Chromasens GEN<i>CAM-SDK | Manual

R07 / 2023-10-06

Table of Contents

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	typedef enum csiEventType {
	CSI_EVT_NEWIMAGEDATA = 0x00,
	CSI_EVT_ERROR = 0x01,
	CSI_EVT_MODULE = 0x02,
	CSI_EVT_CUSTOM = 0x1000
	} csiEventType;

Elements	CSI_EVT_NEWIMAGEDATA: New image data received CSI_EVT_ERROR: Error occurred in the SDK CSI_EVT_MODULE: General event notification CSI_EVT_CUSTOM: A custom event was triggered

Structures:

For example:

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

Functions:

For example:

csiDiscoverDevices	Searches for the devices currently connected to the system
Syntax	csiErr csiDiscoverDevices(csiDiscoveryInfo* discoveryInfoOut, uint64_t timeoutMilliseconds, csiDiscoveryInfoCallbackFunc discCallbackFunc = NULL, const char* additionalSearchPaths = NULL, bool overrideSearchPath CSI_DEFAULT_PARAM_FALSE);
Parameters:	In: 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 AdditionalSearchPaths: as default only the paths given in the system variable “GENICAM_GENTL64_PATH” are being searched for the used transport layers overrideSearchPath: If set, only the given path is searched for transport layers to use. Out: discoveryInfoOut: The structure will be filled with the available devices
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	

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

Requirements:

Supported operating systems:

Windows: Windows 10 Version 2

Linux: Ubuntu >= 18.X

Supported compiler:

Visual Studio >= 2015

GCC

3 Getting started

This chapter will describe the basic functions/sequences needed to handle the basic functionality of the camera.

Ready to use-Examples are also shipped with the SDK in order to demonstrate the usage of the SDK regarding getting/setting features and acquiring images.

3.1 Initialization of the SDK

Before accessing any other functions of the SDK, an initialization needs to be done.

Please refer to 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 “`csiGetFeatureFloat`” to get a float parameter.

To retrieve additional information the function “`csiGetFeatureParameter`” exists. This function will fill a csiFeatureParameter-structure which provides information about the display name, minimum and maximum values, etc. This function is especially useful if you do not know the valid thresholds of a parameter.

Please be careful when treating string features. You must not exceed the maximum length! This can also be retrieved with the function “`csiGetFeatureParameter`”. The parameter “`maximumStringLength`” of the 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.

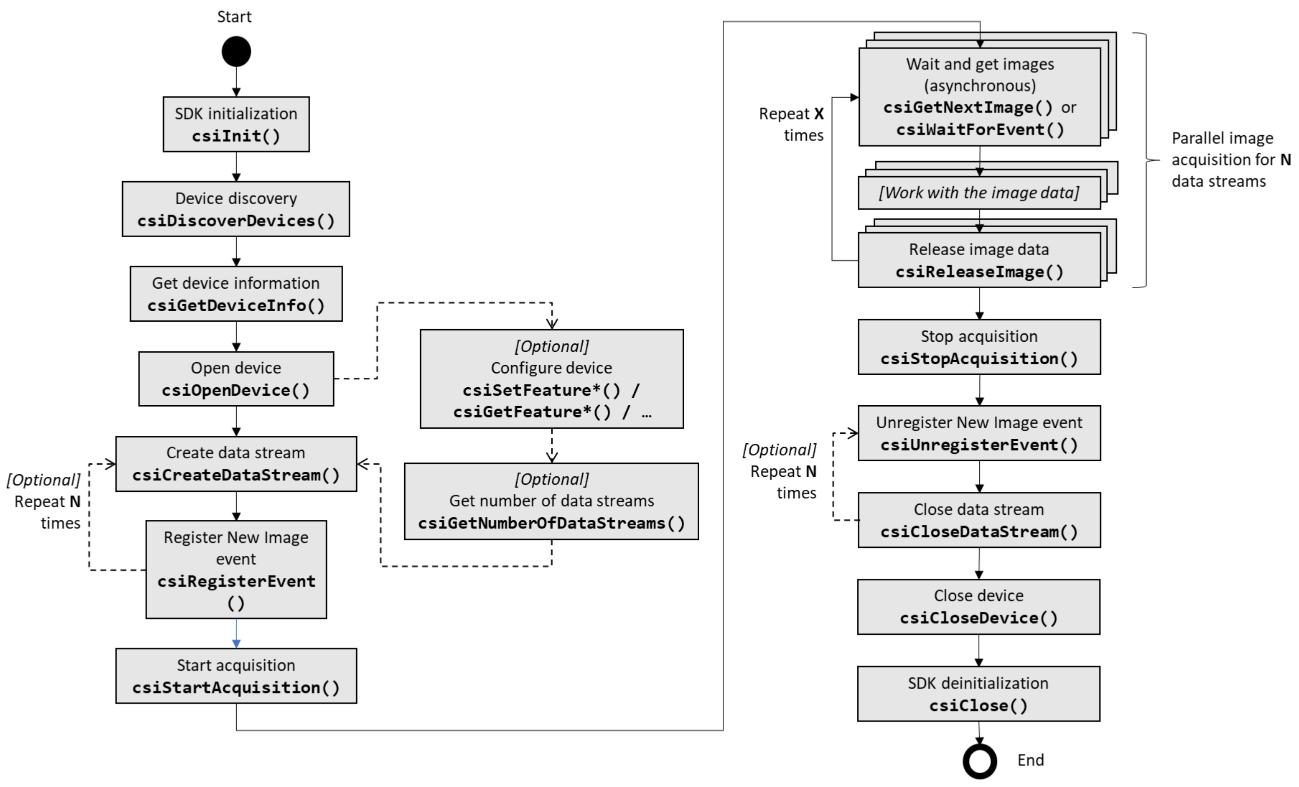
To set the values please use the type-specific set-functions. For example, use “`csiSetFeatureInt`” for an integer value.

