisterInvalidateCB(csiHandle moduleHandle, const char *featureName, CB_OBJECT objCB, CB_FEATURE_INVALIDATED_PFN pfnFeatureInvalidateCB, csiModuleLevel module = CSI_DEVICE_MODULE);</code>
Parameters:	In: moduleHandle: Handle provided by the <code>csiOpenDevice</code> or <code>csiOpenTLInterface</code> function featureName: Name of the feature register the callback to objCB: An user object that is passed as parameter to the callback function pfnFeatureInvalidateCB: The callback function module: Module for which the feature invalidation callback should be registered. Please use the enum <code>csiModuleLevel</code> to select.

	<p>Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module</p> <p>Out: Nothing</p>
<u>Return value:</u>	Returns csiSuccess or an error defined in the csiErr-Enum.
<u>Comment:</u>	<p>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:</p> <pre>void featureInvalidated(const char *featureName, void* userObj);</pre>

csiUnRegisterInvalidateCB Unregister an invalidation callback function from a specific feature	
<u>Syntax</u>	<code>csiErr csiUnRegisterInvalidateCB(csiHandle moduleHandle, const char *featureName, csiModuleLevel module = CSI_DEVICE_MODULE);</code>
<u>Parameters:</u>	<p>In: moduleHandle: Handle provided by the <i>csiOpenDevice</i> or <i>csiOpenTLInterface</i> function featureName: Name of the feature to unregister the callback from module: Module for which the feature invalidation callback should be registered. Please use the enum <i>csiModuleLevel</i> to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface-stream- or buffer-module</p> <p>Out: Nothing</p>
<u>Return value:</u>	Returns csiSuccess or an error defined in the csiErr-Enum.
<u>Comment:</u>	

4.5 Functions related to image acquisition

csiGetNumberOfDataStreams		Return the available number of data streams of the device
Syntax	<i>csiErr csiGetNumberOfDataStreams(csiHandle device, uint32_t* numberOfStreamsOut)</i>	
<u>Parameters:</u>	In: device:	Handle provided by the <i>csiOpenDevice</i> -function
	Out: numberOfStreamsOut	Number of available data streams for the given device
<u>Return value:</u>	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .	
<u>Comment:</u>	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 <i>csiGetDataStreamInfo()</i> which receives an index to a data stream as parameter.	

csiGetDataStreamInfo		Return information about the desired data stream
Syntax	<i>csiErr csiGetDataStreamInfo(csiHandle device, uint32_t dsIndex, csiDataStreamInfo* dataStreamInfoOut)</i>	
<u>Parameters:</u>	In: device:	Handle provided by the <i>csiOpenDevice</i> -function
	dsIndex:	An index to a data stream. Starting from 0 to the number of available data streams as returned by <i>()</i> .
	Out: dataStreamInfoOut	Information about the selected data stream given by the <i>dsIndex</i> parameter or NULL if the data stream is not found. See documentation on <i>csiDataStreamInfo</i> for more information.
<u>Return value:</u>	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .	
<u>Comment:</u>		

csiCreateDataStream		Create a data stream to receive images
Syntax	<i>csiErr csiCreateDataStream(csiHandle device, uint32_t dsIndex, csiHandle* dataStreamOut, uint32_t numberOfBuffers, size_t bufferSize = 0)</i>	
<u>Parameters:</u>	In: device:	Handle provided by the <i>csiOpenDevice</i> -function
	dsIndex:	An index to a data stream. Starting from 0 to the number of available data streams as returned by <i>()</i>
	numberOfBuffers:	The number of internal buffers to be allocated for the created data stream. This number must be at least 1, recommended is ≥ 3 .
	bufferSize:	(Optional) The size of one buffer in bytes. This parameter can be 0 in which case the size of a buffer will be defined from the standard 'PayloadSize' feature of a device.
	Out: dataStreamOut	A handle to the data stream that was created.
<u>Return value:</u>	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .	
<u>Comment:</u>	Use this function in combination with <i>csiGetNumberOfDataStreams()</i> to get the total number of available data streams in the camera.	

csiCloseDataStream	
Close the data stream	
Syntax	<code>csiErr csiCloseDataStream(csiHandle dataStream)</code>
<u>Parameters:</u>	In: <code>dateStream</code> A handle to the data stream to be closed. Out: None
<u>Return value:</u>	Returns <code>csiSuccess</code> or an error defined in the <code>csiErr-Enum</code> .
<u>Comment:</u>	Make sure to release all used buffers with <code>csiReleaseBuffer()</code> and unregister all events with <code>csiUnregisterEvent()</code> 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 <code>csiStopAcquisition()</code> .

csiRegisterEvent	
Register an event which will be signaled in the case the desired event is triggered	
Syntax	<code>csiErr csiRegisterEvent(csiHandle moduleHandle, csiEventType evtType, csiHandle* eventOut, csiModuleLevel module = CSI_DEVICE_MODULE)</code>
<u>Parameters:</u>	In: <code>moduleHandle</code> Handle to the module that is used to register an event. This can be either a device handle or a data stream handle. <code>evtType</code> The type of event that should be registered. <code>module</code> The module level where the event should be registered on. Out: <code>eventOut</code> A handle to the event that was registered. Use this handle to wait for events using the <code>csiWaitForEvent()</code> or <code>csiGetNextImage()</code> functions.
<u>Return value:</u>	Returns <code>csiSuccess</code> or an error defined in the <code>csiErr-Enum</code> .
<u>Comment:</u>	

csiWaitForEvent	
Wait for a desired event to happen	
Syntax	<code>csiErr csiWaitForEvent(csiHandle evt, uint64_t timeoutMilliseconds, csiEventData** evtDataOut)</code>
<u>Parameters:</u>	In: <code>evt</code> the handle of the event to wait for <code>timeoutMilliseconds</code> Timeout after the waiting stops if no event was received Out: <code>evtDataOut</code> the event data for further use. See <code>csiEventData</code> . In case of an error or timeout the output will be NULL, therefore please check the return value before using it.
<u>Return value:</u>	Returns <code>csiSuccess</code> or an error defined in the <code>csiErr-Enum</code> .
<u>Comment:</u>	After the event was registered with <code>csiRegisterEvent()</code> 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 (<code>csiEventData</code>). There exists also an event data structure for image data (<code>csiImageEventData</code>) which contains more information on the image that was received. Note: Also see <code>csiGetNextImage()</code> which can be used as convenience function to wait for new image data events.

