SIPS Camera Driver Interface

Scientific Image Processing System (SIPS) is designed to operate with arbitrary camera, providing the camera driver DLL is available. Even Moravian Instruments Gx (CCD) and Cx (CMOS) series of cameras, with which the SIPS package is shipped as a default camera control software, use a regular driver DLLs to interact with SIPS like any other camera.

This document describes the interface between SIPS and camera drivers. It can be used for both implementing of a new camera driver for SIPS and for using of Gx/Cx camera drivers DLLs to control Moravian cameras form other software packages.

Three driver DLLs are used to control all current Moravian cameras:

- **'gxusb.dll'** handles all CCD-based Gx cameras (G0 to G4, Mark I and Mark II) connected directly to the host computer through USB 2.0 hi-speed lines (480 Mbps).

  Also all CMOS-based C0, C1, C1+ and global-shutter sensor based C2, connected using USB 3.0 super-speed lines (5 Gbps), are handled.

  Please note this driver handles G2 cameras revision 3 and higher, as well as all G0, G1, G3 and G4 cameras. Older G2 cameras with USB 1.1 interface (revision 1) and the first USB 2 version (revision 2) are no longer supported. All Gx Mark II cameras are of course supported.

  The 'gxusb' DLL offers two image acquisition interfaces (see the nech chapter for details):

  1. **Camera timing synchronous** interface, used typically to perform precisely timed short exposure (typically shorter than ~0.25s).

  2. **Computer timing** interface, typically used to perform long exposures (typically above ~0.25s).

- **'gxeth.dll'** handles all Gx and Cx cameras (G0 to G4, C0 to C5) connected to the host computer over TCP/IP network. In such case the **Moravian Camera Ethernet Adapter** device is necessary. Up to four Gx/Cx cameras of arbitrary type can be connected to this device using standard USB lines on one side and the standard 1 Gbps Ethernet interface (compatible with older 10/100 Mbps networks) can be attached on the other side. Because the TCP/IP protocol can be routed, the distance between cameras and the host PC is virtually unlimited.

  Please note camera support depends on the firmware version of the Camera Ethernet Adapter device. Refer to the Camera Ethernet Adapter User's Guide for procedures to check and/or update the device firmware.

  The 'gxusb' DLL offers only one image acquisition interface:

  1. **Camera timing asynchronous** interface.

- **'cxusb.dll'** driver handles large cooled C1×, C3, C4 and C5 camera lines, as well as a rolling-shutter sensor based C2-9000. These cameras use different image acquisition interface compared to smaller C1, C1+ and global-shutter sensor based C2 lines, and thus it is necessary to introduce new driver, as single driver DLL cannot handle different camera lines, each requiring different handling of image acquisition.

The 'cxusb' DLL offers only one image acquisition interface:

1. **Camera timing asynchronous** interface.

Please note the USB connected cameras require installation of the kernel driver on the particular computer. The USB kernel drivers are identified by USB Vendor and Product identifiers. Some cameras are software compatible, so they share the Product ID (and thus use the same driver). The operating system takes care of these drivers and the user usually needs not to install them manually (refer to the **Installing and Using Drivers and Software** manual for kernel driver installation instructions, please). Of course, no kernel drivers are installed when the Moravian Camera Ethernet Adapter is used to communicate with cameras.

Driver DLL Interface

SIPS drivers are Dynamic Link Libraries (DLLs) with specific set of exported functions. All exported functions use standard "C" calling conventions, so they can be called from arbitrary software package without problems with name decorations etc.

C calling conventions apply to 32-bit versions of supplied DLLs. There is only one universal calling convention in 64-bit mode, so specifications of calling conventions in the source code are ignored.

Some functions are mandatory and the DLL must implement them to be accepted as a SIPS driver. Other functions are optional and the particular functionality are used only if the driver exports (implements) these functions.

A certain exception from this rule are image acquisition functions. There are three sets of image acquisition functions defined:

1. **Camera timing synchronous** interface.

2. **Camera timing asynchronous** interface.

3. **Computer timing** interface.

Each driver must implement at last one set of these functions, otherwise there is no way how to acquire images and the driver is rejected by SIPS. E.g. the 'gxusb.dll' driver implements the first and third interfaces while the 'gxeth.dll' driver implements only the second one.

The 'camera timing asynchronous' and 'computer timing' interfaces cannot be combined in one driver. Every driver should implement one or another interface, but not both.

When the SIPS can use both 'camera timing synchronous' and the 'computer timing' interfaces, it uses 0.25 seconds threshold. Exposures shorter than 0.25 seconds use camera timing interface and longer exposures are counted in computer, so they can be of arbitrary length. Computer timing is somewhat less precise compared to camera timing (computer-timing uncertainty is in the order of milliseconds, while camera-timing exposure time precision is better than 0.1 millisecond), but there is no limit in exposure time length.

Camera timing synchronous interface

The camera timing synchronous interface consists of single function only 'GetFrameExposure'. Note this function is blocking, it does not return from the call until the exposure time passes and image is read.

Camera timing asynchronous interface

This interface also relies on the camera hardware to count exposure time, but is designed to operate asynchronously, without a possibly long blocking call, waiting until desired exposure time expires and image is downloaded.

The exposure sequence controlled by the camera consists of the following steps:

1. StartExposure(...)
2. ... (wait for exposure time, but can proceed to next step earlier)
3. ImageReady(), if TRUE, continue to next step, if FALSE repeat this step
4. ReadImage(...)

If the exposure is to be terminated earlier that the exposure time expires, it is possible to call 'AbortExposure' function.

Camera timing asynchronous interface continuous (serial) read

Cameras with rolling-shutter CMOS sensors and on-board RAM buffer for multiple frames can be capable to perform in "continuous read" mode, in which the next exposure initiates immediately after the previous exposure finished and image is digitized. The delay between two subsequent exposures is virtually zero and all time is consumed by exposure time only.

The 'GetBooleanParameter(gbpContinuousExposures)' call can be used to determine if this function can be used with connected camera.

If the exposure time is long enough, so the camera interface is capable to transfer images to host PC faster than individual frames are stored into camera memory, the on-board RAM works just like FIFO buffer, capable to bridge potential delays in the read commands issued by the host PC.

For very short exposures, images can be stored into camera RAM faster then the host PC is capable to read them. In such case camera performs a fast sequence of exposures until the on-board RAM is filled and then pauses until the PC reads the oldest image. When the oldest frame is read and the memory slot occupied by it is freed, camera automatically performs next exposure.

