

CamRecord Programmer Guide

CamRecord-Sprinter Series



CamRecord-Runner Series



CAMRECORD PROGRAMMER GUIDE

OVERVIEW.....	2
GETTING STARTED	2
Size/Speed/Exposure parameters handling.....	2
NETWORK CONNECTION.....	3
CONTROL SEQUENCES	5
Video live handling.....	5
Recording handling	5
Camera memory recording cinematic.....	6
SSD Discovery and Formatting Operations (only CamRecord-Sprinter)	7
SSD Read Operations (only CamRecord-Sprinter).....	7
SSD Write Operations (only CamRecord-Sprinter).....	8
CAMERA COMMANDS	9
Camera Information.....	9
CAM_GETSENSORCOLOR	9
CAM_GETSENSORBAYERTYPE	9
CAM_SETSENSORSYNCHRO [SYNC].....	9
CAM_GETFGSIZE	10
CAM_GETSERIALNUMBER.....	10
CAM_GETCAMERATYPE	10
CLOSE_CONNECTION	10
GETSERVER_VERSION.....	11
Sensor size Commands.....	11
CAM_SETSENSORSIZE [W] [H].....	11
CAM_GETSENSORSIZE [bMAX].....	11
CAM_GETSENSORLEFTTOP [W] [H].....	12
CAM_GETSENSORSCALE.....	12
CAM_SETSENSORENSITIVITY [GAIN]	12
CAM_GETSENSORENSITIVITY	13
Sensor speed Commands	13
CAM_SETSENSORSPPEED [SP].....	13
CAM_GETSENSORSPPEED	13
CAM_GETREALSENSORSPEED	13
CAM_GETSENSORSPPEEDRANGE.....	14
CAM_GETSENSORSPPEEDMAX [RES_X] [RES_Y] [0] [0] [0]	14
Sensor Exposure time Commands	14
CAM_SETSENSOREXPOSURE [EXP].....	14
CAM_GETSENSOREXPOSURE	15
CAM_GETSENSOREXPOSURERANGE.....	15
CAM_GETSENSOREXPOSUREMIN	15
CAM_GETSENSOREXPOSUREMAX	15
CAM_GETSENSOREXPOSUREMAX2 [RES_X] [RES_Y] [FPS] [0] [0] [0]	16
Live Commands.....	16
CAM_LIVESTART [Q].....	16
CAM_LIVESTOP	16
GET_IMAGE [T].....	17
CAM_WHITEBALANCEENABLED.....	17
CAM_SETWHITEBALANCE	17
CAM_RESETWHITEBALANCE	18
Memory Commands.....	18
CAM_FG_GETPARAM.....	18
CAM_FG_ISSEQUENCETHERE.....	19
CAM_PLAYSTART.....	19
CAM_PLAYSTOP	19
CAM_FG_GETFRAME_TM [iframenum] [bSnap] [b8] [bRAW].....	20

CAM_FG_SETUP [Numframes] [iTriggerSource] [TriggerPos].....	20
CAM_FG_RECORDSTART	21
CAM_FG_RECORDSTOP	21
CAM_FG_ABORTRECORD	21
CAM_FG_SOFTTRIGGER	21
CAM_FG_ENABLEFORCESOFTTRIGGER [bforce]	22
CAM_FG_SETAUTOTRIGGER [left] [top] [right] [bottom] [per]	22
CAM_FG_GETSTATUS.....	23
CAM_FG_GETMAXFRAMES.....	23
SSD COMMANDS (ONLY CAMRECORD-SPRINTER)	24
SSD Discovery Commands.....	24
CAM_SSD_GETSTATUS.....	24
CAM_SSD_GETSSDPARAM	24
CAM_SSD_FORMAT	25
SSD Read Commands	25
CAM_SSD_SETCURRENTSEQ	25
CAM_SSD_GETCURRENTSEQ	25
CAM_SSD_GETSEQHEADER.....	25
CAM_SSD_GETUSERINFO	27
CAM_SSD_EDITUSERINFO.....	27
CAM_SSD_PLAYSTART.....	27
CAM_SSD_PLAYSTOP	27
CAM_SSD_GETFRAMETM	28
SSD Write Commands	28
CAM_SSD_DELSEQ	28
CAM_SSD_SAVESEQ.....	29
GET SSD Saving Progression (command "REG_READ 1A80").....	29
Return codes from the Camera Commands.....	30
CAMRECORD-SPRINTER DEMO	31
Description:.....	31
1900DLL.DLL	32
Getting started	32
Function description	33
int CAM_IPScan ().....	33
int CAM_IPScanList (char *s).....	33
int CAM_Connect (char *s, int inode)	33
int CAM_DisConnect (int inode).....	33
int CAM_SendCommand (int inode, char *cmd, char *ret).....	34
int CAM_GetBuffer (int inode, char *buf, int sizeX, int sizeY).....	34
SSDLIB.DLL.....	35
int exSSD_IsDrive()	35
int exSSD_OpenDrive().....	35
int exSSD_GetSeqNumber(__int64 *n).....	35
int exSSD_SetCurrentSeq(__int64 seq).....	36
int exSSD_GetSeqHeader(__int64 seq, long *frames, int *resoX, int *resoY, long *fps, long *exptime, long *trigpos, int *mode, __int64 *realfpsHex, int *bFG8bits, unsigned char *Cam_SerialN, int *bSensorColor, int *sensorBayer, int *ssdTimeMarker_en, int *recordLeft, int *recordTop).....	36
int exSSD_GetSeqHeaderTrigTM(__int64 seq, long *frames, int *resoX, int *resoY, long *fps, long *exptime, long *trigpos, int *mode, __int64 *realfpsHex, int *bFG8bits, unsigned char *Cam_SerialN, int *bSensorColor, int *sensorBayer, int *ssdTimeMarker_en, int *recordLeft, int *recordTop, __int64 *trigTM, bool *wbCoeffAvailable, int *wbCoeff1, int *wbCoeff2, int *wbCoeff3, int *wbCoeff4).....	37
int exSSD_GetUserInfo(int *userInfoSize, unsigned char *userInfo).....	38
int __cdecl exSSD_EditUserInfo(char *userInfo)	38
int __cdecl exSSD_GetFrame(__int64 *timeMarker, __int64 frame, unsigned char *pimage)	38
int exSSD_DeINSeq(int _seqToDeINb)	38
SSDLib Example	39
ANNEX.....	41
Color Image Reconstruction.....	41

Overview

CamRecord Programmer Guide provides information to integrate CamRecord-Sprinter series cameras and CamRecord-Runner series cameras into customer specific software environment. Information about network connection and control sequences and control commands are given.

The software described in this CamRecord Programmer Guide is designed to work with Windows 7/10 32/64 bit and allows to operate one or more cameras.

Getting started

The CamRecord Programmer Guide is delivered with the following structure:

- 'redist' folder with the redistributable package to install vcredist.exe.
- 'doc' folder with the current document
- 'lib' folder with the following files: cam1900dll.dll, SSDLib.dll, cam1900dll.lib, SSDLib.lib (the dynamic libraries), cam1900dll.h, SSDLib.h (the header files)
- 'sample': a Microsoft Visual Studio 2013 sample project called 'CamRecord-Sprinter Demo' allows you to access to the main function of the camera

IMPORTANT:

- The cam1900dll.dll, cam1900dll.lib and cam1900dll.h must be copied in the same folder as your application executable.
- For the Microsoft Visual projects settings, the working directory has also to point to the same folder as your application executable.
- IP address handling is described on the user manual of the camera.

Size/Speed/Exposure parameters handling

To set a frame size, call the function CAM_SETSENSORSIZE. The nearest available format will be set and can be retrieved by calling CAM_GETSENSORSIZE.

After the CAM_SETSENSORSIZE call, the camera speed is set to the maximum available speed according to the current sensor size. The current sensor speed can be retrieved by using the function CAM_GETSENSORSPPEED. To set another speed, use the function CAM_SETSENSORSPPEED with a speed in the range returned by CAM_GETSENSORSPPEEDRANGE.

