pco.sdk

SDK-Description

This document describes the application programming interface to pco cameras compatible to pco.sdk.

Table of contents:

A. Command Structure

This document describes the commands for controlling pco cameras at the SDK/API level, which are compatible to this SDK (e.g. pco.1300, pco.1400, pco.pixelfly usb, pco.1600, pco.2000, pco.4000, pco.1200, pco.dimax, pco.edge). Further explanations appear as needed for the commands, settings and mode configurations.

1 General

- Conventions in this manual

1.1 Conventions

The following typographic conventions are used in this manual:

bold: Functions, procedures or modes used	**get camera type**
[words in brackets]: Possible values or "states" of the described functions	[run]
ALL CAPITAL WORDS: Logical or boolean values such as TRUE, FALSE, ON, OFF, 0, 1, RISING, FALLING, HIGH, LOW	TRUE
<words in arrows>: Names of hardware input / output signals	<acq enbl>

2 SDK Overview and function description

The SDK consists of header files for compiling and lib files for linking your project. Visual Studio 2008 has been used to create the SDK-DLLs and it is recommended, but is not restricted, to use the same development environment. There exist other samples in different languages, e.g. C++, Delphi, C#, Visual Basic and Visual Basic dot net. To access the camera you have to call functions inside the SC2_Cam.dll, which should reside in your application directory. The SC2_Cam.dll depends on the SC2_1394.dll, SC2_GigE.dll, SC2_USB.dll or corresponding Camlink-dlls, which should be installed during the driver installation.

The SDK is ***not thread save***. Thus it is not possible to setup two or more threads getting images with different settings and sizes. However threading is possible in case the developer takes care for correct thread synchronization, e.g. one thread changes the settings and a second one grabs the images. In this case the second thread has to stop grabbing till the first one has changed the settings and has executed an ARM. In principle the order of commands shown in the typical implementation must be met.

The SDK consists of the following files:

- Header: SC2_CamExport.h, SC2_SDKStructures.h
- Library: SC2_Cam.lib, SC2_Cam.dll

The SC2_CamExport can be used to access the functions, while SC2_SDKStructures is a help to get easy access to camera data.

Please carefully fill in all wSize parameters inside the SDK-structures. See section '5 Typical Implementation' ***for an easy entry. Mandatory functions are marked with an asterisk (*).***

All camera control commands are encapsulated into SDK functions inside the SC2_Cam.dll. Thus SC2_cam.dll can be seen as a wrapper for all camera control commands. Additionally there exist some SDK functions to control the buffer handling for image transfer from the camera to the PC.

While debugging with the GigE interface, it might be possible to get error 0xA0322005, which means 'timeout'. This is caused by a long break between two debugging steps (usually > 65s). Single stepping stops all threads executed till the next step. This disables the sc2_gige.dll thread to send heartbeat messages to the camera. The camera will generate a timeout due to lost connection. In this case please stop and restart your debug session. Keep in mind that you'll have to step quickly through your code while debugging with a GigE interface.

Logfile: All commands sent to the SC2_Cam dll can be reported to a file. If you like to enable logging, create a file called ‘SC2_Cam.log’ in the same directory as SC2_Cam.dll can be found. SC2_Cam.log will be overwritten with each session start. In case the user likes to keep older sessions, name the logfile SC2_Cam_a.log. This will append further sessions. After ending your logging session please do not forget to delete the SC2_Cam(_a).log file, because it may cut down performance.

The total set of SDK-commands is subdivided into seven sections:

- Camera (General)
- Image Sensor
- Timing
- Storage
- Recording
- Image Read
- API-Management

2.1 Camera

This section contains general instructions to control the camera and to request information about the camera:

- Request camera type, hardware/firmware version, serial number, interface type
- Request camera status (warnings, errors etc.)
- Reset all settings to default values
- Initiate self test procedure
- Get camera / power supply temperature

2.2 Image Sensor

This group contains complete image sensor control instructions and instructions to request information about the sensor. These are:

- Get Camera description: sensor type, standard resolution, extended resolution, dynamic resolution (bit), delay and exposure times…
- Set/request sensor format: [standard] / [extended].
- Set/request ROI settings.
- Set/request binning settings.
- Set/request pixel rate (frequency for shifting the pixels out of the sensor shift registers).
- Set/request conversion factor (gain) settings.
- Set/request double image mode (expose two images one after another immediately).
- Set/request ADC mode (use one or two ADCs for digitizing the pixel data of the sensor).
- Set/request IR sensitivity setting (ON/OFF).
- Set/request cooling set point temperature.
- Set/request Offset Mode.