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.



Start
SDK initialization csiInit()
Device discovery csiDiscoverDevices()
Get device information csiGetDeviceInfo()
Open device csiOpenDevice()
[Optional] Configure device csiSetFeature*() / csiGetFeature*() / …
Create data stream csiCreateDataStream()
[Optional] Get number of data streams csiGetNumberOfDataStreams()
[Optional] Repeat N times
Register New Image event csiRegisterEvent()
Start acquisition csiStartAcquisition()
Wait and get images (asynchronous) csiGetNextImage() or csiWaitForEvent()
Repeat X times
[Work with the image data]
Release image data csiReleaseImage()
Parallel image acquisition for N data streams
Stop acquisition csiStopAcquisition()
Unregister New Image event csiUnregisterEvent()
[Optional] Repeat N times
Close data stream csiCloseDataStream()
Close device csiCloseDevice()
SDK deinitialization csiClose()
End

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 two examples included:

acquisition_basics	Demonstrates how to discover and open a device and how to acquire images. **Locations:** Windows: `C:\Users\Public\Documents\Chromasens\GCT2\examples\basic` Linux: `/usr/share/csgenicam/examples/basic`
feature_iteration	Demonstrates how to iterate through the feature tree of a device and how to set / get features. **Locations:** Windows: `C:\Users\Public\Documents\Chromasens\GCT2\examples\feature_iteration` Linux: `/usr/share/csgenicam/examples/feature_iteration`

3.5.1 Visual Studio Example Projects

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

3.5.2 Build examples

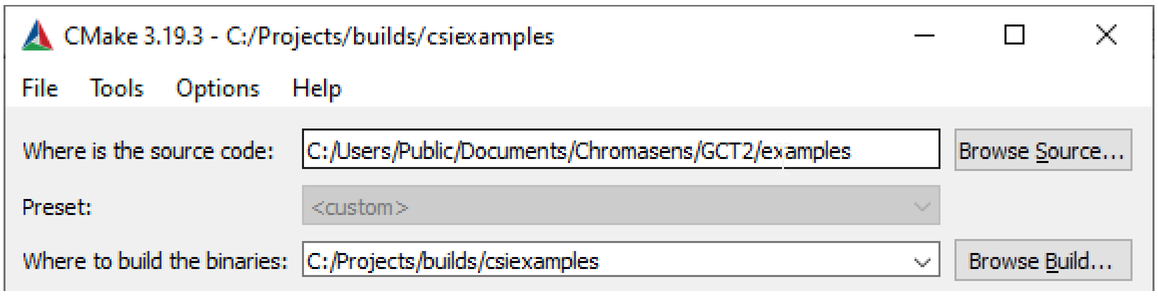
To build the examples requires CMake version > v3.14 and a build environment. The steps to build the examples are the same for both **Windows and Linux**:

1) Open the CMake GUI and select the examples root directory as the source folder of your project.

   ("Where is the source code")

2) Next select a directory where to generate the project files, should be somewhere outside the source tree.

