Chromasens Camera-API

CD40082 Version 1.7

History:

Version	Release	Author	Description
1.60	2017/03/16	Sczech	Included settings for the allPIXA wave. It is now possible to query the type of the connected camera. Several bugs removed.
1.7	2019/01/29	Sczech	Added XLC parameters for the allPIXA wave. It is now possible to query the transport speed and direction in the operational values. Also the error state can be queried in the operational values without clearing the state. Improved behavior when connecting via virtual com-Port of the frame grabbers. CL_BASE_1X3 mode for CameraLink transmission and allPIXA pro is available now. Added global master/slave setting Added setting of internal frame trigger Fixed problem when config files where located in the same directory as the dlls. Extended IO-configuration with new master-slave configurations and possibility to use an internally generated frame trigger. `CS_CAMERA_INFORMATION_STRUCT now` contains information about the camera software package. Possible to get a line profile through the serial interface (get_image_line_data_from_camera) Changed enum "CS_REFERENCE_TYPE" to CS_REFERENCE_VERSION for better understanding. This enum is now also used in the CS_CALCULATE_REFERENCE_STRUCT. In this struct it was also necessary to change the type of the image pointer to unsigned char*. Type of max_no_of_scan_lines, scan_lines_after_stop and no_of_suppressed_lines in the CS_TRIGGER_STRUCT were changed to UINT16. img_height in the CS_IMAGE_PARAMETER_STRUCT was changed to UINT32.

Content

1 Preface

The Chromasens Camera API package is designed specifically for use with Chromasens Cameras.
The package supports Microsoft Windows Versions Windows XP and later. The software can be used in 32- and 64-Bit environments. Linux is currently not supported.

The CSAPI can be used by C++ or C#-implementations. The necessary DLLs are provided within this package.

The package consists of:

- Full Application Programming Interface (API) allowing complete camera control including convenience functions such as generation of shading data sets.
- Image acquisition when using the GigE-Interface to the Chromasens cameras
- Set of C++ examples with full source code (Visual Studio 2005 /2010);
- Set of C# examples (Visual Studio 2010)
- Documentation of the camera features and the corresponding API-functions
- Camera setup tool (CST) for communication with the camera.

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:

CS_WHITE_BALANCING_MODE	Used for choosing the white balancing mode. Currently only one mode is supported.

Definition	enum CS_WHITE_BALANCING_MODE {
	GAIN_CONTROL_USING_AREA_RANGE=0
	};

Elements	GAIN_CONTROL_USING_AREA_RANGE : The White balancing will be done in the defined area

Structures:

For example:

Variable type	Element name	Description
INT16	firmwareVersion	
INT16	fpgaVersion	
Char	firmwareDescription	
Char	fpgaDescription	

Functions:

For example:

get_camera_white_balancing_setup	Retrieves the currently set parameters for the white balancing setup
Syntax	get_camera_white_balancing_setup(CS_WHITE_BALANCE_STRUCT *whiteBalanceSetup, bool bUpdate=false)
Parameters:	whiteBalanceSetup : pointer to a CS_WHITE_BALANCE_STRUCT-structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

1.1 Additional documents

"Chromasens Camera HSI"

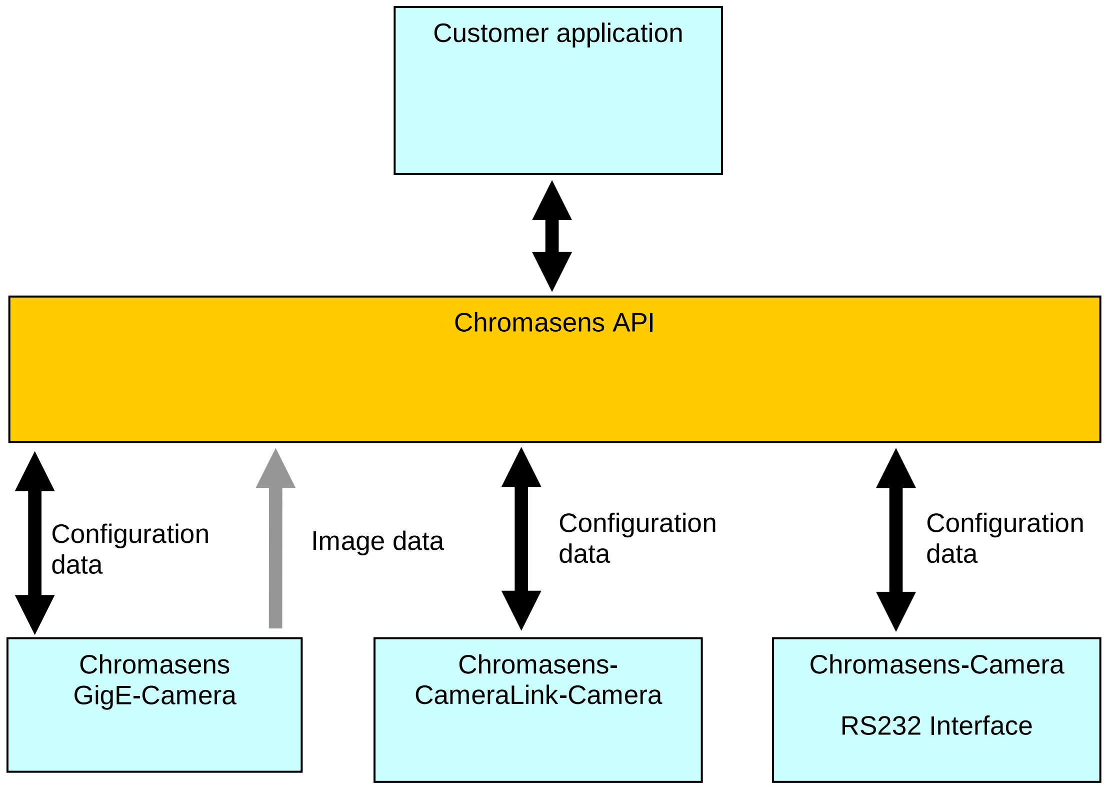
Documentation and definition of the Hardware-Software-Interface for Chromasens Cameras.

This Document describes the commands for controlling Chromasens cameras. Data structures and values of parameters are defined there.

"Manual allPIXA Camera"

Documentation for Chromasens allPIXA cameras

2 Software Architecture



2.1 Communication concept

The Chromasens API does provide a set of function for communication with Chromasens cameras.

With these functions connection to the camera can be opened and closed via delivered interfaces. Internal working parameters of the cameras can be read and modified.

Convenience functions for calibration of the camera are provided. Calibration is done to adapt to actual illumination surroundings.

Functions for low level communication are implemented in the Chromasens API.

Working with GigE interface image capturing is integrated.

2.2 Internal camera parameter

Chromasens cameras provide a set of working parameters which adapt the behavior of the camera to the suggested customer application.

Well known parameters for digital cameras like “image size”, “integration time”, “camera gain”, “trigger mode”, etc. are implemented.

But there are much more features provided (encoder setup, automatic gain adjustment, etc.) by the Chromasens cameras.

All the provided functions are initialized by a set of working parameters which are held in the camera. These parameters can be read and modified by the Chromasens API.

The different internal parameters have, depending on their functions, different data lengths.

Binary values

Parameters which enable or disable specific functions are implemented as binary values (BIN).

Integer values

Parameters which setup certain values in the camera are implemented as 16 bit short (**SHORT**) or 32 bit long (**LONG**) values.

Structured values

Functions which need a set of parameters are implemented as variable structures (VAR).

For detailed definition of camera parameters see documentation ***“Chromasens Camera HSI”***

The complete setup of the camera (called ***setting***) can be stored non-volatile in the camera and can be re-called by the API.

The cameras provides 18 banks (1 .. 18) for storing a complete working set of the camera. Storing and re-calling of a setting is managed by the Chromasens API.

At start-up after power up or after reset of the camera setting no. 1 is activated internally. It is not possible to change this behavior.

Alternatively these settings can be read and modified by the CST-tool provided by Chromasens.

Furthermore calibration data like offset and shading references can be loaded into the cameras. Depending on the available features of the camera common or customized gamma correction tables or filters can be downloaded to the camera.

3 Getting started

3.1 Hardware Requirements

There are different ways to communicate with a Chromasens camera. Depending on the available interfaces different hardware is required.

Standard serial RS232-interface:

Serial interface card for the PC, USB to serial interface adapter or built in RS232-interface

CameraLink:

Standard Frame Grabber and clAllSerial.dll required (usually shipped with the grabber).

GigE:

In order to avoid problems with lost image frames we recommend using interfaces board from the Intel® Gigabit CT series.

Please make sure that the receive buffer sizes of the network adapter are set to the maximum values. We also recommend enabling the Checksum offload functions of the network adapter to minimize CPU load.

Windows 7 and GigE cameras:

A new feature sets the priority of multimedia data to a higher Level. In order to avoid the loss of image data the following registry-entry should be set:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\NetworkThrottlingIndex = 0xFFFFFFFF.

This disables the Multimedia Class Scheduler Service (MCSS).

3.2 Software

Supported operating systems (32- and 64-bit versions):

Windows XP (Service Pack 3 recommended),
Windows Vista (Service Pack 1 recommended)
Windows 7

Windows 8.1 (Currently under test)

Redistributable for Visual Studio 2010 (x64 or w32).

On Systems with Windows XP also the redistributable for Visual Studio 2005 (x64 or w32) is required.

Development environment:

MS Visual Studio 2005 SP1

MS Visual Studio 2005 SP1 and SP1 Update for Vista (Needed on Vista and Windows 7)

MS Visual Studio 2010 SP1

3.3 Installation

The Chromasens API is provided in a single .zip file: ChromasensAPI_VER_X.XX.zip. This .zip-file will extract into different directories

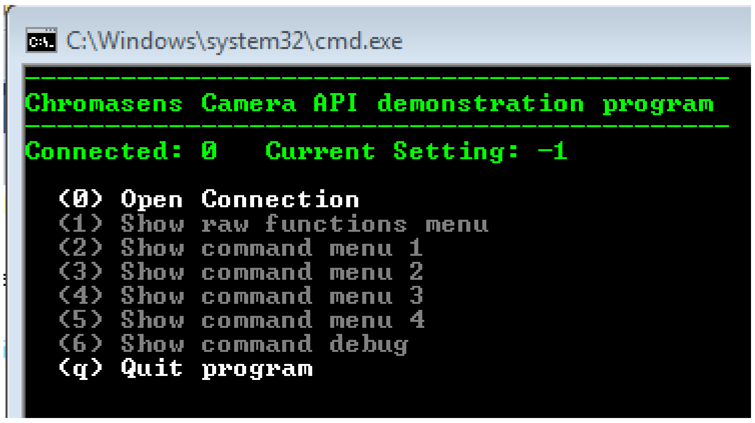
Following directories exist:

./CS_API	Contains all necessary include files. Only “ChromasensCameraIf.h“ needs to be included by the application software
./lib/win32	Contains the 32 bit library files to integrate into your solution.
./lib/win64	Contains the 64 bit library files to integrate into your solution.
./bin/win32	The DLLs (32 bit) needed for the communication can be found here.
./bin/win64	The DLLs (64 bit) needed for the communication can be found here.
./bin/config	contains all necessary configuration files which need to be included also into your application. This directory should be put inside the directory where the csapi.dll is located.
./API_Test	Example code which demonstrates how to use the single functions of the API.
./csharp_CS_API_Test	Example code for C#.

The libraries and DLLs are available for Debug and Release. They can be found in the appropriately named directories.

3.4 The API_Test-program

All functions which are available in the CS-API are demonstrated in the sample program API_Test.
The binary version can be found in the directory: ./bin/x64/release/API_Test.exe.
The source code is delivered as well so it is possible for the developer to debug into the program and see what steps need to be done in order to connect to the camera and use the single functions.
There is no graphical user interface. The program will be running in a command-shell with a textual interface.



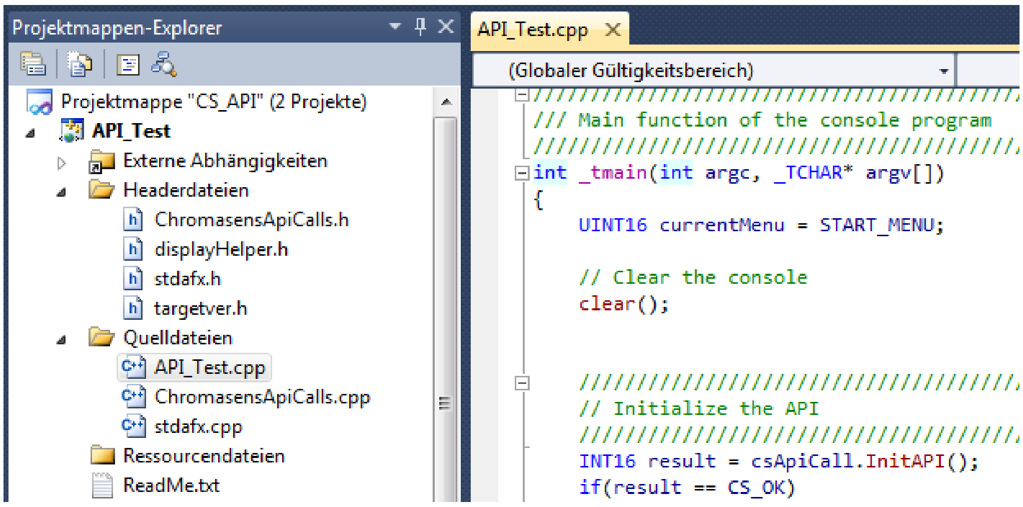
```
C:\Windows\system32\cmd.exe
--------------------------------------------
Chromasens Camera API demonstration program
--------------------------------------------
Connected: 0   Current Setting: -1

   (0) Open Connection
   (1) Show raw functions menu
   (2) Show command menu 1
   (3) Show command menu 2
   (4) Show command menu 3
   (5) Show command menu 4
   (6) Show command debug
   (q) Quit program
```

The program will start as demonstrated above. Please make sure that a camera is connected to your system via RS232 or cameraLink.
By pressing the “0” you will be able to enter the desired connection parameters.

As soon as you have established a connection to the camera, the other menu options will become selectable.

In order to step through the program please open the API_Test solution (./API-Test/API_Test.sln) in your development environment.



```
//////////////////////////////////////////
/// Main function of the console program
//////////////////////////////////////////
int _tmain(int argc, _TCHAR* argv[])
{
    UINT16 currentMenu = START_MENU;

    // Clear the console
    clear();


    //////////////////////////////////////
    // Initialize the API
    //////////////////////////////////////
    INT16 result = csApiCall.InitAPI();
    if(result == CS_OK)
```

The main function of the console program can be found in the API_Test.cpp file. It is called “_tmain”.
Please set your first breakpoint inside this function you will then be able to follow the steps necessary for initialization of the API and establishing a connection to the camera.
The file ChromasensApiCalls.cpp contains all the calls to the Chromasens API which are needed to set and get all needed parameters.
The functions available are described on the following pages.

3.5 Programming

In order to work with the Chromasens API and the Chromasens cameras the following steps need to be done.

3.5.1 Initialization of CS-API

The CS-API does need a number of configuration files. These files are usually located in the folder “config” which should be located in the folder of the CS-API-dll.
Please use the files provided by the installation in the “bin”-folder.

Initialize the class **CChromasensCameraIf**

If you use the standard setup where the “config” directory resides inside the dll-directory, use the following code:

```
CChromasensCameraIf* camera=NULL;
camera = InitCsCameraIf();
```

Or choose where the configuration files are located:

```
CChromasensCameraIf* camera=NULL;
camera = InitCsCameraIf(“C:/myDirectory/configFiles”);
```

Remark:

A CChromasensCameraIf-object can only handle one connection!

To open multiple connections, create another object.

3.5.2 Open a connection to the camera

There are two possibilities to open a connection:

Use a port with a known identifier
or
let the API display a dialog with the available interfaces and its parameters.

1: display a dialog:

```
CS_CONNECTION_STRUCT connectInfo;
  nReturn = camera->open_control(&connectInfo);
```

This is demonstrated in the Chromasens API-Demo “API_TEST” in the function “connectCamera”.

2: call a specific function for the different connection types

RS232:

```
nReturn = camera->open_RS232_control(nPort, nBaudRate);
```

CameraLink:

```
nReturn = camera->open_CL_control(nPort, nBaudRate);
```

Ethernet:

```
nReturn = open_ETH_control(DEFAULT_IP_ADDRESS, USE_DEFAULT_PORT);
```

In order to check for available interfaces two functions do exist:
The function `get_number_of_available_interfaces` will return the number of available interfaces including serial and CameraLink connections.
To obtain more detailed information about the single interfaces the user has to call the function `get_interface_description`.
The CameraLink connections will be listed at first.
To distinguish in between cameraLink and RS232-connections, the number of detected ports can be retriebved by calling: get_no_of_cameraLink_ports and get_no_of_rs232_ports. The sequence of the ports in the list is always the same: At the beginningh of the list the cameraLink-ports are lsited(if present). Afterwards you can ind the RS232 ports in the list.

3.5.3 Get current parameters

Raw HSI-Commands:

Different types of HSI-parameters do exist. The CS-API offers different functions to retrieve these parameters (TAGs):

- binary tags:	get_bin_tag (page 97)
- short tags :	get_short_tag (page 97)
- long tags:	get_long_tag (page 98)
- var tags:	get_var_tag (page 98)

Example for retrieving the gain values (TAG_SET_GAIN, 0x1c0 hex) from the camera:

```
WORD value[12];
int nReturn = camera->get_var_tag(TAG_SET_GAIN, 12, value, true);
```

This function call will request the VAR-TAG "**TAG_SET_GAIN**" with 12 contained values from the camera. The received data is written to the array referenced with pointer "value"

Convenience functions:

As an example we retrieve the current gain values from the camera. Depending on the type of the camera, 6 or 12 gain values are present (Aleos: 6 gain values, allPIXA 12). The number of valid entries is returned in the parameter returned from the get_gain_values function.

```
cs_gain_struct gainValues;
get_gain_values(&gainValues, true);
```

Following convenience functions are currently implemented:

- get_gain_values (page 43)
- get_initial_gain_values (page 44)
- get_coarse_gain_values (page 45)
- get_integration_time (page 46)
- get_image_parameters (page 48)
- get_camera_trigger_mode (page 51)
- set_global_master_slave_settings (page 52)
- get_encoder_parameters (page 53)
- get_reference_settings (page 55)
- get_camera_white_balancing_setup (page 56)
- get_current_reference_values (page 58)
- get_brightness_and_contrast (page 59)
- get_image_processing_parameters (page 60)
- get_current_setting_number (page 65)
- get_available_settings (page 65)
- get_available_settings (page 65)
- get_current_setting_number (page 65)
- get_setting_description (page 66)
- get_camLink_properties (page 68)
- get_test_pattern_settings (page 70)
- get_video_output_parameters (page 69)
- get_camera_information (page 71)
- get_camera_operating_values (page 79)
- get_camera_state (page 86)
- get_camera_state (page 86)
- get_error_text (page 86)
- get_trace_mode (page 89)
- get_trace_from_camera (page 89)

- get_logging_parameter (page 100)
- get_led_flash_parameters (page 78)
- determineCameraType (page 86)
- get_roi_parameters (page 81)
- get_xlc_parameters (page 84)
- get_internal_light_barrier_settings (page 85)

3.5.4 Set camera parameters

<u>**Raw HSI-Commands:**</u>

Please be cautious when using the raw commands together with the high level API functions! After using the raw functions you should retrieve the current camera settings by calling any "get_xxx"-function with the bUpdate parameter set to true. By doing this the internal data structure will represent the current data of the camera again.