After the CAM_SETSENSORSPPEED call, the camera exposure time is set to the maximum available exposure time according to the current sensor speed. The current sensor exposure time can be retrieved by using the function CAM_GETSENSOREXPOSURE. To set another exposure time, use the function CAM_SETSENSOREXPOSURE with an exposure time less or equal than the value returned by CAM_GETSENSOREXPOSUREMAX.

Network Connection

Each camera (CamRecord-Sprinter and CamRecord-Runner controller) on the network can be addressed through its IP address via ports 5000 and 5500. To communicate with the CamRecord-Sprinter camera and CamRecord-Runner controller, ethernet command strings have to be send and an ethernet answer string is replied.



On the **Camera Commands** section prototypes of all commands are described.

Communication with the camera can be done by using:

Open a **socket** at the specific IP address and port for simple tests and debugging.



Use your **own code**.



Use Optronis examples program described on the **CamRecord-Sprinter Demo**



Use 1900dll.dll.



To access data on the SSD when removed from the camera and installed on a separate docking station the **SSDLIB.dll** has to be used.



Each command consists in a string formed by the command name [COMMAND_NAME] followed by its parameters [Input_ParamXX] and using space delimiter.

- Command prototype with no input parameter:
 - o "[COMMAND_NAME]"
- Command prototype with 1 input parameter:
 - o "[COMMAND_NAME] [Input_Param1]"
- Command prototype with N input parameters:
 - o "[COMMAND_NAME] [Input_Param1] [Input_Param2] ... [Input_ParamN]."

Each answer consists in a string formed by command return code [RET] followed by its returned parameters [Return_paramXX] and using space delimiter.

- Command answer prototype with no return parameter:
 - o "[RET]"
- Command answer prototype with 1 return parameter:
 - o "[RET] [Return_Param1]"
- Command answer prototype with N return parameters:
 - o "[RET] [Return_Param1] ... [Return_ParamN]"

Camera control summary:

- Open a socket to camera IP address using 1 of camera control ports (5000 or 5500)
- Send command
- Get and process command answer

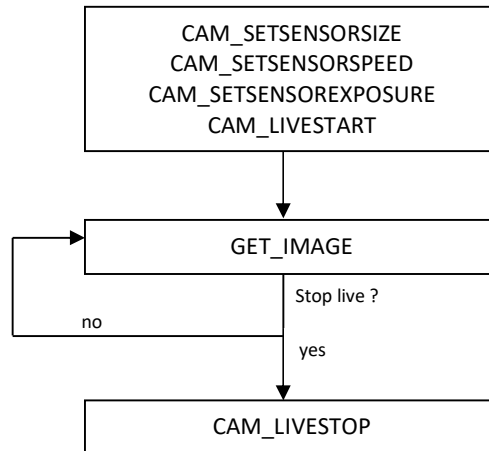
Examples:

- send_command("CAM_GETSENSORCOLOR"); => Get answer="0 0"
Answer follows prototype [RET] [COLOR], so answer means RET=0=Successfully operation and COLOR=0=monochrom sensor.
- send_command("CAM_SETSENSOREXPOSURE 1000"); => Get answer="0"
Answer follows prototype [RET], so answer means RET=0=Successfully operation.

Control Sequences

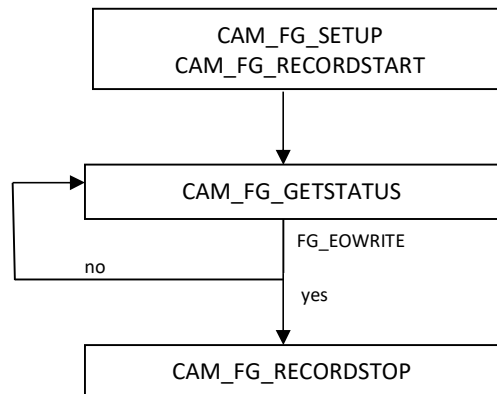
Video live handling

The typical video live schema can be given by:



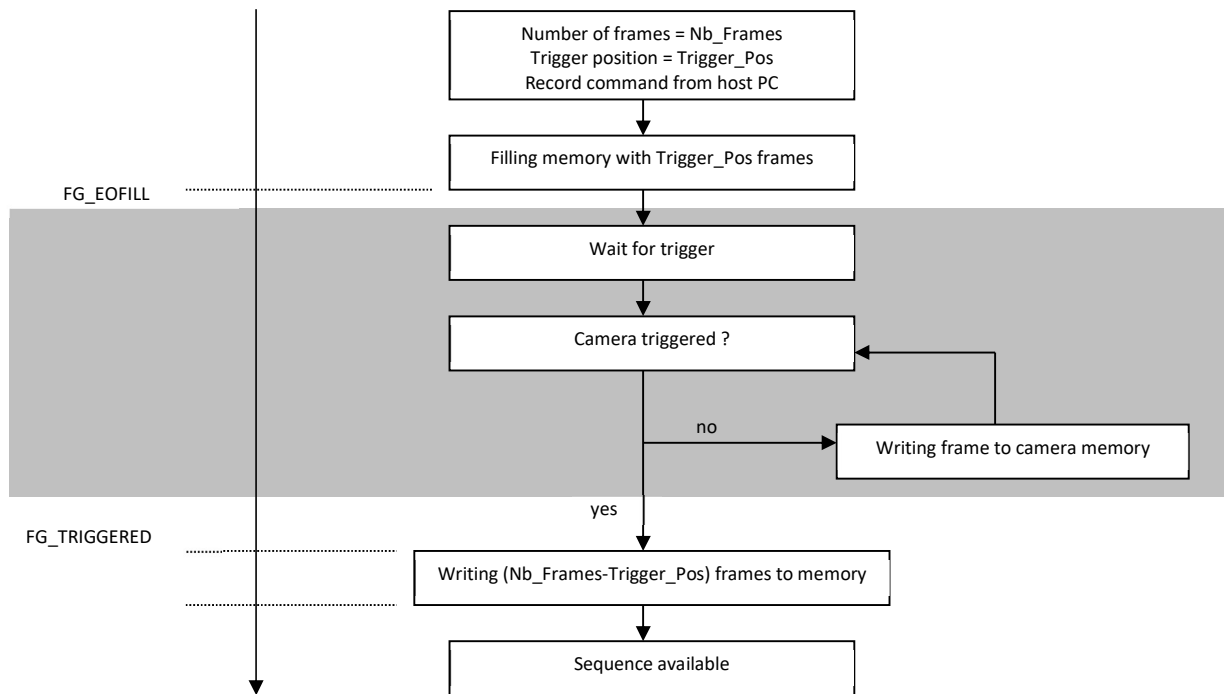
Recording handling

The typical recording schema can be given by:

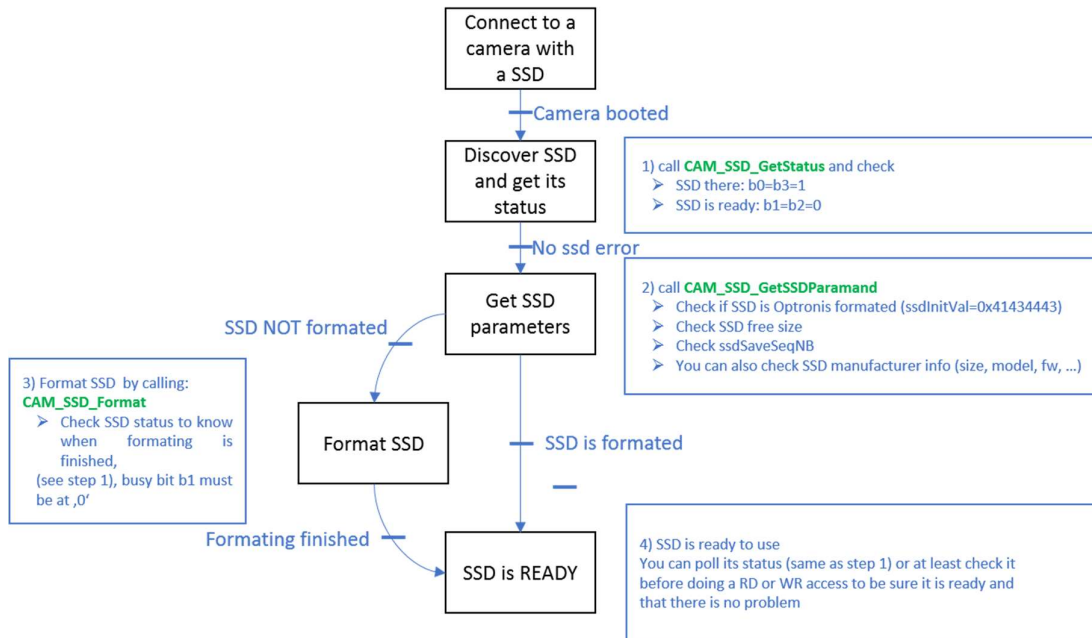


Camera memory recording cinematic

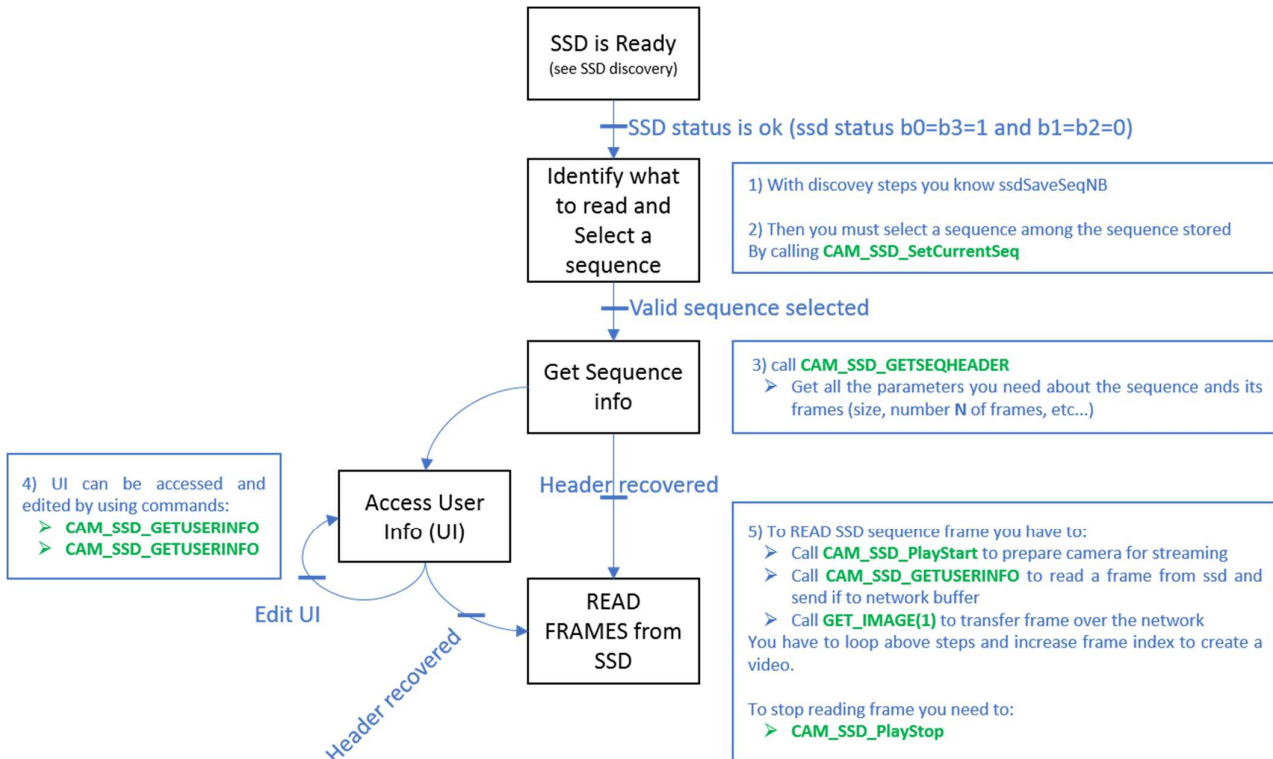
To acquire a sequence, it is useful to know the mechanism of the camera memory (see schema below). The first step is to define the number of frames to acquire (Nb_Frames) and the position of the trigger in the sequence (Trigger_Pos). The possibility to place the trigger position is very interesting when the user wants to analyze what happened before the trigger. This feature is available thanks to the ring memory of the camera. After the record command is sent by the host computer, camera is filling its memory with Trigger_Pos frames, so that user is able to analyze the event before the trigger. After that, the memory status is FG_EOFILL. The camera is waiting for a trigger and continuously writes frames into memory. When arriving at the end of the memory capabilities, the camera continues to write from the beginning of the memory: that is called ring memory. When the camera is triggered, (Nb_Frames-Trigger_Pos) frames are written to the memory. After that, the memory status is FG_EOWRITE and the sequence is available for reading.



SSD Discovery and Formatting Operations (only CamRecord-Sprinter)



SSD Read Operations (only CamRecord-Sprinter)



SSD Write Operations (only CamRecord-Sprinter)



Camera Commands

Camera Information

CAM_GETSENSORCOLOR

Return the sensor type of the camera.

Note:

Input parameters:

- none

Return string: [RET] [COLOR]

[COLOR] is 1 for a color sensor, else 0

CAM_GETSENSORBAYERTYPE

Return the sensor Bayer type of the camera.

Note:

Input parameters:

- none

Return string: [RET] [BAYER]

[BAYER] is

0 for GRBG bayer pattern

1 for GBRG bayer pattern

2 for RGGB bayer pattern

3 for BGGR bayer pattern

4 for no bayer pattern (monochrome)

CAM_SETSENSORSYNCHRO [SYNC]

Set the synchronization mode of the camera.

Note: By default, at camera power up, the synchronization mode is 0 (intern)

Input parameters:

- [SYNC]: 0 for internal synchronization, 1 for external synchronization.

Return string: [RET]

CAM_GETFGSIZE

Return the DDR internal memory size (in giga bytes) of the camera.

Note:

Input parameters:

- none

Return string: [RET] [SIZE]

[SIZE]: DDR internal camera memory in giga bytes.

CAM_GETSERIALNUMBER

Return the serial number of the camera.

Note:

Input parameters:

- none

Return string: [RET] [SN]

[SN]: camera serial number

CAM_GETCAMERATYPE

Return the type of the camera

Note:

Input parameters:

- none

Return string: [RET] [TYPE]

[TYPE] of the camera

11 CR-S3500 or Sprinter-HD

12 Sprinter-FHD

CLOSE_CONNECTION

Close the connection to the camera

Note:

Input parameters:

- none

Return string: [RET]

GETSERVER_VERSION

Returns the cr_server version

Note:

Input parameters:

- none

Return string: [RET] [VERSION]

- Version of the cr_server.

Sensor size Commands

CAM_SETSENSORSIZE [W] [H]

Set the current size of the camera.

Note:

Input parameters:

- [W]: sensor width
- [H]: sensor height

Return string: [RET]

CAM_GETSENSORSIZE [bMAX]

Return the current or maximal sensor size of the camera

Note:

Input parameters:

- [bMAX] : 1 if you want the function to return the maximal size or 0 if you want the current size.

Return string: [RET] [W] [H]

- [W]: current sensor width if [bMAX] is 0 else maximal sensor width.
[H]: current sensor height if [bMAX] is 0 else maximal sensor height.

CAM_GETSENSORLEFTTOP [W] [H]

Return the (left,top) position of the sensor active area of the camera node [node] for a (sx,sy) frame format

Note:

Input parameters:

- [W]: width of the frame
- [H]: height of the frame

Return string: [RET] [L] [T]

[L]: left corner of the active area
[T]: top corner of the active area

CAM_GETSENSORSSCALE

This command read the minimum, maximum and the increment for the width and height.