Continuous image read begins the same way like normal image read by calling the 'StartExposure' function. Only if the camera has to perform continuous read, image is not read using the 'ReadImage' function, but its alternative 'ReadImageExposure'. This informs the camera is should continue with next exposure. Last image in the sequence should be read using standard 'ReadImage' function, or alternatively the series can be aborted using 'AbortExposure' call.

The continuous image commands can be as follows:

1. StartExposure(...)
2. ... (wait for exposure time, but can proceed to next step earlier)
3. ImageReady(), if FALSE repeat this step
4. ReadImageExposure(...)
5. ... (wait for exposure time, but can proceed to next step earlier)
6. ImageReady(), if FALSE repeat this step
7. ReadImageExposure(...)
8. ... (wait for exposure time, but can proceed to next step earlier)
9. ...
10. ImageReady(), if FALSE repeat this step
11. ReadImage(...)

Number of frames, which can be stored in the camera on-board memory, naturally depends on the memory size, but also on the defined sub-frame, on the binning (if hardware binning is used) etc.

Camera timing asynchronous interface triggered exposures

Cameras can be equipped with hardware input, which allows external devices to define exact time of exposure start. The 'GetBooleanParameter(gbpTrigger)' call can be used to determine if the connected camera is equipped with hardware trigger input.

If the exposure should wait for the trigger input, it should be started with the 'StartExposureTrigger', call. The 'ImageReady' call can be used to determine if the trigger was activated and exposure already finished or the image is still not acquired. The 'AbortExposure' can be used to cancel the started exposure, regardless if the camera waits for the trigger or exposure was already started and exposure is in progress.

If the used camera is not equipped with hardware trigger input, the 'StartExposureTrigger' function is identical to 'StartExposure', which means the exposure starts immediately upon the function call.

Computer timing interface

The exposure sequence controlled by the computer consists of the following steps:

1. BeginExposure(...)
2. ... (wait for exposure time)
3. EndExposure(...)
4. GetImage(...)

Providing the optional 'BeginExposure' and 'EndExposure' functions are not exported from the driver DLL, computer timing exposure sequence is even more complex:

1. ClearSensor()
2. Open()
3. ... (wait for exposure time)
4. Close()
5. GetImage(...)

Note steps 2 ('Open') and 4 ('Close') are skipped if dark or bias frame is exposed.

Driver API function description

Exported functions are grouped according to functionality. Individual function description follows this list of all exported functions.

- Functions marked * are optional, SIPS does not require them to be implemented (exported from the driver DLL).

- Functions marked ** create image acquisition interfaces, at last one interface must be present. All functions from the particular interface must be present for the interface to be considered as implemented.

Driver life-cycle functions:

Enumerate
Initialize
Configure*
Release
RegisterNotifyHWND

Driver information functions:

GetBooleanParameter
GetIntegerParameter
GetStringParameter
GetValue
GetLastErrorString

Image parameter functions:

AdjustSubFrame*
SetBinning*
SetTemperature*
SetTemperatureRamp*
SetFan*
EnumerateReadModes*
SetReadMode*
SetGain*
ConvertGain*
SetWindowHeating*
SetPreflash*

Image acquisition 'camera timing synchronous' functions:

GetImageExposure**

Image acquisition 'camera timing asynchronous' functions:

StartExposure**
StartExposureTrigger**
AbortExposure**
ImageReady**
ReadImage**
ReadImageExposure**

Image acquisition 'computer timing' functions:

BeginExposure*,**
EndExposure*,**
ClearSensor**
Open**
Close**
GetImage**

Filter wheel functions:

EnumerateFilters*
EnumerateFilters2*
SetFilter*

```
    ReinitFilterWheel*
```

Guider functions:

```
    MoveTelescope*
    MoveInProgress*
```

GPS functions:

```
    GetImageTimeStamp*
    GetGPSData*
```

Data types used in Gx/Cx camera Application Programming Interface:

```
typedef int              INTEGER;
typedef short            INT16;
typedef unsigned int     CARDINAL;
typedef unsigned char    CARD8;
typedef float            REAL;
typedef double           LONGREAL;
typedef unsigned char    CHAR;
typedef unsigned char    BOOLEAN;
typedef void *           ADDRESS;
```

```
struct CCamera;
```

Driver API function description

```
void __cdecl Enumerate(
     void ( __cdecl *CallbackProc )( CARDINAL )
)
```

Mandatory function.

Enumerate allows discovering of all cameras currently connected to the host PC. Its argument is a pointer to callback function 'CallbackProc' with single unsigned integer argument. This callback function is called for each connected camera and the camera identifier is passed as an argument. Application can then offer the user a list of all connected cameras to choose the one with which the user wants to work etc.

If the application is designed to work with one camera only or the camera identifier is known (for instance the application scans some .INI file where the identifier is defined), 'Enumerate' needs not to be called at all.

Please note the callback function uses the "C" calling conventions, as opposed to "stdcall" conventions often used in the Windows OS.

```
CCamera* __cdecl Initialize(
    CARDINAL Id
)
```

Mandatory function.

User-space driver is designed to handle multiple cameras at once. Driver distinguishes individual cameras using handles to individual instances. Handle is defined as unsigned integer of the size corresponding to the size of the address (32 bits or 64 bits). This allows the driver implementation to allocate class/structure and return a pointer to it as a handle.

This handle is returned by the 'Initialize' function. It is necessary to pass the camera identifier to the Initialize (see the 'Enumerate' function description). If the 'Initialize' returns 0xFFFFFFFF (INVALID_HANDLE_VALUE), the particular camera cannot be used. It is either not connected or is already used by some other application.

```
BOOLEAN __cdecl Configure(
    CCamera *PCamera,
    ADDRESS  ParentHWND
)
```

Optional function.

If the driver requires configuration (for instance TCP/IP based camera requires definition of IP address), it should implement this function. This function can open a dialog box (and use the ParentHWND passed from the user application).

This function can use called already enumerated Handle to re-configure existing instance of driver or the Handle can be -1, in which case the configuration affects all instances of the particular driver. User application should allow calling of 'Configure' function with Handle = -1 also in the case 'Enumerate' did not return any camera.

SIPS always offer an empty line marked "unconfigured" for the drive exporting this function. After the configuration finished, SIPS calls 'Enumerate' to get a fresh list of available cameras (e.g. from new Moravian Camera Ethernet Adapter device).