2.3 Timing

This group contains all available commands for control of imaging process timing:

- Set / request delay and exposure time (timebase, timetable) for taking images.
- Set / request trigger mode for exposures: [auto trigger], [force trigger], [extern edge triggered], [extern exposure pulse trigger][1]. Controls the usage of the <exp trig> control input. See below for a detailed description of the trigger modes.
- Force trigger: this software command starts an exposure if the trigger mode is in the state [auto trigger], [force trigger] or [extern edge triggered]. If in [extern exposure pulse trigger] mode nothing happens.

- Request busy status: A trigger is ignored if the camera is still busy (exposure or readout). In case of [force trigger] command, the user may request the camera's busy status in order to generate a valid [force trigger] command.
- Set / request power down time (threshold value, which becomes available in case of exposure times longer than 1s)
- Read control input (<exp trig>): read TRUE or FALSE level of external control input[2] (<control in>).

Notes:

(1) Edge type (FALLING edge / RISING edge) as well as the electrical sensitivity (trigger level) are selected by DIP switches at the power supply unit near the trigger input(<control in>). In double image mode, the first exposure time is affected by the trigger commands. The duration of the second exposure is always given by the readout time of the first image.

(2) If the DIP switch shows a RISING edge, then the HIGH level signal is TRUE and the LOW level signal is FALSE. If the DIP switch shows a FALLING edge, then the HIGH level signal is FALSE and the LOW level signal is TRUE.

The following table shows how the different trigger modes work:

Trigger mode	Operation Description
auto trigger	A new image exposure is automatically started best possible compared to the readout of an image. If a CCD is used and images are taken in sequence, then exposures and sensor readout are started simultaneously.
software trigger	An exposure can only be started by a **force trigger** command.
extern exposure & software trigger	A delay / exposure sequence is started at the RISING or FALLING edge [1] of the trigger input (<control in>) or by a [force trigger] command.
extern exposure control	The exposure time is defined by pulse length at the trigger input (<control in>). The delay and exposure time values defined by the **set / request delay and exposure** command are ineffective.

2.4 Storage

This set contains all commands needed for controlling the memory and storage process.

The total camera memory is divided into four segments (similar to partitions on hard discs).

- Request RAM size (pages) and page size (pixels)
- Request / set RAM segment size in pages
- Clear RAM segment
- Get / set active RAM segment

Note:
Consistency check (e.g., in order to avoid buffer overlap) must be done by the application software!
Each segment also contains information about the image settings (ROI / binning etc.) for the images stored within this segment (all images must have the same format).

2.5 Recording

- Set / request storage mode: [recorder mode] / [FIFO buffer mode] (see insert box 2.5.1 for further explanations)
- Set / request recorder submode: [sequence] / [ring buffer] (see insert box 2.5.2 for further explanations)
- Set / request recording state: [run] / [stop] (see insert box 2.5.3 for further explanations)
- Arm: prepare camera for recording command
  This function is necessary before a new recording (**set recording** = [run]) command is released. This function takes the delay, exposure, triggering, recorder mode (etc.) settings, compiles them and prepares the camera to start immediately when a start of recording (**set recording** = [run]) is performed.
- Set / request acquire mode: [auto] / [external], controls the usage of the <acq enbl> control input
  - [auto]: the external control input <acq enbl> is ignored
  - [external]: the external control input <acq enbl> is a static enable signal of images. If this input is TRUE, then exposure triggers are accepted and images are taken. If this signal is set FALSE, then all exposure triggers are ignored and the sensor readout is stopped.
- Read control input (<acq enbl>): read TRUE or FALSE level of external control input(1) (<control in>)
- Set date / time
- Set / request timestamp mode

Notes:
Active (TRUE) level (LOW/HIGH) as well as the electrical sensitivity is selected by DIP switches at the power supply unit near the acquire enable input(<acq enbl>).

(1) If the DIP switch shows ⊓ then the HIGH level signal is TRUE and the LOW level signal is FALSE. If the DIP switch shows ⊔ then the HIGH level signal is FALSE and the LOW level signal is TRUE.