Note:

Input parameters:

- none

Return string: [RET] [X_MIN] [X_MAX] [X_INC] [Y_MIN] [Y_MAX] [Y_INC]

[X_MIN]: Minimum image width
[X_MAX]: Maximum image width
[X_INC]: Increment for the image width
[Y_MIN]: Minimum image height
[Y_MAX]: Maximum image height
[Y_INC]: Increment for the image height

CAM_SETSENSORSSENSITIVITY [GAIN]

Get the current active gain. Available at cr_server version 0.90.17

Note:

Input parameters:

- [GAIN]: Value for the gain, valid values are 1, 2 or 4.

Return string: [RET]

CAM_GETSENSORENSITIVITY

Get the current active gain.

Note:

Input parameters:

- none

Return string: [RET] [GAIN]

[GAIN]: Current gain possible values are 1, 2 or 4.

Sensor speed Commands

CAM_SETSENSORSPEED [SP]

Set the current speed of the camera.

Note:

Input parameters:

- [SP]: sensor speed in fps

Return string: [RET]

CAM_GETSENSORSPEED

Return the current speed of the camera.

Note:

Input parameters:

- none

Return string: [RET] [S]

[S]: Current sensor speed in fps.

CAM_GETREALSENSORSPEED

Return the estimated speed coming out from the camera (only used with internal synchronization).

Note:

Input parameters:

- none

Return string: [RET] [S]

[S]: Current estimated sensor speed in fps.

CAM_GETSENSORSPEEDRANGE

Return the current speed range of the camera.

Note:

Input parameters:

- none

Return string: [RET] [MIN] [MAX]

- [MIN]: lowest speed available with current sensor size
- [MAX]: highest speed available with current sensor size

CAM_GETSENSORSPEEDMAX [RES_X] [RES_Y] [0] [0] [0]

Return the maximum available speed for the current sensor size.

Note:

Input parameters:

- [RES_X]: Width of sensor.
- [RES_Y]: Height of sensor.
- [0]: Set always to 0.
- [0]: Set always to 0.
- [0]: Set always to 0.

Return string: [RET] [SPEED_MAX]

[SPEED_MAX]: Maximum frame rate.

Sensor Exposure time Commands

CAM_SETSENSOREXPOSURE [EXP]

Set the current exposure time of the camera.

Note:

Input parameters:

- [EXP]: sensor exposure time in us

Return string: [RET]

CAM_GETSENSOREXPOSURE

Return the current exposure time of the camera.

Note:

Input parameters:

- none

Return string: [RET] [EXP]

[EXP]: Current sensor exposure time in us.

CAM_GETSENSOREXPOSURERANGE

Return the current exposure time of the camera.

Note:

Input parameters:

- none

Return string: [RET] [MIN] [MAX]

[MIN]: lowest available exposure time with current speed and sensor size for the camera.

[MAX]: highest available exposure time with current speed and sensor size for the camera.

Current sensor exposure time in us.

CAM_GETSENSOREXPOSUREMIN

Return the lowest available exposure time.

Note:

Input parameters:

- none

Return string: [RET] [EXP_MIN]

[EXP_MIN]: Minimum exposure time.

CAM_GETSENSOREXPOSUREMAX

Return the highest available exposure time.

Note:

Input parameters:

- none

Return string: [RET] [EXP_MAX]

[EXP_MAX]: Maximum exposure time.

CAM_GETSENSOREXPOSUREMAX2 [RES_X] [RES_Y] [FPS] [0] [0] [0]

Return the maximum available exposure time for a given sensor speed and size.

Note:

Input parameters:

- [RES_X]: Width of sensor
- [RES_Y]: Height of sensor
- [FPS]: Speed of sensor
- [0]: Set always to 0.
- [0]: Set always to 0.
- [0]: Set always to 0.

Return string: [RET] [EXP_MAX]

[EXP_MAX]: Maximum exposure time.

Live Commands

CAM_LIVESTART [Q]

Start video live mode for the camera:

- Initialize transfer
- Send live mode command to camera

Note:

After call of this function, camera sends live frames from camera to PC.

To get a frame into a buffer, call the function GET_IMAGE.

To stop video live mode, call the function CAM_LIVESTOP.

Input parameters:

- [Q]: please always set it to 0

Return string: [RET]

CAM_LIVESTOP

Stop video live mode for the camera:

- Stops the transfer
- Send stop live mode command to camera

Note:

After call of this function, no more frames are sent from camera

Input parameters:

- none

Return string: [RET]

GET_IMAGE [T]

Get a 8-bits frame buffer from the camera.

Note: If live is started, the camera returns a live frame.
When you want to get a frame from an acquired sequence, you have to call CAM_FG_GETFRAME_TM before.
When you want to get a frame from a sequence stored in the internal SSD, you have to call CAM_SSD_GETFRAME_TM before.
Format:
For cameras with monochrome sensor, frame buffer contains intensity values.
For cameras with color sensor, frame buffer contains intensity values after white balance correction in case this correction is active. A Bayer reconstruction needs to be applied to calculate the RGB values.

Input parameters:

- [T]: please always set it to 1

Return string: [RET] [Px1y1] [Px2y1]... [PxWyH]

where Pxiyi is the pixel value of coordinates (xi,yi)

CAM_WHITEBALANCEENABLED

Returns if the white balance is activated or not.

Note:

Input parameters:

- none

Return string: [RET] [ENA]

[ENA]: 1 if white balance is activated in the camera else 0

CAM_SETWHITEBALANCE

Calculates the white balance coefficient for the camera (only for a color camera) and apply it to the camera.

Note: The white balance coefficients are calculated in a 20x20 window on the center of a camera frame. Camera needs to be directed onto a white reference surface in the center. White balance needs to be done after each power-up for firmware version till

Input parameters:

- none

Return string: [RET]

CAM_RESETWHITEBALANCE

Resets the white balance coefficient for the camera (only for a color camera).

Input parameters:

- none

Return string: [RET]

Memory Commands

CAM_FG_GETPARAM

Retrieves the acquired sequence parameters of the camera.

Note:

Input parameters:

- none

Return string: [RET] [frames] [resoX] [resoY] [fps] [exptime] [trigpos] [bFG8bits] [mode] [nbSeq] [realfps]

[frames]: Number of acquired frames

[resoX]: Width of frame

[resoY]: Height of frames

[fps]: frame rate

[exptime]: exposure time (in us)

[trigpos]: trigger position

[bFG8bits]: TRUE if memory is configured in 8-bits mode, FALSE if memory is configured in 10-bits mode

[mode]: 0

[nbseq]: 1

[realfps]: estimated frame rate coming out from the camera (only used with internal synchronisation)

CAM_FG_ISSEQUENCETHERE

Return if a sequence has been acquired into camera memory.

Note:

Input parameters:

- none

Return string: [RET] [there]

[there] 0 if a sequence has been successfully acquired into camera memory else 4.

CAM_PLAYSTART

Switch memory into the playing mode.

Note: Before calling the function CAM_Get_FG_Frame, camera memory has to be switched into the playing mode by calling CAM_PlayStart.

Input parameters:

- none

Return string: [RET]

CAM_PLAYSTOP

Exits memory from the playing mode.

Note:

Input parameters:

- none

Return string: [RET]

CAM_FG_GETFRAME_TM [iframenum] [bSnap] [b8] [bRAW]

Asks the camera to send a memory frame with its time marker.
To get the image, then call GET_IMAGE

Input parameters:

- [iframenum]: index of frame to retrieve from camera memory. Index must be between 1 and the number of acquired frames.
- [bSnap]: by default, bSnap is set to false. In that case, CAM_PlayStart has to be started before the call of the function. This is used to play a sequence for example: first, the initialization of the play mode is made by calling the function CAM_PlayStart; then, in a loop, you can play every frame by calling CAM_FG_GetFrame. If bSnap is set to true, you don't need to start the CAM_PlayStart function before. In that case, the CAM_FG_GetFrame can be called without the CAM_PlayStart initialization.
- [b8]: please set to 1
- [bRAW]: please set to 1

Return string: [RET] [ITimeMarker]

[ITimeMarker] : time marker buffer (unsigned 64-bits integer) expressed in microseconds.