csiUnregisterEvent	
Unregister a specific event from the event handler	
Syntax	<i>csiErr csiUnregisterEvent(csiHandle evt)</i>
<u>Parameters:</u>	In: evt the handle to the event that should be unregistered. Out: None
<u>Return value:</u>	Returns csiSuccess or an error defined in the csiErr-Enum.
<u>Comment:</u>	Unregistering the event will cancel all active pending calls to <i>()</i> or <i>csiGetNextImage()</i>

csiEventKill	
Cancel all waiting functions related to this event.	
Syntax	<i>csiErr csiEventKill(csiHandle evt)</i>
<u>Parameters:</u>	In: evt the handle of the event to be canceled Out: None
<u>Return value:</u>	Returns csiSuccess or an error defined in the csiErr-Enum.
<u>Comment:</u>	Any pending call to <i>()</i> or <i>csiGetNextImage()</i> on this event will be canceled.

csiStartAcquisition	
Start the acquisition on the device and created data streams	
Syntax	<i>csiErr csiStartAcquisition(csiHandle device, csiAcquisitionMode mode)</i>
<u>Parameters:</u>	In: device: Handle provided by the <i>csiOpenDevice</i> -function mode: Acquisition mode as defined in csiAcquisitionMode Out: None
<u>Return value:</u>	Returns csiSuccess or an error defined in the csiErr-Enum.
<u>Comment:</u>	This function will start the acquisition on the camera device passed with the <i>device</i> parameter and on all created data streams of that device.

csiStopAcquisition	
Stop the acquisition on the device	
Syntax	<i>csiErr csiStopAcquisition(csiHandle device)</i>
<u>Parameters:</u>	In: device: Handle provided by the <i>csiOpenDevice</i> -function Out:
<u>Return value:</u>	Returns csiSuccess or an error defined in the csiErr-Enum.
<u>Comment:</u>	This function will stop the acquisition on the camera device passed with the <i>device</i> parameter and on all created data streams of that device.

csiGetNextImage Get the next image from the data stream	
Syntax	<code>csiErr csiGetNextImage(csiHandle dataStream, csiNewBufferData** bufferInfoOut, uint64_t timeoutMilliseconds)</code>
Parameters:	<p>In: <code>dataStream</code> the handle of the data stream to wait for images on. Requires a previous call to <code>csiNewBufferData()</code> to register for new image events on that stream.</p> <p><code>timeoutMilliseconds</code> Timeout after the waiting stops if no event was received.</p> <p>Out: <code>bufferInfoOut</code> the event data structure containing the image data and information, see <code>csiNewBufferData()</code>.</p>
Return value:	Returns <code>csiSuccess</code> or an error defined in the <code>csiErr-Enum</code> .
Comment:	<p>This is a convenience function that is recommended to use for image acquisition. It is an alternative to <code>csiNewBufferData()</code> that returns a more general representation of the event data. Please note that a <code>CSI_EVT_NEWIMAGEDATA</code> event must be registered on the data stream to be able to wait for new images.</p> <p>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 <code>csiReleaseImage()</code>.</p>

csiReleaseImage Release the image back into the processing buffer from the device	
Syntax	<code>csiErr csiReleaseImage(csiHandle dataStream, csiNewBufferData* bufferInfo)</code>
Parameters:	<p>In: <code>dataStream</code> the handle to the data stream this buffer belongs to. This handle is also part of the <code>csiNewBufferData()</code> structure</p> <p><code>bufferInfo</code> Pointer to the buffer event data that was previously received from <code>csiNewBufferData()</code> or <code>csiGetNextImage()</code></p> <p>Out: None</p>
Return value:	Returns <code>csiSuccess</code> or an error defined in the <code>csiErr-Enum</code> .
Comment:	<p>Releases 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.</p> <p>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.</p>

csiGetAcquisitionStatistics Retrieve the statistical buffer regarding the data stream (e.g. transmitted frames, etc.)	
Syntax	<code>csiErr csiGetAcquisitionStatistics(csiHandle dataStream, csiAcquisitionStatistics* stats)</code>
Parameters:	<p>In: <code>datastream</code> Handle to the data stream the acquisition statistics should be collected on.</p> <p>Out: <code>stats</code> The acquisition statistics, see Fehler! Verweisquelle konnte nicht gefunden werden..</p>
Return value:	Returns <code>csiSuccess</code> or an error defined in the <code>csiErr-Enum</code> .
Comment:	

4.6 File transfer functions

csiGetUpdateFileType		Request the update file type of a file (if available)
Syntax	<i>csiErr csiGetUpdateFileType(csiHandle device, const char* fileName, char* fileTypeOut, size_t bufferSize)</i>	
Parameters:	In: device: Handle provided by the <i>csiOpenDevice</i> -function filename: The path to a file that should be checked bufferSize: the size of the output buffer <i>fileTypeOut</i> Out: <i>fileTypeOut</i> If the file is a valid file that can be used to update on the device, this buffer should contain the type of that file as string representation	
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .	
Comment:	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 camera (for example firmware, sensor file, user settings, XML description, reference files, etc.). It can be used to detect if a file is a valid file that can be used for an update and to detect the type of the file. It is required to first get the type of a file before uploading it to the device to see if it is a valid file.	

csiFileDownloadToDevice		Downloads a file located on the local PC to the camera
Syntax	<i>csiErr csiFileDownloadToDevice(csiHandle device, csiHandle localDevice, const char* fileName, const char* fileType, uint64_t timeoutMilliseconds, csiMemTransferCallbackFunc listener = NULL, csiMemTransferUserData* userdata = NULL)</i>	
Parameters:	In: device: Handle provided by the <i>csiOpenDevice</i> -function localDevice: Handle of the local device module (usually the same handle value of device) filename: Full path of the file to be uploaded fileType: Type of the update file as returned from <i>()</i> timeoutMilliseconds: Timeout for the update procedure in milliseconds. 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. listener: Optional callback function that will be called during the update process to inform about the progress. userdata: Optional user data that will be passed as parameter to the progress callback function. Out: None	
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .	
Comment:		

csiFileUploadFromDevice		Uploads a file from device to the local PC
Syntax	<i>csiErr csiFileUploadFromDevice(csiHandle device, csiHandle localDevice, const char* fileName, const char* fileType, uint64_t timeoutMilliseconds, csiMemTransferCallbackFunc listener = NULL, csiMemTransferUserData* userdata = NULL)</i>	
Parameters:	In: device: Handle provided by the <i>csiOpenDevice</i> -function localDevice: Handle of the local device module (usually the same handle value of device) filename: Name of the file on the local PC fileType: the type of the file, corresponds to the name of the file on the device. timeoutMilliseconds: Timeout for the upload procedure in milliseconds. 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. listener: Optional callback function that will be called during the transfer process to inform about the progress. userdata: Optional user data that will be passed as parameter to the progress callback function. Out: None	
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .	
Comment:		