   ("Where to build the binaries")

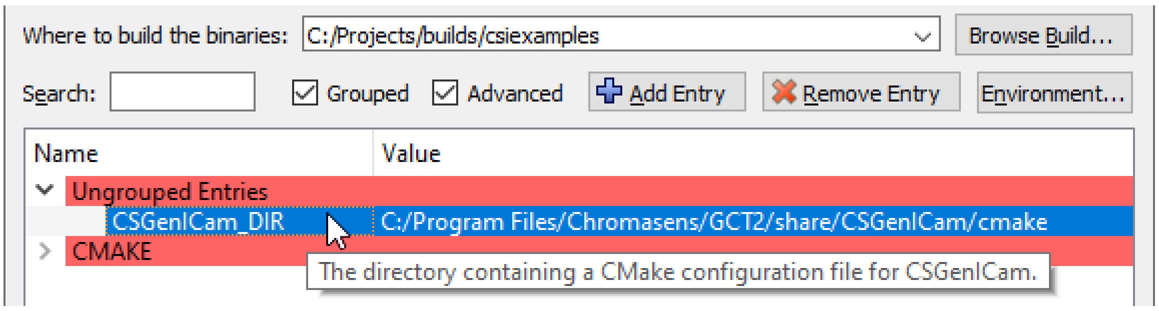

CMake 3.19.3 - C:/Projects/builds/csiexamples

Where is the source code: C:/Users/Public/Documents/Chromasens/GCT2/examples

Preset: <custom>

Where to build the binaries: C:/Projects/builds/csiexamples

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


Where to build the binaries: C:/Projects/builds/csiexamples

Search: ☑ Grouped ☑ Advanced

Name	Value
Ungrouped Entries	
CSGenICam_DIR	C:/Program Files/Chromasens/GCT2/share/CSGenICam/cmake
CMAKE	

The directory containing a CMake configuration file for CSGenICam.

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	*csiErr csiInit(csiLogLevel logLvl = CSI_LOGLEVEL_WARN, , csiLogSinkCallbackFunc logCallbackFunc = NULL, csiLogUserData* userdata = NULL)*
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	*csiErr csiClose();*
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	*csiErr csiDiscoverDevices(csiDiscoveryInfo* discoveryInfoOut, uint64_t timeoutMilliseconds, csiDiscoveryInfoCallbackFunc discCallbackFunc = NULL, const char* additionalSearchPaths = NULL, bool overrideSearchPath = false)*
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	*csiErr csiGetDeviceInfo(uint32_t deviceIndex, csiDeviceInfo* deviceInfoOut)*
Parameters:	In: deviceIndex — Index of the found device from the *csiDiscoverDevices*-function Out: deviceInfoOut — Detailed information of the found device. The information will be provided in this structure. The structure must be allocated on the caller side.
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	*csiErr csiGetNumberOfTLProducers(int32_t *numTLProducers);*
Parameters:	In: Nothing Out: *numTLProducers* — The number of transport layers detected in the environment.
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	*CSI_DLL_EXPORT csiErr csiGetTLProducerInfoByIndex(csiTLProducerInfos *tlProducerInfos, uint32_t indexTL);*
Parameters:	In: intexTL: — The index of the transport layer in the list. Out: *tlProdcrInfos:* — The structure holding additional information about the transport layer.
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	*csiErr csiGetTLProducerInfobyFilePath(csiTLProducerInfos *tlProdcrInfos, const char* producerName)*
Parameters:	In: *producerName*: — The name of the producer (usually the file path to the producer *.cti file) Out: *tlProdcrInfos:* — The structure holding additional information about the transport layer.
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	*csiErr csiOpenDevice(const char* deviceIdentifier, csiHandle* deviceHandleOut, uint64_t timeoutMilliseconds, csiDeviceAccessMode openMode)*
<u>Parameters:</u>	In: deviceIdentifier — Index of the found device from the *csiDiscoverDevices*-function 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. The following modes might be used: CSI_DEV_MODE_EXCLUSIVE: Only this process can communicate with the camera CSI_DEV_MODE_READ: Camera-parameters can be read and images can be acquired CSI_DEV_MODE_CONTROL: Camera-parameters can be read and written. Read-access by another process to the device is still possible Out: deviceHandleOut: Handle to the device. This handle needs to be used to any successive call.
<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.

csiCloseDevice	Close the connection to the specific device
Syntax	*csiErr csiCloseDevice(csiHandle device*
<u>Parameters:</u>	In: device: Handle provided by the *csiOpenDevice*-function Out:
<u>Return value:</u>	Returns csiSuccess or an error defined in the csiErr-Enum.
<u>Comment</u>:	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	*csiErr csiDiscoverTLInterfaces(csiTLInterfaceDiscoveryInfo* discoveryInfoOut, uint64_t timeoutMilliseconds, csiDiscoveryTLInterfaceInfoCallbackFunc discCallbackFunc = NULL, const char* additionalSearchPaths = NULL, bool overrideSearchPath = false)*
Parameters:	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
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).