Different types of HSI-parameters do exist. The CS-API offers different functions to set these parameters (TAGs):

- binary tags:	send_bin_tag (page 94)
- short tags :	send_short_tag (page 94)
- long tags:	send_long_tag (page 95)
- var tags:	send_var_tag (page 95)

Example for setting the gain values (TAG_SET_GAIN, 0x1c0 hex) in the camera:

```
WORD value[12];
bool bSendImmediately = true;
value[0] = 320;
value[1] = 320;
….
value[11] = 300;
int nReturn = camera->send_var_tag(0x1c0, 12, value, bSendImmediately);
```

If the flag bSendImmediately is not set, it is possible to collect several commands and send them all together in order to minimize traffic to the camera.

If using this method you need to call "send_stored_hsi_commands" in order to transfer the stored commands to the camera.

<u>**Convenience functions:**</u>

As an example we set the actual gain values of the camera. Depending on the type of the camera, 6 or 12 gain values are present (Aleos: 6 gain values, allPIXA 12).
The size of the array with the gain values to set must be given to the function set_gain_value.

```
cs_gain_struct gainValues;
gain.gain[0] = 200;
gain.gain[1] = 200;
…
gain.gain[11] = 200;
gain.no_of_valid_entries = 12;

set_gain_values (gainValues, true);
```

Currently these convenience functions are implemented:

- set_gain_values (page 44)
- set_initial_gain_values (page 44)
- set_coarse_gain_values (page 46)
- set_integration_time (page 47)
- set_image_parameters (page 49)
- set_camera_trigger_mode (page 52)
- set_global_master_slave_settings (page 53)
- set_encoder_parameters (page 54)
- set_reference_settings (page 55)
- set_camera_white_balancing_setup (page 58)
- set_brightness_and_contrast (page 60)
- set_image_processing_parameters (page 62)
- burn_current_setting_to_camera (page 65)
- save_current_setting_2Disk (page 65)
- select_active_setting (page 66)
- set_setting_description (page 66)
- set_video_output_parameters (page 68)
- set_test_pattern_settins (page 70)
- perform_white_balancing (page 87)
- reset_camera (page 87)
- do_tap_balancing (page 87)
- prepare_cam_4reference (page 88)
- set_trace_mode (page 89)
- send_hsi_file_2_camera (page 89)
- calculate_reference (page 90)
- create_reference_internally (page 92)
- set_led_flash_parameters (page 77)
- set_roi_parameters (page 82)
- set_xlc_parameters (page 83)
- set_internal_light_barrier_settings (page 85)

3.5.5 Close the connection to the camera

```
camera->close_control();
```

3.5.6 Close the Chromasens Camera API (free the allocated memory)

```
if(camera)
{
        delete camera;
        camera = NULL;
}
```

3.5.7 Constants

Gain control constants

Constants used to set and get the parameters for the gain control of the camera. The allPIXA camera has 12 independent channels for the gain control, Therefore the maximum channel count is set to 12.

Other Chromasens cameras may have only 6 channels. This will be indicated by the entry “no_of_valid_entries” in the CS_CHANNEL_STRUCT-structure.

```
const INT16 MAX_CHANNELS = 12;
const INT16 MAX_ANALOG_COARSE_GAIN = 6;
const INT16 COLOR_CHANNELS = 3;
const INT16 MAX_CAM_CONNECTIONS = 20;
```

Used for parameters belonging to the allPIXA wave camera which usually consist out of a red, green, blue and white(grey) sensor line

```
const INT16 MAX_SENSOR_LINES = 4;
```

This indicates the maximum possible ROIs for an allPIXA wave camera.

```
const INT16 MAX_ROIS = 4;
```

Indicates that this parameter is not used!

```
const INT16 PARAMETER_NOT_USED = -9999;
```

Used in the operational values of the camera. The speed data indicates if the detected speed is too fast regarding the minimum integration speed.

```
const double SPEED_2_HIGH = 9999.0;     // Indicates that the detected speed of the encoder is too
                                           high
const double SPEED_2_SLOW = -9999.0;    // Indicates that the detected speed of the encoder is too
                                           slow
const double SPEED_NO_DATA = -9999.9; // No speed data received
```

Constants used to access the single fields in the channel struct

```
const INT16 RED_ODD = 0;
const INT16 RED_EVEN = 1;
const INT16 GREEN_ODD = 2;
const INT16 GREEN_EVEN = 3;
const INT16 BLUE_ODD = 4;
const INT16 BLUE_EVEN = 5;
const INT16 RED_ODD_REAR = 6;
const INT16 RED_EVEN_REAR = 7;
const INT16 GREEN_EVEN_REAR = 8;
const INT16 GREEN_ODD_REAR = 9;
const INT16 BLUE_ODD_REAR = 10;
const INT16 BLUE_EVEN_REAR = 11;
```

Constants used to access the single fields in the analog coarse gain struct, the settings for the linear gain or the sensitivity.

```
const INT16 RED = 0;
const INT16 GREEN = 1;
const INT16 BLUE = 2;
const INT16 WHITE = 3;
const INT16 RED_REAR = 3;
const INT16 GREEN_REAR = 4;
const INT16 BLUE_REAR = 5;
```

Connection related constants

The following constants are used for the connection to the camera.

Possible Connection Speeds (not needed when connecting via Ethernet).

If the desired baud rate is set to USE_MAX_BAUD_RATE the API will determine the maximum possible connection speed by itself. This may extend the time needed to connect to the camera but of cause decrease transferring time of commands and responses.

```
const INT32 BAUD_RATE_19K = 19200;
const INT32 BAUD_RATE_38K = 38400;
const INT32 BAUD_RATE_56K = 57600;
const INT32 BAUD_RATE_115K = 115200;
const INT32 USE_MAX_BAUD_RATE = 0;
```

These parameters are only needed when connecting to an Ethernet camera.

```
const INT32 USE_DEFAULT_PORT = 0;
const char*     const DEFAULT_IP_ADDRESS= "192.168.128.200";
```

PortNumber for an ethernet connection

```
const INT16 ETHERNET_CONNECTION = 1003;
```

Maximum and minimum setting numbers possible.

```
const int MAX_SETTING_NO =   18;
const int MIN_SETTING_NO =    1;
```

Maximum setting comment length

```
const UINT16 MAX_SETTING_COMMENT_LENGTH = 128;
```

Maximum string length for the connection port description

```
const UINT16 MAX_SETTING_COMMENT_LENGTH = 256;
```

Maximum text length for an error description

```
const int  MAX_ERROR_TEXT_LENGTH = 1000;
```

Maximum length for a camera trace

```
const int  MAX_TRACE_MESSAGE_TEXT_LENGTH = 15000;
```

General Error Codes

```
const INT16 CS_OK = 0;
const INT16 CS_GENERAL_ERROR = -1;
const INT16 CS_POINTER_GIVEN_IS_ZERO = -2;
const INT16 CS_CAM_NOT_CONNECTED = -3;
const INT16 CS_HSI_INTERPRETER_NOT_PRESENT = -4;
const INT16 CS_FILE_OPEN_ERROR = -5;

const INT16 CS_BAUD_RATE_NOT_SUPPORTED = -1000;
const INT16 CS_SERIAL_CONNECTION_OPEN_ERROR = -1001;
const INT16 CS_CAMERA_INTERFACE_MISSING = -1002;
const INT16 CS_DATA_RECEPTION_FROM_CAM_FAILED = 1003;
const INT16 CS_CL_PORT_NOT_PRESENT = -1004;
```

Remarks:
The error code CS_CAMERA_INTERFACE_MISSING will be returned when trying to open a CameraLink-port without the clallSerial-DLL available. This DLL is necessary to open CLPorts independent from the grabber manufacturer. It should get installed by the grabber installer. See Chapter 6 for further information.

Error codes related to setting and getting camera parameters

```
const INT16 CS_ERROR_PARAMETER_OUT_OF_RANGE = -2000;
const INT16 CS_TAG_NOT_SUPPORTED = -2004;
const INT16 CS_ERROR_COMMAND_UNKNOW = -2005;
const INT16 CS_NO_STORED_TAGS_PRESENT = -2006;
const INT16 CS_ERROR_RETRIEVING_VALUE = -2007;
const INT16 CS_ERROR_HSI_SEND = -2008;
const INT16 CS_ERROR_INVALID_DATA_LENGTH = -2009;
const INT16 CS_ERROR_INVALID_COMMAND_TYPE = -2010;
const INT16 CS_ERROR_SETTING_NO_OUT_OF_RANGE = -2011;
const INT16 CS_ERROR_IN_CAMERA_RESPONSE = -2012;
const INT16 CS_ERROR_SETTING_NOT_AVAILABLE = -2013;
const INT16 CS_ERROR_TAP_BALANCING_FAILED = -2014;
const INT16 CS_ERROR_WHITE_BALANCING_FAILED = -2015;

const INT16 CS_ERROR_CREATING_REFERENCE_PARAMS = -2200;
const INT16 CS_ERROR_SENDING_REFERENCE_PARAMS = -2201;
const INT16 CS_ERROR_SAVING_REFERENCE_PARAMS = -2202;
const INT16 CS_ERROR_REFERENCE_NO_OUT_OF_RANGE = -2203;
const INT16 CS_ERROR_REFERENCE_MODE_INVALID = -2204;

const INT16 CS_ERROR_SAVING_SETTING_2_DISK = -2210;
const INT16 CS_ERROR_TRIGGER_INPUT_NO_OUT_OF_RANGE = -2211;

const INT16 CS_ERROR_XLC_NOT_FOUND = -2220;

const INT16 CS_NOT_IMPLEMENTED_YET = -9999;
```

Bit definitions for the image insert mode

```
const UINT16    INSERT_FIRST_LINE_INFO_BLOCK = 0x01;
const UINT16    INSERT_TEST_RAMP_IN_LAST_LINE = 0x02;
const UINT16    INSERT_CHECKSUM_IN_LAST_LINE = 0x04;
const UINT16    INSERT_INFO_EACH_LINE = 0x08;
const UINT16    INSERT_ACTIVATE_PIX_9_16 = 0x10;
const UINT16    INSERT_CONTRAST_VALUE_SUM = 0x20;
```

Bit definitions for the camera trace

```
const UINT16    TRACE_GENERAL_DEBUG_INFO = 0x0001;
const UINT16    TRACE_TRANSPORT_LAYER = 0x0002;
const UINT16    TRACE_TRANSPORT_LAYER_DETAILS = 0x0004;
const UINT16    TRACE_WHITE_CONTROL = 0x0010;
const UINT16    TRACE_ENVIRONMENT_VALUES = 0x80;
const UINT16    TRACE_IMAGE_COUNTER = 0x0100;
```

Constants used to set the parameter for the IO configuration struct

```
const UINT16 ENCODER_DISABLE = 0x40;                 // encoder function is disabled generally
const UINT16 ENCODER_ENABLE = 0x80;                  // encoder function is enabled generally
const UINT16 ENCODER_ENABLE_WITH_CC2 = 0x100;        // encoder function is enabled with CC2 input
const UINT16 ENCODER_ENABLE_WITH_CC4 = 0x200;        // encoder function is enabled with CC4 input
const UINT16 ENCODER_INVERT_SCAN_DIR = 0x800;        // encoder inputs are inverted

const UINT16 ENCODER_INCR0_CC1 = 0x1;                // CC1 signal is used as encoder clock 0
const UINT16 ENCODER_INCR0_GPIO_P0_N0 = 0x4;         // GPIO_P0/N0(X5-1/9) is used as encoder
                                                     // clock 0
const UINT16 ENCODER_INCR0_GPIO_P2= 0x10;            // GPIO_P2 (X5-4) signal is used as encoder
                                                     // clock 0
const UINT16 ENCODER_INCR0_LOW = 0x1000;             // encoder clock 0 is set to static low
const UINT16 ENCODER_INCR0_HIGH = 0x4000;            // encoder clock 0 is set to static high

const UINT16 ENCODER_INCR1_CC2 = 0x2;                // CC2 signal is used as encoder clock 1
const UINT16 ENCODER_INCR1_GPIO_P1_N1 = 0x8;         // GPIO_P1/N1 (X5-2/10) is used as encoder
                                                     // clock 1
const UINT16 ENCODER_INCR1_GPIO_N2= 0x20;            // GPIO_N2 (X5-12 )signal is used as encoder
                                                     // clock 1
const UINT16 ENCODER_INCR1_LOW = 0x2000;             // encoder clock 1 is set to static low
const UINT16 ENCODER_INCR1_HIGH = 0x8000;            // encoder clock 1 is set to static high

const UINT16 LIGHTBARRIER_CC3 = 0x1;                 // CC3 is used for LB0 + LB2
const UINT16 LIGHTBARRIER_CC4 = 0x2;                 // CC4 is used for LB1 + LB2
const UINT16 LIGHTBARRIER_GPIO_P0_N0 = 0x4;          // GPIO_P0/N0(X5-1/9)  is used for LB2
const UINT16 LIGHTBARRIER_GPIO_P1_N1 = 0x8;          // GPIO_P1/N1(X5-2/10)  is used for LB2 + LB3
const UINT16 LIGHTBARRIER_INTERNAL = 0x10;           // Frame trigger will be generated by the camera
                                                     // due to the set image parameters
const UINT16 LIGHTBARRIER_GPIO_P4 = 0x80;            // GPIO_P4 (X5-3) is used for LB2 + LB3

const UINT16 SELECT_SCAN_DIR_CC2 = 0x100;            // Scan Dir selection via CC2
const UINT16 SELECT_SCAN_DIR_GPIO_P1_N1 = 0x200;// Scan Dir selection via GPIO P1/N1(X5-2/10)
const UINT16 SELECT_SCAN_DIR_GPIO_P2 = 0x400;        // Scan Dir selection via GPIO P2(X5-4)

const UINT16 SELECT_MASTER_GPIO_4 = 0x01;          // GPIO_P4 (X5-3) is used to select the master
                                                   // camera
const UINT16 SELECT_MASTER_CC4 = 0x02;             // CC4 signal is used to select the master camera

const UINT16 MASTER_SLAVE_IF_X5_4_12 = 0x800;      // Master/Slave signals are transmitted via X5 over
                                                   // Pins 4 and 12 !Only firmware packages <P2.22
                                                   // or Classic >P1.42
const UINT16 MASTER_SLAVE_IF_X5_6_8 = 0x1000;      // Master/Slave signals are transmitted via X5 over
                                                   // Pins 6 and 8
const UINT16 MASTER_SLAVE_IF_INTERNAL_X9_1_8 = 0x2000;          // Master/Slave signals are
                                                   // internally transmitted via X9 over Pins 1
                                                   // and 8  ! Only allPIXA pro
```

```
const UINT16 MASTER_SLAVE_IF_INTERNAL_X12_2_4 = 0x4000;       // Master/Slave signals are
                                                     // internally transmitted via X12 over Pins 2 and 4
                                                     // ! Only allPIXA pro
const UINT16 MASTER_SLAVE_IF_HUB_X5 = 0x8000;        // Master/Slave signals are transmitted via X5 over
                                                     // Pins 6-4 and 8-12  possibility to daisy chain
                                                     // cameras (More than 2)! Only allPIXA pro

const UINT16 GENERALFUNCTION_VSY = 0x1;               // GPIO_P2(X5-4)  is output and shows frame
                                                      // signal
const UINT16 GENERALFUNCTION_HSY = 0x4;               // GPIO_N2 (X5-12) is output and shows line
                                                      // signal
const UINT16 GENERALFUNCTION_FVAL_WHILE_FLASHING = 0x4000;  // Framing signal transmitted
                                                     // during Flash sequence   ! Only allPIXA pro
```

Constants for the basic HSI-communication

Tag-formats

```
const UINT16 TAG_BIN0 =        0x0000;
const UINT16 TAG_BIN1 =        0x2000;
const UINT16 TAG_SHORT=        0x4000;
const UINT16 TAG_LONG =        0x6000;
const UINT16 TAG_VAR =         0x8000;
const UINT16 TAG_CONT =        0xA000;
```

Standard command type for HSI-commands

```
const char* const STD_HSICMD_TYPE = "MK";
```

3.5.8 Enumerations

These enumerations must be used by the caller to select functions, settings or certain parameters.

It is recommended not to use the specific integer values in order to be compliant with any changes that may be done to the basic values.

CS_WHITE_BALANCING_MODE	Used for choosing the white balancing mode. Currently only one mode is supported.

Definition	enum CS_WHITE_BALANCING_MODE {
	GAIN_CONTROL_USING_AREA_RANGE=1
	};

Elements	GAIN_CONTROL_USING_AREA_RANGE: The White balancing will be done in the defined area

CS_AVERAGING	Used for selecting the used averaging mode for the calculation of the white balancing or the encoder averaging.

Definition	enum CS_WHITE_BALANCING_AVERAGING {
	NO_AVERAGING = 0,
	USE_2_SAMPLES_FOR_AVERAGING,
	USE_4_SAMPLES_FOR_AVERAGING,
	USE_8_SAMPLES_FOR_AVERAGING,
	USE_16_SAMPLES_FOR_AVERAGING,
	USE_32_SAMPLES_FOR_AVERAGING,
	};

Elements	The specific entry will determine how many samples will be used to generate a control value for the white balancing process or the averaging of encoder pulses. Please note that 32 Samples averaging is only valid for the white reference averaging!

CS_ENCODER_SYNC_MODE	Used to set the synchronization of the encoder mode. Line Trigger Mode means that for every encoder pulse a line in the camera is captured. If set to STANDARD_ENCODER the encoder settings will determine the behavior of the acquisition.

Definition	enum CS_ENCODER_SYNC_MODE{
	STANDARD_ENCODER = 1,
	LINE_TRIGGER = 5
	};

Elements	STANDARD_ENCODER: Camera will treat the set encoder input as an encoder. Several parameters can be set in this mode: encoder tics per mm, etc. LINE_TRIGGER: The camera will output a line for each incoming pulse

CS_MULTI_TAP_MODE	Used for the setting of the multi-tab mode in the reference struct

Definition	enum CS_MULTI_TAP_MODE{
	FRONT_TAP_IS_MASTER = 0,
	REAR_TAP_IS_MASTER,
	INDEPENDENT_WHITE_BALANCING,
	AUTOMATIC_MASTER_DETECTION
	};

Elements	FRONT_TAP_IS_MASTER: White balancing is done only on the front tap. The rear tap follows the front tap REAR_TAP_IS_MASTER: White balancing is done only on the rear tap. The front tap follows the rear tap INDEPENDENT_WHITE_BALANCING: White balancing is done only for both tab independent AUTOMATIC_MASTER_DETECTION: The determination which tap is the master tap is done automatically by the position of the reference. This is the default and recommended setting.

CS_COARSE_GAIN	Used for the setting of the analog coarse gain

Definition	enum CS_COARSE_GAIN{
	MINUS_3DB = 0,
	_0_DB,
	_3_DB,
	_6_DB
	};

Elements	The amount of the coarse gain to set

CS_CDS_GAIN	Used for the setting of the CDS gain for the allPIXA wave

Definition	enum CS_CDS_GAIN{
	CDS_X1 = 0,
	CDS_X2
	};

Elements	The amount of the CDS gain to set (Factor 1 or 2)

CS_SENSITIVITY	Used for the setting of the sensor sensitivity

Definition	enum CS_SENSITIVITY{
	    HIGH_DYNAMIC,
	    HIGH_SENSITIVITY
	};

Elements	The amount of the sensitivity to set

CS_CAMLINK_SPEED	Used to set speed mode for the cameraLink connection. This parameter sets the pixel clock speed of the CameraLink transmission. Reduced pixel clock might solve transmission problems when camera is operating at the edge of the CameraLink specifications or when using cables of less quality. There will be no limitations when using CS-cameras up to 4K. When using a 7K-camera restrictions in regards of the line period/integration time may apply.