csiFileDownloadToDeviceEx	
Downloads a file located on the local PC to the camera providing custom options	
Syntax	<i>csiErr</i> csiFileDownloadToDeviceEx(<i>csiHandle</i> device, <i>csiHandle</i> localDevice, const <i>csiDownloadParams</i> params, <i>uint64_t</i> timeoutMilliseconds, <i>csiMemTransferCallbackFunc</i> listener = NULL, <i>csiMemTransferUserData*</i> userdata = NULL)
Parameters:	<p>In: device: Handle provided by the <i>csiOpenDevice</i>-function</p> <p>localDevice: Handle of the local device module (usually the same handle value of device)</p> <p>params: File related parameter and options, see csiDownloadParams</p> <p>timeOutMilliSeconds: Timeout for the update procedure in milliseconds.</p> <p>listener: Optional callback function that will be called during the update process to inform about the progress.</p> <p>userdata: Optional user data that will be passed as parameter to the progress callback function.</p> <p>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.</p> <p>Out: None</p>
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr</i> -Enum.
Comment:	

csiFileUploadFromDeviceEx	
Uploads a file from device to the local PC providing custom options	
Syntax	<i>csiErr</i> csiFileUploadFromDeviceEx(<i>csiHandle</i> device, <i>csiHandle</i> localDevice, const <i>csiUploadParams</i> params, <i>uint64_t</i> timeoutMilliseconds, <i>csiMemTransferCallbackFunc</i> listener = NULL, <i>csiMemTransferUserData*</i> userdata = NULL)
Parameters:	<p>In: device: Handle provided by the <i>csiOpenDevice</i>-function</p> <p>localDevice: Handle of the local device module (usually the same handle value of device)</p> <p>params: File related parameter and options, see csiUploadParams</p> <p>timeoutMilliseconds: Timeout for the upload procedure in milliseconds.</p> <p>listener: Optional callback function that will be called during the transfer process to inform about the progress.</p> <p>userdata: Optional user data that will be passed as parameter to the progress callback function.</p> <p>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.</p> <p>Out: None</p>
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr</i> -Enum.
Comment:	

4.7 Calibration Data Generator functions

csiGenerateAndUploadCalibrationData	
Generate and upload calibration file to the camera	
Syntax	<i>csiErr</i> csiGenerateAndUploadCalibrationData(<i>csiHandle</i> devHandle, <i>csiCalibrationLUT</i> LUTSelector, <i>csiReferenceImgLoadMode</i> imgLoadMode, <i>csiCalibrationParams</i> calibParam, <i>csiNewBufferEventData*</i> reflImage, const char* imgFile)
Parameters:	<p>In: devHandle: Handle provided by the <i>csiOpenDevice</i>-function</p> <p>LUTSelector: The desired LUT in camera memory where the data to be uploaded</p> <p>imgLoadMode: Specifying the source of the input image used to generate calibration data. The input image can be acquired from camera of loaded from PC memory</p> <p>calibParam: The input parameters to configure and generate calibration data</p> <p>reflImage: If the input image is captured from the camera, it can be loaded in this argument if the input image is loaded from the PC, this can be a "nullptr"</p> <p>imgFile: The absolute path of the input image file to be used for calibration generation. if the input image is acquired from camera, this can be a "nullptr"</p>
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr</i> -Enum.
Comment:	In this function, even if input data is provided to both arguments of type " <i>csiNewBufferEventData</i> " and <i>imageFile</i> path, the argument " <i>imgLoadMode</i> " of type " <i>csiReferenceImgLoadMode</i> " determines which input image data should be used for calibration generation.

csiGenerateAndSaveCalibrationDataToFile		Generate and save calibration file to the PC memory in provided file path
Syntax	<i>csiErr csiGenerateAndSaveCalibrationDataToFile (csiHandle devHandle, CalibrationMode calibMode, csiReferenceImgLoadMode imgLoadMode, csiCalibrationParams calibParam, csiNewBufferEventData *refImage, const char* imgFile, const char* calibrationFile)</i>	
Parameters:	In: devHandle:	Handle provided by the <i>csiOpenDevice</i> -function
	calibMode:	The desired calibration mode. Either DSNU or PRNU
	imgLoadMode	Specifying the source of the input image used to generate calibration data. The input image can be acquired from camera or loaded from PC memory
	calibParam	The input parameters to configure and generate calibration data
	refImage	If the input image is captured from the camera, it can be loaded in this argument if the input image is loaded from the PC, this can be a "nullptr"
	imgFile	The absolute path of the input image file to be used for calibration generation if the input image is acquired from camera, this can be a "nullptr"
	calibrationFile	The absolute path of the output calibration file, where the file to be saved. The appropriate file type of ".dsnu" or ".prnu" should be mentioned
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .	
Comment:	In this function, even if input data is provided to both arguments of type "csiNewBufferEventData" and imageFile path, the argument "imgLoadMode" of type "csiReferenceImgLoadMode" determines which input image data should be used for calibration generation.	

csiGenerateReferenceImage		Generates calibration data and applies it on the input image to generate calibrated output image
Syntax	<i>csiErr csiGenerateAndSaveCalibrationDataToFile (csiHandle devHandle, CalibrationMode calibMode, csiReferenceImgLoadMode imgLoadMode, csiCalibrationParams calibParam, csiNewBufferEventData *refImage, const char* imgFile, int *referenceImage, int &resMaxValue)</i>	
Parameters:	In: devHandle:	Handle provided by the <i>csiOpenDevice</i> -function
	calibMode:	The desired calibration mode. Either DSNU or PRNU
	imgLoadMode	Specifying the source of the input image used to generate calibration data. The input image can be acquired from camera or loaded from PC memory
	calibParam	The input parameters to configure and generate calibration data
	refImage	If the input image is captured from the camera, it can be loaded in this argument if the input image is loaded from the PC, this can be a "nullptr"
	imgFile	The absolute path of the input image file to be used for calibration generation if the input image is acquired from camera, this can be a "nullptr"
	refImage	The absolute path of the output calibration file, where the file to be saved. The appropriate file type of ".dsnu" or ".prnu" should be mentioned
	OUT: referenceImage	The calibration data is applied on the input image and generates calibrated image
	resMaxValue	The resultant max value of a pixel present in the calibrated output image
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .	
Comment:	In this function, even if input data is provided to both arguments of type "csiNewBufferEventData" and imageFile path, the argument "imgLoadMode" of type "csiReferenceImgLoadMode" determines which input image data should be used for calibration generation.	

