CEMES CCD CAMERA USER'S GUIDE

- **Low level driver functions for power supply and camera control**

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

CEMES

Table of contents

1. Introduction 3

1.1. Glossary 3
1.2. Contents 3
1.3. Dictionary 4
1.4. DLL (Dynamic Link Library) 4
1.5. Supported Operating Systems 5
1.6. Supported Communication Buses 5
1.7. System Requirements 5
1.8. Installation 5

2. Software structure 6

2.1. Characteristics 6

3. Hardware structure 6

3.1. Communications 7
3.2. Acquiring modes 8
3.3. CCD TH7899M structure 9
3.4. CCD specifications 10
3.5. Side effects 11
3.6. Windowing 11
3.7. CCD cleaning 12
3.8. Shutter modes 12
3.9. Thermal functioning 12
3.10. Camera performance 13

4. Driver classes 19

4.1. Power Supply class 19
4.2. Control and acquiring class 24
4.3. Communications class 32

5. Using the classes functions in C 35

5.1. Settings to be chosen before interacting with the dll 35
5.2. Communication examples 36
5.3. Error codes 39

1. Introduction

1.1 Glossary

Abbreviation	Definition
API	Application Program Interface
DLL	Dynamic Link Library
LIB	Library

Table 1: Glossary

The main role of the low level driver functions capsulated in the CID_HL.dll are:

- To generate the communication through several existing communication buses such as the serial port and the PCI.

- To give the user a group of functions allowing him to have access to the main capabilities of the camera, regardless its knowledge about the register fashion, low level functions. This also permits the user to write is own code in a more structured way, choosing which class of functions (power supply, control and acquiring or communications) he pretends to use and then, which function of that particular class he needs to call in his code.

- To give a homogeneous interface regardless the product and the firmware version.

This 32-bit Dynamic Link Library “CID_HL.dll” is totally developed in C++ and can be used from all the other common programming languages such as C, C++ or Visual Basic. This library is compatible with Windows 2000 XP. The dll provides a suite of functions allowing the user to configure the data acquisition process, the CCD temperature and the shutter operations, as well as other camera processes. To use this driver effectively, the user must develop a software package to configure the acquisition, provide memory management, process the data scan(s) and create the user interface.

1.2 Contents

This document provides a description of all the functions encapsulated in the CID_HL.DLL as well as their way of use.

1.3 Dictionary

The following terms are constantly used in this manual. It is essential for the user to get familiarized with these definitions before using the manual:

DLL – A .dll file contains one or several functions compiled, linked, and stored separately from the processes using them. The operating system maps the DLLs into the address space of the process when this process is starting up or while it is running. The process then executes the functions in the DLL which means "dynamic link library"

LIB – Library of information used by a specific program

OS – Operating System

ADU – Analog Digital Unit: Electronic value that represents the coder levels. It's equivalent to the grey level of one pixel

Binning – To sum the photons charge of different adjacent CCD pixels to reconstruct one single pixel

Windowing – To only acquire a fraction of the total image

1.4 DLL (Dynamic Link Library)

This type of library is a standard for Windows and all the programmers have to deal with it nowadays. Actually, all the Windows libraries are in this format and all the compilers available for Windows use this kind of libraries. The main points characterizing a DLL are:

- A Windows standard
- Its format is known by all the compilers
- A DLL is dynamically linked. It allows the user to modify or to update it without compiling again the application using it.

It is important to correctly understand the functioning of a DLL. Please refer to the manuals of your compiler to know in detail how the DLLs work and how the compiler is able to manage them. Be careful to understand where the DLLs have to be stored and where the application goes and gets them.

1.5 Supported Operating Systems

At the moment the CID_HL.dll is suitable for use in all Windows versions since Windows 95.

1.6 Supported Communication Buses

At the moment the CID_HL.dll supports the following communication bus:

- RS232 serial bus
- PCI bus

1.7 System Requirements

The system (PC) requirements to run the camera with the CID_HL.dll are:

- 2 RS232 communication ports
- 1 motherboard free slot to install the ADLink board
- 100 MB free disk space

Note: The experience has shown that the power management on laptops often induces a malfunctioning of the serial port.

1.8 Installation

1) Copy the CID_HL.DLL file into your project folder.

2) Place the CID_HL.LIB file into “Library to import” and “other dependencies”

3) Include the CID_HL.DLL in one of the .h files of your project by typing:
    #define c:\project\CID_HL.DLL

IMPORTANT: If you cannot see the DLL, please go to the Explorer’s View menu. Select “Folder Options” and then “View”. Then choose either the “Show all files” button or the “Do not show hidden files”—anything but the “Do not show hidden or system files” button (Windows 98 now seems to regard all DLLs as system files and not “application extensions” as they usually are).

WARNING: Do NOT keep multiple versions of CID_HL.DLL (or any other module) in the Modules folder with simple renaming. If they are in that folder and still have the file type “DLL” they will still be loaded and used by the project. If you want to keep older versions of any modules, make a separate folder (e.g. “OldModules”) and put them in there with any other name.

2. Software Structure

The following figure demonstrates the software modules organization and its communication with the hardware drive systems level.

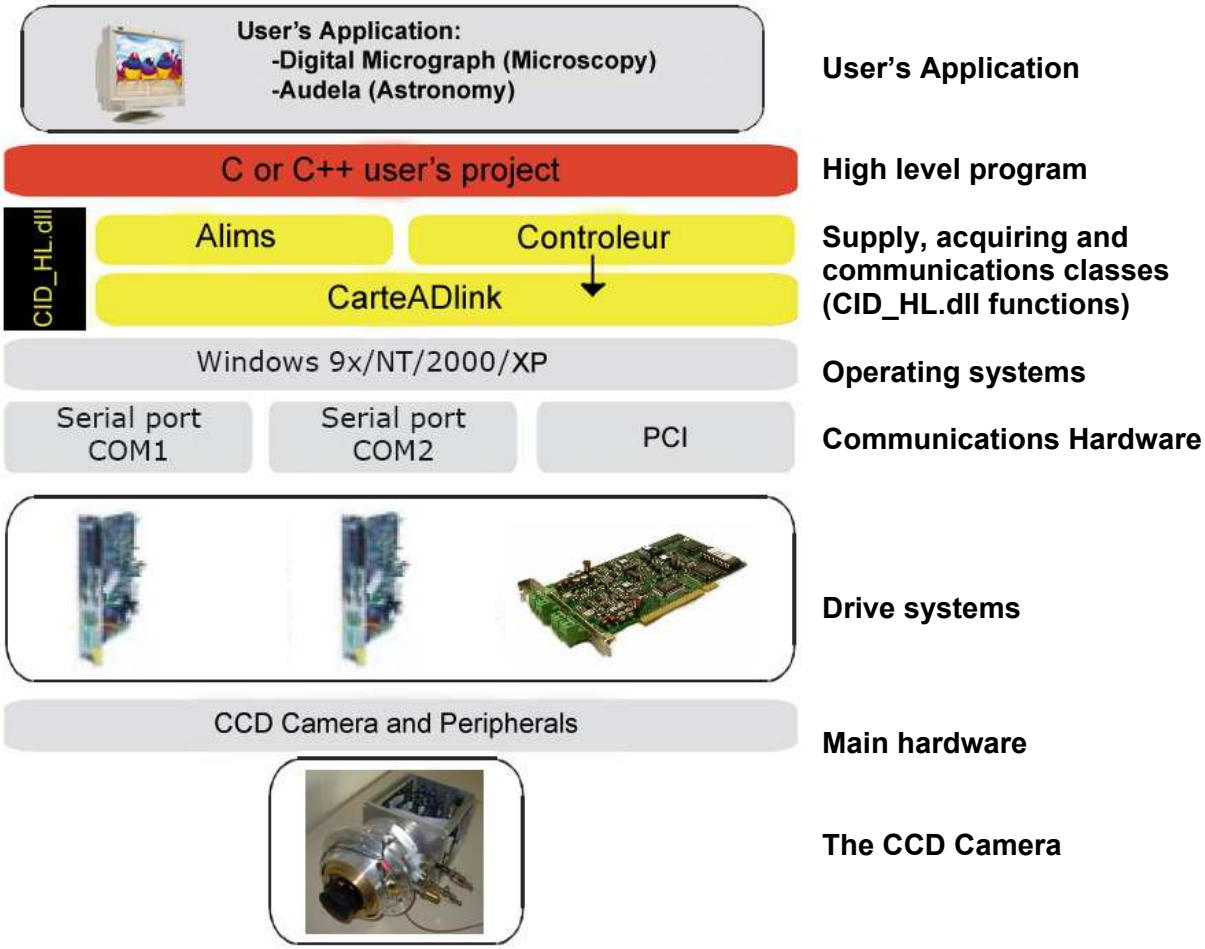

Figure 1: Overview on software structure

2.1 Characteristics

- Implemented in ANSI C++
- Can be used from C, C++ (Visual C++ and Borland C++ Builder)
- Common source files for multiple platforms and operating system
- C++ wrapper classes included in header files
- Object-oriented architecture