If bit[63] = '1' IrigB is connected else no IrigB detected

If Irig-B is present or detected (only for CR600 with IRIGB option), the time marker buffer is composed like this :

- [63..60] status bits
- [59..44] Irig-B days buffer
- [43..36] Irig-B hours buffer
- [35..28] Irig-B minutes buffer
- [27..20] Irig-B seconds buffer
- [19..0] Irig-B μ seconds buffer

If Irig-B isn't present or detected, then the time marker buffer is composed like this :

- [63..60] status bits
- [59..0] Irig-B μ seconds buffer

CAM_FG_SETUP [Numframes] [iTriggerSource] [TriggerPos]

Setup of the camera memory recording parameters

Note: To call before CAM_FG_RecordStart

Input parameters:

- [Numframes]: number of frames to acquire
- [iTriggerSource]: trigger source of the recording.
 - 0 trigger source is software
 - 1 trigger source is extern TTL rising edge
 - 2 trigger source is extern falling edge
 - 3 trigger source is AutoTrigger
- [TriggerPos]: trigger position (must be between 1 and [Numframes]-1)

Return string: [RET]

CAM_FG_RECORDSTART

Start memory recording mode for the camera.

Note:

Input parameters:

- none

Return string: [RET]

CAM_FG_RECORDSTOP

Stop memory recording mode for the camera.

Note: To be called when recording is finished, i.e. when CAM_FG_GETSTATUS returns FG_EOWRITE.

Input parameters:

- none

Return string: [RET]

CAM_FG_ABORTRECORD

Abort memory recording mode and resets memory for the camera.

Note:

Input parameters:

- none

Return string: [RET]

CAM_FG_SOFTTRIGGER

Send software trigger command to camera when trigger source is ,Software'

Note:

Input parameters:

- none

Return string: [RET]

CAM_FG_ENABLEFORCESOFTTRIGGER [bforce]

Enable a software trigger to be active when another trigger mode has been activated: in the standard mode, if an external trigger is active, a software trigger is not taken into account. If this function is called with the parameter bforce set to true, the software trigger is always taken into account.

Note:

Input parameters:

- [bforce] : if set to 1, the software trigger is always taken into account (also if the trigger mode is set to external or auto). If bforce is set to 0, a software trigger is not taken into account when trigger mode is external or auto.

Return string: [RET]

CAM_FG_SETAUTOTRIGGER [left] [top] [right] [bottom] [per]

Send software auto-trigger parameters to camera when trigger source is ,Auto' and activates the auto-trigger process.

Note:

Input parameters:

- [left], [top], [right], [bottom]: the region of interest coordinates where the auto-trigger takes place and is processed is transmitted through the ([left],[top]) and ([right],[bottom]) corners
- [per] : percentage threshold when the camera has to trigger : if the mean value in the ROI varies less than [per] %, the trigger is activated.

Return string: [RET]

CAM_FG_GETSTATUS

Return memory status for the camera.

Note: See Annex for complete description of memory processing.

Input parameters:

- none

Return string: [RET] [status]

[status]:

2 memory is in idle mode (FG_IDLE, in that mode, camera waits for a command)
4 memory is currently reading a frame (FG_READ)
8 memory is currently writing a frame (FG_WRITE)
16 notice at end of read (FG_EOREAD)
32 notice at end of write (FG_EOWRITE)
64 memory is currently filling (FG_FILL)
128 notice at end of filling cycle (FG_EOFILL)
256 camera memory had been triggered (FG_TRIGGERED)

CAM_FG_GETMAXFRAMES

Return the maximal number of frames that can be acquired into camera memory with the current sensor size for the camera.

Note: See Annex for complete description of memory processing.

Input parameters:

- none

Return string: [RET] [maxframes]

[maxframes]: Maximal number of frames that can be acquired into camera memory with the current sensor size

SSD Commands (only CamRecord-Sprinter)

SSD Discovery Commands

CAM_SSD_GETSTATUS

Returns the SSD status.

Note: Only 8 bits are used, use mask 0x000000FF to extract them.

Input parameters:

- none

Return string: [RET] [ssdStatus]

[ssdStatus]: integer. Use mask 0x000000FF to access status (SSD status information is stored on bit0 to bit7. Bit description is below:

b0 = sata_init_ok_0 = SSD init ok

b1 = sata_busy_0 = SSD is used

b2 = sata_error_0 = an error occurred, reboot the camera

b3 = ssd_detected_0 = an SSD is HW detected

b4 = ssd_version_ok[0], not used now

b5 = ssd_format_ok[0], not used now

b6 = ssd_standby_0, not used now

b7 = reserved

b0=b3=1 ensure that the SSD is well connected and initialized

b1=b2=0 ensure that the SSD is ready for operations

CAM_SSD_GETSSDPARAM

Get SSD parameters.

Note: Available size can be computed with formula:

FreeSize_sect (in sector) = [SSDSize] - [SSD_NextFreeAddr]

FreeSize_Bytes = FreeSize_sect x 512

Input parameters:

- none

Return string: [RET] [SSDSize] [SSD_Serial] [SSD_Firmware] [SSD_Model] [SSD_InitVal] [SSD_FormatVersion] [SSD_SavedSeqNB] [SSD_NextFreeAddr]

[SSDSize] = unsigned long long

[SSD_Serial] = char*

[SSD_Firmware] = char*

[SSD_Model] = char*

[SSD_InitVal] = int, when then SSD is Optronis Formated this value = 0x41434443

[SSD_FormatVersion] = int

[SSD_SavedSeqNB] = int, it is the number of sequences stored on the SSD

[SSD_NextFreeAddr] = unsigned long long, see note

CAM_SSD_FORMAT

Format SSD to Optronis format.

Note: Once formatted, `ssdInitVal` takes value `0x41434443` (variable returned by command `Command_CAM_SSD_GetSSDParam`).

Input parameters:

- **[option]**: must be set to 0, reserved for future use.

Return string: [RET]

SSD Read Commands

CAM_SSD_SETCURRENTSEQ

Select a sequence on the SSD to access its parameter and read frames.

Note: `CAM_SSD_GETSSDPARAM` has to be called before to know how many sequences are stored on the SSD.

Be careful, sequence index starts at 0.

This command is mandatory because it initialize the SSD reading path in the camera.

Input parameters:

- **[sequenceID]**: integer, valid values are `[0..N-1]` if N sequences are stored in the SSD

Return string: [RET]

CAM_SSD_GETCURRENTSEQ

Returns current selected sequence. See `CAM_SSD_SETCURRENTSEQ` command to modify this value.

Note:

Input parameters:

- none

Return string: [RET] [selectedSequenceID]

CAM_SSD_GETSEQHEADER

Each SSD sequence has a header containing useful information about the sequence, frame characteristics, etc... Each header is an array of `WORD[256]`.

This command returns selected sequence header.

Note: Ex: "`CAM_SSD_GETSEQHEADER 0 1`" returns 4 bytes representing sequence frame number. If the sequence has 305.419.897 frames = `0x12345679`, return string will be "`12345679`". You will have to convert this string to an integer to recover the correct value.