Example of driver selection tool in SIPS, showing a camera connected over USB, which does not require any configuration. Another two grayed lines indicate presence of drivers (Gx Camera Ethernet Adapter and ASCOM) marked "unconfigured", which allow call of 'Configure' function. 'Configure' function of Gx Camera Ethernet Adapter driver was called, opening the Ethernet Adapter IP address selection dialog box.

```
void __cdecl Release(
    CCamera *PCamera
)
```

Mandatory function.

When the camera is no longer used, the handle must be released by the 'Release' call. No other function (with the exception of 'Enumerate' and 'Initialize') may be called after the 'Release' call.


```
void __cdecl RegisterNotifyHWND(
    CCamera *PCamera,
    ADDRESS  NotifyHWND
)
```

Mandatory function.

The driver can notify the application the the camera was plugged or unplugged. Notifications are sent as Windows messages to the windows, which HWND was passed as and argument to the 'RegisterNotifyHWND' function. Notification messages are:

#define WM_CAMERA_CONNECT   1034
#define WM_CAMERA_DISCONNECT   1035

When the application is no longer interested in this notification, it can call 'RegisterNotifyHWND' with NULL as a handle.

Calling Release automatically calls 'RegisterNotifyHWND' with NotifyHWND = NULL.


```
BOOLEAN __cdecl GetBooleanParameter(
    CCamera  *PCamera,
    CARDINAL  Index,
    BOOLEAN  *Boolean
)
```

Mandatory function.

Function 'GetBooleanParameter' returns boolean value depending on the Index parameter. If the function does not "understand" passed Index, it returns FALSE.

gbpConnected	= 0	TRUE if camera currently connected
gbpSubFrame	= 1	TRUE if camera supports sub-frame read
gbpReadModes	= 2	TRUE if camera supports multiple read modes
gbpShutter	= 3	TRUE if camera is equipped with mechanical shutter
gbpCooler	= 4	TRUE if camera is equipped with active CCD cooler
gbpFan	= 5	TRUE if camera fan can be controlled (switched on and off)
gbpFilters	= 6	TRUE if camera controls filter wheel
gbpGuide	= 7	TRUE if camera is capable to guide the telescope mount
gbpWindowHeating	= 8	TRUE if camera can control the CCD window heating
gbpPreflash	= 9	TRUE if camera can use CCD preflash
gbpAsymmetricBinning	= 10	TRUE if camera horizontal and vertical binning can differ
gbpMicrometerFilterOffsets	= 11	TRUE if filter focusing offsets are expressed in micrometers
gbpPowerUtilization	= 12	TRUE if camera can return power utilization in GetValue
gbpGain	= 13	TRUE if camera can return used gain in GetValue
gbpElectronicShutter	= 14	TRUE if the sensor is equipped with electronic shutter
gbpGPS	= 16	TRUE if the sensor is equipped with GPS receiver
gbpContinuousExposures	= 17	TRUE if the camera is capable of serial exposures

gbpTrigger	= 18	TRUE if the sensor is equipped with hardware trigger port
gbpConfigured	= 127	TRUE if camera is configured
gbpRGB	= 128	TRUE if camera has Bayer RGBG filters on sensor
gbpCMY	= 129	TRUE if camera has CMY filters on sensor
gbpCMYG	= 130	TRUE if camera has CMYG filters on sensor
gbpDebayerXOdd	= 131	TRUE if camera Bayer masks starts on horizontal odd pixel
gbpDebayerYOdd	= 132	TRUE if camera Bayer masks starts on vertical odd pixel
gbpInterlaced	= 133	TRUE if CCD detector is interlaced (else progressive)

```
BOOLEAN __cdecl GetIntegerParameter(
    CCamera  *PCamera,
    CARDINAL  Index,
    INTEGER  *Num
)
```

Mandatory function.

Function 'GetIntegerParameter' returns integer value depending on the Index parameter. If the function does not "understand" passed Index, it returns FALSE.

gipCameraId	= 0	Identifier of the current camera
gipChipWidth	= 1	Sensor width in pixels
gipChipDepth	= 2	Sensor depth in pixels
gipPixelWidth	= 3	Sensor pixel width in nanometers
gipPixelDepth	= 4	Sensor pixel depth in nanometers
gipMaxBinningX	= 5	Maximum binning in horizontal direction
gipMaxBinningY	= 6	Maximum binning in vertical direction
gipReadModes	= 7	Number of read modes offered by the camera
gipFilters	= 8	Number of filters offered by the camera
gipMinimalExposure	= 9	Shortest exposure time in microseconds ($\smile$s)
gipMaximalExposure	= 10	Longest exposure time in milliseconds (ms)
gipMaximalMoveTime	= 11	Longest time to move the telescope in milliseconds (ms)
gipDefaultReadMode	= 12	Read mode to be used as default
gipPreviewReadMode	= 13	Read mode to be used for preview (fast read)
gipMaxWindowHeating	= 14	Maximal value for 'SetWindowHeating' call
gipMaxFan	= 15	Maximal value for 'SetFan' call
gipMaxGain	= 16	Maximum value for 'SetGain' call
gipMaxPossiblePixelValue	= 17	Maximum value of (saturated) pixel Note the value may vary with read mode and binning, read only after 'SetReadMode' and 'SetBinning' calls.
gipLineTime	= 18	Time to digitize one image line of rolling-shutter cameras equpped with GPS receiver
gipFirmwareMajor	= 128	Camera firmware version (optional)
gipFirmwareMinor	= 129	
gipFirmwareBuild	= 130	
gipDriverMajor	= 131	Driver version (optional)
gipDriverMinor	= 132	
gipDriverBuild	= 133	

gipFlashMajor	= 134	Flash version (optional)
gipFlashMinor	= 135	
gipFlashBuild	= 136	

```
BOOLEAN __cdecl GetStringParameter(
    CCamera  *PCamera,
    CARDINAL  Index,
    CARDINAL  String_HIGH,
    CHAR      *String
)
```

Mandatory function.

Function 'GetStringParameter' returns string value depending on the Index parameter. If the function does not "understand" passed Index, it returns FALSE.

The string is copied to the buffer pointed by the String parameter. The function checks the maximum buffer size not to cause buffer overrun. The highest character index (index starts with 0) of the buffer must be passed as String_HIGH parameter. If the buffer is longer than the passed string, terminating zero character is also copied.

gspCameraDescription	= 0	Camera description
gspManufacturer	= 1	Manufacturer name
gspCameraSerial	= 2	Camera serial number
gspChipDescription	= 3	Used CCD detector description

```
BOOLEAN __cdecl GetValue(
    CCamera  *PCamera,
    CARDINAL  Index,
    REAL      *Value
)
```

Mandatory function.