3. Hardware Structure

3.1 Communications

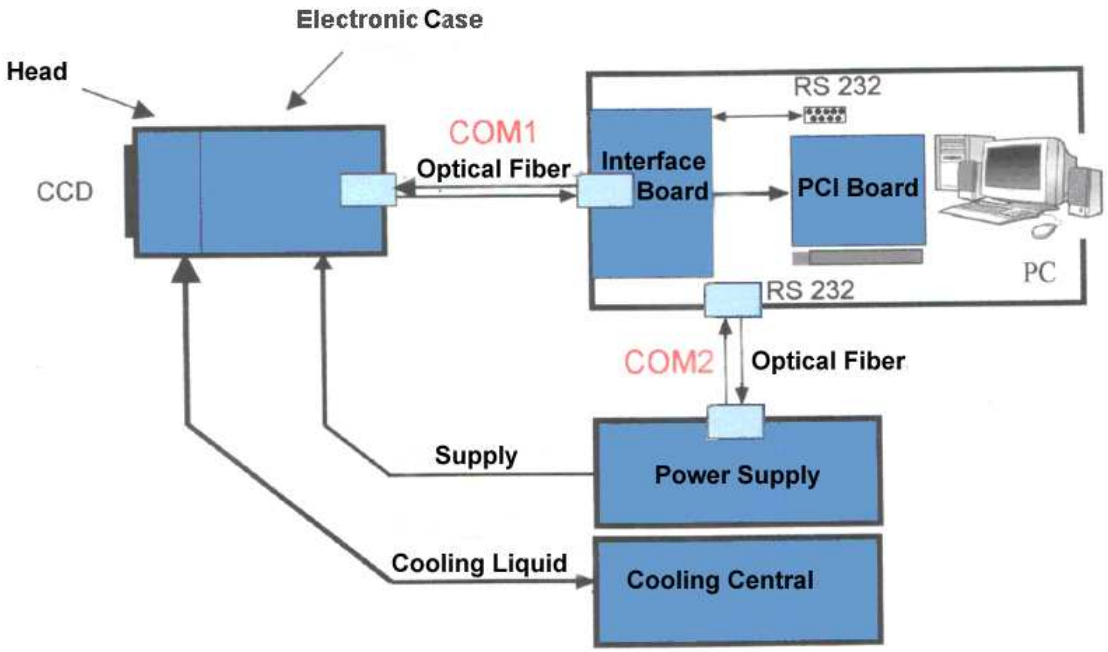

Figure 2: Communications overview

The CEMES CCD camera is a high sensibility numerical CCD camera for use in electronic microscopy and astronomy. The camera is controlled by the C++ low-level functions (CID_HL.dll) presented in this user's guide, which communicates with the camera by an I/O card (card AD-Link 7300A), which is reconnected to the camera's electronic by a serial connection RS232, has shown in figure 2.

As one can see in figure 2, the CCD is supplied by a power supply which is controlled by the PC via a RS232 connection optically coupled. The CCD is cooled by a cooling liquid provided by a cooling supply.

The Interface board communicates with the camera by optical fiber as it is represented in the image above.

3.2 Acquiring modes

The camera has two distinct operation modes with different goals.

The **High Speed mode** (HV) is a video mode and is normally used for adjusting the camera and for registering transitory phenomena. Under this mode the CCD is read at 8 MHz. This high frequency value permits obtaining a 25 images/s ratio. One can choose one of the following modes:

HV2: High speed with 2x2 binning (1024x1024)
HV4: High speed with 4x4 binning (512x512)
HV8: High speed with 8x8 binning (256x256)

The most important advantage when comparing with an independent video camera is that the field of view is the same, which simplifies the image adjustments.

The **High Resolution mode** (HR) is a "slow scan" mode that allows the user to choose between 4 different speed/resolution ways of imaging:

HR1: High resolution with 1x1 binning (2048x2048)
HR2: High resolution with 2x2 binning (1024x1024)
HR4: High resolution with 4x4 binning (512x512)
HR8: High resolution with 8x8 binning (256x256)

The following table has further information about the camera operation:

Mode	Binning Spatial resolution (µm)		Number of pixels Image size	Reading time	Exposure time	Max. Charge Ké/pixel		Max ADC level (ADU)	Elect-rons/ ADU	Equivalent Exposure time	
HR Slow-Scan	HR1 PHOTO	1 x 1 14x14	2080 x 2080 8,65 MO	1.27 s	1 s 0.44 l/s	115	132	2^{16} ADU	2	500 ms	
	HR2 SLOW	2 x 2 28x28	1040 x 1040 2,16 MO	0.618 s	0.5 s 0.9 l/s	178/ 4	51.2	2^{16} ADU	3.203	195 ms	
	HR4 SLOW	4 x 4 56x56	520 x 520 0.541 MO	0.311 s	0.1 s 2.43 l/s	178/ 16	12.8	2^{16} ADU	3.203	48.6 ms	
	HR8 SLOW	8 x 8 128x128	260 x 260 0.135 MO	0.159 s	0.024 s 5.5 l/s	178/ 64	3.2	2^{16} ADU	3.203	12.15 ms	
HV Video	HV2 FAST	2 x 2 28x28	1040 x1040 2,16 MO	76.31 ms	123.7 ms 5 l/s	178/ 4	51.2	2^{14} ADU	12.81	195 ms	3.7 l/s
	HV4 FAST	4 x 4 56x56	520 x 520 0.541 MO	41.27 ms	59 ms 10 l/s	178/ 16	12.8	2^{14} ADU	12.81	48.6 ms	11.1 l/s
	HV8 FAST	8 x 8 128x128	260 x 260 0.135 MO	23.75 ms	16.2 ms 25 l/s	178/ 64	3.2	2^{14} ADU	12.81	12.1 ms	28 l/s

Table 2: Camera operation

The exposure time should not be inferior to 1ms regardless the acquisition mode. The maximum charge corresponds to the electron number that saturates the measure. 132000 electrons will be converted to the level of 65536 in the ADC (analog/digital converter), which results in about 2 electrons/level without binning and 3.2 in binning mode.

3.3 CCD specifications

The CCD used in CEMES camera is a 2048x2048 full frame Charge Couple Device designed for a wide range of applications due to both its operating mode flexibility and its high dynamic range combined with its high resolution. The device is 180º symmetrical, so if it is not plugged in the right side it will not be damaged. The nominal photosensitive area is made of 2048x2048 useful pixels divided vertically in 4 zones of 520 lines each. Each zone can be driven separately by four-phase clocks, allowing different operating modes. There are two identical horizontal shift registers: one at the top of the image and the other at the bottom. At each end of the two readout registers is located a summing gate which can be clocked to allow a horizontal pixel summation before the on-chip output amplifier.