Input parameters:

- [min] [max]

Return string: [RET] [selectedSequenceHeader]

selectedSequenceHeader = Word *, it is a string formed by the concatenation of N WORD (16bit). its size N = [max]-[min] * 16b

Header information:

```
// frame information
//-----
SSD_header[0] = (m_issdFrames);
SSD_header[1] = (m_issdFrames>>16);
SSD_header[2] = (resoX);
SSD_header[3] = (resoX>>16);
SSD_header[4] = (resoY);
SSD_header[5] = (resoY>>16);
SSD_header[6] = (fps);
SSD_header[7] = (fps>>16);
SSD_header[8] = (exptime);
SSD_header[9] = (exptime>>16);
SSD_header[10] = (trigpos);
SSD_header[11] = (trigpos>>16);
SSD_header[12] = (mode);
SSD_header[13] = (mode>>16);
SSD_header[14] = (nbseq);
SSD_header[15] = (nbseq>>16);
SSD_header[16] = (realfpsHex);
SSD_header[17] = (realfpsHex>>16);
SSD_header[18] = (realfpsHex>>32);
SSD_header[19] = (realfpsHex>>48);
SSD_header[20] = (bFG8bits);
SSD_header[21] = (bUF) + ((WORD)(bUS)<<1);

WORD 22 to 29 are Cam_SerialN

WORD 30:
// b0 = color sensor (see cam600dll.h: SENSOR_MONO=0 / SENSOR_COLOR=1)
// b2..b1= bayerType (see cam600dll.h: BAYER_GRGB=0 / BAYER_GBRG=1 / BAYER_RGGB=2 / BAYER_BGGR=3)
// b3 = m_issdTimeMarkerEN (0=disable / 1=enable)

// DDR and SSD sequence information
//-----
SSD_header[31] = ((m_RecordLeft & 0x0000FFFF));
SSD_header[32] = ((m_RecordLeft & 0xFFFF0000)>>16);
SSD_header[33] = ((m_RecordTop & 0x0000FFFF));
SSD_header[34] = ((m_RecordTop & 0xFFFF0000)>>16);
SSD_header[35] = stopposL;
SSD_header[36] = stopposM;

SSD_header[37] = (WORD)(m_bSSD_DDRSubSeqEn) | ((WORD)(m_bSSD_DDRAutoInc)<<1);

WORD 38..41 are trig frame time marker

// Reserved for future use
//-----
WORD 42..46 not used, available for new data

// User Information (can be retrieved using CAM_SSD_GETUSERINFO)
//-----
WORD 48 to 175 = 128 WORD

// SSD information
//-----
WORD 176 to 255
```

CAM_SSD_GETUSERINFO

User Information (**UI**) is a dedicated User area in the sequence header to allow user saving custom information (date, names, etc...). This area size as a maximum capacity of 256Bytes

This command is returning selected sequence User Information.

Note: User information are a

Input parameters:

- none

Return string: [RET] [UserInformation]

UserInformation = string containing UI, size max = 256 bytes

CAM_SSD_EDITUSERINFO

This command allows to edit selected sequence User Information (**UI**). Previous UI is lost and replaced by the new one.

Note:

Input parameters:

- **[NewUserInformation]** string, size max = 256 bytes

Return string: [RET]

CAM_SSD_PLAYSTART

Prepare camera for SSD streaming.

Note:

Input parameters:

- None

Return string: [RET]

CAM_SSD_PLAYSTOP

Stop SSD streaming and set camera and SSD ready for other operation than reading frames.

Note:

Input parameters:

- none

Return string: [RET]

CAM_SSD_GETFRAMETM

Stop SSD streaming and set camera and SSD ready for other operation than reading frames.

Note:

Input parameters:

- [iframenum] [bSnap] [b8] [bRAW]

Return string: [RET] [timemarker]

- **iframenum** = frame index/number you want to read. If the sequence has N frames, iframenum valid values are [0..N-1]
- **bSnap** = set to 0, reserved for compatibility / future use
- **b8** = set to 1, reserved for compatibility / future use
- **bRAW** = set to 1, reserved for compatibility / future use
- **timemarker** = (unsigned long long) frame n° iframenum time marker. The camera has a rolling 64b μ s counter and each frame gets a unique timemarker representing the end of exposure.

SSD Write Commands

CAM_SSD_DELSEQ

This commands allows to delete some sequence on the SSD to free space.

Note:

Only the last sequences are deleted.
 EX: 100 sequences are stored on the SSD
 „CAM_SSD_DelSeq 20“ command is send.(i.e. nbOfSeqToDelete=20)
 The last 20 saved sequences are deleted (sequences 80 to 99) and the SSD has 80 sequences left, (seq_0 to seq_79).
 If nbOfSeqToDelete = numberOfStoredSeq, then the SSD is empty.
 Alternatively, calling CAM_SSD_FORMAT also erase all sequences.

Input parameters:

- [nbOfSeqToDelete] must valid (\leq number of stored sequences)

Return string: [RET] [selectedSequenceID]

CAM_SSD_SAVESEQ

Save sequence from DDR to SSD.

While saving, no operation are possible, the user will have to wait to use the camera.

Note: Check SSD saving progression and SSD status to know If the SSD saving operations are finished.

Input parameters:

- [1stFrame] [LastFrame]

Return string: [RET]

This function allows to save a subpart of the sequence using input parameters.

Ex: if a sequence has N=1000 frames, you can:

- Save all the sequence on the SSD calling "CAM_SSD_SAVESEQ 0 999"
(1stFrame=0 and LastFrame=999)
- Save only a few framson the SSD calling "CAM_SSD_SAVESEQ N1 N2"
With $0 \leq N1 \leq N2 \leq N$
(1stFrame=100 and LastFrame=199 to save only 100 frames starting at frame 100)

GET SSD Saving Progression (command "REG_READ 1A80")

To get the SSD write progression you must call command "REG_READ 1A80"

This command returns the progression in %:

Return string: [RET] [progressionPercent]

Return codes from the Camera Commands

0	Successfully operation
1	-
2	Video live must be started first
3	Camera is busy
4	No sequence is available
5	No CamRecord is connected
6	-
7	The input parameter is out of range
8	GigE initialization error
9	An error has occurred when writing into the camera registers
10	The selected option is not available
11	IP has not been saved in flash
12	Frame corrupt
13	NIC and camera IPs don't match
14	-
15	No lens is connected
16	Error in lens command
17	Format is not available
18	Out of range (frame nb > frame max or sequence nb > seq max)
19	SSD full

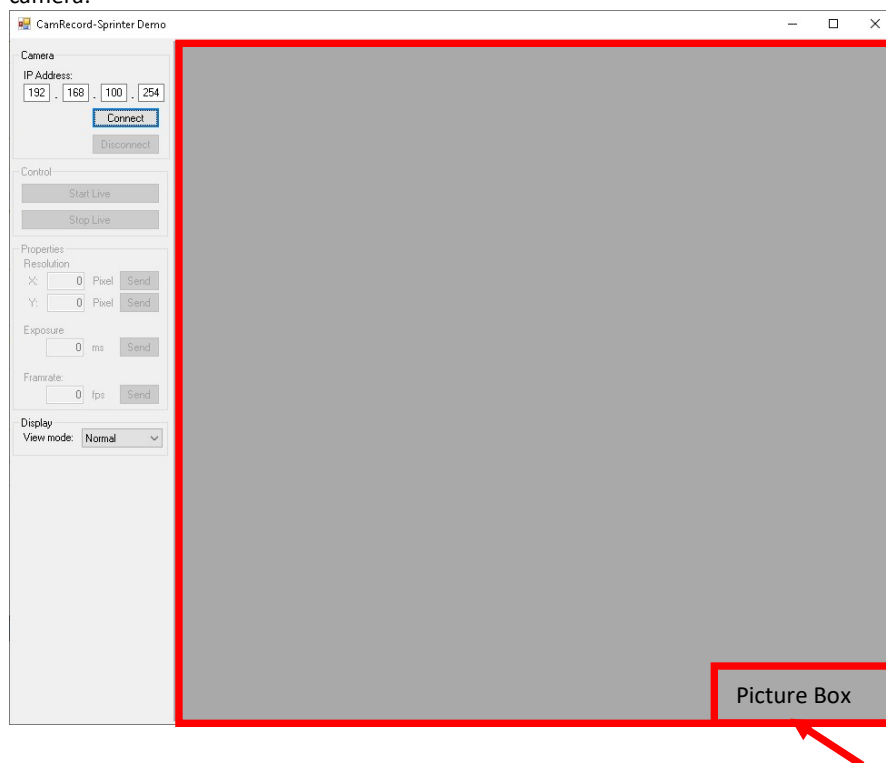
CamRecord-Sprinter Demo

The CamRecord-Sprinter Demo application is a Visual Studio 2013 C++ project. For color cameras, no color reconstruction is displayed. You will find the source code of CamRecord-Sprinter Demo in the "Sample" folder of the CamRecord programmer Kit. The Demo application uses the following resources:

- Microsoft .Net Framework 4.5
- Optronis Camera Class (Camera.cpp and Camera.h files)

Camera.cpp and **Camera.h** contain source code to show how to send and receive **Camera Commands**. This can be integrated into your application software.