4.8 Memory transfer functions

csiReadMemory		Read memory from a register address on the device
Syntax	<i>csiErr csiReadMemory(csiHandle device, uint64_t address, char* buffer, size_t sizeBytes)</i>	
Parameters:	In: device:	Handle provided by the <i>csiOpenDevice</i> -function
	address:	The register address to read from
	buffer:	The buffer to which the data should be read
	sizeBytes:	The size of the buffer to which the buffer should be read and at the same time the number of bytes to read from the address.
	Out:	
Return value:	Returns <i>csiSuccess</i> or an error defined in the <i>csiErr-Enum</i> .	

<u>Comment:</u>	
-----------------	--

csiWriteMemory Write memory to a register address on the device	
<u>Syntax</u>	<code>csiErr csiWriteMemory(csiHandle device, uint64_t address, const char* buffer, size_t sizeBytes)</code>
<u>Parameters:</u>	<p>In:</p> <ul style="list-style-type: none"> device: Handle provided by the <code>csiOpenDevice</code>-function address: Address of the register on the device to which the memory should be written buffer: Buffer holding the data to write sizeBytes: The number of bytes to write from buffer to the register address <p>Out:</p>
<u>Return value:</u>	Returns <code>csiSuccess</code> or an error defined in the <code>csiErr-Enum</code> .
<u>Comment:</u>	

4.9 Helper functions

csiBitsPerPixelFromFormat	
Syntax	<i>unsigned char csiBitsPerPixelFromFormat(const csiPixelFormat format)</i>
<u>Parameters:</u>	In: format Out:
<u>Return value:</u>	Returns the number of bits per pixels for the given pixel format or an error defined in the csiErr-Enum.
<u>Comment:</u>	

csiGetErrorDescription	
Returns a human readable description of an error code	
Syntax	<i>csiErr csiGetErrorDescription(csiErr error, char* bufferOut, size_t bufferSize)</i>
<u>Parameters:</u>	In: error: error code to retrieve the text for bufferSize: size of the provided text buffer Out: bufferOut: char-buffer where the error text will be written to
<u>Return value:</u>	Returns csiSuccess or an error defined in the csiErr-Enum.
<u>Comment:</u>	

csiGetLibraryVersion	
Returns the current library version	
Syntax	<i>csiErr csiGetLibraryVersion(uint32_t* major, uint32_t* minor, uint32_t* patch, uint32_t* revision, uint32_t* build)</i>
<u>Parameters:</u>	In: None Out: major, minor, patch, revision: The different version numbers. Format: major.minor.patch.revision. If any of the input values is NULL, it will be ignored.
<u>Return value:</u>	Always returns csiSuccess.
<u>Comment:</u>	



4.10 Enumerations

csiPixelFormat	
Defines the currently supported pixel data formats	
Definition	<pre>typedef enum csiPixelFormat { CSI_PIX_FORMAT_UNKNOWN = 0x00000000, /// Mono formats CSI_PIX_FORMAT_MONO8 = 0x01080001, CSI_PIX_FORMAT_MONO10 = 0x01100003, CSI_PIX_FORMAT_MONO10_PACKED = 0x010A0046, CSI_PIX_FORMAT_MONO12 = 0x01100005, CSI_PIX_FORMAT_MONO12_PACKED = 0x010C0047, CSI_PIX_FORMAT_MONO16 = 0x01100007, /// Color formats CSI_PIX_FORMAT_RGB8 = 0x02180014, CSI_PIX_FORMAT_RGB10_PACKED = 0x0220001D, CSI_PIX_FORMAT_RGBA8 = 0x02200016, CSI_PIX_FORMAT_BGR8 = 0x02180015, CSI_PIX_FORMAT_RGB16 = 0x02300033, /// Bayer formats CSI_PIX_FORMAT_BayerGR8 = 0x01080008 CSI_PIX_FORMAT_BayerRG8 = 0x01080009 CSI_PIX_FORMAT_BayerGB8 = 0x0108000A CSI_PIX_FORMAT_BayerBG8 = 0x0108000B CSI_PIX_FORMAT_BayerGR12 = 0x01100010 CSI_PIX_FORMAT_BayerRG12 = 0x01100011 CSI_PIX_FORMAT_BayerGB12 = 0x01100012 CSI_PIX_FORMAT_BayerBG12 = 0x01100013 } csiPixelFormat;</pre>
Elements	The value of each entry corresponds to its value in PFNC standard. Please refer to the PFNC standard for more information on each specific format: https://www.emva.org/standards-technology/genicam/genicam-downloads/

csiDeviceAccessMode											
Defines the mode in which a device will be opened											
Definition	<pre>typedef enum csiDeviceAccessMode { CSI_DEV_MODE_UNKNOWN = 0x00, CSI_DEV_MODE_NONE = 0x01, CSI_DEV_MODE_EXCLUSIVE, CSI_DEV_MODE_READ, CSI_DEV_MODE_CONTROL } csiDeviceAccessMode;</pre>										
Elements	<table border="0"> <tr> <td>CSI_DEV_MODE_UNKNOWN:</td> <td>Undefined access mode</td> </tr> <tr> <td>CSI_DEV_MODE_NONE:</td> <td>No device access mode specified</td> </tr> <tr> <td>CSI_DEV_MODE_EXCLUSIVE</td> <td>The device will be opened exclusively; no other application will be allowed to open the device.</td> </tr> <tr> <td>CSI_DEV_MODE_READ</td> <td>The device will be opened in read only mode, other application might open it in read only mode too.</td> </tr> <tr> <td>CSI_DEV_MODE_CONTROL</td> <td>The device will be opened in control mode (read/write), other application might still be able to open it in read mode.</td> </tr> </table>	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.
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													
Defines the current access status of a device as returned from device discovery													
Definition	<pre>typedef 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 } csiDeviceAccessStatus;</pre>												
Elements	<table border="0"> <tr> <td>CSI_DEV_ACCESS_STATUS_READWRITE</td> <td>Device is not yet open and can be opened in read/write mode.</td> </tr> <tr> <td>CSI_DEV_ACCESS_STATUS_READONLY</td> <td>Device is not yet open and can be opened in read only mode.</td> </tr> <tr> <td>CSI_DEV_ACCESS_STATUS_NOACCESS</td> <td>Device is listed but cannot be opened.</td> </tr> <tr> <td>CSI_DEV_ACCESS_STATUS_BUSY</td> <td>Device is open by another process thus cannot be opened again.</td> </tr> <tr> <td>CSI_DEV_ACCESS_STATUS_OPEN_READWRITE</td> <td>Device already owned by this producer in read write mode.</td> </tr> <tr> <td>CSI_DEV_ACCESS_STATUS_OPEN_READ</td> <td>Device already owned by this producer in read only mode.</td> </tr> </table>	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.
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.												