The matrix has also:

- 7 columns of dark reference, formed by photo-elements that are not exposed to photons, showing thou the thermal behavior of the detector and are used as reference level;
- 5 lines of isolating pixels, formed by identical photo-elements to those used in the matrix, used to ensure the 2048 active columns;
- The TH7889M is read at a frequency of 8 MHz in High Speed (HV – haute vitesse) mode and at 888.89 KHz at High Resolution (HR – haute résolution) mode;
- The exposure times can be adjusted between 10µs and 167s;
- Binning: 2x2, 4x4, 8x8;
- Working at non MPP in HV mode and MPP with HR mode;
- The 4 reading ways can be selected as strong gain (G_{NB}) without binning (212 ke-) or as reduced gain (G_B) with binning (532 ke-);
- Band-pass limited at 14.1 MHz at HR (BP_{HR}) and at 37.9 MHz at HV (BP_{HV});
- CCD working temperature adjusted at -40 ºC with a cooling liquid at +5 ºC
- The window is centered;
- Converting factor, without binning (F_{NB}): 7µV/e-;
- Converting factor, with binning (F_B): 7µV/e-;
- Maximum read charge without binning is 212 ke- and 532 ke- with binning;
- Pixel size: 14µmx14µm with 100% aperture;
- Image zone: 28,67mmx28,67mm;
- Data rates up to 4 x 20MHz (compatibility with 15 frames/second);
- Very low dark current (MPP mode);
- Optimized resolution and response in the 400-1100nm spectrum;
- Compatible with fiber optic face plate coupling;
- Operating temperature range: -55 ºC to +85 ºC;
- The read duration of a vertical transfer is 6µs.

3.4 CCD TH7899M structure

As one can see in the following picture, the CCD ATMEL TH7899MRCH assembled in our camera, has 4 quadrants and 2 output (read only) registers. After each exposure the electric charges formed in each pixel are vertically and then horizontally transferred to this output registers. The amount of time expended to read all the pixels can be diminished by reading all the 4 quadrants simultaneously.

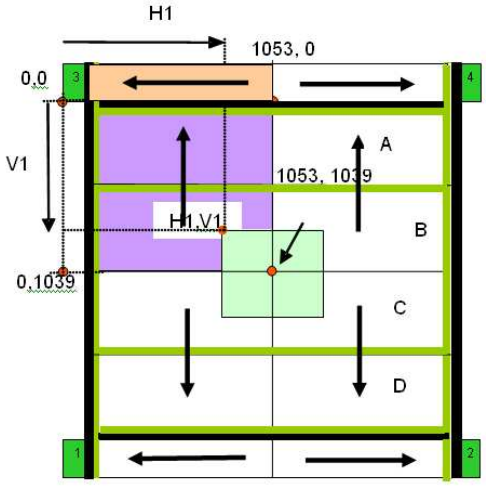

Figure 3: CCD's structure

The next table explains de meaning of the distinct zones of the CCD. The number of pixels to transmit is 1054, and 1040 of each are read, the 18 extra pixels aren't in the sensible area of the CCD. They belong just to the output registers (14 are transmitted without being read), 7 pixels are masked, they are only sensible to the thermal effect. They're goal is to be used in the image correction process. The 5 remaining pixels are often called the isolation pixels between the calibrating zone and the 1024 pixels image zone. Each of the 4 zones (A, B, C and D) has 8 supplementary lines of witch 3 are masked in the superior (A) and inferior (D) extremities.

Quadrants			Binning 1	Binning 2	Binning 4	Binning 8
Pixels	Transferred		1054	1054	1054	1054
	Read	Total	1040	520	260	130
		Extra	4	2	1	2
		Non exposed	7	4	3	
		Isolation	5			
		Image	1024	512	256	128
Lines	Transferred		1040	520	260	130
	Read	Total	1040	520	260	130
		Thermal	3	3	3	3
		Image	1037	519	259	129
Image (Pixels x Lines)			2080x2080	1040x 1040	520x520	260x260
bytes			8 652 800	2 163 200	540 800	135 200

Table 3: Reading modes

3.5 Side effects

Due to electrical imperfections the user should not expect to have a similar response from the 4 quadrants. In order to adjust the grey level of the image in running time (instead of executing a posterior image processing, which can also be done with great results) it is possible to use a dll function to adjust the 4 quadrants. This process should be taken each time the image doesn't present similar grey levels in the 4 quadrants.

3.6 Windowing

Consist on reading only the useful part of the matrix, that means, the part which contains the image information.
In HR mode, this part is read at low speed (0.89 MHz), while the rest is transferred in a vertical binning of 8 lines at 8 MHz. In HV mode, the window is read at 8 MHz.

The window is centred; the zone to be read must, therefore, be at the image's centre and one can select the window by choosing one of the following options: "1/1", "1/2" or "1/4".

So, in order to correctly centre this zone, this functionality allows to diminish the reading time, therefore it allows increasing image cadence, always keeping desired spatial resolution.

	1/1	**1/2**	**1/4**
Mode HR1	2048x2048	1024x1024	512x512
Mode HR2 or HV2	1024x1024	512x512	256x256
Mode HR4 or HV4	512x512	256x256	128x128
Mode HR8 or HV8	256x256	128x128	64x64

Table 4: Size of an image, according the mode and chosen window option

The table CC shows that with 500ms of exposure, the cadence on HR1 changes from 0.59 to 1.60 images per second, reducing the window by a factor of 4. Diminishing the exposure time, one can obtain until a factor of 10, reducing the window by 4.

Window	**Image size**	**Reading time (sec)**	**Images/s ($T_{exposure}$=0.5s)**	**Cadence max Images/s ($T_{exposure}$=0s)**	**Cadence without windowing (2080x2080)**
1/1	2048x2048	1.19	0.59	0.84	0.58
1/2	1024x1024	0.39	1.13	2.59	0.58
1/4	512x512	0.12	1.60	8.13	0.58

Table 5: Windowing – example for the HR1 mode

3.7 CCD cleaning

The CCD cleaning process is executed at the end of each capture and it is considered as a background task. The CCD is read at 0.89 MHz with four vertical transfers between each line transfer. The cleaning process takes 314ms to be accomplished, although this process might be stopped at any moment when it is requested. At the end of each cleaning process the start/stop state is verified, and if there's any request, this process will be stopped.

3.8 Shutter modes

About the control over the camera's shutter it can be said that there are 3 modes of operation, open, closed and synchro. The synchro mode opens the shutter before each capture and then closes the shutter in an automatic fashion.

Note: Due to a non-infinitesimal opening time of the shutter (about 300 ms), one should not demand to the camera an exposure time inferior to 0.3 seconds when capturing in the synchro mode.

3.9 Thermal functioning

A start request is sent for the Peltier cooler (Peltier M/A = 1) and the cold finger temperature is measured, then the program sends the temperature exchanger instruction (Temp instruction) and reads thermal exchanger temperature (Temp ETH). After this, the program reads the current Peltier value. It takes between 5 and 10 minutes for the temperature to stabilize below instruction temperature (about -30 ºC). Although every temperature between -30 and -50 can be demanded, if one demands a temperature above -30, the program will read a -30. The same process is executed for a value below -50. If after a time out the temperature is not stabilized, an alarm signal is sent with a stop request for Peltiers supplies. If the cold finger temperature is below the instruction value, the program continues.

3.10 Camera performance

Processes that were took in account in order to achieve the best response:

- 3 stage peltier design with an high current supply
- Water cooling (ethanol solution at -10ºC) of the first stage of the peltiers with a cooling central to reduce the operating temperature of the camera to -55ºC, a temperature 15ºC below the minimal operating temperature recommended to this CCD to avoid malfunctioning.
- Reading frequency of the CCD reduced to 112 KHz, which is 8 times slower than 900KHz, the original operating frequency.

Although the camera was designed by spending the above efforts to achieve the maximal performance in long exposure acquiring, there are always thermal noise in this kind of cameras, and it always increases when augmenting the exposure time, which is in fact, the really important proposal of the camera. Nevertheless, by comparing this camera to other similar cameras, it can be said after the testing and the noise measures, that this camera has a good long exposure response, and a practical noise as low as ??? electron/volt.