Additionally, a demo application **CamRecordSprinterDemo.cpp** is provided to allow first operation with the camera.



Description:

Camera group:

Allows to connect and disconnect to the camera with the given IP address.

Control group:

Allows to Start and stop the live acquisition.

This controls are only available if the software is connected to the camera.

Properties group:

Allow to set camera properties like resolution, exposure time and framerate.

This controls are only available if the software is connected to the camera and the camera isn't in ,Live' mode.

Display group:

Allows to change the display mode in the Picture Box. ("Normal", "Center Image" and "Strech")

1900dll.dll

Getting started

The 1900dll.dll is another way to communicate with the camera. It allows to find a camera on a network, connect/disconnect a camera, send a command and receive an image buffer. In this part, we will give some basic way to handle with the 1900dll.dll. Please refer to “Camera Commands” section for detailed description of camera commands.

IMPORTANT:

The 1900dll.dll is not necessary to develop your own application. It is a help to find the camera(s) on a network if you don’t have the camera(s) IP address. If the camera(s) keeps the same IP you can connect directly by opening a socket at the specific IP address and port.

First step: Initialization of the Gigabit Ethernet Interface

The first step for your application is to scan the network to find your camera(s). That is done by calling the function CAM_IPScan. That function scans the network to find the attached cameras that are in the same subnet as the Network Interface Cards (NICs). Use CAM_IPScanList to get the list of the attached devices.

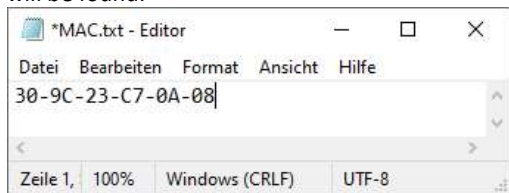
CamRecord-Runner particularity:

1900dll.dll is connecting to CamRecord-Runner systems also by using Ethernet connection. The same requirements as for CamRecord-Sprinter cameras related to the network apply. As long as standard controller for example Controller-RC1 is used, 1900dll.dll will connect to the CamRecord-Runner camera and no particular modifications are needed.

In case a non-standard controller is used, the MAC address (media access control address also called physical address) of the Ethernet interface used on this controller needs to be known.

To find controller MAC address, use Windows setup on the controller and go to the adapter settings of the adapter used. Click on “Details” and note MAC (physical) address.

Create a text file named “MAC.txt” on the same directory as 1900dll.dll software. Enter MAC address on the file. Only first 3 bytes are relevant. In this case 30-9C-23. Last 3 bytes can be set to 00-00-00. All CamRecord-Runner cameras with Ethernet adapters having the MAC address starting with these 3 bytes will be found.



This is also needed in case 1900dll.dll is run on a non-standard CamRecord-Runner controller itself. On the standard controllers (example: Controller-RC1) no MAC.txt file is needed.

Second step: Initialization of a camera on the network

After the first step previously described, each camera on the network can be addressed. Call CAM_Connect with a given IP address, you will get a welcome message from the camera.

Then the camera is ready to get commands from the host.

Third step: Communicate with the camera

To communicate with the camera, you only have to send ethernet command strings (the prototypes of the commands are described in the next section) by calling the function CAM_SendCommand. Then you get an ethernet answer string from the camera.

Function description

int CAM_IPScan ()

Scans the network for the CamRecord-Sprinter CR IP-compatible cameras (i.e. cameras that are in the same subnet as the NICs).

Note: Must be the first function called in the main program.

Input parameters:

None

Return value:

CAMDLL_OK.

int CAM_IPScanList (char *s)

Returns the list of the attached devices.

Note: Must be the first function called in the main program.

Input parameters:

None

Return value:

- [s]: string containing the list of cameras connected with IP and serial numbers. The format is the following:

[First Camera IP][First Camera Serial Number][Second Camera IP][Second Camera Serial Number]... [Last Camera IP][Last Camera Serial Number]

For example:

192.254.128.45|1902-ST-010|192.254.128.12|1902-ST-025|

int CAM_Connect (char *s, int inode)

Open a socket for a given camera IP address.

Note:

Input parameters:

- [s]: NIC IP address

- [inode]: integer containing the id of the camera associated to the IP address. User must choose an id associated the IP address. That id will be used in all other function to identify the camera.

Return value:

CAMDLL_OK.

int CAM_DisConnect (int inode)

Close a socket for a given camera id.

Note:

Input parameters:

- [inode]: camera id.

Return value:

CAMDLL_OK.

int CAM_SendCommand (int inode, char *cmd, char *ret)

Sends a command string a given camera. The command strings are described in the next section.

Note:

Input parameters:

- [inode]: camera id.
- [cmd]: command string

Return value:

- [ret]: string containing the answer of camera.

int CAM_GetBuffer (int inode, char *buf, int sizeX, int sizeY)

Gets the image buffer if an image command request has been called.

Note:

Input parameters:

- [inode]: camera id.
- [buf]: 8-bit buffer that contains the image datas
- [sizeX]: width of buffer to retrieve
- [sizeY]: height of buffer to retrieve

Return value:

CAMDLL_OK.

SSDLib.dll

A 2.5" SSD (Solid-State-Drive) is installed inside the camera and used to save sequences recorded in the volatile DDR memory. As long as SSD is installed inside the camera, **camera commands** allow access and control the data on the SSD.

However, the SSD can be removed from camera and be directly connected to a standard USB docking station (SATA III to USB 3 for example). As data are saved on the SSD by using an Optronis custom file format, you can't access directly data using standard Windows functions.

To see and extract video data without the camera, Optronis provides a Windows DLL "SSDLib.dll".

int exSSD_IsDrive()

Check if a SSD drive is connected to the PC.

Note:

Input parameters:

- None

Return value:

- SUCCESS if a SSD drive is connected to the PC
- ERROR_NO_SSD_DETECTED if no SSD drive is found

int exSSD_OpenDrive()

Open the SSD Drive.

Note:

Input parameters:

- None

Return value:

- SUCCESS if the SSD drive has been successfully open
- ERROR_NO_SSD_DETECTED if no SSD drive is found
- ERROR_SSD_NOT_OPEN if SSD drive opening process went wrong
- ERROR_NO_SEQUENCE_FOUND if no sequence is stored on the SSD drive

int exSSD_GetSeqNumber(__int64 *n)

Get the number of stored sequences.

Note:

Input parameters:

- None

Output parameters:

- __int64 *n: Number of sequence stored in the SSD drive

Return value:

- SUCCESS if no error.
- ERROR_SSD_NOT_OPEN if SSD drive opening process went wrong.

`int exSSD_SetCurrentSeq(__int64 seq)`

Select the current sequence.

Note:

Input parameters:

- `__int64 seq`: Index of the sequence to select. Index must be between 0 and the number of sequences - 1.

Return value:

- SUCCESS if no error.

`int exSSD_GetSeqHeader(__int64 seq, long *frames, int *resoX, int *resoY, long *fps, long *exptime, long *trigpos, int *mode, __int64 *realfpsHex, int *bFG8bits, unsigned char *Cam_SerialN, int *bSensorColor, int *sensorBayer, int *ssdTimeMarker_en, int *recordLeft, int *recordTop)`

Get the sequence header from the selected sequence.