csiFeatureType									
Defines the data type of a feature									
Definition	<pre>typedef enum csiFeatureType { CSI_UNKNOWN_TYPE, CSI_BOOLEAN_TYPE, CSI_INT_TYPE, CSI_FLOAT_TYPE, CSI_STRING_TYPE, CSI_ENUMERATION, CSI_CATEGORY, CSI_COMMAND, CSI_REGISTER, CSI_PORT } csiFeatureType;</pre>								
Elements	<table border="0"> <tr> <td>CSI_UNKNOWN_TYPE</td> <td>Unknown type</td> </tr> <tr> <td>CSI_BOOLEAN_TYPE</td> <td>Boolean data type</td> </tr> <tr> <td>CSI_INT_TYPE</td> <td>Integer data type</td> </tr> <tr> <td>CSI_FLOAT_TYPE</td> <td>Floating point data type</td> </tr> </table>	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_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	Enumeration feature type
	CSI_CATEGORY	Category feature type
	CSI_COMMAND	Command feature type
	CSI_REGISTER	Register feature type
	CSI_PORT	Port of the feature note map

csiAccessMode		Defines the access mode of a feature
Definition	<pre>typedef enum csiAccessMode { CSI_ACCESS_UNKNOWN, CSI_ACCESS_NOT_AVAILABLE, CSI_ACCESS_READ_ONLY, CSI_ACCESS_READ_WRITE, CSI_ACCESS_WRITE_ONLY } csiAccessMode;</pre>	
Elements	CSI_ACCESS_UNKNOWN CSI_ACCESS_NOT_AVAILABLE CSI_ACCESS_READ_ONLY CSI_ACCESS_READ_WRITE CSI_ACCESS_WRITE_ONLY	Unknown access mode, feature might not be accessible Feature is flagged as Not Available (N/A). 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. Feature is read only. Feature can be accessed in read and write mode. Feature can only be written.

csiFeatureVisibility		Defines the visibility of a feature depending on the role of a user
Definition	<pre>typedef enum csiFeatureVisibility { CSI_VISIBILITY_BEGINNER=1, CSI_VISIBILITY_EXPERT, CSI_VISIBILITY_GURU, CSI_VISIBILITY_DEVELOPER, CSI_VISIBILITY_INVISIBLE } csiFeatureVisibility;</pre>	
Elements	CSI_VISIBILITY_BEGINNER CSI_VISIBILITY_EXPERT CSI_VISIBILITY_GURU CSI_VISIBILITY_DEVELOPER CSI_VISIBILITY_INVISIBLE	Feature is visible to beginner users and higher Feature is visible to expert users and higher Feature is visible to guru users and higher Feature is visible to developer users only Feature is invisible to any user

csiModuleLevel		Defines the module level on which a specific action should be performed
Definition	<pre>typedef enum csiModuleLevel { CSI_UNKNOWN_MODULE, CSI_TRANSPORTLAYER_MODULE, CSI_INTERFACE_MODULE, CSI_DEVICE_MODULE, CSI_LOCAL_DEVICE_MODULE, CSI_STREAM_MODULE, CSI_BUFFER_MODULE } csiModuleLevel;</pre>	
Elements	CSI_UNKNOWN_MODULE CSI_TRANSPORTLAYER_MODULE CSI_INTERFACE_MODULE CSI_DEVICE_MODULE CSI_LOCAL_DEVICE_MODULE CSI_STREAM_MODULE CSI_BUFFER_MODULE	Unknown module level Transport layer module (System module) Interface module Device module Local device module Data stream module Buffer module

csiDisplayNotation		Defines the display notation for a floating-point feature
Definition	<pre>typedef enum csiDisplayNotation { CSI_NOTATION_AUTOMATIC, CSI_NOTATION_FIXED, CSI_NOTATION_SCIENTIFIC, } csiDisplayNotation;</pre>	
Elements	CSI_NOTATION_AUTOMATIC CSI_NOTATION_FIXED CSI_NOTATION_SCIENTIFIC	Notation not specified, can be decided by the application Fixed notation Scientific notation

csiRepresentation		Defines how a feature value should be represented when printed in UI
Definition	<pre>typedef enum csiRepresentation { CSI_REPRESENTATION_LINEAR, CSI_REPRESENTATION_LOGARITHMIC, CSI_REPRESENTATION_BOOLEAN, CSI_REPRESENTATION_PURENUMBER, CSI_REPRESENTATION_HEX, CSI_REPRESENTATION_IP, CSI_REPRESENTATION_MAC, CSI_REPRESENTATION_UNDEFINED } csiRepresentation;</pre>	
Elements	CSI_REPRESENTATION_LINEAR CSI_REPRESENTATION_LOGARITHMIC CSI_REPRESENTATION_BOOLEAN CSI_REPRESENTATION_PURENUMBER CSI_REPRESENTATION_HEX CSI_REPRESENTATION_IP CSI_REPRESENTATION_MAC CSI_REPRESENTATION_UNDEFINED	Linear representation (default) Logarithmic representation Boolean representation (true / false) Represent as pure number Hexadecimal representation (0x...) IP address representation Mac address representation Not defined, use default

csiLogLevel	
	Defines the severity of log messages coming from the SDK
Definition	<pre>typedef enum csiLogLevel { CSI_LOGLEVEL_ERROR = 1, CSI_LOGLEVEL_WARN = 2, CSI_LOGLEVEL_INFO = 4, CSI_LOGLEVEL_DEBUG = 8, CSI_LOGLEVEL_TRACE = 16, } csiLogLevel;</pre>
Elements	<pre>CSI_LOGLEVEL_ERROR CSI_LOGLEVEL_WARN CSI_LOGLEVEL_INFO CSI_LOGLEVEL_DEBUG CSI_LOGLEVEL_TRACE</pre>