Expected noise formulae:

$$BthrnbCCD := \sqrt{\frac{Bllhr^2}{Néchan} + Btrnb^2 + \frac{Bélechrnb^2}{Néchan} + \frac{Bmis^2}{Néchan} + \frac{Bps^2}{Néchan} + Bthhr^2}$$

- **BthrnbCCD:** Total noise of the CCD camera
- **Néchan:** Number of samples
- **BLhr:** Reading noise of the CCD in High Resolution mode (HR)
- **Btrnb:** Transfer noise of the CCD in HR mode
- **Bélechrnb:** Noise due to the amplification and conversion stages
- **Bmis:** Other noises
- **Bps:** Noise due to the power supply
- **Bthhr:** Thermal noise in HR mode

Expected noise:

Number of samples	N=4	N=8	N=16	N=32	N=64
Total Noise HR (TEMPERATURE = - 40ºC)	16,15	14,87	14,18	13,83	13,65
Total Noise HR (TEMPERATURE = - 55ºC)	11,12	9,16	8,00	7,35	7,00

Table 6: Expected noise

Experimental noise:

Number of samples	N=4	N=8	N=16	N=32	N=64
Total Noise HR (TEMPERATURE = - 40ºC)					
Total Noise HR (TEMPERATURE = - 55ºC)					

Table 7: Experimental noise

Graphics on expected noise:

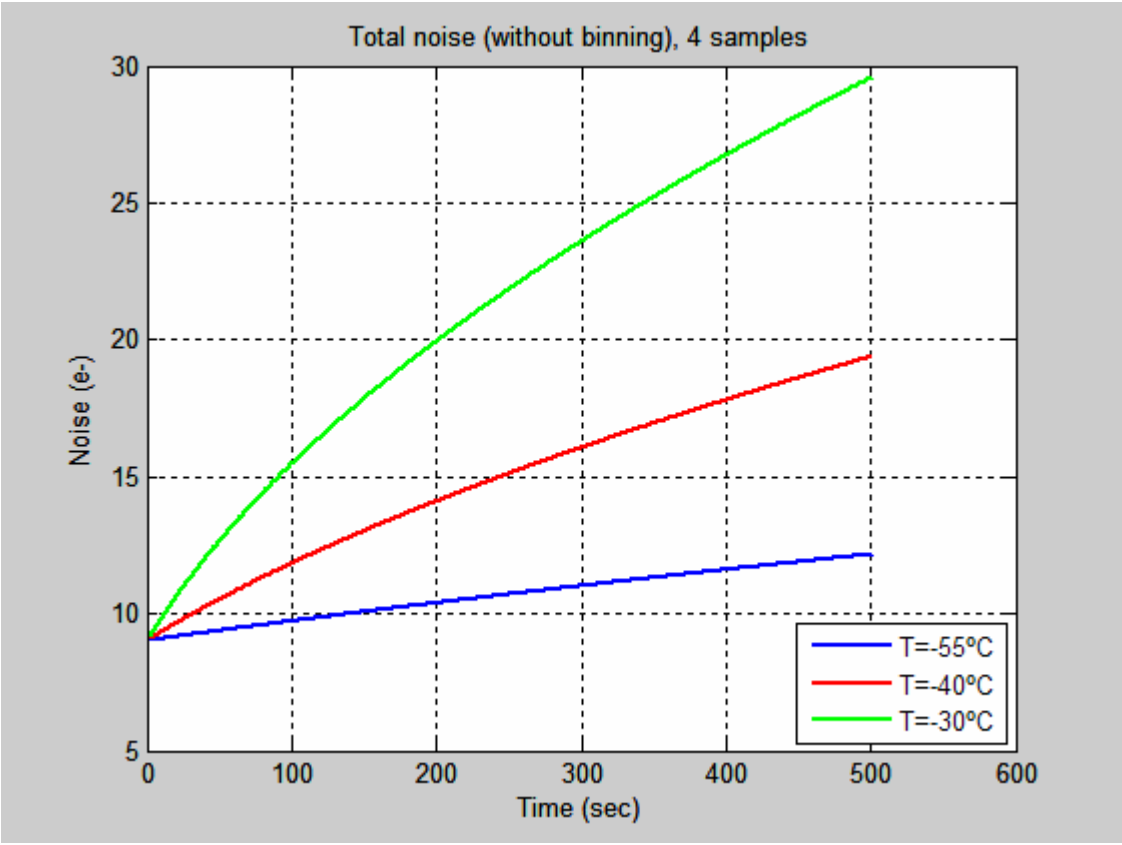

Figure 4: CCD's total noise without binning, samples=4

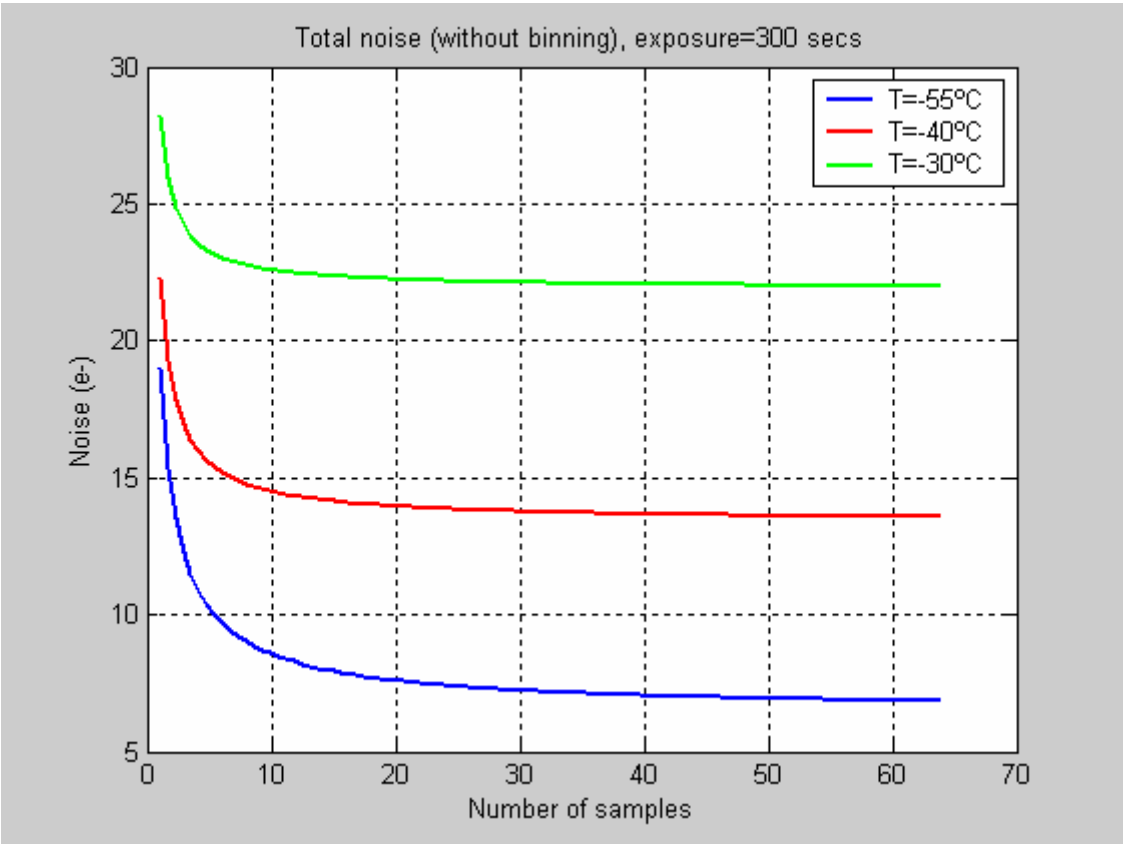

Figure 5: CCD's total noise without binning, exposure=300 secs

Graphics on experimental noise:

M42 nebulae taken in audela with a 16'' telescope:

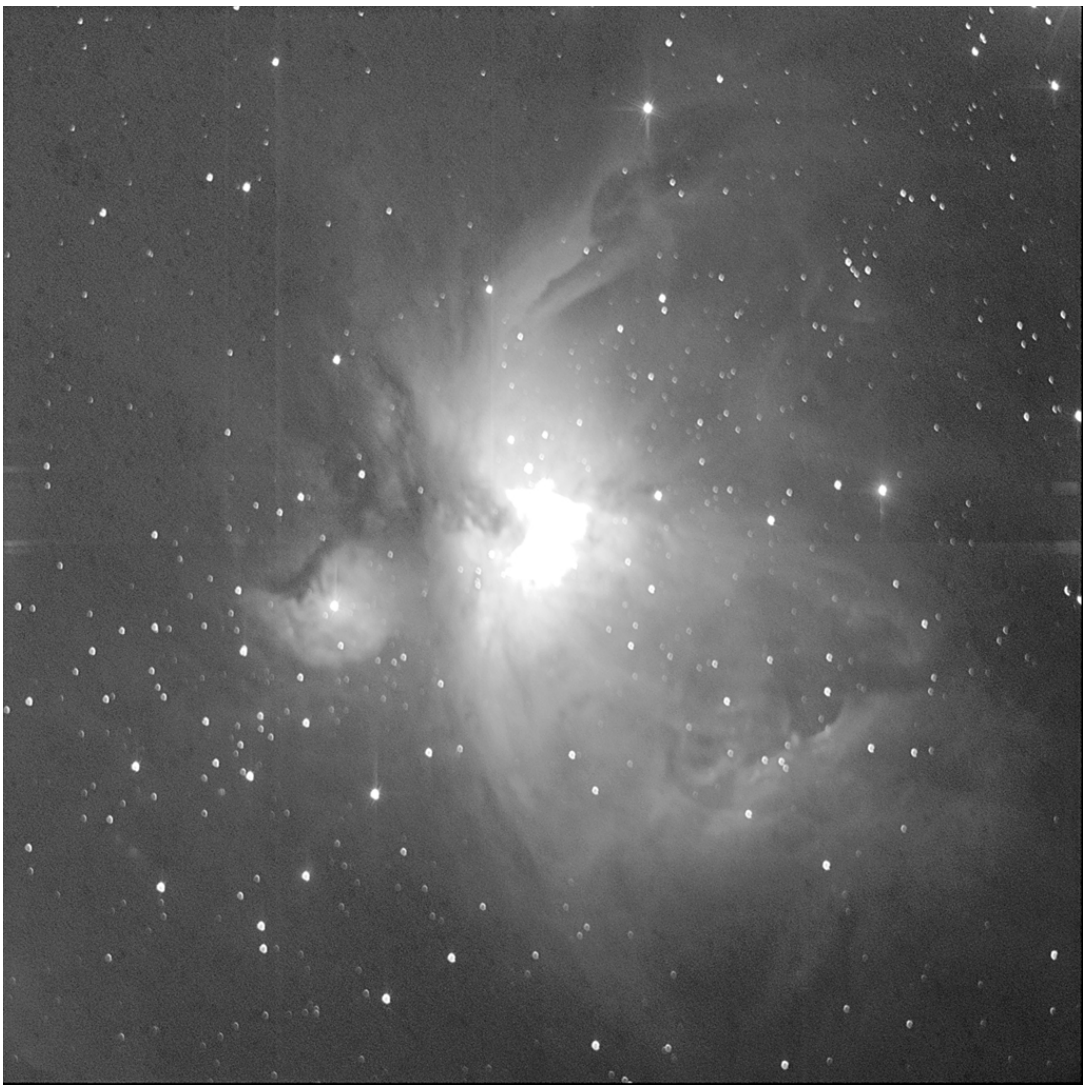

Figure 6: M42 nebulae taken in audela with a 16'' telescope

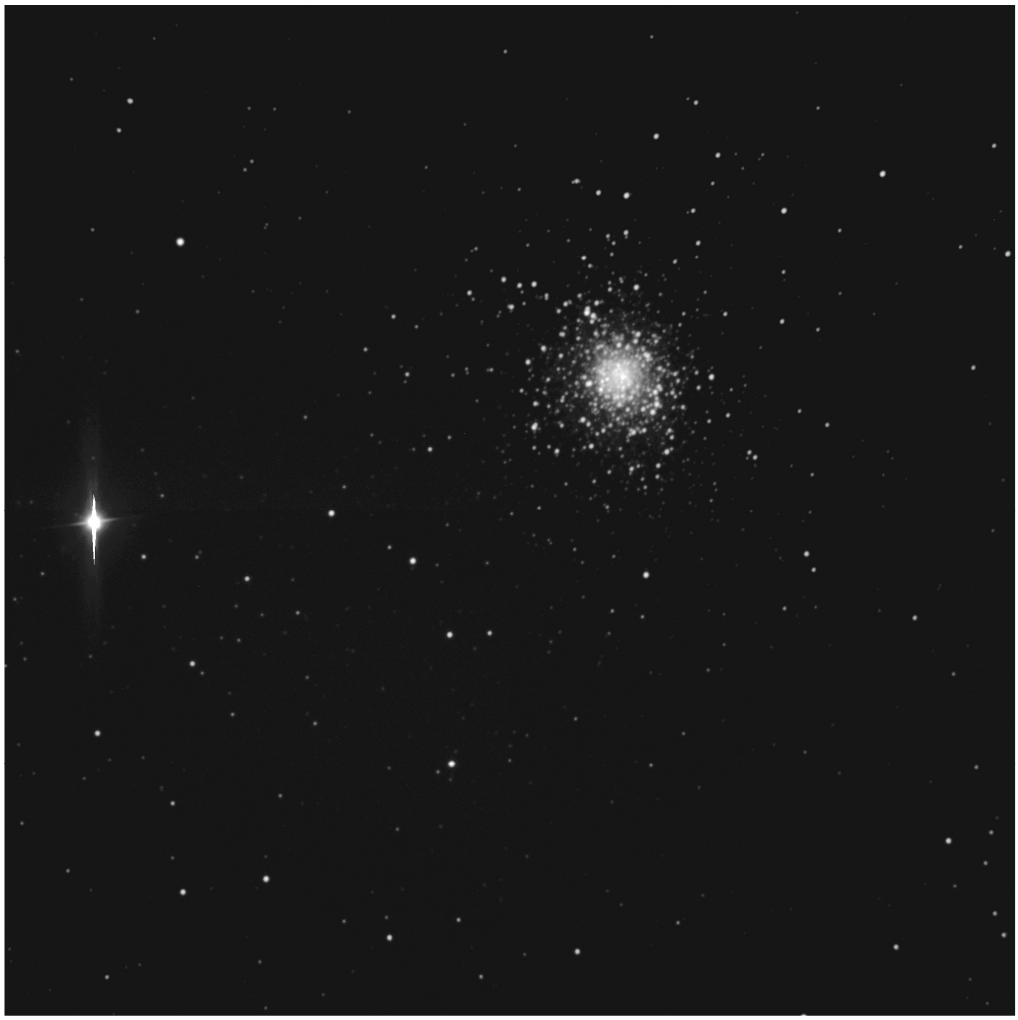

Figure 7: M5 Cluster taken in audela with a 16'' telescope (1x60s)

Figure 8: M5 Cluster (amplification over the previous image)

Figure 9: M81 and M82 Galaxies taken in audela with a 16'' telescope(3x60s)

Figure 10: M51 Galaxy taken in audela with a 16'' telescope (7x60s)