Function 'GetValue' returns float value depending on the Index parameter. If the function does not "understand" passed Index, it returns FALSE.

It is similar to 'GetBoolean/Integer/StringParameter'. But while the three previous functions return static values (independent on current camera state), GetValue returns numbers reflecting current state of the camera (e.g. CCD temperature etc.). The difference is that it is typically not necessary to communicate with attached camera to get 'GetBoolean/Integer/StringParameter', but 'GetValue' typically needs to perform communication before it returns.

gvChipTemperature	= 0	Current temperature of the CCD detector in deg. Celsius
gvHotTemperature	= 1	Current temperature of the cooler hot side in deg. Celsius
gvCameraTemperature	= 2	Current temperature inside the camera in deg. Celsius
gvEnvironmentTemperature	= 3	Current temperature of the environment air in deg. Celsius
gvSupplyVoltage	= 10	Current voltage of the camera power supply
gvPowerUtilization	= 11	Current utilization of the CCD cooler (number 0 to 1)
gvADCGain	= 20	Current gain of A/D converter in electrons/ADU

```
BOOLEAN __cdecl GetLastErrorString(
    CCamera  *PCamera,
    CARDINAL  ErrorString_HIGH,
    CHAR      *ErrorString
);
```

Mandatory function.

If any call fails (returns FALSE), GetLastErrorString return failure description. ErrorString is passed similarly to GetStringParameter call.


```
BOOLEAN __cdecl AdjustSubFrame(
    CCamera *PCamera,
    INTEGER *x,
    INTEGER *y,
    INTEGER *w,
    INTEGER *d,
)
```

Optional function.

Digitization of individual pixels of CCD sensors is performed by camera electronics, outside of the sensor itself. So, the sub-frame read of a CCD is implemented in the camera and the sensor itself has little or no influence on this function. Restriction of image read to a sub-frame is naturally available in all CCD based Gx cameras without any limitations on sub-frame position and/or size. This function has no meaning if used with CCD camera.

Situation with CMOS sensors is just opposite — reading of a sub-frame has to be implemented in the sensor hardware/firmware and camera electronics cannot affect it. But sub-frame (or ROI – Region Of Interest) support by CMOS sensor hardware may impose number of limitations on the sub-frame position and size. What's more, these limitations differ among various sensors and also between 8-bit and 12-bit read modes of the same sensor etc. For instance, sub-frame origin must be aligned to certain multiply of pixels (typically 4 or 8 pixels). The same limitation is valid for sub-frame width and height. Also, minimal sub-frame width is often several hundred pixels etc.

But SIPS camera drivers as well as user interface were not designed to impose any sub-frame position/size restrictions. So, when a sub-frame read is requested by the user, the driver must combine two approaches:

1. Cx camera driver checks if the sub-frame position and size comply to sensor-imposed limitations. If yes, camera hardware is programmed to limit read to specific ROI only and resulting image is directly returned to the user.

2. If the requested sub-frame position and/or size does not fit sensor limitations, the driver enlarges the sub-frame size and position to satisfy the sensor and at the same time to be the smallest possible one, but containing the user requested sub-frame. This greater sub-frame is then read from camera and cropped by driver to the size requested by the user. Differences in sub-frame read from camera and sub-frame returned to the user are typically very small, a few pixels necessary to align position and size to multiply of 4 or 8 pixels etc. So, amount of data transferred from camera is almost the same, only the software cropping consumes some time.

The purpose of the 'AdjustSubFrame' function is to adjust sub-frame position and/or size to comply to sensor-imposed limitations. The application may offer the user to adjust requested sub-frame coordinates, so the driver can return images directly without software cropping and thus to achieve higher performance.

Note:

The support for hardware sub-frames must be implemented also by camera firmware. For C1, C1+ and C2 CMOS cameras, make sure the firmware if of version 7.6 or higher. Otherwise the driver always reads full-size image from the camera and implements sub-frame by software cropping. The impact on FPS, when only a small portion of image is requested, is obvious.

```
BOOLEAN __cdecl SetBinning(
    CCamera  *PCamera,
    CARDINAL  x,
    CARDINAL  y
)
```

Optional function.

Sets the required read binning. If the camera does not support binning, this function has no effect.

```
BOOLEAN __cdecl SetTemperature(
    CCamera *PCamera,
    REAL      Temperature
)
```

Optional function.

Sets the required chip temperature, 'Temperature' parameter is expressed in degrees Celsius. If the camera has no cooler, this function has no effect.

```
BOOLEAN __cdecl SetTemperatureRamp(
    CCamera *PCamera,
    REAL      Ramp
)
```

Optional function.

Sets the maximum speed with which the driver changes chip temperature. The 'Ramp' parameter is expressed in degrees Celsius per minute. If the camera has no cooler, this function has no effect.

```
BOOLEAN __cdecl SetFan(
    CCamera *PCamera,
    CARD8    Speed
)
```

Optional function.

If the camera is equipped with cooling fan and allows its control, this function sets the fan rotation speed. The maximum value of the Speed parameter should be determined by **GetIntegerParameter** call with **gipMaxFan** parameter index. If the particular camera supports only on/off switching, the maximum value should be 1 (fan on), while value 0 means fan off.

```
BOOLEAN __cdecl EnumerateReadModes(
    CCamera  *PCamera,
    CARDINAL  Index,
    CARDINAL  Description_HIGH,
    CHAR      *Description
);
```

Optional function.

Enumerates all read modes provided by the camera. This enumeration does not use any callback, the caller passes index beginning with 0 and repeats the call with incremented index until the call returns FALSE. Description string is passed similarly to **GetStringParameter** call.

```
BOOLEAN __cdecl SetReadMode(
    CCamera  *PCamera,
    CARDINAL  mode
);
```

Optional function.

Sets required read mode.

```
BOOLEAN __cdecl SetGain(
    CCamera  *PCamera,
    CARDINAL  gain
);
```

Optional function.

Sets required gain. Range of parameter **gain** depends on particular camera hardware, as it typically represents directly a register value. This method is chosen to allow to control gain as precisely as each particular camera allows. Low limit is 0, high limit is returned by function **GetIntegerParameter** with index **gipMaxGain**.

```
BOOLEAN __cdecl ConvertGain(
    CCamera  *PCamera,
    CARDINAL  gain,
    LONGREAL *dB,
    LONGREAL *times
);
```

Optional function.