Box 2.5.1

recorder mode	FIFO buffer mode
• images are recorded and stored within the internal camera memory (camRAM) • "live view" transfers the most recent image to the PC (for viewing / monitoring) • indexed or total image readout after the recording has been stopped	• all images taken are transferred to the PC in chronological order • camera memory (camRAM) is used as a huge FIFO buffer to bypass short bottlenecks in data transmission. If buffer overflows, the oldest images are overwritten. In **FIFO buffer mode,** images are send directly to the PC interface (FireWire, USB …) like a continuous data stream. Synchronization is done with the interface.

Box 2.5.2

recorder submode: sequence	recorder submode: ring buffer
• Recording is stopped when the allocated buffer is full.	• Camera records continuously into ring buffer. • If the allocated buffer is full, the older images are overwritten. Recording is stopped by software command.

Box 2.5.3

Recording: [run] / [stop]

The recording command controls the camera status. If the recording state is [run], images can be released by **exposure trigger** and **acquire enable**. If the recording state is [stop] all image readout or exposure sequences are stopped and the sensors (CCDs or CMOS) are running in a special idle mode to prevent dark charge accumulation.

The recording state has the highest priority compared to functions like **acquire enable** or **exposure trigger**.

The recording state is started by:

- software command: **Set recording** = [run]

The recording state is stopped by:

- powering on the camera
- software command: **Set recording** = [stop]
- software command: Reset all settings to default values.
- in recorder submode = [sequence], if the buffer overflows.

2.6 Image Read

- Request image settings for this segment (ROI, binning, horizontal x vertical resolution)
- Request number of images in segment

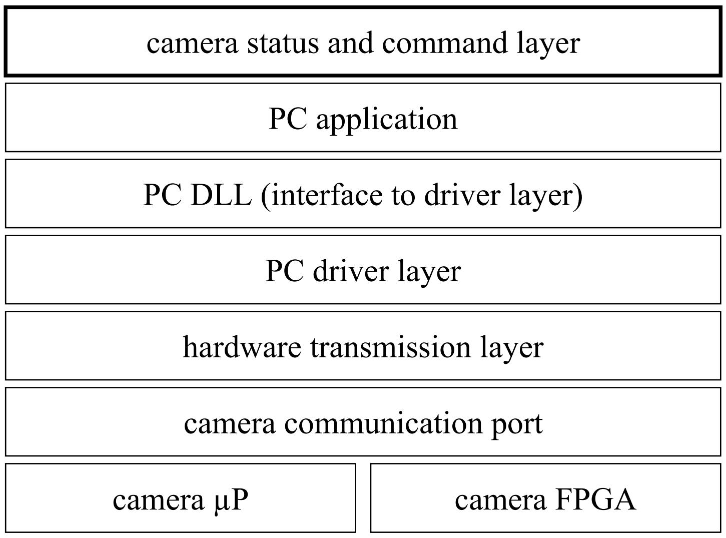
The image readout is part of the API-management commands. If the camera is in recording state the PCO_AddBuffer command must be used. If the camera is not in recording state, the PCO_GetImage command must be used.

2.7 API Management

- Open and close the camera device
- Buffer management (allocate, free, add buffer, get status) and image access
- Device availability during runtime