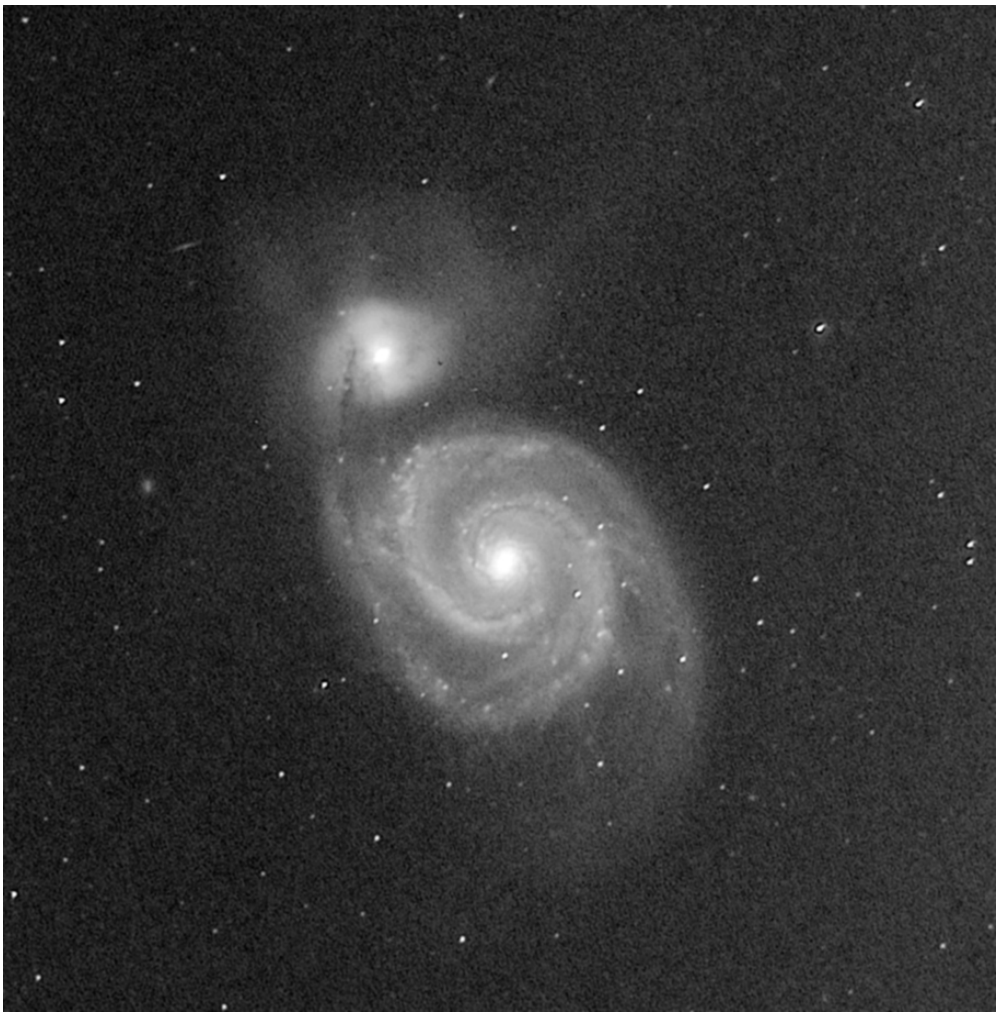

Figure 11: M51 Galaxy (amplification over the previous image)

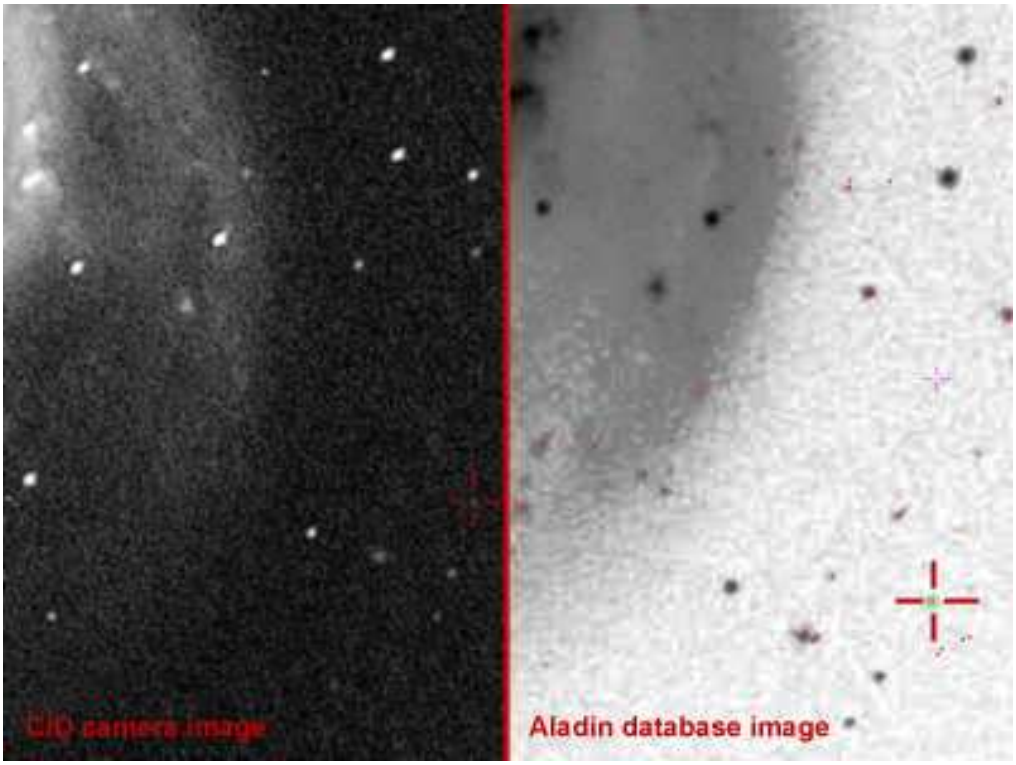

Figure 12: M51 Galaxy (A 20.44 magnitude star was found among the field)

4. Driver classes

The commands supported by the CID_HL.dll are grouped into the following sections discussed individually below:

- **Calims** (power supply)
- **Ccontroleur** (control and acquiring)
- **CarteADlink** (communications)

4.1 Power supply class (Calims)

This class contains the functions that control the power supply and that directly or indirectly manipulate the voltages of the camera. Thus the temperature control and the surveillance of all the voltages within the camera are executed by the functions of this class.

GetStatus

bool GetStatus(bool *alert, bool *supply, bool *peltier, bool *bt, bool *tachedefond)

Description Verifies the status of the camera, the peltier and the power supply and if they work correctly together.

Parameters bool *alert: not used;
bool *supply: verifies the status of the case connection;
bool *peltier: verifies the status of the peltier and stores 0 if there is a connection problem or too high temperature exchanger
bool *bt: stores the camera's head power supply status.
bool *tachedefond: not used.

Return bool: true if the status of the camera, the peltier and the power supply are correctly read and if they work correctly together.

See also

GetSupply

bool GetSupply(float *v5, float *v15, float *vm15, float * v18, float *vm18);

Description Reads the 5 supply voltages to independent registers, one for each supply: 5V, 15V, -15V, 18V and -18V.

Parameters float *v5: stores value in the 5V supply register;
float *v15: stores value in the 15V supply register;
float *vm15: stores value in the -15V supply register;
float * v18: stores value in the 18V supply register;
float *vm18: stores value in the -18V supply register.

Return bool: true when the 5 supply voltage values are correctly demanded and settled to the respective registers.

See also

GetTemperature

bool GetTemperature(int n, double *temp)

Description Reads the temperature of the peltier, the thermal exchanger or the cold finger temperature.

Parameters int n: selects peltier (0), thermal exchanger (1) or cold finger (2)
double *temp: pointer to storage of temperature value.

Return bool: true when the temperature is correctly read from the chosen device.

See also

GetVR

bool GetVR(unsigned int n, float *val);

Description

Parameters unsigned int n:
float *val:

Return bool:

See also

GetVRConsigne

double GetVRConsigne(unsigned int n)

Description Returns the rail (n) voltage value

Parameters unsigned int n: selects the rail number n

Return double: rail voltage value.

See also

GetVRLimits

void GetVRLimits(int voie, float *min, float *max);

Description Gets the minimum voltage (*min) and the maximum voltage (*max) of one of the 8 rails (voie)

Parameters int voie: value from 1 to 8, to select one of the 8 rails;
float *min: stores the minimum voltage of the selected rail;
float *max: stores the maximum voltage of the selected rail.

Return None.

See also

SetBasseTension

bool SetBasseTension(bool on)

Description Turns on/off the power supply and sets the default rail voltages.

Parameters bool on: turns the power supply on (1) or off (0).

Return bool: true when the power supply is correctly turned on/off.

See also

SetModeTension

bool SetModeTension(bool HR, unsigned int binning);

Description Returns true if the rail voltages are correctly settled according to the high resolution mode (HR=1) or the high speed mode.

***: VR1 >>> n=1 VR2 >>> n=2 VR3 >>> n=3 VR4 >>> n=4**

Parameters bool HR:
unsigned int binning:

Return bool:

See also

SetPeltier

bool SetPeltier(bool on);

Description Turns the peltier on or off.

Parameters bool on: turns on (true) or off (false) the peltier.

Return bool: true when the peltier is correctly turned on/off.

See also

SetPeltierConsigne

bool SetPeltierConsigne(unsigned int cons);

Description Demands the peltier to achieve a temperature value.

Parameters unsigned int cons: value between 0 and 65535, which corresponds to a certain temperature (in ºC) to be settled on the peltier, according to the expression: $(10^{2.6551*10^{-6}*\ \ ^{3}+4.9964*10^{-4}*\ \ ^{2}+1.516*10^{-3}+3.565199})$

Example: temperature= -30ºC -> consigne=7900
temperature= -35ºC -> consigne=10250
temperature= -40ºC -> consigne=13600
temperature= -45ºC -> consigne=18500
temperature= -50ºC -> consigne=25500

Return bool: true when it is demanded the peltier to achieve a temperature value (cons).