As the **SetGain** function accepts camera-dependent parameter **gain**, which typically does not represent actual gain as signal multiply or value in dB, a helper function **ConvertGain** is provided to convert register value into gain in logarithmic dB units as well as in linear times-signal units.

```
BOOLEAN __cdecl SetWindowHeating(
    CCamera *PCamera,
    CARD8    Heating
)
```

Optional function.

If the camera is equipped with CCD cold chamber front window heater and and allows its control, this function sets heating intensity. The maximum value of the Heating parameter should be determined by **GetIntegerParameter** call with **gipMaxWindowHeating** parameter index. If the particular camera supports only on/off switching, the maximum value should be 1 (heating on), while value 0 means heating off.

```
BOOLEAN __cdecl SetPreflash(
    CCamera  *PCamera,
    LONGREAL  PreflashTime,
    CARDINAL  ClearNum
)
```

Optional function.

If the camera is equipped with preflash electronics, this function controls it. Parameter 'PreflashTime' defines time for which the preflash LED inside the camera is switched on. Second parameter 'ClearNum' defines how many times the CCD has to be cleared after the preflash.

Actual values of these parameters depends on the particular camera model (e.g. number and luminance of the LEDs used etc.). Gx series of CCD cameras typically need less than 1 second to completely saturate the CCD ('PreflashTime'). Number of subsequent clears should be at last 2, but more than 4 or 5 clears is not useful, too.

```
BOOLEAN __cdecl GetImageExposure(
    CCamera   *PCamera,
    LONGREAL  ExpTime,
    BOOLEAN   UseShutter,
    INTEGER   x,
    INTEGER   y,
    INTEGER   w,
    INTEGER   d,
    CARDINAL  BufferLen,
    ADDRESS   BufferAdr
);
```

Optional function.

This functions implement the whole 'Camera timing synchronous' interface.

Note this function is blocking, it does not return from the call until the exposure time passes and image is read.

For exposures of tens of seconds or minutes, the uncertainty of timing is not important (depending on the PC hardware, the uncertainty can be between 2 to 15 milliseconds). For short exposures this uncertainty can be important. This is why the 'Camera timing synchronous' interface is introduced. This function lets the camera itself to count the exposure with the precision of tens of microseconds.

The 'ExpTime' is the required exposure time in seconds. 'UseShutter' parameter tells the driver of the dark frame (without light) has to be acquired, or the shutter has to be opened and closed to acquire normal light image. Sub-frame coordinates are passed in the 'x', 'y', 'w' and 'd' parameters. If the camera does not support sub-frame read, 'x' and 'y' must be 0 and 'w' and 'd' must be the chip pixel dimensions.

It is expected the driver returns 16 bits per pixel (2 bytes) 'w' * 'd' matrix copied to 'BufferAdr' address. The buffer must be allocated by the caller, driver does not allocate any memory. The 'BufferLen' parameter specifies allocated memory block length in bytes (not in pixels!). It has to be greater or equal to image size in bytes else the function fails.

Maximum exposure time depends on the particular camera model. Other image exposition interface has to be used if longer than maximum exposure time is desired.

Please note the maximum exposure time is 65.5 seconds for the G2, G3 and G4 cameras and 8.192 seconds for the G0 and G1 cameras. If longer exposures are necessary, use the computer timing image acquisition interface.

CMOS cameras of all types (Cx series) can pass exposure time as long as approximately 150 hours, which is way above any reasonable exposure time.

```
BOOLEAN __cdecl GetImageExposure16b(
    CCamera  *PCamera,
    LONGREAL  ExpTime,
    BOOLEAN   UseShutter,
    INTEGER   x,
    INTEGER   y,
    INTEGER   w,
    INTEGER   d,
    CARDINAL  BufferLen,
    ADDRESS   BufferAdr
);
```

Optional function.

This function is equivalent to normal 'GetImageExposure' call, which also returns 16-bit pixels.

```
BOOLEAN __cdecl GetImageExposure8b(
    CCamera  *PCamera,
    LONGREAL  ExpTime,
    BOOLEAN   UseShutter,
    INTEGER   x,
    INTEGER   y,
    INTEGER   w,
    INTEGER   d,
    CARDINAL  BufferLen,
    ADDRESS   BufferAdr
);
```

Optional function.

Providing the used camera supports 8-bit read mode, using of 'GetImageExposure' or 'GetImageExposure16b' call forces the driver to internally expand the buffer from 8-bit pixel to 16-bit pixel format, which consumes some time. If the client application can handle image frames with single byte per pixel, using ' GetImageExposure8b' function can save time.

Note the particular camera hardware must be able to return 8-bit pixels and the read mode must also be set to 8-bit per pixel (if the camera offers multiple read modes and only some of them are 8-bit). Otherwise this call returns FALSE without even performing any exposure. In such case, normal 'GetImageExposure' must be called to return 16-bit pixel frame.

```
BOOLEAN __cdecl StartExposure(
    CCamera  *PCamera,
    LONGREAL  ExpTime,
    BOOLEAN   UseShutter,
    INTEGER   x,
    INTEGER   y,
    INTEGER   w,
    INTEGER   d
);
```

Optional function.

When the 'Camera timing asynchronous' interface is used, every exposure starts with 'StartExposure' call. Driver accepts exposure time, whether to open and close shutter (light or dark image) and sub-frame coordinates. If the camera does not support sub-frame read, 'x' and 'y' must be 0 and 'w' and 'd' must be the chip pixel dimensions. It is on the camera/driver to start the exposure.

```
BOOLEAN __cdecl StartExposureTrigger(
    CCamera  *PCamera,
    LONGREAL  ExpTime,
    BOOLEAN   UseShutter,
    INTEGER   x,
    INTEGER   y,
    INTEGER   w,
    INTEGER   d
);
```

Optional function.

This function works similarly to 'StartExpopsure', only the exposure does not start upon function call, but waits for hardware trigger. The 'ImageReady' call can be used to determine if the trigger was activated and exposure already finished or the image is still not acquired. The 'AbortExposure' can be used to cancel the started exposure, regardless if the camera waits for the trigger or exposure was already started and exposure is in progress.