csiErr																																																							
	Defines possible error values																																																						
Definition	<pre>typedef 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, csiOError = -118, csiParsingError = -119, csiInvalidValue = -120, csiResourceExhausted = -121, csiOutOfMemory = -122, csiBusy = -123, csiUnknown = -200, csiCustomErr = 0x0f000000 } csiErr;</pre>																																																						
Elements	<table border="0"> <tr> <td>csiSuccess</td> <td>No error</td> </tr> <tr> <td>csiNotInitialized</td> <td>System is not initialized, call <i>csiInit()</i> first</td> </tr> <tr> <td>csiInvalidState</td> <td>An invalid state occurred, see log output for more information</td> </tr> <tr> <td>csiNotOpened</td> <td>There was an action that requires the device / network / stream to be opened</td> </tr> <tr> <td>csiNoImageDataAvailable</td> <td>There was no image data available</td> </tr> <tr> <td>csiNotFound</td> <td>General error that the requested information was not found, see log for more detailed info on this error.</td> </tr> <tr> <td>csiInvalidParameter</td> <td>A parameter had an invalid value</td> </tr> <tr> <td>csiNotAvailable</td> <td>An expected result or a resource was not available</td> </tr> <tr> <td>csiFunctionNotAvailable</td> <td>The called function or a sub-function is not available</td> </tr> <tr> <td>csiTimeout</td> <td>A timeout occurred</td> </tr> <tr> <td>csiAborted</td> <td>A pending operation was aborted</td> </tr> <tr> <td>csiFileOperationFailure</td> <td>There was an error during file operation, see log for more information</td> </tr> <tr> <td>csiFileOperationFatalError</td> <td>There was a fatal error during file operation, see log for more information</td> </tr> <tr> <td>csiNoAccess</td> <td>Access denied (e.g., when trying to write a read only feature)</td> </tr> <tr> <td>csiWrongBufferSize</td> <td>A given buffer was too small to store the requested data</td> </tr> <tr> <td>csiInvalidBuffer</td> <td>The requested buffer is not valid</td> </tr> <tr> <td>csiResourceInUse</td> <td>The requested resource is already in use by the transport layer</td> </tr> <tr> <td>csiNotImplemented</td> <td>A function that was called is not yet implemented</td> </tr> <tr> <td>csiInvalidHandle</td> <td>A handle passed as parameter is not valid</td> </tr> <tr> <td>csiOError</td> <td>The was an error during an IO operation (e.g. file or network)</td> </tr> <tr> <td>csiParsingError</td> <td>An error occurred when parsing an XML node map file</td> </tr> <tr> <td>csiInvalidValue</td> <td>A value that was passed parameter is not valid</td> </tr> <tr> <td>csiResourceExhausted</td> <td>A requested resource is exhausted (e.g. hard disk space)</td> </tr> <tr> <td>csiOutOfMemory</td> <td>Memory allocation failed, there is no more memory available</td> </tr> <tr> <td>csiBusy</td> <td>The requested operation cannot be executed because the system is busy</td> </tr> <tr> <td>csiUnknown</td> <td>Generic error, see log for more information</td> </tr> <tr> <td>csiCustomErr = -0x0f000000</td> <td>Custom error codes defined by specific transport layers</td> </tr> </table>	csiSuccess	No error	csiNotInitialized	System is not initialized, call <i>csiInit()</i> 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	csiOError	The was an error during an IO 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 = -0x0f000000	Custom error codes defined by specific transport layers
csiSuccess	No error																																																						
csiNotInitialized	System is not initialized, call <i>csiInit()</i> 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																																																						
csiOError	The was an error during an IO 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 = -0x0f000000	Custom error codes defined by specific transport layers																																																						

csiAcquisitionMode					
	Defines acquisition mode				
Definition	<pre>typedef enum csiAcquisitionMode { CSI_ACQUISITION_SINGLE_FRAME = 0x00000001, CSI_ACQUISITION_CONTINUOUS = 0xFFFFFFFF } csiAcquisitionMode;</pre>				
Elements	<table border="0"> <tr> <td>CSI_ACQUISITION_SINGLE_FRAME</td> <td>Acquire a single frame only</td> </tr> <tr> <td>CSI_ACQUISITION_CONTINUOUS</td> <td>Perform continuous frame acquisition</td> </tr> </table>	CSI_ACQUISITION_SINGLE_FRAME	Acquire a single frame only	CSI_ACQUISITION_CONTINUOUS	Perform continuous frame acquisition
CSI_ACQUISITION_SINGLE_FRAME	Acquire a single frame only				
CSI_ACQUISITION_CONTINUOUS	Perform continuous frame acquisition				

csiEventType		Defines event types that the user application can listen for
Definition	<pre>typedef enum csiEventType { CSI_EVT_NEWIMAGEDATA = 0x00, CSI_EVT_ERROR = 0x01, CSI_EVT_MODULE = 0x02, CSI_EVT_CUSTOM = 0x1000 } csiEventType;</pre>	
Elements	CSI_EVT_NEWIMAGEDATA CSI_EVT_ERROR CSI_EVT_MODULE CSI_EVT_CUSTOM	New image data event, can be registered on data stream module only Error event, can be registered on all module levels Generic module event, can be registered on all module levels Custom user defined event types

csiMemTransferStatus		Defines the status of memory transfer functions as it is provided in the tranfer callback
Definition	<pre>typedef enum csiMemTransferStatus { csiTransferStatusInit, csiTransferStatusInProgress, csiTransferStatusInProgressWaiting, csiTransferStatusFinishSuccess, csiTransferStatusFinishError, csiTransferStatusCancelOnError } csiMemTransferStatus;</pre>	
Elements	csiTransferStatusInit csiTransferStatusInProgress csiTransferStatusInProgressWaiting csiTransferStatusFinishSuccess csiTransferStatusFinishError csiTransferStatusCancelOnError	Transfer was initialized Transfer is in progress Transfer process is waiting for response from device Transfer finished successfully Transfer finished with an error Transfer was canceled after an error occurred

csiUploadOptions		Defines the file transfer options for upload (bit flag enum)
Definition	<pre>typedef enum csiUploadOptions { CSI_UPLOAD_NO_OPTIONS, CSI_UPLOAD_IGNORE_CHECKSUM } csiUploadOptions;</pre>	
Elements	CSI_UPLOAD_NO_OPTIONS CSI_UPLOAD_IGNORE_CHECKSUM	Default behaviour, i.e. checksum comparison No checksum comparison done

csiDownloadOptions		Defines the file transfer options for download (bit flag enum)
Definition	<pre>typedef enum csiDownloadOptions { CSI_DOWNLOAD_NO_OPTIONS } csiDownloadOptions;</pre>	
Elements	CSI_DOWNLOAD_NO_OPTIONS	no options, i.e. default behaviour