csiOpenTLInterface	Opens the interface given by the interface information
Syntax	*csiErr csiOpenTLInterface(csiTLInterfaceInfo interfaceInfo, csiHandle* interfaceHandleOut, uint64_t timeoutMilliseconds)*
Parameters:	In: interfaceInfo — Interface information of the found TL interface from the *csiDiscoverTLInterfaces* 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.
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.

csiCloseTLInterface	Closes the connection to the specific interface
Syntax	*csiErr csiCloseTLInterface(csiHandle interfaceHandle)*
Parameters:	In: interfaceHandle — Handle provided by the *csiOpenTLInterface* function. Out:
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	To grant access to the underlying interface for other applications, the connection should be closed when it is not needed anymore. The API will cleanup no longer needed memory when this command is executed.

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	*csiErr csiGetFeatureBool (csiHandle moduleHandle, const char* parameterName, bool* valueOut,* *csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface-stream- or buffer-module Out: valueOut: Pointer to a bool-value where the current value of the feature will be written to
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	

csiSetFeatureBool	Set a boolean feature on the device
Syntax	*csiErr csiSetFeatureBool(csiHandle moduleHandle, const char* parameterName, bool value,* *csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be set on the device-, transport layer-, interface- stream- or buffer-module Out:
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	

csiGetFeatureInt	Retrieve an integer feature from the device
Syntax	*csiErr csiGetFeatureInt(csiHandle moduleHandle, const char* parameterName, int64_t* valueOut,* *csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module Out: valueOut: Pointer to an in64_t-value where the current value of the feature will be written to
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.

Comment:	

csiSetFeatureInt	Set an integer feature on the device
Syntax	*csiErr csiSetFeatureInt(csiHandle moduleHandle, const char* parameterName, int64_t value, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be set on the device-, transport layer-, interface- stream- or buffer-module Out:
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	

csiGetFeatureFloat	Retrieve a floating point feature from the device
Syntax	*csiErr csiGetFeatureFloat(csiHandle moduleHandle, const char* parameterName, double* valueOut, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module Out: valueOut: Pointer to a double-value where the current value of the feature will be written to
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	

csiSetFeatureFloat	Set a floating point value feature on the device
Syntax	*csiErr csiSetFeatureFloat(csiHandle moduleHandle, const char* parameterName, double value, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be set on the device-, transport layer-, interface- stream- or buffer-module Out:
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	

csiGetFeatureString	Retrieve a string feature from the device
Syntax	*csiErr csiGetFeatureString(csiHandle moduleHandle, const char* parameterName, char* valueOut, size_t* sizeOut, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module Out: valueOut: Pointer to a char-value where the current value of the feature will be written to sizeOut: size of the read string
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	To avoid unexpected behavior, you should first retrieve the length of the string to be received! 1. Call the function with valueOut set to NULL. The function will return the current size of the string parameter. 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	*csiErr csiSetFeatureString(csiHandle moduleHandle, const char* parameterName,const char* value, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be set on the device-, transport layer-, interface- stream- or buffer-module Out:
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	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

csiExecuteCommand	Execute a command on the device
Syntax	*csiErr csiExecuteCommand(csiHandle moduleHandle, const char* parameterName, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be executed on the device-, transport layer-, interface- stream- or buffer-module Out:
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
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 "*csiIsCommandActive*".

csiIsCommandActive	Check if a command is still active
Syntax	*csiErr csiIsCommandActive(csiHandle moduleHandle, const char* parameterName, bool *isActive, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module Out: isActive: Pointer to a bool-value where the current state of the command is written to (true: active, false: inactive)
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
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	*csiErr csiGetFeatureReg(csiHandle moduleHandle, const char* parameterName, char* buffer, size_t* length, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module Out: buffer: Pointer to a char-array where the current value of the feature will be written to length: Current length of the retrieved data
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	To avoid unexpected behavior, it is recommended to retrieve the maximum buffer length before getting it from the device. This can be achieved by using the function "*csiGetFeatureParameter*". This function will provide all necessary information about the parameter (including min and max values). The register length must not exceed the length given in the "featureRegLength"-parameter