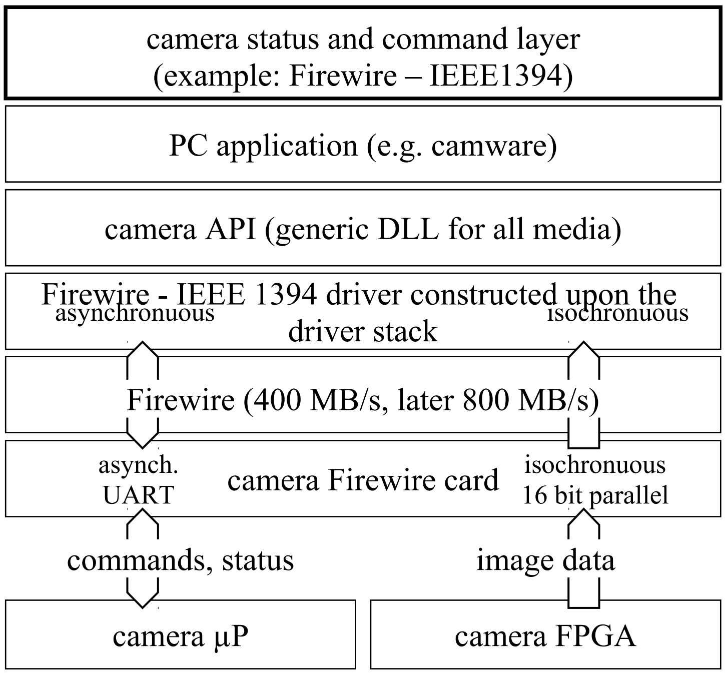
B. Implementation Details

3 Communication Layers


camera status and command layer
PC application
PC DLL (interface to driver layer)
PC driver layer
hardware transmission layer
camera communication port
camera µP
camera FPGA

The application software running on the PC is able to send commands to the camera as well as request status information from the camera. There is also a channel for transmitting image data.

The DLL links the application software to the camera device driver layer. Commands sent to the driver should be common for all camera versions as well as for all types of interfaces (FireWire, USB etc.). Thus, the sdk and driver converts the commands to the used hardware port.


camera status and command layer (example: Firewire – IEEE1394)
PC application (e.g. camware)
camera API (generic DLL for all media)
Firewire - IEEE 1394 driver constructed upon the driver stack
asynchronuous
isochronuous
Firewire (400 MB/s, later 800 MB/s)
asynch. UART
camera Firewire card
isochronuous 16 bit parallel
commands, status
image data
camera µP
camera FPGA

Example of Layer structure applied to the FireWire interface between PC and camera.

Commands and status information are sent between the PC and the camera µP, the image data are transferred by the camera FPGA to the FireWire interface.

Interfaces, which will be implemented, are FireWire – IEEE1394, Camera Link, USB 2.0 and Ethernet (TCP/IP). The latter is somewhat different since within the PC, the layers up to the application layer are already implemented within the operating system.

The communication port, that is the path from the PC driver layer down, separates the data path into channels for commands, status messages and image data.

4 SDK function sections

Group codes:
General Control / Status
Image Sensor
Timing Control
Storage Control
Recording Control
Image Buffer Data
API managament

4.1 General Control / Status

This function group defines general access to the camera, especially camera type, sensor type, status, temperatures, etc.
Overview:

Command:
PCO_GetGeneral
PCO_GetCameraType
PCO_GetCameraHealthStatus
PCO_ResetSettingsToDefault
PCO_InitiateSelftestProcedure
PCO_GetTemperature
PCO_GetCameraName
PCO_GetFirmwareInfo

4.1.1 PCO_GetGeneral

Request all info contained in the following function descriptions, especially:

- camera type, hardware/firmware version, serial number etc.
- Request the current camera and power supply temperatures.

PCO_ResetSettingsToDefaultand PCO_InitiateSelftestProcedure functions are not called. Please fill in all wSize parameters, even in embedded structures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetGeneral(HANDLE ph, PCO_General* strGeneral)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- PCO_General* strGeneral: Address of a PCO_General structure.

Please fill in all wSize parameters, even in embedded structures.

The structure will be filled with the parameters followed by this function description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.1.2 PCO_GetCameraType

Request camera type, hardware/firmware version, serial number etc. Please fill in all wSize parameters, even in embedded structures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetCameraType(HANDLE ph, PCO_CameraType* strCamType)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- PCO_CameraType* strCamType: Address of a PCO_CameraType structure.

Please fill in all wSize parameters, even in embedded structures.

The structure will be filled with the following parameters:

- camera type as word, see table below
- camera sub type as word
- serial no. as long word.
- hardware version as long word, where the most significant word is the version no. and the lower significant word is the revision no. (ver.rev e.g. 2.01 = [0x00020001])
- firmware version as long word, where the most significant word is the version no. and the lower significant word is the revision no. (ver.rev e.g. 2.01)
- interface type as word, see table below

Camera Type codes:					
pco.1200 hs	0x0100	pco.4000	0x0260	pco.edge	0x1300
pco.1300	0x0200	pco.1400	0x0830	pco.(future 1)	0xyyyy
pco.1600	0x0220	pco.newgen	0x0840	pco.(future 2)	0xyyyy
pco.2000	0x0240	pco.dimax	0x1000	pco.(future 3)	0xyyyy

See sc2_defs.h for more camera types.