Note:

Input parameters:

- `__int64 seq`: The sequence number to access

Output parameters:

- `long *frames`: The number of frames stored in the sequence
- `int *resoX`: The image width
- `int *resoY`: The image height
- `long *fps`: The framerate in frames per second
- `long *exptime`: The exposure time in us
- `long *trigpos`: The trigger position within the sequence
- `int *mode`: The acquisition mode
- `__int64 *realfpsHex`: the real framerate
- `int *bFG8bits`: The pixel depth
0 = 10bits
1 = 8bits
- `unsigned char *Cam_SerialN`: The camera serial number
- `int *bSensorColor`: The sensor color format
0 = Monochrome sensor
1 = Color sensor
- `int *sensorBayer`: The bayer pattern format
0 = GRBG
1 = GBRG
2 = RGGB
3 = BGGR
4 = None
- `int *ssdTimeMarker_en`: Internal parameter, please do not use
- `int *recordLeft`: Offset x
- `int *recordTop`: Offset y

Return value:

- SUCCESS if no error.

```
int exSSD_GetSeqHeaderTrigTM(__int64 seq, long *frames, int *resoX, int *resoY, long *fps, long *exptime, long *trigpos, int *mode, __int64 *realfpsHex, int *bFG8bits, unsigned char *Cam_SerialN, int *bSensorColor, int *sensorBayer, int *ssdTimeMarker_en, int *recordLeft, int *recordTop, __int64 *trigTM, bool *wbCoeffAvailable, int *wbCoeff1, int *wbCoeff2, int *wbCoeff3, int *wbCoeff4)
```

Get the sequence header from the selected sequence.

Note:

Input parameters:

- __int64 seq: The sequence number to access

Output parameters:

- long *frames: The number of frames stored in the sequence
- int *resoX: The image width
- int *resoY: The image height
- long *fps: The framerate in frames per second
- long *exptime: The exposure time in us
- long *trigpos: The trigger position within the sequence
- int *mode: The acquisition mode
- __int64 *realfpsHex: the real framerate
- int *bFG8bits: The pixel depth
 - 0 = 10bits
 - 1 = 8bits
- unsigned char *Cam_SerialN: The camera serial number
- int *bSensorColor: The sensor color format
 - 0 = Monochrome sensor
 - 1 = Color sensor
- int *sensorBayer: The bayer pattern format
 - 0 = GRBG
 - 1 = GBRG
 - 2 = RGGB
 - 3 = BGGR
 - 4 = None
- int *ssdTimeMarker_en: Internal parameter, please do not use
- int *recordLeft: Offset x
- int *recordTop: Offset y
- __int64 *trigTM: Time marker reference
- bool *wbCoeffAvailable: White balance coefficients available
- int *wbCoeff1: White balance coefficient 1
- int *wbCoeff2: White balance coefficient 2
- int *wbCoeff3: White balance coefficient 3
- int *wbCoeff4: White balance coefficient 4

Return value:

- SUCCESS if no error.

`int exSSD_GetUserInfo(int *userInfoSize, unsigned char *userInfo)`

Get the user information.

Note:

Input parameters:

- int *userInfoSize: Pointer to the buffer size

Output parameters:

- unsigned char *userInfo: Pointer to the buffer

Return value:

- SUCCESS if no error.

`int __cdecl exSSD_EditUserInfo(char *userInfo)`

Edit the user information.

Note:

Input parameters:

- char *userInfo:

Return value:

- SUCCESS if no error.

`int __cdecl exSSD_GetFrame(__int64* timeMarker, __int64 frame, unsigned char *pimage)`

Get a frame on the current selected sequence by exSSD_SetCurrentSeq.

Note: __int64 frame: Index of frame to retrieve from the sequence. Index must be between 0 and the number of acquired frames -1

Input parameters:

- int *userInfoSize: Pointer to the buffer size

Output parameters:

- __int64* timeMarker: Time marker of the image in μ s
- unsigned char *pimage: Pointer to the destination image buffer

Return value:

- SUCCESS if no error.

`int exSSD_DeINSeq(int _seqToDelNb)`

Delete N sequence from the end of the SSD.

Note:

Input parameters:

- int _seqToDelNb: Number of sequences to delete

Return value:

- SUCCESS if no error.

SSDLib Example

```

#include "SSDLib.h"
#include <memory.h>

int main(void)
{
    int dll_result(SUCCESS);
    __int64 sequences(0);
    int resolution_x(0), resolution_y(0), selected_seq(0);
    long frames(0);

    // Search for SSD drive
    dll_result = exSSD_IsDrive();
    if(dll_result == SUCCESS)
    {
        // Open SSD drive
        dll_result = exSSD_OpenDrive();
    }
    if(dll_result == SUCCESS)
    {
        // Get number of stored sequences
        dll_result = exSSD_GetSeqNumber(&sequences);
    }
    if(dll_result == SUCCESS)
    {
        if(sequences > 0)
        {
            // Select the first sequence
            dll_result = exSSD_SetCurrentSeq(selected_seq)
        }
    }
    if(dll_result == SUCCESS)
    {
        // Read sequence header from the first sequence
        long fps(0), exposure_time(0), trigger_position(0);
        __int64 real_fps_hex(0);
        int mode(0), bitdepth8(0), offset_x(0), offset_y(0);
        int color_mode(0), bayer_format(0), time_marker_en(0);
        unsigned char serial_number_camera[128];
        __int64 time_marker_reference;
        int wb_c1(0), wb_c2(0), wb_c3(0), wb_c4(0);
        bool wb_available(false);

        memset(serial_number_camera, 0, 128 * sizeof(unsigned char));

        dll_result = exSSD_GetSeqHeaderTrigTM(selected_seq,
                                                &frames,
                                                &resolution_x,
                                                &resolution_y,
                                                &fps,
                                                &exposure_time,
                                                &trigger_position,
                                                &mode,
                                                &real_fps_hex,
                                                &bitdepth8,
                                                serial_number_camera,
                                                &color_mode,
                                                &bayer_format,
                                                &time_marker_en,
                                                &offset_x,
                                                &offset_y,
                                                &time_marker_reference,
                                                &wb_available,
                                                &wb_c1,
                                                &wb_c2,
                                                &wb_c3,
                                                &wb_c4);
    }

    if(dll_result == SUCCESS)

```

```
{
    unsigned char* image_buffer(NULL);
    __int64 timme_marker(0), image_number(0);

    if(frames > 0)
    {
        // Create image buffer
        image_buffer = new unsigned char[resolution_x * resolution_y * sizeof(unsigned char)];

        // Read the first image of the sequence
        result = exSSD_GetFrame(&timme_marker, image_number, image_buffer);

        // Process the image data
        // ...

        delete[] image_buffer;
    }
}
```

Annex

Color Image Reconstruction

For CamRecord-Sprinter cameras with color sensor, frame buffer contains intensity information as provided by sensor with Bayer filter. White balance might be applied in case this is activated but RGB values for each pixel need to be calculated. The following description is considered to be an example how the RGB image can be calculated. As public libraries are used, command names or source locations might change at any time. Alternative solutions can be used.

Free library **OpenCV** is used and needs to be downloaded. <https://opencv.org/>

Used **CAM_GETSENSORBAYERTYPE** to get the Bayer type of the sensor.
It should return 0 for GRBG Bayer pattern type for the Sprinter-FHD.

Use the **cvtColor** function of OpenCV library

cvtColor(cv::Mat src, cv::Mat dst, int code)

Converts Bayer pattern image into Color coded image

Parameters:

[cv::Mat src] input image, 8-bit unsigned

[cv::Mat dst] output image

[int code] color space conversion code

For the Sprinter-FHD camera code should be CV_BayerGR2RGB

(see https://docs.opencv.org/3.4.15/d8/d01/group_imgproc_color_conversions.html#ga4e0972be5de079fed4e3a10e24ef5ef0)

Example:

```
cv::Mat src;
```

```
cv::Mat dst;
```

```
src.create(image_height, image_width, CV_8U); // Copy image buffer to src object // ...
```

```
cvtColor(src, dst, CV_BayerGR2RGB);
```

You will find more information about cvtColor here:

https://docs.opencv.org/3.4.15/d8/d01/group_imgproc_color_conversions.html#ga397ae87e1288a81d2363b61574eb8cab