See also

4.2 Control and acquiring class (Ccontroleur)

This class contains the functions that control the main capabilities of the camera, like the filling of the parameters of the image (binning, speed, shutter, mode, image size and exposure time) and the image acquiring commands.

ecrit_registre

bool ecrit_registre(unsigned char add, long val);

Description Writes a value in a FPGA registers.

Parameters unsigned char add: register's address;
long val: value to write in the register.

Return bool: true if correctly write to the FPGA register.

See also

GetArea

bool GetArea(unsigned short *x0, unsigned short *y0, unsigned short *xb, unsigned short *yb);

Description Returns true if it gets correctly the parameters chosen earlier in the call of the SetArea function.

Parameters unsigned short *x0, unsigned short *y0: stores the coordinates of the top left pixel of the screen;
unsigned short *xb, unsigned short *yb: stores the default values used in order to keep the image centred.

Return bool: true if it correctly gets the parameters chosen earlier by calling the SetArea function.

See also

GetNextImage

unsigned long GetNextImage(unsigned char * data, bool parbandes, bool *fin_image, int *nbimage);

Description Gets the parameters of the next image to be taken.

Parameters unsigned char *data: pointer to image data;
bool parbandes:

bool *fin_image: register which returns true when the image is totally captured to the array (only for binning=1 mode)
int *nbimage: stores the image number

Return bool: true when the data image array is full.

See also

GetPOLAConsigne

void GetPOLAConsigne(unsigned int n);

Description Reads supplies instruction of the different polarization tensions.*see GetPolaLimits

Parameters unsigned int n:

Return None.

See also GetPolaLimits

GetPolaLimits

void GetPolaLimits(int voie, float *min1, float *max1);

Description Reads rail tension limits of the different power supplies.

Parameters int voie: selects the rail:
1=VDR1 – Polarization 1
2=VDR2 – Polarization 2
3=VDR3 – Polarization 3
4=VDR4 – Polarization 4
5=VGS2 – Polarization 5
6=VGS1 – Polarization 6
7=VGS4 – Polarization 7
8=VGS3 – Polarization 8
9=VGL1 – Polarization 9
10=VGL2 – Polarization 10
11=VGL3 – Polarization 11
12=VGL4 – Polarization 12
float *min1: stores the minimum voltage of the selected rail;
float *max1: stores the maximum voltage of the selected rail.

Return None.

See also

GetStatusAmplis

bool GetStatusAmplis(bool *comG, bool *comBP, bool *coml, bool *ampliam, bool *ampliof);

Description Verifies the camera's amplifier status.

Parameters bool *comG:
bool *comBP:
bool *coml:
bool *ampliam:
bool *ampliof:

Return bool: true if correctly receives the camera's amplifier status.

See also

GetTempsExposition

bool GetTempsExposition(double *pose, bool *erreur);

Description Verifies the exposure time and if it's a correct value.

Parameters double *pose: stores the exposure time previously selected;
bool *erreur: receives true if *pose* (exposure time) is out o its limits.

Return bool: true if correctly receives the exposure time to the register pointed by the pointer *pose.

See also

Initialise

void Initialise(bool fromregistry = true);

Description Initializes the settling of the default polarization and shift values for the high speed mode

Parameters bool fromregistry:

Return None.

See also

---

lit_registre

bool lit_registre(unsigned char add, long val);

Description Returns true when correctly read from the FPGA's registers. *Add* is the register's address, *val* is returned value.

Parameters unsigned char add: register's address;
long val: stores the read value from the register.

Return bool: true if correctly write to the FPGA register.

See also

Reset

bool Reset(void);

Description Resets the camera and periperals

Parameters None.

Return bool: true if correctly resets the camera and peripherals.

See also

SaveRegistry

void SaveRegistry();

Description Saves the shift tensions in the windows registers. It is called after a calibration.

Parameters None.

Return None.

See also

SerialDownload

bool SerialDownload();

Description Programs the FPGAs.
NOTE: One must verify that the files needed to the programming are present in the PC folders.

Parameters None.

Return bool: true if correctly programs the FPGA.

See also

SetAmplisObtu

bool SetAmplisObtu(bool on, bool on2, bool on3, bool on4, int on5);

Description Sets the camera's shutter and amplifier.

Parameters bool on: amplifier (0=AUTO/1=MAN);
bool on2: shutter (0=AUTO/1=MAN);
bool on3: amplifier (0=OFF/1=ON);
bool on4: shutter (0=OFF/1=ON);
int on5: mode (0=IMAGING/1=DETECTOR/2=ASTRO).

Return bool: true when correctly sets the camera's shutter and amplifier.

See also

SetArea

bool SetArea(unsigned short x0, unsigned short y0, unsigned short xb = 10000, unsigned short yb = 10000);

Description Sets the image window's size.

Parameters unsigned short x0, unsigned short y0: coordinates of the top left pixel of the screen (one should chose 16, 16 for a 2048x2048 image; 8, 8 for a 1024x1024 image; 4, 4 for a 512x512 image or 2, 2 for a 256x256 image);
unsigned short xb = 10000, unsigned short yb = 10000: default values used in order to keep the image centred.

Return bool: true if the image window's size is correctly settled.

See also

---

bool SetConfCalcul(short ConfCalcul);

Description Reads in an independent way the different samplings. It's used for the camera's calibration at the same time that shift tensions are regulated (setdecalage).

In HR (High Resolution) mode:

-ConfCalcul: 0 >> Ref 1
-ConfCalcul: 1 >> Ref 2
-ConfCalcul: 2 >> Sigma ref 1
-ConfCalcul: 3 >> Signal 1
-ConfCalcul: 4 >> Signal 2
-ConfCalcul: 5 >> Signal 3
-ConfCalcul: 6 >> Signal 4
-ConfCalcul: 7 >> Sigma Signal
-ConfCalcul: 8 >> Sigma Signal – Sigma Ref

In HV (High Speed) mode:

-ConfCalcul (0,1,2) >> Ref
-ConfCalcul (3,4,5,6,7) >> Signal
-ConfCalcul 8 >> Signal - Ref

Parameters short ConfCalcul:

In HR (High Resolution) mode:

0 >> Ref 1
1 >> Ref 2
2 >> Sigma ref 1
3 >> Signal 1
4 >> Signal 2
5 >> Signal 3
6 >> Signal 4
7 >> Sigma Signal
8 >> Sigma Signal – Sigma Ref

In HV (High Speed) mode:

(0,1,2) >> Ref
(3,4,5,6,7) >> Signal
8 >> Signal - Ref

Return bool: true when correctly reads in an independent way the different samplings.

See also

SetDebugLevel

void SetDebugLevel(int level);

Description Sets the debug level (level)

Parameters int level: 0- normal functioning
1- 4000x1000 image to test the lines
2- The FPGA sends constant values

Return None

See also

SetDECALAGE

void SetDECALAGE(unsigned int n, unsigned int val);

Description Sets the shift tension (val) to balance one of the 4 zones of the CCD. There are two balances that can be made to the image, the static balance (n=13-16) and the dynamic balance (n=17-20)

Parameters unsigned int n: 13-16 >> Static level of each of the 4 CCD zones
17-20 >> Dynamic level of each of the 4 CCD zones

unsigned int val: value between 0 and 4095;

Return None.

See also **SetConfCalcul** (to choose either the static or the dynamic mode)

SetModeBinning

bool SetModeBinning(bool HV, unsigned int binning, unsigned int vitesse, bool debug);

Description Turns the high speed mode On/Off (HV 1/0), sets the binning (1x1, 2x2, 4x4 or 8x8), sets the speed value (0,1 or 2) and turns the debug On/Off.

Parameters bool HV: turns high speed mode on (1) or off (0);
unsigned int binning: 0 for 1x1, 1 for 2x2, 2 for 4x4 and 3 for 8x8;
unsigned int vitesse: 0, 1 or 2
bool debug: turns the debug on (1) or off (0).

Return bool: true if the binning mode is correctly chosen.

See also

SetPOLA

void SetPOLA(unsigned int n, float val);

Description Sets the camera's polarization tensions.*see GetPolaLimits

Parameters unsigned int n:
float val:

Return None.

See also GetPolaLimits

SetTempsExposition

bool SetTempsExposition(double pose, bool *erreur);

Description Sets the exposure time to the FPGA.