Definition	enum CS_CAMLINK_SPEED {
	    CL_STANDARD=0,
	    CL_HIGH_SPEED,
	    CL_REDUCED_SPEED
	};

Elements	CL_STANDARD: reduced pixel clock of the Camera Link transmission is used. allPIXA camera: 75 MHz CL_HIGH_SPEED: maximum pixel clock is used. allPIXA: 85 MHz CL_REDUCED_SPEED: Slower pixel clock is used allPIXA: 63 MHz

CS_CAMLINK_MODE	Used to set mode of the cameraLink connection. If CL_BASE_MODE is selected only the first CameraLink connector is used for the transmission of the image data. In the base mode the speed of the camera is limited to the amount of data which can be transmitted via a CameraLink base connection. Be sure that integration time and line length fit these parameters.

Definition	enum CS_CAMLINK_MODE{
	CL_BASE_MODE=0,
	CL_MEDIUM_MODE=1,
	CL_MEDIUM_MODE_1X2=2,
	CL_FULL64_1X8=3,
	CL_FULL80_1X8=4,
	CL_FULL80_1X10=5,
	CL_BASE_1X3=6,
	CL_BASE_MODE_WAVE = 0,
	CL_MEDIUM_1X4_MONO_WAVE = 1,
	CL_MEDIUM_1X4_4T_12BIT_MONO_WAVE = 2,
	CL_MEDIUM_1X2_2T_8BIT_RGB_WAVE = 3,
	CL_MEDIUM_1X4_1T_12BIT_RGB_WAVE = 4,
	CL_FULL64_1X8_8T_8BIT_RAW_WAVE = 5,
	CL_FULL80_1X10_10T_8BIT_RAW_WAVE = 6,
	CL_FULL80_1X8_8T_10BIT_RAW_WAVE = 7
	};

Elements	CL_BASE_MODE: Only one CL-channel is used for the image transfer CL_MEDIUM: Both CL connectors are used for image data transmission 2XE geometry is used ⚠ *The following modes are supported only by the allPIXA pro and require both CL connectors:* CL_MEDIUM_MODE_1X2: Camera is used in medium CameraLink mode. 2 Taps are transmitted per clock. CL_FULL64_1X8_RAW: Camera is used in full CameraLink mode. 8 Taps are transmitted per clock in 1X8 raw format. If a mono camera is used the Camera is set into 1X8 mode CL_FULL80_1X8: Camera is used in full 80 CameraLink mode. 8 Taps with 10 bits are transmitted per clock. CL_FULL80_1X10: Camera is used in full 80 CameraLink mode. 10 Taps with 8 bits are transmitted per clock. CL_BASE_1X3: Only one CL-channel is used for the image transfer. 3 Pixels are transmitted per cycle. This mode is only available for the grey modes in the allPIXA pro! This enables lower integration times when using base mode.

	For more detailed information about the different CameraLink modes please refer to the allPIXA manual or the CameraLink reference. ⚠ *The following modes are supported only by the allPIXA wave* CL_BASE_MODE_WAVE : Camera is used in base mode: 1x RGB (1 tabs with 8 bits information) RGB (only allPIXA wave)) CL_MEDIUM_1X4_MONO_WAVE: Camera is used in base mode: 1x4 mono (4 tabs with 8 bits information) gray (only allPIXA wave)) CL_MEDIUM_1X4_4T_12BIT_MONO_WAVE: Camera is used in base mode: 1x4 mono (4 tabs with 12 bits information) gray (only allPIXA wave)) CL_MEDIUM_1X2_2T_8BIT_RGB_WAVE: Camera is used in medium mode: 1x2 RGB (2 tabs with 12 bits information), RGB (only allPIXA wave)) CL_MEDIUM_1X4_1T_12BIT_RGB_WAVE: Camera is used in medium mode: 1x4 RGB (4 tabs with 12 bits information), RGB (only allPIXA wave)) CL_FULL64_1X8_8T_8BIT_RAW_WAVE: Camera is used in full mode: 1x8 raw (8 tabs each with 8 bits information), RGB or gray (only allPIXA wave)) CL_FULL80_1X10_10T_8BIT_RAW_WAVE: Camera is used in full mode: 1x10 raw, (10 tabs each with 8 bits information) RGB or gray (only allPIXA wave) CL_FULL80_1X8_8T_10BIT_RAW_WAVE: Camera is used in full mode: 1x8 raw, (8 tabs each with 10 bits information) RGB or gray (only allPIXA wave)

CS_VIDEO_OUTPUT_MODE	Used to set the grey video output mode

Definition	enum CS_VIDEO_OUTPUT_MODE {
	    _3x8_BIT_COLOR_PARALLEL = 0,
	    _2x8_BIT_GREY_FIRST_CL_PORT,
	    _2x10_BIT_GREY_FIRST_CL_PORT,
	    _2x12_BIT_GREY_FIRST_CL_PORT,
	    _2x8_BIT_GREY_CL_DUAL_BASE = 6,
	    _2x10_BIT_GREY_CL_DUAL_BASE,
	    _2x12_BIT_GREY_CL_DUAL_BASE
	
	    // Definitions for the wave camera
	        GREY = 0,
	        COLOUR,
	        GREY_AND_COLOUR
	};

Elements	Description of the different modes: _3x8_BIT_COLOR_PARALLEL: CL-MEDIUM, Standard color mode _2x8_BIT_GREY_FIRST_CL_PORT: CL-BASE: Grey output on the first CL-port with 8Bit data depth _2x10_BIT_GREY_FIRST_CL_PORT: CL-BASE: Grey output on the first CL-port with 10Bit data depth _2x12_BIT_GREY_FIRST_CL_PORT: CL-BASE, Grey output on the first CL-port with 12Bit data depth _2x8_BIT_GREY_CL_DUAL_BASE: CL-MEDIUM, Grey output on both CL-ports with 8Bit data depth _2x10_BIT_GREY_CL_DUAL_BASE: CL-MEDIUM, Grey output on both CL-ports with 10Bit data depth _2x12_BIT_GREY_CL_DUAL_BASE: CL-MEDIUM, Grey output on both CL-ports with 12Bit data depth These definitions arte only valid if connected to an allPIXA wave camera, the other definitions are not used in this case!: GREY: Only the data from the “white” sensor line will be transmitted COLOUR: Only the data from the “red, gree and blue” sensor lines will be transmitted GREY_AND_COLOUR: data from all four sensor lines will be transmitted

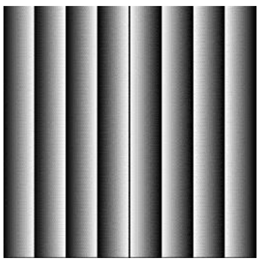
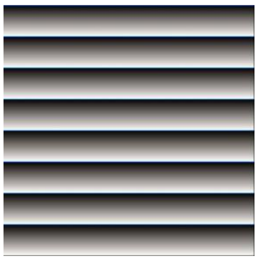
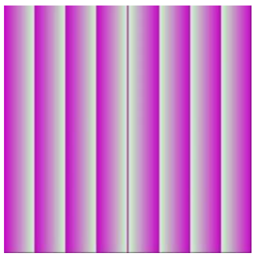
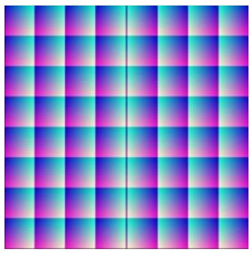
CS_SETTINGS_FORMAT	Indicates how the data of a camera setting should be saved to disk

Definition	enum CS_SETTINGS_FORMAT {
	        FORMAT_BINARY = 0,
	        FORMAT_XML
	};

Elements	FORMAT_BINARY: Binary data format which will be a HSI command set FORMAT_XML: XML data format which can be read and displayed by common editors

CS_TEST_PATTERN	This enumeration is used to set the different test patterns in the camera. The grey ramps will be generated from 0 to 255 grey levels. Increasing by 1 digit per pixel. If set to SEQUENCE_OF_PATTERNS the camera will output alternating the first three test patterns and at last a live image.

Definition	enum CS_TEST_PATTERN {
	TEST_PATTERN_OFF = 0,
	GREY_RAMP_IN_CCD_DIR,
	GREY_RAMP_IN_TRANSPORT_DIR,
	INPUT_RAMP_ON_GREEN_CHANNEL,
	SEQUENCE_OF_PATTERNS,
	CHANGE_VIDEO_LEVEL_AT_EVERY_PIXEL
	};

Elements	GREY_RAMP_IN_CCD_DIR:  Values from 0 to 255 in transport direction GREY_RAMP_IN_TRANSPORT_DIR:  Values from 0 to 255 in transport direction INPUT_RAMP_ON_GREEN_CHANNEL:  Values from 0 to 255 in the green channel and fixed values(Test pattern level) in the other channels. SEQUENCE_OF_PATTERNS  Four alternating test patterns including 1 live image CHANGE_VIDEO_LEVEL_AT_EVERY_PIXEL: Not supported yet

CS_REFERENCE_MODE	This enumeration is used in combination with the function prepare_cam_4reference. It sets the parameter to the desired acquisition mode. According to the reference to acquire the camera parameter are set differently to fit the correct reference image.

Definition	enum CS_REFERENCE_MODE {
	    BLACK_LEVEL_CORRECTION = 0,
	    SHADING_CORRECTION
	};

Elements	BLACK_LEVEL_CORRECTION: Sets the camera into the correct mode for acquiring a black level reference SHADING_CORRECTION: Sets the camera into the correct mode for acquiring a shading reference

CS_CALC_REFERENCE_MODE	This enumeration is used to set the calculation mode for the convenience function calculate_reference. The user may choose here if the API should determine a suitable area where to calculate the reference or if the user sets the area manually.

Definition	enum CS_CALC_REFERENCE_MODE{
	    AUTOMATIC_DETECTION=0,
	    MANUAL_SETTING
	};

Elements	AUTOMATIC_DETECTION: The parameters for the reference generation will be determined automatically MANUAL_SETTING: Set the used parameters for the reference generation manually

CS_REFERENCE_PLANE_DEFINITION	This enumeration is used to set the type of the reference plane that is used for the reference generation. Only needed if single color images are provided to the API.

Definition	enum CS_REFERENCE_PLANE_DEFINITION{
	    RED_REF_PLANE = 0,
	    GREEN_REF_PLANE,
	    BLUE_REF_PLANE,
	    MONOCHROME_REF_PLANE,
	    SPECIAL_REF_PLANE1,
	    SPECIAL_REF_PLANE2,
	    SPECIAL_REF_PLANE3,
	    SPECIAL_REF_PLANE4,
	    SPECIAL_REF_PLANE5,
	    SPECIAL_REF_PLANE6,
	    SPECIAL_REF_PLANE7,
	    SPECIAL_REF_PLANE8,
	    SPECIAL_REF_PLANE9
	  };

Elements	Elements will define the usage of the given single color image for the reference generation. The SPECIAL_REF_PLANEX planes are not yet defined and serve as placeholders

CS_REFERENCE_VERSION	This enumeration is used to indicate which kind of reference will be generated. If this is a wave camera, always use CS_ALLPIXA_WAVE_VERSION

Definition	enum CS_REFERENCE_TYPE{
	CS_GENERAL_CAM_REF_VERSION = 0,
	CS_ALLPIXA_WAVE_VERSION = 10
	};

Elements	For all other cameras than the wave camera use the CS_GENERAL_CAM_REF_VERSION type.

CS_TRIGGER_MODE	The trigger mode enumeration is used in the function set_camera_trigger_mode, respectively in the structure CS_TRIGGER_STRUCT. It sets the trigger mode of the camera. In “FREE_RUNNING” and “START_CONDITION_ONLY” mode the length of the images is determined by the image parameters. If “START_STOP_CONDITION” is selected the length of the images is determined by the length of the trigger signal.

Definition	enum CS_TRIGGER_MODE {
	FREE_RUNNING = 0,
	START_CONDITION_ONLY,
	START_STOP_CONDITION = 3
	};

Elements	FREE_RUNNING: Camera is not waiting for a trigger signal. START_CONDITION_ONLY: The camera will look for a start condition on the configured input to start outputting image data START_STOP_CONDITION: The camera will be in frame mode. The external inputs will be controlling the length of the acquired image.

CS_TRIGGER_INPUT	The trigger input enumeration is used to select which configured light barrier (digital input) is used to generate the trigger conditions.

Definition	enum CS_TRIGGER_INPUT {
	LIGHT_BARRIER_0 = 0,
	LIGHT_BARRIER_1,
	LIGHT_BARRIER_2,
	LIGHT_BARRIER_3
	};

Elements	LIGHT_BARRIER_X: This will select which input will be relevant for the start or stop condition.

CS_TRIGGER_CONDITION	The trigger condition enumeration will be used to select if the camera will look for a raising or falling edge to set the image

	start or stop.

Definition	enum CS_TRIGGER_CONDITION {
	RAISING_EDGE = 0,
	FALLING_EDGE
	};

Elements	RAISING_EDGE: The camera will check for a raising edge on the input FALLING_EDGE: The camera will check for a falling edge on the input

CS_MASTER_SLAVE_OPERATION	The trigger condition enumeration will be used to select if the camera will look for a raising or falling edge to set the image start or stop.

Definition	enum CS_MASTER_SLAVE_OPERATION {
	USE_SELECTION_FROM_SETTING = 0,
	NO_MASTER_SLAVE_OPERATION = 0,
	CAMERA_IS_MASTER,
	CAMERA_IS_SLAVE,
	SELECT_MASTER_SLAVE_BY_INPUT
	};

Elements	USE_SELECTION_FROM_SETTING: Used only for global master slave setting. If this is set, the setting from the active setting is used -> functions: set_global_master_slave_settings and get_global_master_slave_settings NO_MASTER_SLAVE_OPERATION: Camera is operating alone CAMERA_IS_MASTER The camera is the master and controls all other cameras regarding image acquisition, trigger and integration time CAMERA_IS_SLAVE The camera is controlled by another camera SELECT_MASTER_SLAVE_BY_INPUT: The behavior of the camera is selected by an external IO

CS_GAIN_CONTROL_DISABLED	The gain control disabled enumeration will be used to get the current state of the white balancing. It must be used to evaluate the state returned in the function get_camera_operating_values by the CS_CAMERA_OPERATING_VALUES.

Definition	enum CS_GAIN_CONTROL_DISABLED{
	GAIN_CONTROL_RUNNING = 0,
	DISABLED_LIGHT_SWITCHED_OFF_INTERNALLY,
	DISABLED_LIGHT_SWITCHED_OFF_EXTERNALLY,
	DISABLED_BY_GAIN_CONTROL_CONDITION,
	NO_REFERENCES_SET
	};

Elements	GAIN_CONTROL_RUNNING: Gain control is running DISABLED_LIGHT_SWITCHED_OFF_INTERNALLY: Gain control stopped because the illumination has been

	switched off internally DISABLED_LIGHT_SWITCHED_OFF_EXTERNALLY Gain control stopped because the illumination has been switched off externally DISABLED_BY_GAIN_CONTROL_CONDITION: Gain control stopped because the “Gain stop” condition has become true(Must have been activated by the user) NO_REFERENCES_SET: No reference data have been set by the user

CS_OPERATING_STATE	The CS_OPERATING_STATE enumeration is used to indicate the state of different devices of a Chromasens system. Possible devices are the camera itself or attached LED-modules.

Definition	enum CS_OPERATING_STATE{
	DEVICE_IN_ERROR_STATE,
	DEVICE_IS_READY,
	DEVICE_IS_WARMING_UP
	};

Elements	DEVICE_IN_ERROR_STATE: An error is currently present DEVICE_IS_READY: Device is ready for operation DEVICE_IS_WARMING_UP: Device is booting.

CS_PIXEL_FORMAT	The CS_PIXEL_FORMAT enumeration will indicate how the pixels are organized in the byte array. Currently only BGR (blue, green, red) or RGB (red, green, blue) images are supported.

Definition	enum CS_PIXEL_FORMAT{
	BGR = 0,
	RGB
	};

Elements	BGR: 🟦🟩🟥 RGB 🟥🟩🟦

CS_LED_FLASH_SYNC	This enumeration is used to select if the camera will output a synchronization signal when the flash function is active.

Definition	enum CS_LED_FLASH_SYNC {
	NO_LED_DRIVER_SYNC = 0,
	LED_885KHZ_SYNC
	};

Elements	NO_LED_DRIVER_SYNC: The camera will not output a synchronization signal when using the LED flash function LED_885KHZ_SYNC: The camera will output a 885kHz synchronization signal for LED flashing

CS_LED_FLASH	This enumeration is used to enable or disable the LED flash function

Definition	enum CS_LED_FLASH {
	FLASH_DISABLED = 0,
	FLASH_ENABLED
	};

Elements	FLASH_DISABLED: Do not use the LED flash function FLASH_ENABLED Use the flash function

CS_CAMERA_TYPE	This enumeration is used to distinguish in between the single camera types. The type can be obtained by calling the function `‘determineCameraType’`

Definition	enum CS_CAMERA_TYPE {
	ALLPIXA,
	ALLPIXA_PRO,
	ALLPIXA_WAVE
	};

Elements	ALLPIXA: standard allPIXA-Classic-camera ALLPIXA_PRO: allPIXA pro camera ALLPIXA_WAVE: allPIXA wave camera (CMOS sensor)

CS_INTERNAL_LB_ROI_LENGTH	This enumeration is used to set the size of the internal light barrier/frame trigger signal in pixel.

Definition	CS_INTERNAL_LB_ROI_LENGTH {
	INTERNAL_LIGHT_BARRIER_ROI_LENGTH_32P = 0,
	INTERNAL_LIGHT_BARRIER_ROI_LENGTH_64P,
	INTERNAL_LIGHT_BARRIER_ROI_LENGTH_128P,
	INTERNAL_LIGHT_BARRIER_ROI_LENGTH_256P,
	};

Elements	Length of the signal in pixels

CS_INTERNAL_LB_COLOR_SELECT	This enumeration is used to set the color channel for the internal light barrier/frame trigger

Definition	CS_INTERNAL_LB_COLOR_SELECT {
	INTERNAL_LIGHT_BARRIER_ALL_COLORS = 0,
	INTERNAL_LIGHT_BARRIER_RED_COLOR,
	INTERNAL_LIGHT_BARRIER_GREEN_COLOR,
	INTERNAL_LIGHT_BARRIER_BLUE_COLOR,
	};

Elements	Color channel which is used for the evaluation of the internal light barrier/frame trigger

CS_IMAGE_DATA_TYPE	Defines what kind of data is going to be transmitted when requesting image data from the camera

Definition	CS_IMAGE_DATA_TYPE {
	RAW_PIXEL_VALUES,
	BMP_IMAGE
	};

Elements	RAW_PIXEL_VALUES: Only the pixel values will be transmitted in the format BGR BMP_IMAGE: The received data can be stored as is into a file to be displayed in an image viewer such as IrfanView

CS_XLC4_IDS	This enumeration is used to distinguish in between the single connected XLC controllers. This is only available when connecting to allPIXA wave cameras.

Definition	enum CS_XLC4_IDS {
	XLC_BROADCAST = 0,
	XLC_ID_2 = 1,
	XLC_ID_3,
	XLC_ID_4,
	XLC_ID_5,
	XLC_ID_6,
	XLC_ID_7,
	XLC_ID_8,
	XLC_ID_9,
	XLC_ID_10,
	XLC_ID_11,
	XLC_ID_12,
	XLC_ID_13,
	XLC_ID_14,
	XLC_ID_15
	};

Elements	XLC_BROADCAST: Set this ID if you want to transmit data to all connected XLC-controllers XLC_ID_XX: ID of the connected XLC controller.