csiCalibrationLUT		Defines which calibration LUT it represents in device(camera) space
Definition	<pre>typedef enum csiCalibrationLUT { CSI_DSNU_LUT1 = 1, CSI_DSNU_LUT2, CSI_PRNU_LUT1, CSI_PRNU_LUT2 } csiCalibrationLUT;</pre>	
Elements	CSI_DSNU_LUT1 - DSNU's LUT 1, CSI_DSNU_LUT2 - DSNU's LUT 2 CSI_PRNU_LUT1 - PRNU's LUT 1 CSI_PRNU_LUT2 - PRNU's LUT 2	

csiReferenceImgLoadMode		Defines the kind of source image to be used for generating calibration data
Definition	<pre>typedef enum csiReferenceImgLoadMode { CSI_LOAD_REF_IMG_FROM_DISC, CSI_ACQUIRE_REF_IMG_FROM_CAMERA } csiReferenceImgLoadMode;</pre>	
Elements	CSI_LOAD_REF_IMG_FROM_DISC Input image to be loaded from the PC memory CSI_ACQUIRE_REF_IMG_FROM_CAMERA input image to be acquired from the camera	

CalibrationMode		Defines the calibration mode
Definition	<pre>typedef enum CalibrationMode { DSNU_CALIBRATION = 1, PRNU_CALIBRATION } CalibrationMode;</pre>	
Elements	DSNU_CALIBRATION PRNU_CALIBRATION	Dark Signal Non Uniformity Calibration mode Photo Response Non Uniformity Calibration mode

4.11 Structures

Struct-name csiFeatureParameter		
Variable type	Element name	Description
csiFeatureType	type	Data type of the feature, see csiFeatureType
csiFeatureVisibility	visibility	The visibility of a feature, see csiFeatureVisibility
csiAccessMode	access	How a feature can be accessed, see csiAccessModecsiAccessMode
csiDisplayNotation	displayNotation	How to display floating point features, see csiDisplayNotation . (Optional)
csiRepresentation	representation	How feature data should be represented, see csiRepresentation (Optional)
char	displayPrecision	Precision of floating-point value representation
int64_t	valueInt	Value of the feature, in case of integer feature type
int64_t	incrementInt	Possible increment for the feature value, in case of integer feature type
int64_t	minimumInt	Minimum for the feature value, in case of integer feature type
int64_t	maximumInt	Maximum for the feature value, in case of integer feature type
int64_t	validValueSetInt	The set of valid values for the feature, in case of integer feature type
size_t	validValueSetSizeInt	The number of elements in set of valid values, in case of integer feature type
double	valueFlt	Value of the feature, in case of floating-point feature type
double	incrementFlt	Possible increment for the feature value, in case of floating-point feature type
double	minimumFlt	Minimum for the feature value, in case of floating-point feature type
double	maximumFlt	Maximum for the feature value, in case of floating-point feature type
char[]	valueStr	Value of the feature, in case of string feature type
size_t	maximumStringLength	Maximum length of the string feature value
Int64_t	level	The level of a feature in the tree (for graphical representation)
uint32_t	enumCounter	Number of elements in the enumeration feature
char	enumIndex	The index of an enumeration entry
char[]	displayName	Display name of the feature for UI display
char[]	name	The name that identifies a feature
char[]	tooltip	Additional information about the feature that can be shown as tooltip in a GUI
char[]	valueUnit	Unit string to append to the value representation in a GUI
size_t	featureRegLength	Length of a register feature
int64_t	featureRegAddress	Address of a register feature
Bool	isFeature	Requested node is a feature

Struct-name csiEventData		
Variable type	Element name	Description
csiEventType	type	Type of the event, see csiEventType
csiHandle	sender	Handle to the sender of the event
csiModuleLevel	senderType	Module level of the sender handle, see csiModuleLevel
char*	tl_rawEventData	Raw data pointer to the event data as it was sent by the producer. This is just the raw data of the event which contains information about the type of event itself and not the value behind the event. See eventValue to get the actual value (e.g., image data) behind the event, if any.
size_t	tl_rawEventDataSizeBytes	Size of the eventData member in bytes.
char*	eventValue	The received value that was shipped together with the event. This can be for example the image data or a error description text in case of an error event. How to interpret the value depends on the type of event.
size_t	eventValueSizeBytes	Size of the eventValue member in bytes.
uint64_t	eventIdentifier	Event identifier

Struct-name csiMemTransferInfo		
---------------------------------------	--	--

Variable type	Element name	Description
csiHandle	device	Handle to the device where the transfer is running on
size_t	totalBytesToTransfer	Total number of bytes to be transferred
size_t	bytesTransferred	Current number of bytes already transferred
csiMemTransferStatus	status	Status of the memory transfer, see Fehler! Verweisquelle konnte nicht gefunden werden.
csiErr	errorCode	Error code in case an error occurred.
const char*	progressText	Progress information text

Struct-name csiTLProducerInfos		
Variable type	Element name	Description
char[]	transportLayerName	Name of a transport layer
char[]	transportLayerDisplayName	Display name of a transport layer for GUI representation
char[]	transportLayerType	Type of the transport layer as string
char[]	transportLayerPath	Full path to the transport layer library file (.cti file)
char[]	transportLayerID	Unique identifier of the transport layer as string
size_t	pathSizeInBytes	Length of the transport layer path

Struct-name csiTLInterfaceInfo		
Variable type	Element name	Description
char[]	interfaceDescription	Name or description of the interface
char[]	interfaceID	Unique identifier of the interface
<i>csiTLProducerInfos</i>	tlProducerInfos	Information about the transport layer the interface is connected to, see

Struct-name csiTLInterfaceDiscoveryInfo		
Variable type	Element name	Description
uint32_t	numInterfaces	Current number of interfaces found during discovery
double	progress	Discovery progress
bool	discoveryRunning	Indicates if the discovery is still ongoing (true) or finished (false)
<i>csiTLInterfaceInfo</i> []	interfaces	The list of interfaces found.