If the used camera is not equipped with hardware trigger input, the 'StartExposureTrigger' function is identical to 'StartExposure', which means the exposure starts immediately upon the function call. The 'GetBooleanParameter(gbpTrigger)' call can be used to determine if the used camera has trigger input.

```
BOOLEAN __cdecl AbortExposure(
    CCamera *PCamera,
    BOOLEAN  Download
);
```

Optional function.

When the exposure already started by 'StartExposure' call has to be terminated before the exposure time expires, 'AbortExposure' has to be called. The 'Donwload' parameter indicates

whether the image should be digitized, because the user will call 'ReadImage' later or the image should be discarded.

```
BOOLEAN __cdecl ImageReady(
    CCamera *PCamera,
    BOOLEAN *Ready
);
```

Optional function.

When the exposure already started by 'StartExposure' call, the 'ImageRady' function returns FALSE in the 'Ready' parameter if the exposure is sill running. When the exposure finishes and it is possible to call 'ReadImage', 'ImageReady' function returns TRUE.

It is recommended to count the exposure time in the user application despite the fact the exact exposure time is determined by the camera/driver and to start calling of 'ImageReady' only upon the exposure time expiration. Starting to call 'ImageReady' in the infinite loop immediately after 'StartExposure' and call it for the whole exposure time (and thus keeping at last one CPU core at 100% utilization) is a bad programming manner (politely expressed).

```
BOOLEAN __cdecl ReadImage(
    CCamera  *PCamera,
    CARDINAL  BufferLen,
    ADDRESS   BufferAdr
);
```

Optional function.

When 'ImageReady' returns 'Ready = TRUE', it is possible to call 'ReadImage'. It is expected the driver returns 16 bits per pixel (2 bytes) matrix copied to 'BufferAdr' address. The buffer must be allocated by the caller, driver does not allocate any memory. The 'BufferLen' parameter specifies allocated memory block length in bytes (not in pixels!). It has to be greater or equal to image size in bytes else the function fails.

```
BOOLEAN __cdecl ReadImageExposure(
    CCamera  *PCamera,
    CARDINAL  BufferLen,
    ADDRESS   BufferAdr
);
```

Optional function.

'ReadImageExposure' function initiates exposed image read exactly like the 'ReadImage' function, but lets the camera immediately start the subsequent exposure. Individual exposures follow immediately one after another, without the overhead of clearing the sensor.

If the function is called and the connected camera does not support serial read, functions returns FALSE. The 'GetBooleanParameter(gbpContinuousExposures)' call can be used to determine if this function can be used in forward.

For more details, see the Serial Image Read sub-chapter.

```
BOOLEAN __cdecl ReadImage16b(
    CCamera *PCamera,
    CARDINAL BufferLen,
    ADDRESS BufferAdr
);
```

Optional function.

This function is equivalent to normal 'ReadImage' call, which also returns 16-bit pixels.

```
BOOLEAN __cdecl ReadImage8b(
    CCamera  *PCamera,
    CARDINAL  BufferLen,
    ADDRESS   BufferAdr
);
```

Optional function.

Providing the used camera supports 8-bit read mode, using of 'ReadImage' or 'ReadImage16b' call forces the driver to internally expand the buffer from 8-bit pixel to 16-bit pixel format, which consumes some time. If the client application can handle image frames with single byte per pixel, using ' ReadImage8b' function can save time.

Note the particular camera hardware must be able to return 8-bit pixels and the read mode must also be set to 8-bit per pixel (if the camera offers multiple read modes and only some of them are 8-bit). Otherwise this call returns FALSE without even performing any exposure. In such case, normal 'ReadImage' must be called to return 16-bit pixel frame.

```
BOOLEAN BeginExposure(
    CCamera *PCamera,
    BOOLEAN  UseShutter
);
```

Optional function.

This function, if implemented, is used instead of of 'ClearSensor' and 'Open' calls (note 'Open' has to be called only if no dark or bias frame is opened, this option is replaced by the 'UseShutter' parameter of 'BeginExposure' function).

```
BOOLEAN EndExposure(
    CCamera *PCamera,
    BOOLEAN  UseShutter,
    BOOLEAN  AbortData
);
```

Optional function.

This function complements the 'BeginExposure' function and has to be called instead of 'Close' call if present. Parameter 'UseShutter' indicates shutter operation (should be FALSE if it was FALSE in the corresponding 'BeginExposure' call).

Typically the 'GetImage' call follows the 'EndExposure' call to actually read the image frame. If the exposure was canceled by the user and no image has to be read from camera, parameter 'AbortData' must be set to TRUE.

```
BOOLEAN __cdecl ClearSensor(
    CCamera *PCamera
);
```

Optional function.

The 'ClearSensor' functions performs detector clearing, which is necessary before each exposure has to be started in the case optional 'BeginExposure' and 'EndExpopsure' functions are missing.

```
BOOLEAN __cdecl Open(
    CCamera *PCamera
);
```

Optional function.

Opens camera shutter. If the camera does have mechanical shutter, this function has no effect.

```
BOOLEAN __cdecl Close(
    CCamera *PCamera
);
```

Optional function.

Closes camera shutter. If the camera does have mechanical shutter, this function has no effect.

```
BOOLEAN __cdecl GetImage(
    CCamera  *PCamera,
    INTEGER  x,
    INTEGER  y,
    INTEGER  w,
    INTEGER  d,
    CARDINAL BufferLen,
    ADDRESS  BufferAdr
);
```

Optional function.

Reads image from the camera. Image is a 'w' * 'd' matrix of 16-bit pixels. BufferAdr points to the buffer to which the image has to be copied. BufferLen is the size of the buffer in bytes (not pixels!). Is the passed size is not big enough to hold the image, the call fails.

If the camera does not support sub-frame read, 'x' and 'y' must be 0 and 'w' and 'd' must be the chip pixel dimensions.

```
BOOLEAN __cdecl GetImage16b(
    CCamera  *PCamera,
    INTEGER  x,
    INTEGER  y,
    INTEGER  w,
    INTEGER  d,
    CARDINAL BufferLen,
    ADDRESS  BufferAdr
);
```

Optional function.

This function is equivalent to normal 'GetImage' call, which also returns 16-bit pixels.

```
BOOLEAN __cdecl GetImage8b(
    CCamera  *PCamera,
    INTEGER  x,
    INTEGER  y,
    INTEGER  w,
    INTEGER  d,
    CARDINAL BufferLen,
    ADDRESS  BufferAdr
);
```

Optional function.