csiSetFeatureReg	Set a register value on the device
Syntax	*csiErr csiSetFeatureReg(csiHandle moduleHandle, const char* parameterName, const char* buffer, size_t length, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* function parameterName: name (Not the display name!) of the feature to set buffer: pointer to the data which will be written to the r egister length: number of bytes to write to the register module: Module for which the register should be set. Please use the enum csiModuleLevel to select. Determines if the register should be set on the device-, transport layer-, interface- stream- or buffer-module Out:
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	To avoid unexpected behavior, it is recommended to retrieve the maximum buffer length before setting it to the device. This can be achieved by using the function "*csiGetFeatureParameter*". This function will provide all necessary information about the parameter (including min and max values). The register length must not exceed the length given in the "featureRegLength"-parameter

csiGetFeatureParameter	Retrieve a specific feature from the device. Detailed information about this feature will be returned
Syntax	*csiErr csiGetFeatureParameter(csiHandle moduleHandle, const char* parameterName, csiFeatureParameter* featureParamOut, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module Out: featureParamOut
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	

csiIterateFeatureTree	Provides a possibility to iterate through all available features on the camera
Syntax	*csiErr csiIterateFeatureTree(csiHandle moduleHandle, const char* rootFeatureName, uint32_t index, char* featureNameOut, size_t nameBuffSize, csiFeatureType* type, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the feature tree on the device-, transport layer-, interface- stream- or buffer-module should be used Out: featureNameOut: name of the retrieved feature type: type of the retrieved feature
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	If starting from the very beginning, use "Root" as rootFeatureName. From there call this function for each returned feature in order to get all features of the device. Please check the provided example "feature_iteration" for a template of usage.

csiGetFeatureEnum	Retrieve an enumeration feature from the device
Syntax	*csiErr csiGetFeatureEnum(csiHandle moduleHandle, const char* parameterName, csiFeatureParameter *featureParamOut, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module Out: featureParamOut: Structure which contains all necessary information about the requested feature: the relevant entries of the csiFeatureParameter-structure: enumCounter: Number of different enum-entries for the enumeration enumIndex: Currently selected enumeration index valueStr: name of the enum-entry
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	

csiSetFeatureEnum	Set an enumeration feature on the device
Syntax	*csiSetFeatureEnum(csiHandle moduleHandle, const char* parameterName, const char* value, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module Out:
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	To retrieve the possible values for this enumeration, two functions need to be called: 1. *csiGetFeatureEnum*: in the returned structure, the element "enumCounter" indicates the number of available entries 2. The different entries can be retrieved by using the function "*csiGetFeatureEnumEntryByIndex*" simply by iterating from 0 until the "enumCounter"-1

csiGetFeatureEnumEntryByIndex	Retrieve an enumeration feature from the device by using its index
Syntax	*csiErr csiGetFeatureEnumEntryByIndex(csiHandle moduleHandle, const char* parameterName, int32_t enumIndex, csiFeatureParameter *featureParamOut, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module Out: featureParamOut: Name of the enumeration of the requested index
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	The enumeration string will be given in the *csiFeatureParameter*-structure: valueStr

csiGetFeatureEnumEntryByName	
Syntax	*csiErr csiGetFeatureEnumEntryByName(csiHandle moduleHandle, const char* parameterName, const char* enumValue, csiFeatureParameter *featureParamOut, csiModuleLevel module = CSI_DEVICE_MODULE)*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module Out: featureParamOut
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	

csiRegisterInvalidateCB	Register an invalidation callback function to a specific feature by name
Syntax	*csiErr csiRegisterInvalidateCB(csiHandle moduleHandle, const char *featureName, CB_OBJECT objCB, CB_FEATURE_INVALIDATED_PFN pfnFeatureInvalidateCB, csiModuleLevel module = CSI_DEVICE_MODULE);*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select.

	Determines if the parameter should be retrieved from the device-, transport layer-, interface- stream- or buffer-module Out: Nothing
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	This function is useful to get informed about changes in the feature tree which lead to an invalidation of features. Whenever a feature changes its value or another attribute, the application should get informed about it. The callback function must be of the form: `void featureInvalidated(const char *featureName, void* userObj);`

csiUnRegisterInvalidateCB	Unregister an invalidation callback function from a specific feature
Syntax	*csiErr csiUnRegisterInvalidateCB(csiHandle moduleHandle, const char *featureName, csiModuleLevel module = CSI_DEVICE_MODULE);*
Parameters:	In: moduleHandle: Handle provided by the *csiOpenDevice* or *csiOpenTLInterface* 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 csiModuleLevel to select. Determines if the parameter should be retrieved from the device-, transport layer-, interface-stream- or buffer-module Out: Nothing
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	