CS_XLC4_CHANNELS	Defines the active channels of the connected XLC controller. This is only available when connecting to allPIXA wave cameras.

Definition	enum CS_XLC4_CHANNELS {
	XLC_CHANNEL_A = 0,
	XLC_CHANNEL_B,
	XLC_CHANNEL_C,

	XLC_CHANNEL_D };
Elements	XLC_CHANNEL_X: Bit is set if the channel is active.

3.5.9 Structures

All structures used in the Chromasens API do follow the same philosophy. They use a constructor in order not to use any random values from memory.

They are prefilled with meaningful values which either indicates that the value needs to be filled or that the value should be ignored by the API.

The structures are documented together with the functions where they are used.

3.5.10 Error codes

The used error codes are listed above in section “constants” and in the header file “ChromasensCameraIf.h”.

4 Description of the API functions

4.1 Initialization of the API and the camera:

Before calling any other function a connection to the camera needs to be established by calling one of these open-functions.

Following error codes are returned at initialization functions:

CS_BAUD_RATE_NOT_SUPPORTED:
Wrong baud rate chosen

CS_SERIAL_CONNECTION_OPEN_ERROR:
serial interface could not be opened. One reason for the failure might be that the port is still in use by another program or not existing.

CS_CAMERA_INTERFACE_MISSING:
The camera interface DLL is missing.

CS_DATA_RECEPTION_FROM_CAM_FAILED:
Connection to the camera could be established but the reception of data from the camera failed.

open_control	Opens a connection to a camera by providing a connection dialog from which the user can choose the appropriate connection.
Syntax	INT16 open_control (CS_CONNECTION_STRUCT* *pConnectInfo*)
Parameters:	CS_CONNECTION_STRUCT* pConnectInfo: Pointer to a structure that will contain the chosen connection parameters by the user. If this information is not needed use NULL as parameter. If a special configuration should already be preselected the given structure should be set to these parameters.
Return value:	If successful CS_OK will be returned. If the user cancelled the dialog or an error an error code will be returned, the value will be < 0 Error codes are defined in the file ChromasensCameraIf.h.
Comment:	When calling this function, the API searches for possible connections and displays a dialog where the user can choose the desired connection to the camera.

The structure "CS_CONNECTION_STRUCT"will be filled when calling the "open_control" function with this parameter.

The structure will contain all necessary information about the chosen interface.

```
struct CS_CONNECTION_STRUCT {
        static const int MAX_CONNECTION_CHARS=200;
        static const int MAX_IP_CHARS = 16;
        INT16 portNumber;
        INT32 baudRate;
        bool doReset;
        char connectionDescription[MAX_CONNECTION_CHARS];
        char ethAddress[MAX_IP_CHARS];
        INT32 ethPort;
        CS_CONNECTION_STRUCT() {
                portNumber = 0;
                baudRate = BAUD_RATE_115K;
                doReset = false;
                strcpy_s(connectionDescription, MAX_CONNECTION_CHARS, "No descriptions");
                strcpy_s(ethAddress, MAX_IP_CHARS, DEFAULT_IP_ADDRESS);
                ethPort = 0;
        }
};
```

Variable type	Element name	Description
static const int	MAX_CONNECTION_CHARS	200 chars
static const int	MAX_IP_CHARS	16 chars
INT16	portNumber	port number of selected interface
INT32	baudRate	Baud rate of RS232 connection, if selected
bool	doReset;	RS232 connection: if true, reset of camera is performed at opening connection
char	connectionDescription[MAX_CONNECTION_CHARS];	Sets description string for connection
char	ethAddress[MAX_IP_CHARS];	Tcp/IP address, if selected
INT32	ethPort	Ethernet port, if selected

open_RS232_control	Opens a serial connection to a camera via RS232 interface
Syntax	INT16 open_RS232_control (INT32 *portNumber* , INT32 b*audRate=USE_MAX_BAUD_RATE* , bool *doReset=false)*
Parameters:	portNumber: the desired com port number to connect to baudRate: the desired connection speed to use for the connection Use the constants BAUD_RATE_19K, BAUD_RATE_38K BAUD_RATE_56K, BAUD_RATE_115K or USE_MAX_BAUD_RATE doReset: If this parameter is set to true a hardware reset is performed upon connection
Return value:	If successful CS_OK will be returned. If an error occurred an error code will be returned, the value will be < 0 Error codes are defined in the file ChromasensCameraIf.h.
Comment:	The baud rate and the “doReset” parameter do not need to be set. If not the default settings are used.

open_CL_control	Opens a serial connection to a camera via CameraLink Interface
Syntax	INT16 open_CL_control (INT16 portNumber, INT32 baudRate=USE_MAX_BAUD_RATE)
Parameters:	portNumber: the desired port number to connect to. The port number will be the index of the entry displayed in the dialog displayed when calling open_control. The index starts with 0 for the first entry. baudRate: the desired connection speed to use for the connection Use the constants BAUD_RATE_19K, BAUD_RATE_38K, BAUD_RATE_56K, BAUD_RATE_115K or USE_MAX_BAUD_RATE
Return value:	If successful CS_OK will be returned. If an error occurred an error code will be returned, the value will be < 0 Error codes are defined in the file ChromasensCameraIf.h.
Comment:	

open_ETH_control	Opens a connection to a camera via Ethernet interface
Syntax	INT16 open_ETH_control (const char* ipAddress=DEFAULT_IP_ADDRESS, INT16 portNumber=USE_DEFAULT_PORT)
Parameters:	ipAddress: pointer to a character array that contains the IP-address in the format 123.123.123.123 port: The desired port number for the TCP-connection to the camera will be set here. The camera uses port number 7501 by default for the communication
Return value:	If successful CS_OK will be returned. If an error occurred an error code will be returned, the value will be < 0 Error codes are defined in the file ChromasensCameraIf.h.
Comment:	

close_control	Closes the connection to the camera
Syntax	INT16 close_control (void)
Parameters:	None
Return value:	If successful CS_OK will be returned. If an error occurred an error code will be returned, the value will be < 0 Error codes are defined in the file ChromasensCameraIf.h.
Comment:	

get_number_of_available_interfaces	Get number of interfaces in the system
Syntax	INT16 get_number_of_available_interfaces(void)
Parameters:	None
Return value:	Returns the number of found interfaces in the system
Comment:	This function will return the number of CameraLink and serial interfaces found in the system.

get_number_of_cameraLinkPorts	Get number of available cameraLinkPorts in the system
Syntax	INT16 get_no_of_cameraLink_ports (void)
Parameters:	None
Return value:	Returns the number of found cameraLinkPorts in the system
Comment:	This function will return the number of CameraLink interfaces found in the system.

get_no_of_rs232_ports	Get number of RS232 interfaces in the system
Syntax	INT16 get_no_of_rs232_ports(void)
Parameters:	None
Return value:	Returns the number of found RS232 ports in the system
Comment:	This function will return the number of serial interfaces found in the system.

get_interface_description	Get the textual description of a desired interface
Syntax	INT16 get_interface_description(INT16 connectionID, char** pConnectionDescription)
Parameters:	INT16 connectionID: The ID is the a number ranging from 0 to maximum interfaces in the system. char** pConnectionDescription: Address of a char pointer. The Pointer will be redirected to the internal memory structure. The char array will have the length
Return value:	If successful CS_OK will be returned. If an error occurred an error code will be returned, the value will be < 0 Error codes are defined in the file ChromasensCameraIf.h.
Comment:	The lower IDs will contain the CameraLink connections. The order of the CameraLink IDs is determined by the order the clallSerial is returning the Interfaces. The IDs for the serial interfaces is determined by the number of the serial port ranging from lower to higher numbers

autodetect_cameras	Get the number of connected Chromasens cameras
Syntax	INT16 autodetect_cameras(bool searchOnCameraLink=true, bool searchOnRS232=true, bool searchOnEthernet=true)
Parameters:	bool searchOnCameraLink: If set to true, the search will be performed on all available CL interfaces. bool searchOnRS232: If set to true, the search will be performed on all available RS232 interfaces. Disable the search on RS232 for speeding-up the search. bool searchOnEthernet: If set to true, the search will be performed on all available ethernet interfaces. Currently NOT available
Return value:	If successful, the number of found cameras will be returned. If an error occurred an error code will be returned, the value will be < 0 Error codes are defined in the file ChromasensCameraIf.h.
Comment:	Use an index number up to [number of cameras]-1 with get_camera_information to get information about the cameras. Important: If using this function, the clALLSerial.dll shipped with this API should be used! The standard DLL does not support parallel access. We added some synchronization methods to it in order to make it thread safe. The DLL can be found in the directory “clAllSerial” of the package. To figure out where to copy this DLL see chapter 6.1 or the readme-file in this folder.

4.2 Getting and setting parameters:

flag “bSend2Camera”

Parameters can be sent immediately to the camera by setting the “bSend2Camera” = true.

In order to reduce communication bandwidth it is also possible to set the parameters at first in memory and then transmit everything to the camera after collecting all parameters. For this set “bSend2Camera” = false.

When setting the last desired parameter, simply set the “bSend2Camera”=true to transmit complete parameters to the camera.

If you want to clear the settings which you prepared in memory, simply request a parameter from the camera and set the “bUpdate”=true. This will get a complete parameter set from the camera and overwrite all changes in memory on the PC done by the set-function.

flag “bUpdate”

At startup of the connection the CS-Api reads the complete parameter set from the camera and holds the data in the memory of the system. When changing parameters the values are updated in the camera and the memory of the PC. By this the data structure in the memory keep the actual data like it is present in the camera.

There are some parameter values which might be changed internally by the camera. For example if automatic gain control is active, actual gain values are adapted frequently. Therefore the parameter values inside the camera and in the CS-Api memory may differ.

Using a “get…..” function for reading parameter data with flag bUpdate = true the complete data set in the CS-Api will be updated with the data from the camera.
With bUpdate = false the data held in the CS-Api is returned at “get…” function call.

This mechanism is implemented to minimize data traffic on communication interface.

These flags will not be explained any further in the parameter descriptions.

4.2.1 General parameters

get_gain_values	Retrieve the gain values from the camera
Syntax	INT16 get_gain_values(CS_CHANNEL_STRUCT * gainValues, bool bUpdate)
Parameters:	gainValues: pointer to a structure with an array of the gain values to set and the number of valid entries. To access single entries use the constants like RED_ODD, BLUE_REAR or similar. Since the allPIXA consist of a two tap sensor, the first 6 gain values are used to control the front tap the second 6 values are used to control the rear tap of the sensor.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Values are in the 10bit-range (0-1023)
Supported cameras:	allPIXA Classic, allPIXA pro

Structure “**CS_CHANNEL_STRUCT**“ is used to get and set values which relate to the setting of the different channels between camera and sensor.

When reading the values from the camera the "no_of_valid_entries" field will be set to indicate if this is a camera with two tabs (12 valid entries) or with a single tab (6 valid entries) e.g. gain, target reference values, etc.

Structure for handling camera channel values:

```
struct CS_CHANNEL_STRUCT   {
        UINT16 no_of_valid_entries;
        UINT16 value[MAX_CHANNELS];
        CS_CHANNEL_STRUCT()
        {
                no_of_valid_entries = 0;
                for(int i = 0; i < MAX_CHANNELS; i++)
                        value[i] = 0;
        }
} ;
```

Variable type	Element name	Description
UINT16	no_of_valid_entries	Number of valid gain channels
UINT16	value[MAX_CHANNELS]	Gain values

set_gain_values	Sets the gain values for the camera.
Syntax	INT16 set_gain_values(CS_CHANNEL_STRUCT gainValues, bool bSend2Camera);
Parameters:	gainValues: structure with an array of the gain values to set and the number of valid entries. To access single entries use the constants like RED_ODD, BLUE_REAR or similar. Since the allPIXA consist of a two tap sensor, the first 6 gain values are used to control the front tap the second 6 values are used to control the rear tap of the sensor.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Values are in the 10bit-range (0-1023)
Supported cameras:	allPIXA Classic, allPIXA pro

get_initial_gain_values	Retrieve the initial gain values. These gain values are used as starting values after the camera was started or the setting was loaded. If these gain values are set close to the ones during operation, the camera is able to be faster within the correct operation range.
Syntax	INT16 get_initial_gain_values(CS_CHANNEL_STRUCT * gainValues, bool bUpdate)
Parameters:	gainValues: pointer to a structure with an array of the gain values to be set and the number of valid entries. To access single entries use the constants like RED_ODD, BLUE_REAR or similar. Since the allPIXA consist of a two tap sensor, the first 6 gain values are used to control the front tap the second 6 values are used to control the rear tap of the sensor.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Values are in the 10bit-range (0-1023)
Supported cameras:	allPIXA Classic, allPIXA pro

set_initial_gain_values	Set the initial gain values. These gain values are used as starting values after the camera was started or the setting was loaded. If these gain values are set close to the ones during operation, the camera is able to correct the gain values faster to be within the correct operation range.
Syntax	INT16 set_initial_gainValues(CS_CHANNEL_STRUCT gainValues, bool bSend2Camera)
Parameters:	gainValues: structure with an array of the gain values to set and the number of valid entries. To access single entries use the constants like RED_ODD, BLUE_REAR or similar. Since the allPIXA consist of a two tap sensor, the first 6 gain values are used to control the front tap the second 6 values are used to control the rear tap of the sensor.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Values are in the 10bit-range (0-1023)
Supported cameras:	allPIXA Classic, allPIXA pro

get_coarse gain_values	Retrieve the analog gain values
Syntax	INT16 get_coarse_gain_values(CS_ANALOG_COARSE_GAIN_STRUCT *coarseGainValues, bool bUpdate)
Parameters:	coarseGainValues: structure which will contain six coarse gain values. The first 3 are used for the front tap and the last three are used for the rear tap. To access them in the array, please use the constants RED, GREEN, BLUE, RED_REAR, etc.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Only four settings are allowed. Use the enum CS_COARSE_GAIN for the possible settings. A range from -3dB to +6dB is possible
Supported cameras:	allPIXA Classic, allPIXA pro

Structure which is used for the configuration of the analog coarse gain of the allPIXA

The coarse gain is not set individually for odd and even channel.
Use the constants RED, GREEN, BLUE, RED_REAR, GREEN_REAR, BLUE_REAR to access the single values.

```
struct CS_ANALOG_COARSE_GAIN_STRUCT{
        UINT16 no_of_valid_entries;
        UINT16 value[MAX_ANALOG_COARSE_GAIN];
        CS_ANALOG_COARSE_GAIN_STRUCT()
        {
                no_of_valid_entries = 0;
                for(int i = 0; i < MAX_ANALOG_COARSE_GAIN; i++)
                        value[i] = 0;
        }
} ;
```

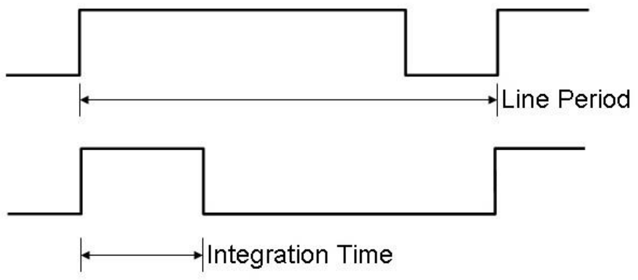
Variable type	Element name	Description
UINT16	no_of_valid_entries	
UINT16	value[MAX_ANALOG_COARSE_GAIN]	

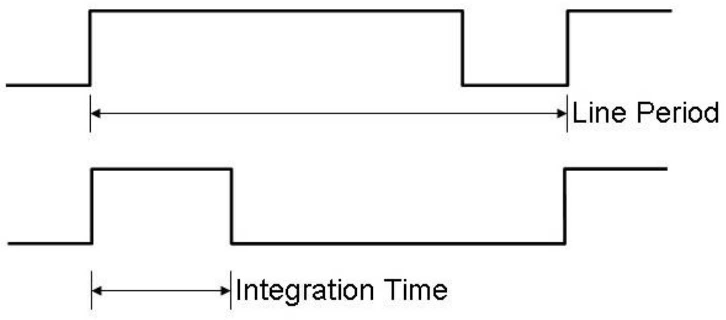
set_coarse_gain_values	Set the analog coarse gain values.
Syntax	INT16 set_coarse_gain_values(CS_ANALOG_COARSE_GAIN_STRUCT coarseGainValues, bool bSend2Camera);
Parameters:	coarseGainValues: structure which will contain six coarse gain values. The first 3 are used for the front tap and the last three are used for the rear tap. To access them in the array, please use the constants RED, GREEN, BLUE, RED_REAR, etc.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Only four settings are allowed. Use the enum CS_COARSE_GAIN for the possible settings. A range from -3dB to +6dB is possible
Supported cameras:	allPIXA Classic, allPIXA pro

get_integration_time	Get the currently set parameters for the integration time and the line period.
Syntax	INT16 get_integration_time(CS_INTEGRATION_TIME_STRUCT *integrationTime, bool bUpdate=false)
Parameters:	integrationTime: pointer to a CS_INTEGRATION_TIME_STRUCT-structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	When using an allPIXA wave camera the 'use_line_period' feature is always active and can not be set!

Structure to set the integration time parameters

```
struct CS_INTEGRATION_TIME_STRUCT{
        UINT32 integration_time_in_ns;
        bool  use_line_period_feature;
        UINT32 line_period_in_ns;
        // Initialization of the struct
        CS_INTEGRATION_TIME_STRUCT() {
                integration_time_in_ns = 50000;
                use_line_period_feature = false;
                line_period_in_ns = 50000;
        }
} ;
```

Variable type	Element name	Description
UINT32	integration_time_in_ns	The used integration time for the sensor
bool	use_line_period_feature	if this feature is used you can set a shorter integration time than the line period  When using an allPIXA wave camera the ‘use_line_period’ feature is always active and can not be set!
UINT32	line_period_in_ns	The line period time. Time from LVAL to LVAL.