Struct-name csiDeviceInfo		
Variable type	Element name	Description
char[]	deviceIdIdentifier	Unique identifier of the device
char[]	name	Name of the device
char[]	model	Model name of the device
char[]	vendor	Vendor of the device
char[]	serialNumber	Serial number of the device
char[]	interfaceDescription	Name or description of the interface the device is connected to
char[]	interfaceID	Unique identifier of the interface the device is connected to
char[]	userName	Username when opening the device
char[]	version	Version of the device
int64_t	cameraSwPackagelsConsistent	
<i>csiTLProducerInfos</i>	tlProducerInfos	Information about the transport layer the device is connected to, see

csiDeviceAccessStatus	accessStatus	The current access status of the device, see csiDeviceAccessStatus
uint64_t	timestampFrequency	Frequency of the timestamps coming from the device

Struct-name csiDiscoveryInfo		
Variable type	Element name	Description
uint32_t	numDevices	Current number of devices found during discovery
double	progress	Discovery progress
bool	discoveryRunning	Indicates if the discovery is still ongoing (true) or finished (false)
csiDeviceInfo[]	devices	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 <i>csiGetDeviceInfo()</i> function.

Struct-name csiDataStreamInfo		
Variable type	Element name	Description
char[]	identifier	Unique identifier of a data stream
char[]	displayName	Display name of a data stream that can be used for GUI representation
uint32_t	index	Internal index of the data stream

Struct-name csiImageInfo		
Variable type	Element name	Description
uint32_t	width	Width of the image
uint32_t	height	Height of the image
uint32_t	linePitch	Line pitch of the image data in bytes
uint32_t	numChannels	Number of channels
csiPixelFormat	format	Pixel format of the image data, see csiPixelFormat

Struct-name csiNewBufferData		
Variable type	Element name	Description
csiEventType	type	Type of the event, this is always CSI_EVT_NEWIMAGEDATA for this type of event
csiHandle	sender	Sender of the event, a stream handle
csiModuleLevel	senderType	Type of the sender, this is always CSI_STREAM_MODULE for this type of event
char*	tl_rawEventData	Raw data pointer to the event data as it was sent by the producer. This is just the raw data of the event which contains information about the type of event itself and not the value behind the event. See eventValue to get the actual value (e.g. image data) behind the event, if any.
size_t	tl_rawEventDataSizeBytes	Size of the eventData member in bytes.
unsigned char*	eventValue	Pointer to the image data
size_t	eventValueSizeBytes	Size of the image data in bytes
uint64_t	eventIdentifier	Unique identifier of this event
csiHandle	bufferHandle	Handle to the buffer holding the image (for internal use)
uint64_t	imageNr	Number of the recorded image
uint64_t	bufferIdentifier	Unique identifier of the image, usually the pointer as integer representation
uint64_t	timestampMS	Timestamp of the image in milliseconds
uint64_t	timestampRaw	Raw timestamp of the image
csiImageInfo	imageInfo	Further image information, see

Struct-name csiAcquisitionStatistics		
Variable type	Element name	Description
Uin64_t	framesUnderrun	The number of frames that were received in the TL but not send to the application because of missing buffers.
uint64_t	framesDropped	Number of frames dropped during acquisition
uint64_t	framesAcquired	Total number of frames acquired in current acquisition
uint64_t	networkPacketsOK	For GigE Vision: The number of network packets received without errors.
uint64_t	networkPacketsError	For GigE Vision: The number of network packets sent with an error.

Struct-name csiFileTransferParams		
Variable type	Element name	Description
const char*	fileName	Full path of the file to be transferred
const char*	fileType	Type of the file as returned from <code>()</code>

Struct-name csiDownloadParams		
Variable type	Element name	Description
csiFileTransferParams	fileParams	File related params, see csiFileTransferParams
csiDownloadOptions	options	Options for download, see csiDownloadOptions

Struct-name csiUploadParams		
Variable type	Element name	Description
csiFileTransferParams	fileParams	File related params, see csiFileTransferParams
csiUploadOptions	Options	Options for upload, see csiUploadOptions

Struct-name csiCalibrationParams		
Variable type	Element name	Description
Boolean	enableROI	To enable ROI based PRNU calibration data generation
Integer	enableROI	The row number of the source image which is a beginning of ROI
Integer	heightROI	Height of the ROI used for PRNU calibration generation
Boolean	enableExtrapolationLeft	To enable/disable extrapolation based PRNU calibration generation on the left side of the image
Boolean	enableExtrapolationRight	To enable/disable extrapolation based PRNU calibration generation on the right side of the image
Integer	extrapolationLeft	The first column on the left side of the image that starts the extrapolation area in left side
Integer	extrapolationRight	The last column on the right side of the image that ends the extrapolation area in right side
Integer	extrapolationWidth	The width of the extrapolation area on both left and right side
Unsigned integer	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.
Integer	TargetValue	For PRNU, the target value of the resultant pixel value. Example, the max value of 8 bit pixel is 255
Boolean	calclmprove	To enable outlier suppression
Double	contrast	To modify contrast of image
Double	brightness	To modify brightness of image

5 Installation

5.1 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.

5.1.1 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\CSGenICam\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**

5.2 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 20.04 LTS

5.2.1 Preparation

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

5.2.2 Step by Step Installation Ubuntu 20.04

- 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.

5.2.3 Installer Contents

The software package is grouped into the following components:

- **SDK** The programming interface and library for customer applications
 Location: **/usr/CSGenICamSDK/lib/libCSI.so**
 /usr/CSGenICamSDK/include/csi/csi.h
 /usr/CSGenICamSDK/share/CSGenICam/cmake/*
- **GCT-2** The camera configuration and acquisition application with graphical interface
 Location: **/usr/CSGenICamSDK/bin/gct_app**
 To run this application, execute the command "GCT-2" in console or in search bar look for "GCT-2". Both these options will execute the shell script in "**usr/CSGenICamSDK/bin/run-gct2.sh**". This shell script configures the dependent library paths and environmental variables for correct functioning of "gct_app" application.
- **SDK Examples** Example source code (C++) that shows the basic usage of the SDK
 Location: **/usr/CSGenICamSDK/share/CSGenICam/examples**
 Note: The precompiled executable for these samples can also be found in the "**usr/CSGenICamSDK/bin/**" folder. These executables requires libraries present in "**usr/CSGenICamSDK/lib/**" which is not a standard library path. Hence before running the executable, it is important to add this path to "LD_LIBRARY_PATH" environmental variable. Please execute the following command,
 -export
 LD_LIBRARY_PATH=/usr/CSGenICamSDK/lib/\${LD_LIBRARY_PATH}:
 +: \$LD_LIBRARY_PATH}
- **Documentation** Documentation of the SDK
 Location: **/usr/CSGenICamSDK/share/CSGenICam/doc/final/CS-GenCam_SDK.pdf**
- **GenTL Producers** A GigE interface Transport Layer developed by Sensor to Image GmbH
 Location: **/usr/CSGenICamSDK/share/CSGenICam/S2I_GenTL**