4.5 Functions related to image acquisition

csiGetNumberOfDataStreams	Return the available number of data streams of the device
Syntax	*csiErr csiGetNumberOfDataStreams(csiHandle device, uint32_t* numberOfStreamsOut)*
Parameters:	In: device: Handle provided by the *csiOpenDevice*-function Out: numberOfStreamsOut Number of available data streams for the given device
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	The returned number of data streams can be 0 if the given device is not a streaming device. The actual number of available data streams depends on the capabilities of the device. Use this function in combination with *csiGetDataStreamInfo()* which receives an index to a data stream as parameter**.**

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

csiCreateDataStream	Create a data stream to receive images
Syntax	*csiErr csiCreateDataStream(csiHandle device, uint32_t dsIndex, csiHandle* dataStreamOut, uint32_t numberOfBuffers, size_t bufferSize = 0)*
Parameters:	In: device: Handle provided by the *csiOpenDevice*-function dsIndex: An index to a data stream. Starting from 0 to the number of available data streams as returned by *csiGetNumberOfDataStreams()* 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.
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	Use this function in combination with *csiGetNumberOfDataStreams()* to get the total number of available data streams in the camera.

csiCloseDataStream	Close the data stream
Syntax	*csiErr csiCloseDataStream(csiHandle dataStream)*
Parameters:	In: dateStream — A handle to the data stream to be closed. Out: None
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	Make sure to release all used buffers with *csiReleaseImage*() and unregister all events with *csiUnregisterEvent()* before closing the data stream. Any buffer or event will be invalid after a call to this function. In addition, acquisition must be stopped before calling this function, see *csiStopAcquisition()*.

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

csiWaitForEvent	Wait for a desired event to happen
Syntax	*csiErr csiWaitForEvent(csiHandle evt, uint64_t timeoutMilliseconds, csiEventData** evtDataOut)*
Parameters:	In: evt — the handle of the event to wait for timeoutMilliseconds — Timeout after the waiting stops if no event was received Out: evtDataOut — the event data for further use. See . In case of an error or timeout the output will be NULL, therefore please check the return value before using it.
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	After the event was registered with *csiRegisterEvent()* it is possible to actively wait for the event using this function. The waiting can be done asynchronously in a separate thread. This function must be called for each event separately. Please note that this function will return a more general representation of the event data ( ). There exists also an event data structure for image data ( ) which contains more information on the image that was received. **Note:** Also see *csiGetNextImage()* which can be used as convenience function to wait for new image data events.

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

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

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

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

csiGetNextImage	Get the next image from the data stream
Syntax	*csiErr csiGetNextImage(csiHandle dataStream, csiNewBufferEventData** bufferInfoOut, uint64_t timeoutMilliseconds)*
Parameters:	In: dataStream the handle of the data stream to wait for images on. Requires a previous call to *csiRegisterEvent*() to register for new image events on that stream. timeOutMilliSeconds Timeout after the waiting stops if no event was received. Out: bufferInfoOut the event data structure containing the image data and information, see .
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	This is a convenience function that is recommended to use for image acquisition. It is an alternative to *csiWaitForEvent()* that returns a more general representation of the event data. Please note that a CSI_EVT_NEWIMAGEDATA event must be registered on the data stream to be able to wait for new images. The image buffer returned from this function will be valid and usable as long as it is not given back to the acquisition engine with *csiReleaseImage()*.

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

csiGetAcquisitionStatistics	Retrieve the statistical buffer regarding the data stream (e.g. transmitted frames, etc.)
Syntax	*csiErr csiGetAcquisitionStatistics(csiHandle dataStream, csiAcquistionStatistics* stats)*
Parameters:	In: datastream Handle to the data stream the acquisition statistics should be collected on. Out: stats The acquisition statistics, see **csi.**
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	

4.6 File transfer functions

csiGetUpdateFileType	Request the update file type of a file (if available)
Syntax	*csiErr csiGetUpdateFileType(csiHandle device, const char* fileName, char* fileTypeOut, size_t bufferSize)*
Parameters:	In: device: Handle provided by the *csiOpenDevice*-function filename: The path to a file that should be checked bufferSize the size of the output buffer fileTypeOut Out: fileTypeOut If the file is a valid file that can be used to update on the device, this buffer should contain the type of that file as string representation
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
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	*csiErr csiFileDownloadToDevice(csiHandle device, const char* fileName, const char* fileType, uint64_t timeoutMilliseconds, csiMemTransferCallbackFunc listener = NULL, csiMemTransferUserData* userdata = NULL)*
Parameters:	In: device: Handle provided by the *csiOpenDevice*-function filename Full path of the fule to be uploaded filetype Type of the update file as returned from *csiGetUpdateFileType()* 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. linstener 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 csiSuccess or an error defined in the csiErr-Enum.
Comment:	