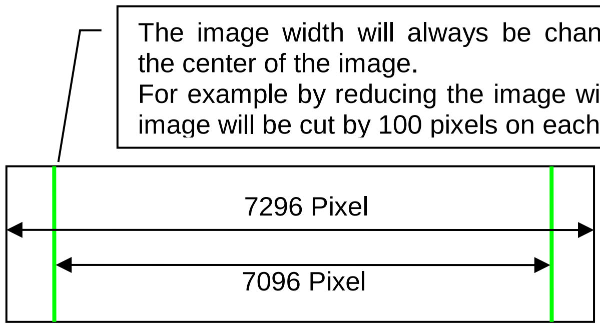
set_integration_time	Sets the parameters regarding the integration time
Syntax	set_integration_time(CS_INTEGRATION_TIME_STRUCT integrationTime, bool bSend2Camera=true)
Parameters:	integrationTime: Sets the parameters regarding the line period and integration time of the camera. If the line period feature is switched off, the line period is equal to the integration time of the camera. If desired the integration time for line capturing may be selected shorter than the line period. The integration time is entered in ns.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Make sure that if you use the line period feature the integration time is always smaller than the line period.  When using an allPIXA wave camera the ‘use_line_period’ feature is always active and can not be set!

get_image_parameters	Returns the currently set image parameters
Syntax	INT16 get_image_parameters(CS_IMAGE_PARAMETER_STRUCT *imageParameters, bool bUpdate=false)
Parameters:	imageParameters: pointer to a CS_IMAGE_PARAMETER_STRUCT-structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Default:

Structure to set the image parameters

```
struct CS_IMAGE_PARAMETER_STRUCT{
        UINT32 img_height;
        UINT16 img_width;
        UINT16 first_scan_line_delay;
        // Initialization of the struct
        CS_IMAGE_PARAMETER_STRUCT() {
                img_height = 500;
                img_width = 500;
                first_scan_line_delay = 0;
        }
} ;
```

Variable type	Element name	Description
UINT32	img_height	Height of the image to acquire. Only used if the camera is not controlled by using external start and stop conditions for the image
UINT16	img_width	Width of the image to acquire.
UINT16	first_scan_line_delay	only used if in frame scan mode -> delays the start of the image if in frame scan mode

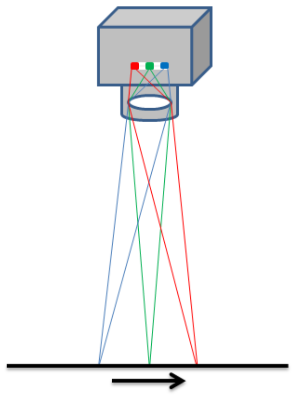
set_image_parameters	Sets the image related parameters like height and width.
Syntax	INT16 set_image_parameters(CS_IMAGE_PARAMETER_STRUCT imageParameters, bool bSend2Camera=true)
Parameters:	imageParameters: This struct contains parameters like width, and height. It is also possible to insert a gap in between images if the camera is in frame scan mode by the first_scan_line_delay parameter. The image will start after n lines. Due to the CCD technology it is not possible to read out ROIs. It is only possible to change the read out width symmetrically to the center. The image width will always be changed symmetrically to the center of the image. For example by reducing the image width to 7096 pixel, the image will be cut by 100 pixels on each side. 
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Default:

get_camera_physical_setup	Retrieves the current settings for distance between the 3 color scan lines (RGB) and the scan direction from the camera.
Syntax	INT16 get_camera_physical_setup (CS_PHYSICAL_SETUP_STRUCT *physicalSetup , bool bUpdate=false)
Parameters:	physicalSetup : pointer to a CS_PHYSICAL_SETUP_STRUCT-structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

Structure to set the trigger settings

```
struct CS_PHYSICAL_SETUP_STRUCT {
        DOUBLE rgb_line_distance;
        UINT16 scan_direction;

        // Initialization of the struct
        CS_PHYSICAL_SETUP_STRUCT (){
                rgb_line_distance = 4.0;
                scan_direction = 0;
        }
} ;
```

Variable type	Element name	Description
DOUBLE	rgb_line_distance;	The allPIXA camera has three individual color lines (RGB). The 3 color lines look at different physical positions in transport direction on the target. By this there is a certain shift between the 3 color planes. The camera is able to correct this color shift internally.  ***rgb_line_distance*** represents the count in pixels in between each color, RED->GREEN and GREEN->BLUE. The camera will compensate this shift internally. Range: 0.0 … 4.0
bool	scan_direction	***scan_direction*** determines the direction in which the color lines are shifted Remark: If a two channel encoder is active, the camera determines the direction out of the angle between the two encoder clocks. Then the flag ***scan_direction*** determines if shift direction is as read from encoder or inverted to this.

set_camera_physical_setup	Configures the current settings for distance between the 3 color scan lines (RGB) and the scan direction from the camera.
Syntax	INT16 set_camera_physical_setup(CS_PHYSICAL_SETUP_STRUCT physicalSetup, bool bSend2Camera)
Parameters:	physicalSetup : structure that contains the settings for physical setup of the camera.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

get_camera_trigger_mode	Retrieves the current trigger mode settings for the camera.
Syntax	INT16 get_camera_trigger_mode(CS_TRIGGER_STRUCT *triggerSettings , bool bUpdate=false)
Parameters:	triggerSettings: pointer to a CS_TRIGGER_STRUCT-structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

Structure to set the trigger settings

```
struct CS_TRIGGER_STRUCT{
        INT16 trigger_mode;
        UINT16 scan_lines_after_stop;
        bool stop_after_max_no_of_scan_lines;
        UINT16 max_no_of_scan_lines;
        UINT16 no_of_suppressed_lines;
        INT16 trigger_input;
        INT16 start_condition;
        INT16 stop_condition;
        INT16 master_slave_operation

        // Initialization of the struct
        CS_TRIGGER_STRUCT(){
                trigger_mode = FREE_RUNNING;
                scan_lines_after_stop = 0;
                stop_after_max_no_of_scan_lines = false;
                max_no_of_scan_lines = 4000;
                no_of_suppressed_lines = 0;
                trigger_input = LIGHT_BARRIER_0;
                start_condition = RAISING_EDGE;
                stop_condition = FALLING_EDGE;
                master_slave_operation = NO_MASTER_SLAVE_OPERATION;
        }

} ;
```

Variable type	Element name	Description
INT16	trigger_mode	Use the enumeration CS_TRIGGER_MODE to select if free running or triggered mode is used.
UINT16	scan_lines_after_stop	Will scan additional scan_lines_after_stop after the stop condition is received. Only used if the camera is using the trigger mode START_STOP_CONDITION
bool	stop_after_max_no_of_scan_lines	If switched on the camera acquires a maximum number of max_no_of_scan_lines in the START_STOP_CONDITION mode
UINT16	max_no_of_scan_lines	This will limit the number of acquired lines per image if the camera is using the trigger mode START_STOP_CONDITION and the parameter stop_after_max_no_of_scan_lines is set.
U INT16	no_of_suppressed_lines	Will suppress every n line **(not allPIXA wave)**
INT16	trigger_input	Sets which trigger input will be used. Use the

		enumeration CS_TRIGGER_INPUT for selecting the correct input.
INT16	start_condition	Selects if the raising or falling edge is relevant for the start condition. Please use the enumeration CS_TRIGGER_CONDITION for the selection
INT16	stop_condition	Selects if the raising or falling edge is relevant for the start condition. Please use the enumeration CS_TRIGGER_CONDITION for the selection
INT16	master_slave_operation	Selects if the camera is used as single camera or is an element of a camera array where the cameras interact with another. These possible settings exist(use enumeration CS_MASTER_SLAVE_OPERATION): NO_MASTER_SLAVE_OPERATION: Camera is operating alone CAMERA_IS_MASTER The camera is the master and controls all other cameras regarding image acquisition, trigger and integration time CAMERA_IS_SLAVE The camera is controlled by another camera SELECT_MASTER_SLAVE_BY_INPUT: The behavior of the camera is selected by an external IO

set_camera_trigger_mode	Configures the camera into free running or triggered mode. The trigger input and the desired mode can be selected here.
Syntax	INT16 set_camera_trigger_mode(CS_TRIGGER_STRUCT triggerSettings, bool bSend2Camera)
Parameters:	triggerSettings: structure that contains all settings for setup the desired trigger mode of the camera. It can be chosen if the trigger signal only controls the start of the image or if it also controls the end of the image acquisition.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

get_global_master_slave_settings	Retrieves the global setting for the master/slave operation
Syntax	INT16 get_global_master_slave_settings (const WORD globalMasterSlaveSetting, bool bSend2Camera)
Parameters:	globalMasterSlaveSetting : Global master slave settings. Parameters are independent of the selected setting. Overrules the locally set master slave behavior.
Return value:	If successful CS_OK will be returned.

	An Error is indicated with values < 0
Comment:	

set_global_master_slave_settings	Configures the global setting for the master/slave operation
Syntax	INT16 set_global_master_slave_settings (WORD* globalMasterSlaveSetting, bool bUpdate)
Parameters:	globalMasterSlaveSetting : Global master slave settings, will be saved permanently and setting independent. Overrules the locally set master slave behavior.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

get_encoder_parameters	Retrieves the currently set encoder parameters
Syntax	INT16 get_encoder_parameters(CS_ENCODER_STRUCT *encoderSetting, bool bUpdate=false)
Parameters:	encoderSetting: pointer to a CS_ENCODER_STRUCT-structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

Structure to set the encoder settings

```
struct CS_ENCODER_STRUCT          {
        bool enable_ecoder;
        DOUBLE um_per_increment;
        DOUBLE vertical_resolution_dpi;
        UINT16 synchronisation_mode;
        UINT16 encoder_channels;
        UINT16 line_trigger_reduction;
        UINT16 averaging;

        CS_ENCODER_STRUCT()      {
                enable_ecoder = false;
                increments_per_um = 10;
                vertical_resolution_dpi = 300.0;
                synchronization_mode = LINE_TRIGGER;
                encoder_channels = ONE_CHANNEL_FULL_STEP;
                averaging = NO_AVERAGING;
        }
} ;
```

If running in encoder mode this formula applies to the calculation of encoder pulses per image line:

$$EncoderPulsesPerLine = \frac{EncoderChannel\left(\frac{Edges}{Step}\right) * 25{,}4\left(\frac{mm}{inch}\right) * 1000}{EncoderResolution\left(\frac{\mu m}{Step}\right) * VerticalImageResolution\left(\frac{Dots}{inch}\right)}$$

Variable type	Element name	Description
bool	enable_ecoder	If set the encoder mode is active, else encode is unused
DOUBLE	um_per_increment	µm per encoder step. This enables you to use e.g. 1.2 encoder tics per image line.
DOUBLE	vertical_resolution_dpi	The desired resolution in transport direction.
UINT16	synchronization_mode	The desired synchronization mode. Use the Enumeration CS_ENCODER_SYNC_MODE. LINE_TRIGGER: Each pulse will generate a line STANDARD_ENCODER: encoder parameter can be set to reach the desired resolution
UINT16	encoder_channels	Parameter is only used if synchronization mode is set to STANDARD_ENCODER
UINT16	line_trigger_reduction	This parameter will provide a divider in whole numbers, regardless in which encoder mode the camera is running.
UINT16	Averaging	Values 0-5 are permissible. Averaging will be set to 2^VALUE, e.g. Value of 3 means 2^3=8 times averaging. 0 = averaging off. Please use the CS_AVERAGING-enumeration.

set_encoder_parameters	Sets all encoder related parameters
Syntax	INT16 set_encoder_parameters(CS_ENCODER_STRUCT encoderSetting, bool bSend2Camera=true)
Parameters:	encoderSetting: A CS_ENCODER_STRUCT-structure which contains elements to switch the encoder mode on or off. The encoder resolution or the encoder mode can be set here. Features like synchronization mode, averaging, line trigger reduction or the type of the connected encoder can be selected.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

get_reference_settings	
Syntax	get_reference_settings(CS_REFERENCE_PARAMETER_STRUCT *referenceParameter, bool bUpdate=true)
Parameters:	referenceParameter: pointer to a CS_REFERENCE_PARAMETER_STRUCT-structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

Structure to set the reference parameters like shading(flat field) or black level compensation

```
struct CS_REFERENCE_PARAMETER_STRUCT{
        bool bUseBlackLevelCorrection;
        bool bUseShadingCorrection;
        UINT16 nBlackLevelReferenceNo;
        UINT16 nShadingReferenceNo;
        // Initialization of the struct
        CS_REFERENCE_PARAMETER_STRUCT(){
                bUseBlackLevelCorrection = true;
                bUseShadingCorrection = true;
                nBlackLevelReferenceNo = 0;
                nShadingReferenceNo = 0;
        }
} ;
```

Variable type	Element name	Description
bool	bUseBlackLevelCorrection	Choose if the clack level correction should be active
bool	bUseShadingCorrection	Choose if the shading correction should be active
UINT16	nBlackLevelReferenceNo	No of the black level reference to apply: set to 0, 1,2, or 3
UINT16	nShadingReferenceNo	No of the shading reference to apply: set to 0, 1,2, or 3

set_reference_settings	Sets the parameters for the references to use.
Syntax	set_reference_settings(CS_REFERENCE_PARAMETER_STRUCT referenceParameter, bool bSend2Camera=true)
Parameters:	referenceParameter: This structure contains the settings if a black level correction and/or a shading(flat field) correction will be applied to the output image. There is also a possibility to select 4 different references for black level and shading correction.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

get_camera_white_balancing_setup	Retrieves the currently set parameters for the white balancing setup
Syntax	get_camera_white_balancing_setup(CS_WHITE_BALANCE_STRUCT *whiteBalanceSetup, bool bUpdate=false)
Parameters:	whiteBalanceSetup: pointer to a CS_WHITE_BALANCE_STRUCT-structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	The white target values are only valid for allPIXA wave cameras

Structure which is used for the configuration of the white reference of the camera

```
struct CS_WHITE_BALANCE_STRUCT{
        bool enable_white_balancing;
        RECT white_reference_area;
        INT16 frames_for_average;
        INT16 white_balancing_mode;
        bool synchronizeWithFrame;
        bool stopGainControl;
        INT16 multi_tap_mode;
        float gain_control_stop_factor;
        INT16 red_target_value;
        INT16 green_target_value;
        INT16 blue_target_value;
        INT16 white_target_value;
        bool showWhiteReferenceBorders;
        CS_WHITE_BALANCE_STRUCT(){
                enable_white_balancing = false;
                white_reference_area.top = 0;
                white_reference_area.left = 0;
                white_reference_area.right = 10;
                white_reference_area.bottom = 10;
                frames_for_average = NO_AVERAGING;
                white_balancing_mode = GAIN_CONTROL_USING_AREA_RANGE;
                synchronizeWithFrame = false;
                stopGainControl = false;
                multi_tap_mode = AUTOMATIC_MASTER_DETECTION;
                gain_control_stop_factor = 1.0;
                red_target_value = 900;
                green_target_value = 900;
                blue_target_value = 900;
                white_target_value = 900;
                showWhiteReferenceBorders = false;
        }
} ;
```

Variable type	Element name	Description
bool	enable_white_balancing	Global switch to turn continuous white balancing on or off
RECT	white_reference_area	Definition where the reference values will be taken from Attention: Area for white reference may not touch the tap border of the sensor. If the area is defined in the front tap, the position is mirrored to the rear tap at the tap border and vice versa.
INT16	frames_for_average	Choose if the reference values should be averaged over a number of frames -> use the CS_AVERAGING enumeration.
INT16	white_balancing_mode	Choose which mode for the white balancing should be selected. Currently only GAIN_CONTROL_USING_AREA_RANGE is supported
bool	synchronizeWithFrame	Select if the camera should do the white balancing only each frame or continuously
bool	stopGainControl	If this parameter is enabled, gain control will stop if (Sum of all current channel values) < gain_control_stop_factor * average of all set point white values. With this parameter it can be avoided that the camera tries to do a white control on dark areas. That would possibvly have an influence on the following areas (gain too high).
float	gain_control_stop_factor	Stops white balancing if stopGainControl is enabled an the described criteria is met
INT16	multi_tap_mode	Defines which tap will be used for white balancing or if this is done independently -> CS_MULTI_TAP_MODE
INT16	red_target_value	Red target value for the white balancing
INT16	green_target_value	Green target value for the white balancing
INT16	blue_target_value	Blue target value for the white balancing
INT16	white_target_value	White target value for the white balancing ➔ Only valid or allPIXA wave cameras
bool	showWhiteReferenceBorders	Display the area where the white balancing reference values are measured inside the image. This will be marked with dark blue lines.

set_camera_white_balancing_setup	Sets the parameters for the white balancing of the camera. Determines if the camera uses the front or rear tap as master, if the taps are controlled independently. The position of the white control area is set. It is also set if the capture of the white control values is synchronized with the start of the image frame. The target values for the white control are also set here.
Syntax	set_camera_white_balancing_setup(CS_WHITE_BALANCE_STRUCT whiteBalanceSetup, bool bSend2Camera=true)
Parameters:	whiteBalanceSetup: A CS_WHITE_BALANCE_STRUCT -structure which contains elements to switch the white balancing on or off. The mode and the working parameters are set here Features like frame synchronization, reference area, tab mode can be set.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

get_current_reference_values	Retrieves the current values of the white control field.
Syntax	INT16 get_current_reference_values (CS_CHANNEL_STRUCT * referenceValues, bool bUpdate)
Parameters:	referenceValues: pointer to a structure where the reference values can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	By checking these values it is possible to check if the camera is in the correct operating range. Values are in the 10bit-range (0-1023)

get_brightness_and_contrast	Returns the currently set parameters for brightness, contrast and gamma correction.
Syntax	INT16 get_brightness_and_contrast (CS_BRIGHTNESS_CONTRAST_STRUCT *brightnessContrast, bool bUpdate=false)
Parameters:	brightnessContrast: pointer to a CS_BRIGHTNESS_CONTRAST_STRUCT -structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

Structure to set the brightness, contrast and gamma correction of the camera

```
struct CS_BRIGHTNESS_CONTRAST_STRUCT{
        bool useBrightnessAndContrast;
        INT16 brightness[MAX_SENSOR_LINES];
        DOUBLE contrast[MAX_SENSOR_LINES];
        DOUBLE gamma_correction;
        // Initialization of the struct
        CS_BRIGHTNESS_CONTRAST_STRUCT() {
                useBrightnessAndContrast = false;
                for(int i = 0; i < MAX_SENSOR_LINES; i++)
                {
                        brightness[i] = 0;
                        contrast[i] = 1.0;
                }
                gamma_correction = 1.0;
        }
} ;
```

Variable type	Element name	Description
Bool	useBrightnessAndContrast	If set to “true” the brightnes and contrast settings will be applied to the sensor output signal.
INT16	brightness[MAX_SENSOR_LINES]	Digital offset to increase/decrease the brightness
DOUBLE	contrast[MAX_SENSOR_LINES]	Digital factor to increase/decrease the contrast in the image
DOUBLE	gamma_correction	Will apply a gamma correction factor to the acquired images

set_brightness_and_contrast	Sets all parameters belonging to brightness, contrast and gamma correction
Syntax	INT16 set_brightness_and_contrast(CS_BRIGHTNESS_CONTRAST_STRUCT brightnessContrast, bool bSend2Camera=true)
Parameters:	brightnessContrast: An additional offset (Brightness) and an additional factor (Contrast) can be added to the video values output by the camera. Also the used value for the gamma correction can be changed in this structure.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	
Supported cameras:	All cameras, allPIXA wave only supports the mirror_image_horizontally flag!

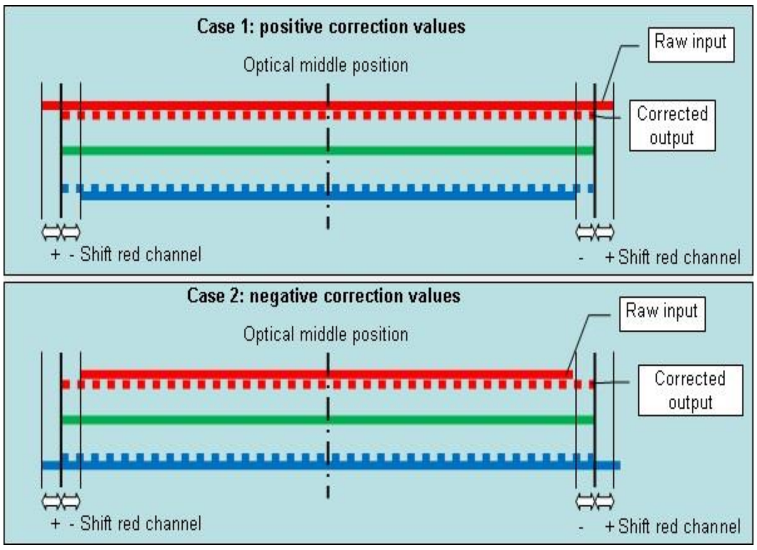
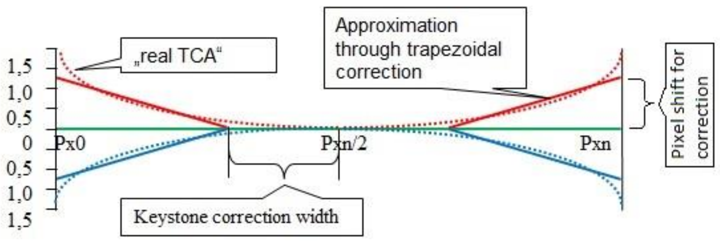
get_image_processing_parameters	Retrieves the currently set parameter for the image processing functions
Syntax	INT16 set_image_processing_parameters (CS_IMAGE_PROCESSING_PARAMETER_STRUCT imageProcessingParameters, bool bSend2Camera=true)
Parameters:	imageProcessingParameters: pointer to a CS_IMAGE_PROCESSING_PARAMETER_STRUCT-structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Default:
Supported cameras:	All cameras, allPIXA wave only supports the “mirror_image_horizontally”-flag!