Providing the used camera supports 8-bit read mode, using of 'GetImage' or 'GetImage16b' call forces the driver to internally expand the buffer from 8-bit pixel to 16-bit pixel format, which consumes some time. If the client application can handle image frames with single byte per pixel, using ' GetImage8b' function can save time.

Note the particular camera hardware must be able to return 8-bit pixels and the read mode must also be set to 8-bit per pixel (if the camera offers multiple read modes and only some of them are 8-bit). Otherwise this call returns FALSE without even performing any exposure. In such case, normal 'GetImage' must be called to return 16-bit pixel frame.

```
BOOLEAN __cdecl EnumerateFilters(
    CCamera  *PCamera,
    CARDINAL  Index,
    CARDINAL  Description_HIGH,
    CHAR     *Description,
    CARDINAL *Color
);
```

Optional function.

Enumerates all filters provided by the camera. This enumeration does not use any callback, but the caller passes index beginning with 0 and repeats the call with incremented index until the call returns FALSE. Description string is passed similarly to GetStringParameter call. Color parameter hints the Windows color, which can be used to draw the filter name in the application.

```
BOOLEAN __cdecl EnumerateFilters2(
    CCamera  *PCamera,
    CARDINAL  Index,
    CARDINAL  Description_HIGH,
    CHAR     *Description,
    CARDINAL *Color,
    INTEGER  *Offset
);
```

Optional function.

Sames as 'EnumerateFilters' function, but returns one more parameter 'Offset'. This parameter indicates the focuser shift when the particular filter is selected.

Units of the 'Offset' can be micrometers or arbitrary focuser specific units (steps). If the units used are micrometers, driver should return TRUE from GetBooleanParameter(gbpMicrometerFilterOffsets, …) call.

```
BOOLEAN __cdecl SetFilter(
    CCamera  *PCamera,
    CARDINAL  FilterIndex
);
```

Optional function.

Sets the required filter. If the camera is not equipped with filter wheel, this function has no effect.

```
BOOLEAN __cdecl ReinitFilterWheel(
    CCamera  *PCamera
);
```

Optional function.

The filter wheel performs the initialization, during which the zero filter position is found and set.

```
BOOLEAN __cdecl MoveTelescope(
    CCamera *PCamera,
    INT16    RADurationMs,
    INT16    DecDurationMs
);
```

Optional function.

Instructs the camera to initiate telescope movement in the R.A. and/or Dec. axis for the defined period of time (in milliseconds). Parameters are passed as 16 bit integers and the sign defines the movement direction in the respective coordinate. The maximum length is approx 32.7 seconds.

If the camera is not equipped with autoguider port, this function has no effect.

```
BOOLEAN __cdecl MoveInProgress(
    CCamera *PCamera,
    BOOLEAN *Moving
);
```

Optional function.

Sets the 'Moving' variable to TRUE if the movement started with 'MoveTelescope' call is still in progress.

If the camera is not equipped with autoguider port, this function has no effect.

```
BOOLEAN __cdecl GetImageTimeStamp(
    CCamera  *PCamera,
    INTEGER  *Year,
    INTEGER  *Month,
    INTEGER  *Day,
    INTEGER  *Hour,
    INTEGER  *Minute,
    LONGREAL *Second
);
```

Optional function.

Obtains the exposure start time of the last acquired image, based on GPS time. The information about the time is available only after the 'ReadImage' or 'ReadImageExposure' call. The time information for the last read image remains valid until the next 'ReadImage' or

'ReadImageExposure' call (time instances are stored into internal FIFO the same way like entire frames are stored into FIFO created in camera on-board memory).

Please note this function must be called after the 'ReadImage' or 'ReadImageExposure' call beginning with the camera firmware v6.6 and later. Previous versions of camera firmware allowed the 'GetImageTimeStamp' call immediately after the 'ImageReady' call returned true, but prior to next 'StartExposure' call.

If the function returns false, either the camera is not equipped with GPS receiver or the GPS receiver did not have valid time lock at the instance of exposure beginning.

The presence of the GPS receiver in the camera can be tested using the 'GetBooleanParameter' call with 'gbpGPS' index.

The exposure time stamp precision is better than 1 µs. However, the GPS receiver must acquire at last 5 satellites to achieve such precision.

```
BOOLEAN __cdecl GetGPSData(
     CCamera  *PCamera,
     LONGREAL *Lat,
     LONGREAL *Lon,
     LONGREAL *MSL,
     INTEGER  *Year,
     INTEGER  *Month,
     INTEGER  *Day,
     INTEGER  *Hour,
     INTEGER  *Minute,
     LONGREAL *Second,
     CARDINAL *Satellites,
     BOOLEAN  *Fix
);
```

Optional function.

Read GPS location and current time in the instance of the call. Location coordinates 'Lat' and 'Lon' are provided in decimal degrees. North and East are positive directions, South and West are marked by negative numbers. MSL (mean sea level) is provided in meters.

The 'Satellites' parameter indicates number of satellites tracked by the GPS receiver and the 'Fix' variable indicates if the receiver acquired position fix.

The location information is updated every minute. However, the time information, also provided by this call, does not contain the time of location information acquisition (it can be up to minute old), but uses the same mechanism like the 'GetImageTimeStamp' call to get precise time. As the USB3 packet round-trip-time is typically around 0.1 ms, the time information can be obtained with better than 1 ms precision. But while the location information is available from 3 acquired satellites (despite with limited precision), time information requires at last 5 satellites. So, if the location information is available, but the number of satellites is not high enough to provide precision time lock, the time-related parameters are returned all zeros.

Document updates

2019-05-14: Version 4.0 of driver DLLs released.

- Added preliminary support for C1 CMOS cameras.

2019-10-10: Version 4.1 of driver DLLs released.

- Added final support for C1 CMOS cameras.

- Added optional API functions 'SetGain' and 'ConvertGain'.

- Added software binning and cropping (sub-frame) for Cx cameras.

2020-02-05: Version 4.2 of driver DLLs released.

- Modified the behavior of the optional 'BeginExposure' and 'EndExposure' functions, used as part of the **Computer Timing Interface**. The 'BeginExposure' call, if present, should not be called in addition to the 'ClearSensor' and 'Open' calls, but instead of them (no 'ClearSensor' and 'Open' have to be called if the 'BeginExposure' is present). Similarly, if the 'EndExposure' is present, it has to be called instead of 'Close', not in addition to 'Close' call.