Parameters double pose: exposure time (between 0.0001 and 165) in seconds;
bool *erreur: receives true if *pose* (exposure time) is out o its limits.

Return bool: true if the exposure time is correctly read to the FPGA.

See also

4.3 Communications class

This class (CarteADLink class) contains the functions related to the communications layer. For being the most low-level class, some of their functions are often called by the functions of the other two classes. This class contains the functions that control the ADlink communications board.

calculecarttype

bool calculecarttype (double *sigma1, double *sigma2, double *sigma3, double *sigma4);

Description Returns true when correctly calculate the standard deviation

Parameters double *sigma1:
double *sigma2:
double *sigma3 :
double *sigma4:

Return bool: true when correctly calculates the standard deviation.

See also

CarteADLink

CarteADLink(bool configurer=true);

Description Constructor of CarteADLink class.

Parameters bool configurer=true.

Return None.

See also

~CarteADLink

CarteADLink(bool configurer=true);

Description Destructor of CarteADLink class.

Parameters bool configurer=true.

Return None.

See also

---

ecrireport

bool ecrireport (char donnee);

Description Writes data in port.

Parameters char donnee: data to be written in port.

Return bool: true when correctly writes data in port.

See also

---

getNextImaqData

bool getNextImaqData(unsigned short *array, unsigned long sx, unsigned long sy, bool *done, int *nbimage);

Description Receive an image with size of *sx* in the X axis and *sy* in the Y axis to the pointer *array*.

Parameters unsigned short *array: pointer to store the received image;
unsigned long sx: horizontal image size;
unsigned long sy: vertical image size;
bool *done: settled with 1 when procedure is finished;
Int *nbimage:

Return bool: true when correctly receive to the pointer *array* an image with a size of *sx* in the X axis and *sy* in the Y axis.

See also

HasWorkingHardware

bool HasWorkingHardware();

Description Verifies if the ADlink Board is connected

Parameters None.

Return bool: true when recognizes that the ADlink Board is connected.

See also

---

lancerAcquisition

bool lancerAcquisition(unsigned long sx, unsigned long sy);

Description Operates the start of the image acquiring procedure.

Parameters long sx: horizontal image size;
unsigned long sy: vertical image size.

Return bool: true when correctly operate the start of the image acquiring procedure.

See also

---

savePLU

bool savePLU(int savePLUon);

Description Returns true when correctly communicates to the capturing process that it should read a flat-field image (savePLUon=1).

Parameters int savePLUon: when savePLUon=1, requests to read a flat-field image.

Return bool: true when correctly communicates to the capturing process that it should read a flat-field image.

See also

---

stopperAcquisition

void stopperAcquisition();

Description Operates the stop of the image acquiring procedure.

Parameters None.

Return None.

See also

---

5. Using the functions in C

5.1 Settings to be chosen before interacting with the dll

When programming in C or C++ environment, one should follow the next 4 steps before using the functions listed:

1) Before all, one should include the **CID_HL.lib** file in the additional dependences list of the C/C++ compiler.

2) Then it must be included in the code (#include) the files alims.h, controleur,h and carteADlink.h.

Important: If the high level program is written in C language one should write the #include commands inside an #ifdef following the next example:

```
Ifdef __cplusplus
#include "c:\alims.h"
Endif
```

Important: If the high level program is written in C language one should write all the code inside an #ifdef following the next example:

```
Ifdef __cplusplus
extern "C" {
Endif

        //CODE

#ifdef __cplusplus
}
#endif
```

3) Once this 2 steps are achieved, one should create a pointer to each of the classes before starting to call the functions:

```
#ifdef __cplusplus
CAlims *calim;
Ccontroleur *control;
CcarteADlink *ADlink;
#endif
```

4) Now the functions can be called in the following way:

```
calim->GetTemperature(n, temp);  //example
control->SetTempsExposition(pose, &erreurbool);   //example
```

5.2 Communication examples

Once the last 4 instructions are accomplished one can write a little program, like the one in the next example, to be sure that the communication with the CID_HL.dll functions is successful:

Example to take a single shot:

```
//To simplify the posterior usages of the classes Ccontroleur and CAlims
control = new Ccontroleur();
calim = new CAlims();


//To turn on the power supply
calim->SetBasseTension(1);


//To check the communications
control->SerialDownload();


//To turn on the peltier
calim->SetPeltier(1);
```

Once none of these callings returned an error code, that means that the hardware is functioning properly and that one can proceed with more specific functions

```
//To set the binning to a value that may be 1, 2, 4 or 8
unsigned int binning=1;


//To set the exposure time to a value between 0.00001 and 165 seconds
float exptime=1.5;


//To send the binning to the internal registers (the other input values doesn't
matter for the present example)
SetModeBinning(0, binning, 0,0);
SetModeTension(0,binning);


//To set the exposure time
SetTempsExposition(exptime);


//To set the image area (10000 is a default value)
SetArea(16/binning, 16/binning, 10000, 10000);
```

```
//To set automatic shutter operation (It opens and closes automatically)
automan=0;
obtu=0;


//To set automatic amplifier operation (It turns on and off automatically)
ampliautoman=0;
amplionoff=0;


//To set the amplifier and shutter modes of operation
SetAmplisObtu(ampliautoman, automan, amplionoff, obtu, 2)


//Starting the acquiring respecting the parameters demanded before
Stop(1);  //Stop(1) because this may not be the first image to be acquired
Start();


//To wait the acquiring process is finished
Sleep(tempexpo+2);


//In this cycle, if the binning is 1, the first part of an image is sent to the sdata array. Otherwise the image is totally sent to the sdata array
while (size == 0)
{
        size = control->GetNextImage(sdata,0,&fin_image, &nb_image);
}
size=0;


//In this cycle the second part of the image which the binning is 1 is sent to the sdata1 array    **
if (cam->binx==1)
{
        while((size == 0) && (!fin_image))
        {
                size = control->GetNextImage(sdata1,0,&fin_image, &nb_image);
        }
}


//To stop the camera once all the pixels of the image are gathered
Stop(1);
size=0;


totalpixels=(xpixels/binning)*(ypixels/binning);
```

```
//In this cycle the *p pointer is filled with the first part of the captured image if the binning is 1 and the total image for other values of the binning
for (k=0;k< totalpixels;k++)
{
        p[k] = sdata[k];
}


//In this cycle the *p pointer is filled with the second part of the captured image if the binning is 1   **
if (cam->binx==1)
{
        for (k= totalpixels /4;k<( totalpixels *0.75);k++)
        {
                p[k] = sdata[k];
        }
}
```

Once this point is reached, a single image was captured to the pointer *sdata.

Now the user should use this pointer to show or to save the image by respecting the rules of the particular program where he wants to deliver the image. This means that some programs expect a particular organization of the image array instead of a simple organization of the image array. If so, the *sdata array should be re-arranged with a cycle in order to respect that particular order of the program. In our case the pixels are organized in the image array as the following example:

SDATA array (binning=1):	**0,1,2,3,4**	**…**	**2047**
	2048,2049	**…**	**4095**
	.		**.**
	.		**.**
	.		**.**
	totalpixels-2048	**…**	**totalpixels-1**

totalpixels=(xpixels/binning) x (ypixels/binning)

** **Note:** When the binning of the captured image is 1, the intermediate buffer has not enough space to handle at once a 2048, 2048 image. The solution that the user will find to resolve this inconvenient should be similar to the one used in the example showed above. First we read the top half of the image and then the bottom half, which makes 2 images of 2048x1024 pixels able to fit in the buffer.

5.3 Error codes

Error code (example):

Figure 13: Error code example

Every time the user sees an error code like the one in the example above when calling one of the low level functions of the CID_HL.dll that may indicate a malfunctioning of:

- **Hardware**: - Power supply disconnected
  - Interface board disconnected
  - PCI board uninstalled
  - Optical fibre communications
  - Cooling central (when calling peltier functions) *****

- **Software**: - The functions may have been called in a bad order (to give an example, it is impossible to call SerialDownload before calling SetBasseTension or to call GetTemperature before calling SetPeltier)

*** Note:** One should not call any function to start the cooling procedure without being absolutely sure that the cooling central is functioning properly and that it is thermally connected to the head of the camera. Not respecting this important constraint of the camera may result in an irreparable damage of the camera.