Structure to set the image processing parameters

```
struct CS_IMAGE_PROCESSING_PARAMETER_STRUCT{
        bool mirror_image_horizontally;
        bool use_color_conversion_matrix;
        bool use_linearisation_table;
        bool use_keystone_correction;
        DOUBLE keystone_pixel_shift;
        INT16 keystone_correction_width;
        INT16 select_color_conversion_matrix;

        // Initialization of the struct
        CS_IMAGE_PROCESSING_PARAMETER_STRUCT() {
                mirror_image_horizontally = false;
                use_color_conversion_matrix = false;
                use_linearisation_table = false;
                use_keystone_correction = false;
                keystone_pixel_shift = 0.0;
```

```
            keystone_correction_width = 1;
            select_color_conversion_matrix = 0;
        }
};
```

Variable type	Element name	Description
bool	mirror_image_horizontally	Mirrors the pixels horizontally in the scan line
bool	use_color_conversion_matrix	Enables the saved color conversion matrix. Can be downloaded to the camera.
bool	use_linearisation_table	Uses a stored linearization table
bool	use_keystone_correction	Enables a keystone correction  Case 1: positive correction values Optical middle position Raw input Corrected output + - Shift red channel - + Shift red channel Case 2: negative correction values Optical middle position Raw input Corrected output + - Shift red channel - + Shift red channel
INT16	keystone_pixel_shift	Sets the amount of the keystone correction. The image above visualizes the impact of the parameter. This parameter treats the red pixel shift. The blue channel will be changed accordingly.
INT16	keystone_correction_width	Possibility to correct only the border area of the image The parameter will set the number of unchanged pixels starting from the middle of the image.  „real TCA“ Approximation through trapezoidal correction Pixel shift for correction 1,5 1,0 0,5 0 0,5 1,0 1,5 Px0 Pxn/2 Pxn Keystone correction width
INT16	select_color_conversion_matrix	Selects the conversion matrix, which is used if color conversion is active.

set_image_processing_parameters	The allPIXA camera contains several image processing functions which can be activated and parameterized here.
Syntax	INT16 set_image_processing_parameters (CS_IMAGE_PROCESSING_PARAMETER_STRUCT imageProcessingParameters, bool bSend2Camera=true)
Parameters:	imageProcessingParameters: structure to activate horizontal mirroring, usage of a color conversion matrix, usage of a linearization table or a keystone correction. The parameters fort the keystone correction can be set also in this structure.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

set_IO_signal_configuration	Sets the assignment of IO functions to certain pins of the camera
Syntax	INT16 set_IO_signal_configuration (CS_IO_CONFIG_STRUCT IOConfiguration bool bSend2Camera=true)
Parameters:	brightnessContrast: structure to set assignment of available IO functionality to pre-defined IO pins of the camera
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

Structure to set the IO configuration of the camera:

```
struct CS_IO_CONFIG_STRUCT {
        UINT16 encoder_config;
        UINT16 encoder_incr0;
        UINT16 encoder_incr1;
        UINT16 select_scan_dir;
        UINT16 light_barrier;
        UINT16 select_master;
        UINT16 master_slave_interface;
        UINT16 general_functions;

        // Initialization of the struct
        CS_IO_CONFIG_STRUCT () {
                encoder_config = ENCODER_ENABLE;
                encoder_incr0 = ENCODER_INCR0_CC1;
                encoder_incr1 = ENCODER_INCR1_CC2;
                select_scan_dir = 0;
                light_barrier = LIGHTBARRIER_CC3;
                select_master = SELECT_MASTER_GPIO_4;
                master_slave_interface = MASTER_SLAVE_IF_X5_4_12;
                general_functions = 0;
        }
} ;
```

Variable type	Element name	Description
UINT16	encoder_config	Parameter for enabling encoder functionality Use the constants defined with "ENCODER_ENABLE" to enable the encoder functionality. ENCODER_INVERT_INPUTS can be set additionally to "ENCODER_ENABLE.."values. Use bitwise OR for combination of the parameters.
UINT16	encoder_icr0	Parameter for setting up encoder input signal 0 resp. line trigger input Use the constants defined with "ENCODER_INCR0" to set the parameter
UINT16	encoder_incr1	Parameter for setting up encoder input signal 1 Use the constants defined with "ENCODER_INCR1" to set the parameter
UINT16	select_scan_dir	Parameter to set up the control of the scan direction via IO-Pins. Use the constants defined with "SELECT_SCAN_DIR_" to set the parameter
UINT16	light_barrier	Parameter for setting up light barrier input Use the constants defined with "LIGHTBARRIER_" to set the parameter
UINT16	select_master	Parameter will determine which input for the selection of the master/slave is used. Use the constants defined with SELECT_MASTER_GPIO_ to set the parameter.
UINT16	master_slave_interface	Parameter will determine which pins are used for transmitting the control data from master to slave camera. Use the constants defined with MASTER_SLAVE_IF_ to set this parameter
UINT16	general_functions	Parameter for general function signals Use the constants defined with "GENERALFUNCTION_" to set the parameter

get_IO_signal_configuration	Returns the currently set parameters for assignment of IO functions to certain pins of the camera
Syntax	INT16 get_IO_signal_configuration (CS_IO_CONFIG_STRUCT *IOConfiguration, bool bUpdate=false)
Parameters:	IOConfiguration : pointer to a CS_IO_CONFIG_STRUCT -structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

store_data_2_camera_memory	Function stores data into permanent flash memory of the camera
Syntax	store_data_2_camera_memory(WORD* const data2Store, unsigned long dataLength)
Parameters:	data2Store : pointer to a memory area where the data to transfer is stored dataLength: Length of the data (number of WORDs)
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

get_data_from_camera_memory	Function retrieves data from permanent flash memory of the camera
Syntax	get_data_from_camera_memory (WORD* const dataRead, unsigned long maxDataLength)
Parameters:	dataRead : pointer to a memory area where the data from the camera memory can be written to. maxDataLength : Maximum length of the data (number of WORDs)
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

get_image_line_data_from_camera	Retrieve the current image data from the camera
Syntax	get_image_line_data_from_camera(char* const imageData, const WORD maxDataLength, CS_IMAGE_DATA_TYPE dataType)
Parameters:	imageData : pointer to a memory area where the image data can be written to. maxDataLength: Maximum length of the data (number of chars) datatype: datatype: select if raw data or an BMP-image should be received. Use the CS_IMAGE_DATA-enum for selection
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Data is in the order BGR

4.2.2 Setting related functions

save_current_setting_2Disk	Saves the current settings of the camera as file to disk
Syntax	INT16 save_current_setting_2Disk(char* filename, INT16 settingFormat=FORMAT_BINARY)
Parameters:	In order to save the camera settings for backup or to transfer them to another camera the settings can be stored to disk. Two different file formats may be selected: binary or xml. The binary format is basically a HSI-command that is saved directly to disk. The xml format can be read by using a xml viewer. Both file formats can be sent to the camera by using the CST or the Chromasens-API.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

burn_current_setting_to_camera	Burns the current setting into non volatile camera memory
Syntax	INT16 burn_current_setting_to_camera(INT16 settingNo=-1)
Parameters:	settingNo: If it is desired to store the current setting into another setting number set this parameter to the desired setting. Values from 1 to 15 are valid.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

get_current_setting_number	Retrieves the currently active setting from the camera
Syntax	get_current_setting_number(void)
Parameters:	None
Return value:	If successful a number from 1 to 19 will be returned. An Error is indicated with values < 0
Comment:	

get_available_settings	Returns the available settings
Syntax	INT64 get_available_settings(void)
Parameters:	None
Return value:	If successful the coded value for the available settings will be returned. An Error is indicated with values < 0
Comment:	The returned value will code the available settings bitwise. That means that if for instance settings 1 and 3 are available, a value of 0x0A is returned -> 0000 1010. The bits 1 and 3 are set.

select_active_setting	Selects the setting to use
Syntax	INT16 select_active_setting(INT16 settingNo)
Parameters:	settingNo: the setting to load into active memory Values from 1 to 15 are valid.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Please mind that only available settings can be selected. Before selecting a setting you should check with the function "get_available_settings " if the setting exists.

set_setting_description	Set a comment for the current setting. 128 chars are available for the comment.
Syntax	INT16 set_setting_description(char* description, bool bSend2Camera =true)
Parameters:	description: pointer to a character array where the description is stored
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	For each setting a textual description exists that describes the purpose of it. This might help the user to differentiate the use of the setting. It is also displayed in the CST tool.

get_setting_description	Returns the currently stored setting description
Syntax	INT16 get_setting_description(char* description, bool bUpdate =true)
Parameters:	description: Pointer to a character array with the length MAX_SETTING_COMMENT_LENGTH to store the setting comment to
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

4.2.3 Special functions

set_camLink_properties	Sets the properties for the cameraLink data transmission
Syntax	INT16 set_camLink_properties(CS_CAMLINK_PROPERTIES camLinkProperties, bool bSend2Camera=true)
Parameters:	camLinkProperties: contains the mode and the speed to use for the transmission. The camera link mode can be selected between these modes: CL_BASE_MODE: usage of one cameraLink connector (CL_MEDIUM_MODE): usage of two cameraLink connectors. The camera link speed can be selected between standard (CL_STANDARD) and high speed (CL_HIGH_SPEED). Changing this parameter might help to reduce transmission problems when using longer cameraLink cables. Using CL_Standard will lower the pixel clock from 85MHz to 72.5MHz. This can be done without restrictions for all cameras with a 4K sensor or smaller. When using a 7K camera the integration time must be set accordingly.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	When using an allPIXA wave camera the parameter camLinkSpeed is no longer used! The camera itself will choose the best possible transmission parameters!

Structure for the CameraLink settings

```
struct CS_CAMLINK_PROPERTIES{
        CS_CAMLINK_MODE camLinkMode;
        CS_CAMLINK_SPEED camLinkSpeed;

        CS_CAMLINK_PROPERTIES(){
                camLinkMode = CL_MEDIUM_MODE;
                camLinkSpeed = CL_HIGH_SPEED;
        }
};
```

Variable type	Element name	Description
camLinkMode	camLinkMode	Use the enumeration “CS_CAMLINK_MODE” to set this parameter. This will select if the CameraLink “BASE” or “MEDIUM” is used. Remember to check if the integration time is in the correct range to set this parameter to BASE mode.
camLinkSpeed	camLinkSpeed	Use the enumeration “CS_CAMLINK_SPEED” for setting this variable. If set to CL_STANDARD a slower pixel clock will be used for the data transmission. This might avoid problems when using longer cables. If using a 7K camera please check if the integration time is big enough to use this setting. **This parameter is obsolete when using an allPIXA wave camera!**

get_camLink_properties	Returns the currently set cameraLink properties
Syntax	INT16 get_camLink_properties(CS_CAMLINK_PROPERTIES *camLinkProperties, bool bUpdate=true)
Parameters:	camLinkProperties: pointer to a CS_CAMLINK_PROPERTIES-structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Default:

set_video_output_parameters	Sets the video output mode to color or grey of the camera.
Syntax	INT16 set_video_output_parameters(CS_VIDEO_OUTPUT_STRUCT videoOutParams, bool Send2Camera=true)
Parameters:	videoOutParams: This structure contains all elements to set the output mode. The video_out_mode should be set by using the enum CS_VIDEO_OUTPUT_MODE. If a grey mode is selected the conversion from RGB to grey can be controlled by the color weights. For the color mode the color channels might be swapped by using the variable swap_color_channels_red_blue. The camera provides the possibility to insert information into the image. By setting the insert_mode variable with a bitwise conjunction of the INSERT_ constants like INSERT_IMAGE_COUNT_VALUE. This information will be written into the image.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

Structure to set the video output parameters

```
struct CS_VIDEO_OUTPUT_STRUCT{
        INT16 video_output_mode;
        float color_weight_red;
        float color_weight_green;
        float color_weight_blue;
        bool swap_color_channels_red_blue;
        INT16 insert_mode;
        INT16 position_eachline_data;

        // Initialization of the struct
        CS_VIDEO_OUTPUT_STRUCT(){
                video_output_mode = _3x8_BIT_COLOR_PARALLEL;
                color_weight_red = 1.0;
                color_weight_green = 1.0;
                color_weight_blue = 1.0;
                swap_color_channels_red_blue = false;
                insert_mode = 0;
                position_eachline_data = 0;
        }
} ;
```

Variable type	Element name	Description
INT16	video_output_mode	This defines the output format of the camera. Please use the "CS_VIDEO_OUTPUT_MODE"-structure to set the desired output format. The different settings for the grey modes are defined according to the CameraLink Standard **Please note, that the allPIXA wave does use different settings and enums than the allPIXA and allPIXA pro cameras, please see the description of the `CS_VIDEO_OUTPUT_MODE`-enum**
Float	color_weight_red	The grey modes do not use only one channel, the grey value is composed out of all three color channels. You can set the factor by yourself how much of each color should be used for the generation of the grey value. **Not used in the allPIXA wave camera!**
Float	color_weight_green	See above
Float	color_weight_blue	See above
Bool	swap_color_channels_red_blue	Internally swaps the red and the blue channel. Might be useful if the direction of the object to acquire has changed.
INT16	insert_mode	Use the constants defined with "INSERT_" to bitwise set the information you want to show in the image
INT16	position_eachline_data	Determines position of "Each Line Info" 0: start of line 1: end of line 2: start and end of line

get_video_output_parameters	Retrieves the currently set video output parameters
Syntax	INT16 get_video_output_parameters(CS_VIDEO_OUTPUT_STRUCT *videoOutParams, bool bUpdate=false)
Parameters:	videoOutParams: pointer to a CS_VIDEO_OUTPUT_STRUCT-structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

set_test_pattern_settings	Output artificially generated test patterns
Syntax	set_test_pattern_settings(CS_TEST_PATTERN_STRUCT testPatternSetting, bool bSend2Camera=true)
Parameters:	testPatternSetting: structure in which the desired pattern can be set. To select the desired test pattern use the enum CS_TEST_PATTERN. The member nTestPatternLevel is used only when the test pattern is set to INPUT_RAMP_ON_GREEN_CHANNEL. This will set the channels red and blue to the entered value.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

Structure for the test pattern settings

```
struct CS_TEST_PATTERN_STRUCT{
        UINT16 nTestPattern;
        UINT16 nTestPatternLevel;
        CS_TEST_PATTERN_STRUCT(){
                nTestPattern = TEST_PATTERN_OFF;
                nTestPatternLevel = 0;
        }
};
```

Variable type	Element name	Description
UINT16	nTestPattern	Use the "CS_TEST_PATTERN"-enumeration to set the desired test pattern. Following patterns are possible: TEST_PATTERN_OFF: Live image data will be output GREY_RAMP_IN_CCD_DIR: Horizontal grey ramp increasing by one grey level per pixel GREY_RAMP_IN_TRANSPORT_DIR: Vertical grey ramp increasing by one grey level per pixel INPUT_RAMP_ON_GREEN_CHANNEL: Ramp in the green channel and a fixed value in the two other channels (nTestPatternLevel) SEQUENCE_OF_PATTERNS: Alternating sequence of the first three patterns and a live image. CHANGE_VIDEO_LEVEL_AT_EVERY_PIXEL: Video_Levels are changed in vertical and horizontal direction.
UINT16	nTestPatternLevel	Used when test pattern INPUT_RAMP_ON_GREEN_CHANNEL or the sequence of the test pattern is set for the output of a fixed value in the red and blue channels. The value will be according to the 10bit space 0..1023

get_test_pattern_settings	Retrieves the currently set test pattern settings
Syntax	get_test_pattern_settings(CS_TEST_PATTERN_STRUCT *testPatternSetting, bool bUpdate=false)
Parameters:	testPatternSetting: pointer to a CS_TEST_PATTERN_STRUCT-structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

get_camera_information	Retrieves information about the currently opened camera.
Syntax	INT16 get_camera_information(CS_CAMERA_INFORMATION_STRUCT *cameraInformation)
Parameters:	cameraInformation: pointer to a CS_CAMERA_INFORMATION_STRUCT-structure that contains a textual and numerical description of some basic camera information..
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

get_camera_information	Retrieves information about a camera discovered by autodetect_camera().
Syntax	INT16 get_camera_information(unsigned int index, CS_CAMERA_INFORMATION_STRUCT *cameraInformation)
Parameters:	index: camera index between 0 and [camera-count –1] autodetect_camera() returns. cameraInformation: pointer to a CS_CAMERA_INFORMATION_STRUCT-structure that contains a textual and numerical description of some basic camera information.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	autodetect_cameras() must be executed before.

Structure for the camera informations

This structure will be filled by the function “get_camera_information”

```
struct CS_CAMERA_INFORMATION_STRUCT{
        static const int MAX_CAMINFO_TEXT_LENGTH = 42;
        static const int MAX_CAM_SERIAL_TEXT_LENGTH = 12;

        INT16 firmwareVersion[3];
        INT16 fpgaVersion;
        char firmwareDescription[MAX_CAMINFO_TEXT_LENGTH];
        char fpgaDescription[MAX_CAMINFO_TEXT_LENGTH];
        char cameraSerialNumber[MAX_CAM_SERIAL_TEXT_LENGTH];
```

```
          INT16  packetVersion;
          char  packetDescription[MAX_CAMINFO_TEXT_LENGTH];
          bool  packetConsistent;
          char fullInterfaceDescription[MAX_CAMINFO_TEXT_LENGTH];
          char productType[MAX_CAMINFO_TEXT_LENGTH];
          int productTypeId;
          int productSerial;
          int serialPortNo;
          int camLinkPortNo;
          char ethAddress[MAX_IP_CHARS];
          int ethPort;

          // Constructor of camera information struct
          CS_CAMERA_INFORMATION_STRUCT()
          {
                    firmwareVersion[0] =     0x00;
                    firmwareVersion[1] =     0x00;
                    firmwareVersion[2] =     0x00;
                    fpgaVersion =            0x00;
                    packetVersion = 0x00;
                    packetConsistent = true;
                    memset(packetDescription, 0, MAX_CAMINFO_TEXT_LENGTH);
                    memset(firmwareDescription, 0, MAX_CAMINFO_TEXT_LENGTH);
                    memset(fpgaDescription, 0, MAX_CAMINFO_TEXT_LENGTH);
                    memset(cameraSerialNumber, 0, MAX_CAM_SERIAL_TEXT_LENGTH);
                    memset(fullInterfaceDescription, 0, MAX_CAMINFO_TEXT_LENGTH);
                    memset(productType, 0, MAX_CAMINFO_TEXT_LENGTH);
                    productSerial = -1;
                    serialPortNo = -1;
                    camLinkPortNo = -1;
                    productTypeId = -1;
                    memset(ethAddress, 0, MAX_IP_CHARS);
                    ethPort=-1;
          };
};
```

Variable type	Element name	Description
INT16 [3]	firmwareVersion	Version of the currently running microprocessor software
INT16	fpgaVersion	Version of the currently running FPGA build
char	firmwareDescription	Short textual description of the microprocessor software
char	fpgaDescription	Short textual description of the FPGA software
char	cameraSerialNumber	Serial number of the camera as text
INT16	packetVersion	Version of the camera software packet
char	packetDescription	Detailed description of the camera software packet
bool	packetConsistent	Indicates if the packet is consistent
char	fullInterfaceDescription	Textual description of the interface the camera is connected to.
char	productType	Product / CP-number
int	productTypeId	Serial number, first part (product dependent)

int	productSerial	Serial number, second part
int	serialPortNo	Serial port if the camera is connected via RS232, otherwise -1
int	camLinkPortNo	CameraLink port if the camera is connected via CL, otherwise -1
char	ethAddress	IP-address if the camera is connected via ethernet.

get_camera_operating_values	Retrieves information about the camera operating values like current gain values, voltages and temperatures.
Syntax	INT16 get_camera_operating_values (CS_CAMERA_OPERATING_VALUES_STRUCT *operatingValues)
Parameters:	operatingValues: pointer to a CS_CAMERA_OPERATING_VALUES_STRUCT-structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

int	ethPort	Port number if the camera is connected via ethernet.