csiFileUploadFromDevice	Uploads a file from device to the local PC
Syntax	*csiErr csiFileUploadFromDevice(csiHandle device, const char* fileName, const char* fileType, uint64_t timeoutMilliseconds, csiMemTransferCallbackFunc listener = NULL, csiMemTransferUserData* userdata = NULL)*
Parameters:	In: device: Handle provided by the *csiOpenDevice*-function 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 csiSuccess or an error defined in the csiErr-Enum.
Comment:	

csiFileDownloadToDeviceEx	Downloads a file located on the local PC to the camera providing custom options
Syntax	*csiErr csiFileDownloadToDeviceEx(csiHandle device, const csiDownloadParams params, uint64_t timeoutMilliseconds, csiMemTransferCallbackFunc listener = NULL, csiMemTransferUserData* userdata = NULL)*
Parameters:	In: device: Handle provided by the *csiOpenDevice*-function params File related parameter and options, see **csiDownloadParams** 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. linstener 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 csiSuccess or an error defined in the csiErr-Enum.
Comment:	

csiFileUploadFromDeviceEx	Uploads a file from device to the local PC providing custom options
Syntax	*csiErr csiFileUploadFromDeviceEx(csiHandle device, const csiUploadParams params, uint64_t timeoutMilliseconds, csiMemTransferCallbackFunc listener = NULL, csiMemTransferUserData* userdata = NULL)*
Parameters:	In: device: Handle provided by the *csiOpenDevice*-function params File related parameter and options, see **csiUploadParams** 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 csiSuccess or an error defined in the csiErr-Enum.
Comment:	

4.7 Memory transfer functions

csiReadMemory	Read memory from a register address on the device
Syntax	*csiErr csiReadMemory(csiHandle device, uint64_t address, char* buffer, size_t sizeBytes)*
Parameters:	In: device: Handle provided by the *csiOpenDevice*-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 csiSuccess or an error defined in the csiErr-Enum.
Comment:	

csiWriteMemory	Write memory to a register address on the device

Syntax	*csiErr csiWriteMemory(csiHandle device, uint64_t address, const char* buffer, size_t sizeBytes)*
Parameters:	In: device: Handle provided by the *csiOpenDevice*-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 Out:
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	

4.8 Helper functions

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

csiGetErrorDescription	Returns a human readable description of an error code
Syntax	*csiErr csiGetErrorDescription(csiErr error, char* bufferOut, size_t bufferSize)*
Parameters:	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
Return value:	Returns csiSuccess or an error defined in the csiErr-Enum.
Comment:	

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

4.9 Enumerations

csiPixelFormat	Defines the currently supported pixel data formats

Definition	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;

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	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;

Elements	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	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;

Elements	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	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;
Elements	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	typedef enum csiAccessMode {
	        CSI_ACCESS_UNKNOWN,
	        CSI_ACCESS_NOT_AVAILABLE,
	        CSI_ACCESS_READ_ONLY,
	        CSI_ACCESS_READ_WRITE,
	        CSI_ACCESS_WRITE_ONLY
	} csiAccessMode;	

Elements	CSI_ACCESS_UNKNOWN	Unknown access mode, feature might not be accessible
	CSI_ACCESS_NOT_AVAILABLE	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.
	CSI_ACCESS_READ_ONLY	Feature is read only.
	CSI_ACCESS_READ_WRITE	Feature can be accessed in read and write mode.
	CSI_ACCESS_WRITE_ONLY	Feature can only be written.

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

Definition	typedef enum csiFeatureVisibility {
	        CSI_VISIBILITY_BEGINNER=1,
	        CSI_VISIBILITY_EXPERT,
	        CSI_VISIBILITY_GURU,
	        CSI_VISIBILITY_DEVELOPER,
	        CSI_VISIBILITY_INVISIBLE
	} csiFeatureVisibility;	

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

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

Definition	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;	

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

csiDisplayNotation	Defines the display notation for a floating-point feature	

Definition	typedef enum csiDisplayNotation {
	        CSI_NOTATION_AUTOMATIC,
	        CSI_NOTATION_FIXED,
	        CSI_NOTATION_SCIENTIFIC,
	} csiDisplayNotation;	

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

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

Definition	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;	

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

csiLogLevel	Defines the severity of log messages coming from the SDK

Definition	typedef enum csiLogLevel {
	CSI_LOGLEVEL_ERROR = 1,
	CSI_LOGLEVEL_WARN = 2,
	CSI_LOGLEVEL_INFO = 4,
	CSI_LOGLEVEL_DEBUG = 8,
	CSI_LOGLEVEL_TRACE = 16,
	} csiLogLevel;