- Added support for optional frame reading functions with 8-bit per pixel format 'GetFrameExposure8b/16b', 'ReadImage8b/16b' and 'GetFrame8b/16b'. The *16b versions of respective functions are only aliases to functions without bit size specifications, which always return 16-bit per pixel format.

- Added support for C2 line of cooled CMOS cameras.

- Added support for C1+ line of cooled CMOS cameras.

2020-04-23: Version 4.3 of driver DLLs released.

- Fixed bug causing wrong position of the sub-frame. This bug occurred only when reading image from Cx (CMOS) camera in 12-bit read mode and other than 1×1 binning. Other cameras, 8-bit read modes and unbinned images were unaffected.

2020-10-08: Version 4.4 of driver DLL released.

- Added new 'cxusb.dll' driver file, supporting new C4-16000 scientific CMOS camera.

- Added hardware sub-frame (region-of-interest) handling for C1, C1+ and C2 CMOS cameras into 'gxusb' driver.

  Please note the hardware sub-frame read requires camera firmware version 7.6 or higher.

- Added new optional API function 'AdjustSubFrame', used to adjust coordinates of sub-frames of CMOS cameras to comply to limitations imposed by used sensors.

2020-12-16: Version 4.5 of driver DLL released.

- The 'cxusb.dll' driver now supports C4-16000v2 scientific CMOS camera (PID 0C41H).

2021-04-30: Version 4.6 of driver DLL released.

- The 'cxusb.dll' driver now supports 4th read mode **LoGain "16b"** for C4-16000 scientific CMOS cameras.

2021-05-28: Version 4.7 of driver DLL released.

- The 'cxusb.dll' driver now supports hardware windowing on C4-16000 scientific CMOS cameras. The C4-16000 camera must use firmware version 4.4 or later to enable this feature.

- Preliminary support for the C1x and C3 cameras included into 'cxusb.dll' driver DLL.

2021-09-10: Version 4.8 of driver DLL released.

- Fixed missing 'gipMaximalExposure' parameter in the 'GetIntegerParameter' function of the 'cxusb.dll' driver DLL.

- Added SetGain and ConvertGain function headers to the 'cxusb.h' file.

2022-02-22: Version 4.9 of driver DLL released.

- The 'cxusb.dll' driver newly supports hardware 2×2 binning of the C4-16000 scientific CMOS cameras. The C4-16000 camera must use firmware version 6.6 or later to enable this feature.

  The hardware binning is is an optional featured, turned on/off by the 'HWBinning' flag in the '[device]' section of the 'cXusb.ini' configuration file.

2022-03-23: Version 4.10 of driver DLL released.

- The hardware 2×2 binning of the C4-16000 camera driver 'cxusb.dll' is supported from firmware version 7.7, instead of 6.6 to fix some issues in sub-frame read.

2022-08-09: Version 4.11 of driver DLL released.

- The C1× and C3 camera driver 'cxusb.dll' now supports hardware 2×2 binning. Note this functionality requires camera C1× and C3 firmware version 3.3 or later.

- The 'cxusb.dll' driver now supports the new C5 camera line.

- Added **BinningSaturate** and **BinningSum** flags to the **[driver]** section of the 'cxusb.ini' driver configuration file. These flags allow modification of the default binning behavior of all C1×, C3 and C5 cameras. Note this functionality requires camera firmware version 5.5 or later.

2022-09-03: Version 4.12 of driver DLL released.

- Added index 'gbpGPS' to 'GetBooleanParameter' call.

- Added functions 'GetImageTimeStamp' and 'GetGPSData'. These functions are available in the **'cxusb.dll'** and **'gxeth.dll'** drivers, which can handle new C5 camera with GPS receiver.

2023-03-24: Version 4.13 of driver DLL released.

- Added support to C2-9000 camera.

- Added support for continuous (serial) image read. This mode is supported for C2-9000, all C1×, C3 and C5 cameras, providing the camera firmware is updated to version 6.6 or higher.

  - Added 'ReadImageExposure' function.

  - Added 'gbpContinuousRead' index of the 'GetBooleanParameter' function.

- Added support for hardware trigger input. The hardware trigger is supported by C1× cameras with GPS support.

  - Added 'StartImageTrigger' function.

  - Added 'gbpTrigger' index of the 'GetBooleanParameter' function.

- Fixed the 'GetImageTimeStamp' for exposures shorter than frame digitization time. The returned exposure time is now corrected by all offsets, caused by exposure length, used sub-frame etc. and corresponds to the exact time of the first line exposure start.

2023-05-17: Version 4.14 of driver DLL released.

- Fixed problem in 'ReadImageExposure' function, combined with hardware 2×2 binning.

2023-07-04: Version 4.15 of driver DLL released.

- The 'GetGPSData' function, used on cameras with new GPS receiver module version 2, could falsely return the Fix parameter set to FALSE, when the receiver reported the GPS satellite network in Differential mode in position fix. If the GPS satellite network remained in the Autonomous (non-differential) mode, the Fix parameter was properly set to TRUE.

- The 'Open' and 'Close' functions are exported from the 'gxeth.dll' and 'cxusb.dll' driver libraries, despite they are not normally used in the 'Camera Timing Asynchronous' interface, used by these drivers. The user's software can use these functions e.g. to temporarily close the sensor when a satellite passes over a field of view etc.

2023-08-03: Documentation update.

- Updated the 'GetImageTimeStamp' function description, the validity of the image GPS time stamp for the read image in regard to camera firmware version is specified.

2023-11-02: Version 4.16 of driver DLL released.

- Added serial image read support for C4-16000 cameras.
- Added support for GPS exposure timing for C3 and C2 cameras.

2023-12-08: Version 4.17 of driver DLL released.

- Added **GetIntegerParameter** function index **gipLineTime**.
- Added support for C0 cameras.
- Added support for C1 v3 (compact) cameras.
- Added support for C1+ v3 cameras.
- Added support for C2 v3 cameras.

2024-07-09:   Version 4.18 of camera driver DLL released.

- Added support for Standalone Filter Wheel (SFW) drivers 'sfw.dll' and 'sfweth.dll'.
- The 'gxusb.dll' now supports the C2 cameras with global-shutter sensors and GPS receiver.
- The 'cxusb.dll' now supports C1+ cameras with rolling shutter sensors (C1+9000).

2024-09-19:   Version 4.19 of camera driver DLL released.

- The 'cxusb.dll' driver fixes the wrong image orientation of the C1+9000 camera.