Structure for the current operating values of the camera.

This structure will be filled when calling the function "get_camera_operating_values"

```
struct CS_CAMERA_OPERATING_VALUES_STRUCT{
        static const int MAX_INTERNAL_VOLTAGES = 6;

        // Inputs
        static const int LB_0                   = 0x01;
        static const int LB_1                   = 0x02;
        static const int LB_2                   = 0x04;
        static const int LB_3                   = 0x08;

        INT32 inputStatus;      // Light barrier state
        INT32 gainCtrlDisableState;
        INT32 imageCount;
        INT32 nCurrentWhiteRef[MAX_CHANNELS];
        INT32 nCurrentGain[MAX_CHANNELS];
        INT32 cameraState;
        INT32 ledState;
        double currentTansportSpeed;

        INT32 internalVoltage[MAX_INTERNAL_VOLTAGES];

        INT32 voltageLed[5];
        INT32 inputVoltage;
        INT16 temperatureBoard;
        INT16 temperatureLEDController;
        INT16 temperatureSensor;
        INT16 temperatureLed;
        INT16 masterSlaveMode;        // Constructor of operating values struct
        CS_CAMERA_OPERATING_VALUES_STRUCT()
        {
                inputStatus = 0;
                camState = 0;
                gainCtrlDisableState = -1;
                imageCount = 0;
                cameraState = 0;
                ledState = 0;
                currentTransportSpeed = 0.0;
                inputVoltage = 0;
                temperatureBoard = 0;
                temperatureLed = 0;
                temperatureLEDController = 0;
                temperatureSensor = 0;

                masterSlaveMode = NO_MASTER_SLAVE_OPERATION;
                // Only the first three states are possible!
                // This entry will show how the camera is currently operating
                // NO_MASTER_SLAVE_OPERATION
                // CAMERA_IS_MASTER
                // CAMERA_IS_SLAVE

                for(INT32 i = 0; i < MAX_CHANNELS; i++)
                {
                        nCurrentWhiteRef[i] = 0;
                        nCurrentGain[i] = 0;
                }
                for(INT32 i = 0; i<5;i++)
                {
                        voltageLed[i] = 0;
                }
        };
};
```

Variable type	Element name	Description
INT32	inputState	The current state of the inputs will be set bitwise in this variable. Please use the constants LB_0 to LIGHT_BARRIER_3 to decode the states.
INT32	gainCtrlDisableState	There are some conditions when the gain control will be switched of automatically by the camera. The enumeration CS_GAIN_CONTROL_DISABLED will provide the reasons.
INT32	imageCount	The number of image frames acquired by the camera since startup
INT32	nCurrentWhiteRef[MAX_CHANNELS]	The current levels of the white reference area Please note that the allPIXA wave camera will only use 4 entries (red, green, blue and white)
INT32	nCurrentGain[MAX_CHANNELS]	The currently set gain values Please note that the allPIXA wave

		camera will only use 4 entries (red, green, blue and white)
INT32	cameraState	Use the enumeration CS_DEVICE_STATE to evaluate the state of the camera

INT32	ledState	Use the enumeration CS_DEVICE_STATE to evaluate the state of possible attached LED modules
double	currentTransportSpeed	If the camera is set into encoder/line trigger mode the measured speed will be displayed here. If the speed is too high or too low, the value will be `SPEED_2_HIGH` or `SPEED_2_SLOW`
INT32	internalVoltage[MAX_INTERNAL_VOLTAGES]	This contains the level of different internal voltages. This may differ from camera model to camera model. allPIXA wave: Only 5 internal Voltages are used (0-4)
INT32	voltageLed[5]	If a LED module is connected, the voltages can be monitored with this variable
INT32	inputVoltage	External supply voltage to the camera
INT32	temperatureBoard	Temperature of the camera board
INT32	temperatureLEDController	If attached: temperature of the LED controller
INT32	temperatureSensor	Temperature of the images sensor
INT32	temperatureLed	If attached: Temperature of the LED
INT16	masterSlaveMode	Shows the current master/slave mode of the camera. Use the CS_MASTER_SLAVE_OPERATION-struct to determine which state the camera is currently in. Please note that the state SELECT_MASTER_SLAVE_BY_INPUT Will not be a valid state.

set_led_flash_parameters	Sets all parameters needed to enable or disable the LED flash function
Syntax	INT16 set_led_flash_parameters(const CS_LED_FLASH_CONTROL_STRUCT& ledFlashParameters, bool bSend2Camera)
Parameters:	ledFlashParameters : reference to a CS_LED_FLASH_CONTROL_STRUCT -structure which contains the information to be written to the camera.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	This function can be used to setup the synchronized triggering of illuminations connected to the camera.
Supported cameras:	allPIXA pro, allPIXA wave

```
struct CS_LED_FLASH_CONTROL_STRUCT{
        static const int MAX_LED_FLASH_PATTERNS = 4;
        static const int MAX_OUTPUTS = 4;

        UINT16 enableFlashControl;
        UINT16 numberOfLinePatterns;
        double flashSequenceTime;
        double flashDefinitions[MAX_LED_FLASH_PATTERNS][MAX_OUTPUTS];
        UINT16 ledDriverSync;

        // Constructor LED control struct
        CS_LED_FLASH_CONTROL_STRUCT()
        {
                enableFlashControl = FLASH_DISABLED;
                numberOfLinePatterns = 1;
                flashSequenceTime = 1.0;
                ledDriverSync = NO_LED_DRIVER_SYNC;
                for(int i = 0; i < MAX_LED_FLASH_PATTERNS; i++)
                {
                        for(int outputs = 0; outputs < MAX_OUTPUTS; outputs++)
                        {
                                flashDefinitions[i][outputs] = 0.0;
                        }
                }
        };
};
```

Variable type	Element name	Description
UINT16	enableFlashControl	Outputs only get triggered when this flag is set to enabled
UINT16	numberOfLinePatterns	Number of different flash scenarios
double	flashSequenceTime	time in us for the whole flash sequence in camera free running mode in encoder / line trigger mode the frequency of repeating the given pattern sequence is determined by the period of the encode / line trigger clocks
double	flashDefinitions	Definitions of the duration which the outputs are switched on
UINT16	ledDriverSync	If enabled the camera provides a synchronization clock of 885 kHz **Not available when using allPIXA wave!**

get_led_flash_parameters	Retrieve all parameters related to the LED flash function
Syntax	INT16 set_led_flash_parameters(const CS_LED_FLASH_CONTROL_STRUCT* ledFlashParameters, bool bUpdate)
Parameters:	ledFlashParameters : pointer to a CS_LED_FLASH_CONTROL_STRUCT -structure where the information can be stored to.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	This function can be used to setup the synchronized triggering of illuminations connected to the camera.
Supported cameras:	allPIXA pro

4.2.4 allPIXA wave functions

The following function must only be used with allPIXA wave cameras. They will throw an error when used with allPIXA or allPIXA pro cameras

get_linear_gain_values	Retrieve the gain values from the camera
Syntax	INT16 get_gain_values(CS_LINEAR_GAIN_STRUCT * linearGainValues, bool bUpdate)
Parameters:	gainValues: pointer to a structure with an array of the linear gain values to set and the number of valid entries. To access single entries use the constants like RED, , BLUE or similar.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Values are in the 10bit-range (0-1023)
Supported cameras:	allPIXA wave

Structure "**CS_LINEAR_GAIN_STRUCT**" is used to get and set values which relate to the setting of the different linear gain values of the camera.

When reading the values from the camera the "no_of_valid_entries" field will be set to indicate the valid number of linear gain values for this camera.

To distinguish which color belongs to which channel, please use the constants RED, GREEN, BLUE and WHITE.

```
struct CS_LINEAR_GAIN_STRUCT        {
        UINT16 no_of_valid_entries;
        UINT16 value[MAX_SENSOR_LINES];
        CS_LINEAR_GAIN_STRUCT ()
        {
                no_of_valid_entries = 0;
                for(int i = 0; i < MAX_SENSOR_LINES; i++)
                        value[i] = 0;
        }
} ;
```

Structure for handling camera linear gain values:

Variable type	Element name	Description
UINT16	no_of_valid_entries	Number of valid linear gain channels
DOUBLE	value[MAX_SENSOR_LINES]	Linear gain values

set_linear_gain_values	Sets the gain values for the camera.
Syntax	INT16 set_linear_gain_values(LINEAR_GAIN_STRUCT linearGainValues, bool bSend2Camera);
Parameters:	gainValues: structure with an array of the linear gain values to set and the number of valid entries. To access single entries use the constants like RED, BLUE or similar.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Values are in the 10bit-range (0-1023)
Supported cameras:	allPIXA wave

get_sensor_sensitivity_values	Retrieve the sensitivity values from the camera
Syntax	INT16 get_sensor_sensitivity_values(CS_SENSOR_SENSITIVITY_STRUCT * sensitivityValues, bool bUpdate)
Parameters:	sensitivityValues : pointer to a structure with cds gain and sensor sensitivity settings
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	
Supported cameras:	allPIXA wave

Structure “CS_SENSOR_SENSITIVITY_STRUCT“ is used to get and set values which relate to the setting of the sensor sensitivity such as cds_gain or the sensitivity(fullwell capacity).

When reading the values from the camera the "no_of_valid_entries" field will be set to indicate the valid number of sensitivity values for this camera.

To distinguish which color belongs to which channel, please use the constants RED, GREEN, BLUE and WHITE.
In order to figure out the meaning of the CDS-gain or the sensitivity use the enums CS_SENSITIVITY or CS_CDS_GAIN

Structure for handling sensor sensitivity values:
```
struct CS_SENSOR_SENSITIVITY_STRUCT    {
        UINT16 no_of_valid_entries;
        UINT16 cds_gain[MAX_SENSOR_LINES];
        UINT16 sensitivity[MAX_SENSOR_LINES];
        CS_SENSOR_SENSITIVITY_STRUCT ()
        {
                no_of_valid_entries = 0;
                for(int i = 0; i < MAX_SENSOR_LINES; i++)
                {
                        value[i] = 0;
                        sensitivity[i] = HIGH_DYNAMIC;
                        cds_gain[i] = CDS_X1;
                }
        }
} ;
```

Variable type	Element name	Description
UINT16	no_of_valid_entries	Number of valid linear gain channels
UINT16	cds_gain[MAX_SENSOR_LINES]	CDS gain values for the single color channels
UINT16	sensitivity[MAX_SENSOR_LINES]	Sensor sensitivity values for the single color channels (fullwell capacity)

set_sensor_sensitivity_values	Set the sensor sensitivity values to the camera
Syntax	INT16 set_sensor_sensitivity_values(CS_SENSOR_SENSITIVITY_STRUCT & sensitivityValues, bool bSend2Camera=true)
Parameters:	sensitivityValues : pointer to a structure with the cds gain and sensor sensitivities to set
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	
Supported cameras:	allPIXA wave

get_roi_parameters	Retrieve the ROI settings from the camera
Syntax:	INT16 get_roi_parameters(CS_ROI_PARAMETER_STRUCT * roiParameters, bool bUpdate=true)
Parameters:	roiParameters : pointer to a structure where the roi parameters can be stored to
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	
Supported cameras:	allPIXA wave

Structure “CS_SENSOR_SENSITIVITY_STRUCT“ is used to get and set values which relate to the setting of the sensor sensitivity such as cds_gain or the sensitivity(fullwell capacity).

```
struct CS_ROI_PARAMETER_STRUCT {
        bool roi_active[MAX_ROIS];
        UINT32 roi_start[MAX_ROIS];
        UINT32 roi_width[MAX_ROIS];
        UINT32 img_height;
        UINT16 first_scan_line;
        CS_ROI_PARAMETER_STRUCT ()
        {
                for(int i = 0; i < MAX_ROIS; i++)
                {
                        roi_active[i] = false;
                        roi_start[i] = 500;
                        roi_width[i] = 500;
                }
                Img_height = 500;
                First_scan_line_delay = 0;
                }
} ;
```

Structure for handling the ROI parameters:

Variable type	Element name	Description
bool	roi_active[MAX_ROIS]	Indicates if this ROI is currently used
UINT32	roi_start[MAX_ROIS]	The start ot the ROI
UINT32	roi_width[MAX_ROIS]	The width of the ROI
UINT32	Img_height	Number of sensor lines used in y direction(same size or all ROIs is used)
UINT16	first_scan_line_delay	Delay in scan lines starting from the image trigger (Only active if camera is in triggered mode)

set_roi_parameters	Set the roi parameters of the camera
Syntax:	INT16 (CS_ROI_PARAMETER_STRUCT & roiParameters, bool bSend2Camera=true)
Parameters:	roiParameters : pointer to a structure with the roi parameters to set
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	The total ROI size must not exceed the sensor length! The chosen ROIs may overlap
Supported cameras:	allPIXA wave

```
struct CS_XLC_PARAMETER_STRUCT {
        WORD current;
        WORD channelSelection4Current;
        WORD channelActive;
        DWORD serialNumber; // Only valid for get function! It is not possible to set the serial number!

        // Initialization of the struct
        CS_XLC_PARAMETER_STRUCT () {
                current = 0;
                channelSelection4Current = 0;
                channelActive =0;
                serialNumber = 0;
        }
} ;
```

Structure for handling the XLC illumination parameters:

Variable type	Element name	Description
WORD	current	Current for the selected XLC channels
WORD	channelSelection4Current	The current will be set for the activated channels (use a bitwise or by applying the enum `CS_XLC4_CHANNELS`)
WORD	channelActive	The bit for the active channels is set here (use a bitwise or by applying the enum `CS_XLC4_CHANNELS`)
DWORD	serialNumber	Only used when getting parameters. Contains the serial number of the requested XLC controller

set_xlc_parameters	Set the xlc illumination parameters of the connected XLC device
Syntax	INT16 set_xlc_parameters(WORD xlcID, CS_XLC_PARAMETER_STRUCT &xlcParameters, bool bSend2Camera=true);
Parameters:	xlcID : The ID of the XLC4-controller.Use the enum `CS_XLC4_IDS` to choose which XLC4 to control. Choose “`XLC_BROADCAST`” to broadcast the set parameters to all connected XLC controllers xlcParameters: struct which contains the parameter to set. Please note that the serial number contained in this structure will be ignored by this function.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	The total ROI size must not exceed the sensor length! The chosen ROIs may overlap
Supported cameras:	allPIXA wave

get_xlc_parameters	Get the xlc illumination parameters of the connected XLC device
Syntax	INT16 get_xlc_parameters(WORD xlcID, CS_XLC_PARAMETER_STRUCT *xlcParameters, bool bUpdate=false);
Parameters:	xlcID : The ID of the XLC4-controller.Use the enum `CS_XLC4_IDS` to choose which XLC4 to control. Choose "`XLC_BROADCAST`" to broadcast the set parameters to all connected XLC controllers xlcParameters: struct which contains the parameter to set. Please note that the serial number contained in this structure will be ignored by this function.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	The total ROI size must not exceed the sensor length! The chosen ROIs may overlap
Supported cameras:	allPIXA wave

```
struct CS_INTERNAL_LIGHT_BARRIER_STRUCT{
        INT16 roi_start;
        INT16 roi_length;
        INT16 color_selection_for_edge_detection;
        INT16 mark_roi_in_image;
        INT16 rising_edge_level;
        INT16 falling_edge_level;

        // Initialization of the struct
        CS_INTERNAL_LIGHT_BARRIER_STRUCT(){
                roi_start = 500;
                roi_length = INTERNAL_LIGHT_BARRIER_ROI_LENGTH_32P;
                color_selection_for_edge_detection = INTERNAL_LIGHT_BARRIER_ALL_COLORS;
                mark_roi_in_image = 0;
                rising_edge_level = 100;  // Value in the 8bit range
                falling_edge_level = 200; // Value in the 8bit range
        }
} ;
```

Structure for handling the internal light barrier:

Variable type	Element name	Description
INT16	roi_start	Start of the ROI for determining the edge
INT16	roi_length	Length of the ROI
INT16	color_selection_for_edge_detection	It is possible to determine the edge in a single or in all color channels
INT16	mark_roi_in_image	ROI will be displayed in the image
INT16	rising_edge_level	Minimum value to detect a rising edge
INT16	falling_edge_level	Minimum value to detect a falling edge

set_internal_light_barrier_settings	Set parameters for the internal light barrier
Syntax	INT16 set_internal_light_barrier_settings(CS_INTERNAL_LIGHT_BARRIER_STRUCT& internalLightBarrierSettings, bool bSend2Camera=true)
Parameters:	internalLightBarrierSettings : Reference to a structure where the parameters to set are stored.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	The internal light barrier will determine the frame trigger out of the image content. By setting up these parameters, the camera will determine the average value in the set ROI and trigger the frame trigger as soon as the set criteria are met. It is possible to evaluate only one color channel or all channels. Use the enum "`CS_INTERNAL_LB_COLOR_SELECT`" To set up the length only predefined values are possible. Please use the enum "`CS_INTERNAL_LB_ROI_LENGTH`"

get_internal_light_barrier_settings	Get parameters for the internal light barrier
Syntax	INT16 get_internal_light_barrier_settings(CS_INTERNAL_LIGHT_BARRIER_STRUCT *internalLightBarrierSettings, bool bUpdate=false)
Parameters:	internalLightBarrierSettings : Pointer to a light barrier structure where the current settings of the internal light barrier will be stored.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

4.3 Camera status and type

determineCameraType	Get number of interfaces in the system
Syntax	INT16 determineCameraType(bool bUdate = true)
Parameters:	bUpdate: if set to true, the information will be requested from the camera, otherwise the information will be taken from memory.
Return value:	Returns the type of the connected camera. To check the type of the camera use the `CS_CAMERA_TYPE-enum.`
Comment:	You can use the camera typ to decide which functions to use with the camera. In between allPIXA, allPIXA pro and the allPIXA wave significant differences exist. See the parameter "Supported cameras"

get_error_text	Returns the textual description for a given error number
Syntax	INT16 get_error_text(INT16 nErrorNo, char** pErrorText)
Parameters:	nErrorNo: The error number which corresponds to the returned error text pErrorText: Address of a pointer. The pointer will point to the null terminated error string after successful completion of the function.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

get_camera_state	Returns the current state of the camera
Syntax	INT16 get_camera_state (bool bGetLastState=false, char** pErrorText=NULL)
Parameters:	bGetLastState: If this flag is set the current state will be requested from the camera. If this is not set the state stored during the last calls will be returned. pErrorText: Address of a pointer. The pointer will point to the null terminated error string after successful completion of the function.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	If bGetLastState-flag is set a RS-Request will be sent to the camera. Therefor the current state of the camera will be returned. If an error is present, this will be returned and cleared. The next call will return an OK-state until another error condition will occur.