Elements	CSI_LOGLEVEL_ERROR CSI_LOGLEVEL_WARN CSI_LOGLEVEL_INFO CSI_LOGLEVEL_DEBUG CSI_LOGLEVEL_TRACE

csiErr	Defines possible error values	

Definition	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,
	csiIOError = -118,
	csiParsingError = -119,
	csiInvalidValue = -120,
	csiResourceExhausted = -121,
	csiOutOfMemory = -122,
	csiBusy = -123,
	csiUnknown = -200,
	csiCustomErr = 0x0f0000000
	} csiErr;	

Elements	csiSuccess	No error
	csiNotInitialized	System is not initialized, call *csiInit()* first
	csiInvalidState	An invalid state occurred, see log output for more information
	csiNotOpened	There was an action that requires the device / network / stream to be opened
	csiNoImageDataAvailable	There was no image data available
	csiNotFound	General error that the requested information was not found, see log for more detailed info on this error.
	csiInvalidParameter	A parameter had an invalid value
	csiNotAvailable	An expected result or a resource was not available
	csiFunctionNotAvailable	The called function or a sub-function is not available
	csiTimeout	A timeout occurred
	csiAborted	A pending operation was aborted
	csiFileOperationFailure	There was an error during file operation, see log for more information
	csiFileOperationFatalError	There was a fatal error during file operation, see log for more information
	csiNoAccess	Access denied (e.g., when trying to write a read only feature)
	csiWrongBufferSize	A given buffer was too small to store the requested data
	csiInvalidBuffer	The requested buffer is not valid
	csiResourceInUse	The requested resource is already in use by the transport layer
	csiNotImplemented	A function that was called is not yet implemented
	csiInvalidHandle	A handle passed as parameter is not valid
	csiIOError	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	typedef enum csiAcquisitionMode {
	CSI_ACQUISITION_SINGLE_FRAME = 0x00000001,
	CSI_ACQUISITION_CONTINUOUS = 0xFFFFFFFF
	} csiAcquisitionMode;	

Elements	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	typedef enum csiEventType {
	CSI_EVT_NEWIMAGEDATA = 0x00,
	CSI_EVT_ERROR = 0x01,
	CSI_EVT_MODULE = 0x02,
	CSI_EVT_CUSTOM = 0x1000
	} csiEventType;

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	typedef enum csiMemTransferStatus { csiTransferStatusInit, csiTransferStatusInProgress, csiTransferStatusInProgressWaiting csiTransferStatusFinishSucess, csiTransferStatusFinishError, csiTransferStatusCancelOnError } csiMemTransferStatus;	
Elements	csiTransferStatusInit csiTransferStatusInProgress csiTransferStatusInProgressWaiting csiTransferStatusFinishSucess 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

csiDownloadOptions	Defines the file transfer options for download (bit flag enum)	

Definition	typedef enum csiDownloadOptions
	{
	CSI_DOWNLOAD_NO_OPTIONS
	} csiDownloadOptions;	

Elements	CSI_DOWNLOAD_NO_OPTIONS	no options, i.e. default behaviour

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

4.10 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 **csi**
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

csiTLProducerInfos	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)
csiTLInterfaceInfo[]	interfaces	The list of interfaces found.

Struct-name	**csiDeviceInfo**	
Variable type	**Element name**	**Description**
char[]	deviceIdentifier	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	cameraSwPackageIsConsistent	
csiTLProducerInfos	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 **csiGetDeviceInfo***()* 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	**csiNewBufferEventData**	
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	**csiAcquistionStatistics**	
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 *csiGetUpdateFileType()*

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**

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 18.04 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 18.04

1.) Open a new terminal window

2.) Navigate to the directory where the SDK software package is located. In this example it will be in the Downloads folder:

```
cd ~/Downloads
```

3.) Update the package manager:

```
sudo apt update
```

4.) Install the package using the package manager, replace the **<version>** part by the version of the downloaded package. The package manager might ask to install additional required dependencies if they are not yet present in the system:

```
sudo apt install ./csgenicam-<version>.deb
```

5.) After the installation, a system reboot is required to apply changes to the system environment.

5.2.3 Installer Contents

The software package is grouped into the following components:

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

Chromasens GmbH

Max-Stromeyer-Straße 116
78467 Konstanz
Germany

Phone: +49 7531 876-0
Fax: +49 7531 876-303

www.chromasens.de
info@chromasens.de