4.4 Camera functions

do_tap_balancing	The camera automatically sets the internal parameters to straighten the differences in between the two taps.
Syntax	INT16 do_tap_balancing(bool bSaveWhiteCalibParams=true)
Parameters:	bSaveWhiteCalibParams : Save all parameters related to the white/tap balancing permanently after success.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	This function only needs to be called if something in the basic setup has been changed. Otherwise the camera should already have the correct parameters to eliminate any differences in between the two taps.

perform_white_balancing	Performs a white balancing procedure. The camera will alter the gain values in order that the measured white reference values meet the set target values. The function will check if the gain values will stay in a narrow range after meeting the target white reference values. If the values will change too much an error will be returned indicating an unstable operating condition (presumable the lighting is not stable enough).
Syntax	INT16 perform_white_balancing (bool bSaveWhiteCalibParams=true)
Parameters:	bSaveWhiteCalibParams : Save all parameters related to the white balancing permanently after success.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Make sure that a valid white reference is present in the set field for the white balancing. Otherwise the function will fail.

reset_camera	Resets the camera back into start up state. The microcontroller and the FPGA are reset. This function is needed e.g. after a program shutdown.
Syntax	INT16 reset_camera (void)
Parameters:	None
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

prepare_cam_4reference	Sets the camera into a mode where an image for the generation of a black level or a shading(flat field) reference can be captured.
Syntax	INT16 prepare_cam_4reference(REFERENCE_MODE referenceMode)
Parameters:	referenceMode: set this parameter to the desired reference: BLACK_LEVEL_CORRECTION or SHADING_CORRECTION.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	After the function returns successfully a reference image can be acquired. All necessary settings are done automatically. Remember to restore the regular operating settings after finishing the acquisition. This can be achieved by loading the last active setting or resetting the camera.

4.5 Maintenance functions

send_hsi_file_2_camera	Download a his file to the camera
Syntax	INT16 send_hsi_file_2_camera(char* fileName)
Parameters:	fileName: Pointer to the complete filepath to download.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	All kind of HSI-files can be downloaded to the camera. This includes new FPGA-files, settings or simple HSI-commands. Also .set-files from the CST tool can be downloaded by this function.

get_trace_mode	Gets internal trace functions that can be retrieved by calling "get_trace_from_camera"
Syntax	INT16 get_trace_mode(UINT16 traceOptions, bool bUpdate=false)
Parameters:	traceOptions: bit field where different options can be set. The set options can be decoded bitwise by using the constants defined with the leading TRACE_ like TRACE_GENERAL_DEBUG_INFO
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

set_trace_mode	Sets internal trace functions that can be retrieved by calling "get_trace_from_camera"
Syntax	INT16 set_trace_mode(UINT16 traceOptions, bSend2Camera=true)
Parameters:	traceOptions: bit field where different options can be set. Use a bitwise conjunction with the constants defined with the leading TRACE_ like TRACE_GENERAL_DEBUG_INFO
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	Parameter traceOption will be stored non-volatile in the camera. Even after reset or switch off the camera the traceOption parameter will remain valid.

get_trace_from_camera	Retrieve the last trace messages from the camera.
Syntax	INT16 get_trace_from_camera(char** pTraceData, INT16& traceLength)
Parameters:	pTraceData: Address to a Pointer. This pointer will point to a character buffer which contains the camera trace. The length of the buffer will be returned through the variable traceLength. Simply copy the buffer into local memory. The single entries are divided by a CR.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

4.6 Convenience functions

calculate_reference	Calculates the reference data for black level or shading (flat field) correction out of the provided image.
Syntax	INT16 calculate_reference(CS_CALCULATE_REFERENCE_STRUCT calcRefParams)
Parameters:	calcReferenceParams: this structure contains all information needed to calculate the reference set for the given image (also set in the parameter structure). The user can set the parameters manually or let the API find the best possible parameters. The kind of the reference to calculate must be set manually.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	The provided image should be acquired with the correct parameters set. To do this use the function “prepare_cam_4reference” to let the camera set the correct parameters for the acquisition of the reference image

Structure to set the parameters for the generation of a reference file for the camera

```
struct CS_CALCULATE_REFERENCE_STRUCT{
        CS_REFERENCE_MODE referenceType;
        UINT16 referenceNo;
        CS_CALC_REFERENCE_MODE calculationMode;
        BYTE* pImage;
        UINT32 height;
        UINT32 width;
        UINT16 pixelFormat ;
        UINT16 pixelPitch;
        UINT32 linePitch;
        bool bSend2Camera;
        char* pFileName;
        UINT16 topLine;
        UINT16 numberOfLines;
        UINT16 targetValue;
        bool doExtrapolation;
        UINT16 leftExtrapolationStart;
        UINT16 rightExtrapolationStart;
        UINT16 numberOfColumnsForExtrapolation;
        UINT16 planeType;
        CS_REFERENCE_VERSION referenceVersion;

        CS_CALCULATE_REFERENCE_STRUCT(){
                referenceType = BLACK_LEVEL_CORRECTION;
                referenceNo = 0;
                calculationMode = MANUAL_SETTING;
                height = 100;
                width = 100;
                pixelFormat = BGR;
                pixelPitch = 3;
                linePitch = 300;
                bSend2Camera = true;
                pFileName = NULL;
                topLine = 0;
                numberOfLines = 100;
                targetValue=255;
                doExtrapolation = false;
                leftExtrapolationStart = 0;
```

```
                    rightExtrapolationStart = (UINT16)99999;
                    numberOfColumnsForExtrapolation = 100;
                    planeType = MONOCHROME_REF_PLANE;
                    referenceVersion = CS_GENERAL_CAM_REF_VERSION;
          }
};
```

Variable type	Element name	Description
CS_REFERENCE_MODE	referenceType	Use the “CS_REFERENCE_MODE”-enumeration to set the reference to calculate.
UINT16	referenceNo	Four different references might be stored into the camera. This is useful if your setup involves different illumination scenarios. Please set the number in the range from 0 to 3.
CS_CALC_REFERENCE_MODE	calculationMode	Use the “CS_CALC_REFERENCE_MODE”-enumeration to set the reference to calculate. If set to “AUTOMATIC_DETECTION” the API determines the suitable parameters for the height and position of the area in the image where to calculate the reference.
BYTE*	pImage	Pointer to image data in the RGB-Format
UINT32	height	Height in pixels of the image
UINT32	width	Width in pixels of the image
UINT16	pixelFormat	Indicates in which order the pixels are organized. RGB or BGR order is currently supported.
UINT16	pixelPitch	The amount of bytes to move from one pixel to another pixel of the same color
UINT32	linePitch	Number of pixels to move vertically in the image.
bool	bSend2Camera	If this flag is set, the reference will be sent automatically to the camera after the creation of the reference did complete´successfully.
char*	pFileName	Name of the file where to store the reference. If this pointer is set to NULL no file will be saved.
UINT16	topLine	First line of the reference area in the image
UINT16	numberOfLines	Number of lines used to generate the reference

UINT16	targetValue (in 8bit)	Desired target value to reach when doing a flat field/shading correction. The correction factors will be calculated by using this value. E.g. if the calculated brightness for a pixel is 200, the correction value will be 255/200 = 1,275 This parameter is ignored when calculating the offset-reference.
bool	doExtrapolation	If there is no suitable data at the borders of the image, enable this parameter to calculate the missing data by using linear extrapolation
UINT16	leftExtrapolationStart	Left position from where an extrapolation should be calculated
UINT16	rightExtrapolationStart	Right position from where an extrapolation should be calculated
UINT16	numberOfColumnsForExtrapolation	Number of pixels which should be used to calculate the extrapolation slope
UINT16	planeType	If only a single color image is used, indicate which color it belongs to. References can be generated separately for each color plane. For RGB images this parameter can be ignored.
CS_REFERENCE_VERSION	referenceVersion	If this reference is meant to be for an allPIXA wave, set this parameter to CS_ALLPIXA_WAVE_VERSION, otherwise use CS_GENERAL_CAM_REF_VERSION
CS_ROI_PARAMETER_STRUCT (Only for allPIXA wave !)	roiParameters	If any ROIs in your camera are set, transfer these settings into this struct, otherwise the reference calculated will not correct the corresponding pixels! “first_scan_line_delay” and “imgHeight” are not used in this context!

create_reference_internally	The camera will internally use the currently recorded image data for the generation of the selected reference.
Syntax	INT16 create_reference_internally(const CS_REFERENCE_MODE refType, const UINT refNo, const UINT timeoutInSecconds=60)
Parameters:	refType: The type of the reference to generate. Use the enumerations `BLACK_LEVEL_CORRECTION or SHADING_CORRECTION` refNo: The number of the reference ranging from 0 or 3 timeout: the maximum timeout for the function. Useful if a triggered condition for the image acquisition is used.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	This function should be called with the correct parameters of the future use case set. To do this use the function “prepare_cam_4reference” to let the

	camera set the correct parameters for the acquisition of the reference image. Remember that you should cover the lens for a black level reference and that you need to provide a white sheet in front of the camera for the shading correction.

4.7 Low level communication functions: Sending of HSI-Tags

Please be cautious when using the raw commands together with the high level API functions! After using the raw functions you should retrieve the current camera settings by calling any "get_xxx"-function with the bUpdate parameter set to true. By doing this the internal data structure will represent the current data of the camera again.

4.7.1 Functions which directly send the values to the camera

send_bin_tag	Sends a binary value via the explicit HSI-tag to the camera.
Syntax	INT32 send_bin_tag (DWORD tagID, bool value, bool sendImmediately=true);
Parameters:	tagID: The ID for the binary tag to send to the camera value: Boolean value for the state of the tag to be set sendImmediately: if this flag is set, the data is immediately transferred to the camera, if set to false, the values are stored into memory. By successive calls to the send function it is possible to store several tags into memory. When sending a larger amount of parameters this will save a significant amount of time.
Return value:	CS_OK: if setting of the parameter was successful Values < 0: Error codes for the specific reason: see chapter "Error codes"
Comment:	To send the stored values to the camera (bSendImmediately = false) call the function "send_stored_hsi_tags"

send_short_tag	Sends a short value via the explicit HSI-tag to the camera.
Syntax	INT32 send_short_tag (DWORD tagID, WORD value, bool sendImmediately=true);
Parameters:	tagID: The ID for the binary tag to send to the camera value: Short value for the value of the tag to be set sendImmediately: if this flag is set, the data is immediately transferred to the camera, if set to false, the values are stored into memory. By successive calls to the send function it is possible to store several tags into memory. When sending a larger amount of parameters this will save a significant amount of time.
Return value:	CS_OK: if setting of the parameter was successful Values < 0: Error codes for the specific reason: see chapter "Error codes"
Comment:	To send the stored values to the camera (bSendImmediately = false) call the function "send_stored_hsi_tags"

send_long_tag	Sends a long value via the explicit HSI-tag to the camera.
Syntax	INT32 send_long_tag (DWORD tagID, DWORD value, bool sendImmediately=true);
Parameters:	tagID: The ID for the binary tag to send to the camera value: Long value for the value of the tag to be set sendImmediately: if this flag is set, the data is immediately transferred to the camera, if set to false, the values are stored into memory. By successive calls to the send function it is possible to store several tags into memory. When sending a larger amount of parameters this will save a significant amount of time.
Return value:	CS_OK: if setting of the parameter was successful Values < 0: Error codes for the specific reason: see chapter “Error codes”
Comment:	To send the stored values to the camera (bSendImmediately = false) call the function “send_stored_hsi_tags”

send_var_tag	Sends a variant value via the explicit HSI-tag to the camera.
Syntax	INT32 send_var_tag (DWORD tagID, WORD length, WORD* pData, bool sendImmediately=true);
Parameters:	tagID: The ID for the binary tag to send to the camera length: Length of the elements in the var tag pData: Pointer to the value array of the tag to be set sendImmediately: if this flag is set, the data is immediately transferred to the camera, if set to false, the values are stored into memory. By successive calls to the send function it is possible to store several tags into memory. When sending a larger amount of parameters this will save a significant amount of time.
Return value:	CS_OK: if setting of the parameter was successful Values < 0: Error codes for the specific reason: see chapter “Error codes”
Comment:	To send the stored values to the camera (bSendImmediately = false) call the function “send_stored_hsi_tags”

Container tags:

The handling of container tags is different from the sending of the other tags.

Since this tag contains other tags therefore you need to create a container tag first by calling create_container_tag.

This will create a tag in memory. After this you need to add the tags contained in the container by calling send_XX_tag and set the “sendImmediately” flag to false.

Finally to transfer the container to the camera, call the function “send_stored_hsi_tags”.

create_container_tag	Creates a container structure in memory
Syntax	INT32 create_container_tag (DWORD tagID, WORD length);
Parameters:	tagID: The ID for the binary tag to send to the camera length: Length of the contained data
Return value:	CS_OK: if creating of the container was successful Values < 0: Error codes for the specific reason: see chapter "Error codes"
Comment:	

send_stored_hsi_commands	Sends all stored tags by the use of the set-functions to the camera. After this command the stored tags are deleted.
Syntax	INT32 send_stored_hsi_commands (bool clearCommands=true);
Parameters:	clearCommands: Clear the tags stored in memory after sending the data.
Return value:	CS_OK: if setting of the parameter was successful Values < 0: Error codes for the specific reason: see chapter "Error codes"
Comment:	

4.7.2 Retrieving raw tags from the camera

Attention: The actual values from the camera are only requested when the update flag is set to true. Otherwise the memory cache of the CS-Api is read out.

Usually the user does update the cache with his first call to a get_XXX-function. Then the complete camera parameters are sent to the PC and stored the memory cache.

The user may now evaluate this cache by calling the get_XXX-functions with update set to false.

get_bin_tag	Gets a binary value via the explicit HSI-tag.
Syntax	INT32 get_bin_tag (DWORD tagID, bool & value, bool bUpdate=false);
Parameters:	tagID: The ID for the binary tag to send to the camera value: Reference of a boolean value for the state of the tag bUpdate: If set the current values will be requested from the camera
Return value:	CS_OK: if getting the parameter was successful Values < 0: Error codes for the specific reason: see chapter "Error codes"
Comment:	

get_short_tag	Gets a short value via the explicit HSI-tag.
Syntax	INT32 get_short_tag (DWORD tagID, WORD & value, bool bUpdate=false,);
Parameters:	tagID: The ID for the binary tag to send to the camera value: Reference to a short value for the value of the tag to be set bUpdate: If set the current values will be requested from the camera
Return value:	CS_OK: if getting of the parameter was successful Values < 0: Error codes for the specific reason: see chapter "Error codes"
Comment:	

get_long_tag	Gets a long value via the explicit HSI-tag.
Syntax	INT32 get_short_tag (DWORD tagID, DWORD value, bool bUpdate=false);
Parameters:	tagID: The ID for the binary tag to send to the camera value: Reference to a long value for the value of the tag to be set bUpdate: If set the current values will be requested from the camera
Return value:	CS_OK: if getting of the parameter was successful Values < 0: Error codes for the specific reason: see chapter “Error codes”
Comment:	

get_var_tag	Gets a variant value via the explicit HSI-tag.
Syntax	INT32 get_var_tag (DWORD tagID, WORD length, WORD* pData, bool bUpdate=false);
Parameters:	tagID: The ID for the binary tag to send to the camera length: Length of the elements in the var tag pData: Pointer to the value array to store the values to bUpdate: If set the current values will be requested from the camera
Return value:	CS_OK: if getting of the parameter was successful Values < 0: Error codes for the specific reason: see chapter “Error codes”
Comment:	

5 Logging / Tracing

For analysis and troubleshooting the CS-API generates logging information about internal processes and communication data to the camera.

The way and the amount of generated logging data can be parameterized.

At startup of the API logging parameters are read from registry and are applied to actual processes. The detailed registry entries are described below.

The parameters can be changed with the API function set_logging_parameter ().

set_logging_parameter	Sets the API internal logging parameter
Syntax	INT16 set_logging_parameter (CS_LOGGING_PARAM_STRUCT loggingParams, bool bStoreToRegistry)
Parameters:	loggingParams: this structure contains information to setup API internal logging functions bStoreToRegistry: If set the parameters are stored in the registry and CS-API will use these values at the next start-up.
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

Structure to set the parameters for the API logging.

```
struct CS_LOGGING_PARAM_STRUCT {
        char* pLogFileName;
        UINT32 OutputToDbgViewer;
        UINT32 LoggingMask;
        UINT32 InterfaceDataLength;
        UINT32 TimeStamp;

        CS_LOGGING_PARAM_STRUCT () {
                char* pLogFileName = NULL;
                OutputToDbgViewer = 0;
                LoggingMask =0;
                InterfaceDataLength=0;
                TimeStamp=0;
        }
};
```

Variable type	Element name	Description
char*	pLogFileName	Path and file name to store logging data. If set to Null logging data is not output to a file.
UINT32	OutputToDbgViewer	If > 0 logging data is output as debug strings and can be catched with “the DebugViewer”
UINT32	LoggingMask	Amount and content of logged data can be specified with “LoggingMask”. With the following enums the behavior of the logging is specified. The enums can be combined by OR operation to enable several function levels: LOG_APPCALLS,

		LOG_PROCDATA_MIN, LOG_PROCDATA_DETAILS, LOG_PROCDATA_SPEC LOG_INTERFACEDATA
UINT32	InterfaceDataLength	0: only command type and response type messages are logged > 0: number of bytes to be logged for each message
UINT32	TimeStamp	> 0: timestamp for each logging item is added

Registry values:

The logging parameter are stored at

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Chromasens\CS-API\

The items are listed below:

RegSZ	LogFileName;
RegDWORD	OutputToDbgViewer;
RegDWORD	LoggingMask;
RegDWORD	InterfaceDataLength;
RegDWORD	TimeStamp;
RegDWORD	InternalProcessData;

get_logging_parameter	Gets the API internal logging parameter
Syntax	INT16 get_logging_parameter (CS_LOGGING_PARAM_STRUCT *loggingParams)
Parameters:	loggingParams: Pointer to a structure where the current logging parameter will be stored to
Return value:	If successful CS_OK will be returned. An Error is indicated with values < 0
Comment:	

6 Troubleshooting

6.1 CameraLink related problems

The CSAPI offers connections to the Chromasens Cameras through CameraLink connections.

For this connection the CSAPI will use the generic clallSerial.dll. This DLL uses the driver dlls provided by the specific grabber manufacturer. Usually this DLL gets installed by the grabber setup.

Some grabber installations do not provide this DLL.
If you try to access a camera by using the function `open_CL_control` you will get an error CS_CAMERA_INTERFACE_MISSING.

How to eliminate this problem:

1. Check if the clallSerial.dll is present:
   - The location can be taken from the following registry entry:
   HKEY_LOCAL_MACHINE\SOFTWARE\CameraLink\Serial\CLSERIALPATH
   If the clallserial.dll is not present in this path, please copy it into this directory or install the CST-tool which will do this automatically.
   If the clallserial.dll is present, check if the architecture of the DLL matches the one of the operating system (Use the dependency Viewer tool for this)
2. Additional to the clallSerial.DLL you should also find the grabber specific DLLs in this directory. Usually the naming of this DLLs should be clserXXX.dll. Meaning e.g. a Matrox interface driver would look like this: clsermtx.dll, SISO: clsersis.dll, Matrix Vision: clsermv.dll.
   It is also important that there is no mixture in between the DLL-architectures. Meaning that win32 DLLs must not be mixed with x64 DLLs.
   If the grabber DLLs could not be found please contact your grabber manufacturer.

7 FAQs