Interface Type codes:			
FireWire	0x0001	Ethernet	0x0004
Camera Link	0x0002	Serial Interface	0x0005
USB	0x0003	Reserved	0x0006

Note: The fact that a special interface type is mentioned here does not guarantee its availability!

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.1.3 PCO_GetCameraHealthStatus

Request the current camera health status: warnings, errors.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetCameraHealthStatus(HANDLE ph, DWORD* dwWarn, DWORD* dwErr, DWORD* dwStatus)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD dwWarn: Address of a DWORD to get warning messages.
- DWORD dwErr: Address of a DWORD to get error messages.
- DWORD dwStatus: Address of a DWORD to get the status.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

The input pointers will be filled with the following parameters:

- Warnings encoded as bits of a longword. Bit set indicates warning, bit cleared indicates that the corresponding parameter is OK. See table on the next page.
- System errors encoded as bits of a longword. Bit set indicates error, bit cleared indicates that the corresponding status is OK. See table on the next page.
- System Status encoded as bits of a longword. For bit meanings, see the table on the next page.

The tables on the next page show the mask value (not the bit no.) for requesting the corresponding error / warning status:

```
// -- C/C++ example ----------------------------------------------
if (errorcode & 0x00000001)   //  power supply voltage range error
{
  // report error to user etc.
}
// -------------------------------------------------------------
```

d.) Warnings:

Warning mask codes:	
0x00000001	Power Supply Voltage Range
0x00000002	Power Supply Temperature
0x00000004	Camera temperature (board temperature / FPGA temperature)
0x00000008	Image Sensor temperature (for cooled camera versions only)

e.) Errors:

Error mask codes:	
0x00000001	Power Supply Voltage Range
0x00000002	Power Supply Temperature
0x00000004	Camera temperature (board temperature / FPGA temperature)
0x00000008	Image Sensor temperature (for cooled camera versions only)
0x00010000	Camera Interface failure
0x00020000	Camera RAM module failure
0x00040000	Camera Main Board failure
0x00080000	Camera Head Boards failure

f.) Status:

Status mask codes:	
0x00000001	Default State: • Bit set: No settings changed, camera is in default state. • Bit cleared: Settings were changed since power up or reset.
0x00000002	Settings Valid: • Bit set: Settings are valid (i.e., last "Arm Camera' was successful and no settings were changed since 'Arm camera', except exposure time). • Bit cleared: Settings were changed but not yet not checked and accepted by 'Arm Camera' command.
0x00000004	Recording State: • Bit set: Recording state is on. • Bit cleared: Recording state is off.
0x00000010	Framerate State (dimax only): • Bit set: Framerate setting is active. Timing depends on fps and exposure. • Bit cleared: Framerate settings is off. Timing depends on exposure and delay setting.

4.1.4 PCO_ResetSettingsToDefault

Resets all camera settings to default values. This function is executed during a power-up sequence.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_ResetSettingsToDefault(HANDLE ph)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

d.) The following are the default settings:

Setting:	**Default:**
Sensor Format	standard
ROI	full resolution
Binning	no binning
Pixel Rate	(depending on image sensor)
Gain	Normal gain (if setting available due to sensor)
Double Image Mode	Off
IR sensitivity	Off (if setting available due to sensor)
Cooler Setpoint	-12 C°
ADC mode	Using one ADC
Exposure Time	20 ms
Delay Time	0 µs
Trigger Mode	Auto Trigger
Recording state	stopped
Memory Segmentation	Total memory allocated to first segment
Storage Mode	Recorder Ring Buffer + Live View on
Acquire Mode	Auto

Note: If the camera is running when this command is sent, it will be stopped!

4.1.5 PCO_InitiateSelftestProcedure

Initiate the self-test, which will return the following status: warnings, errors.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_InitiateSelftestProcedure(HANDLE ph, DWORD* dwWarn, DWORD*
dwErr)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD dwWarn: Address of a DWORD to get the warning messages.
- DWORD dwErr: Address of a DWORD to get the error messages.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

The input pointers will be filled with the following parameters:

- see. PCO_GetCameraHealthStatus

4.1.6 PCO_GetTemperature

Request the current camera and power supply temperatures. Power supply temperature is not available with all cameras. If it is not available, the temperature will show 0. In case the sensor temperature is not available it will show 0x8000.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetTemperature(HANDLE ph, SHORT* sCCDTemp, SHORT*
                                            sCamTemp, SHORT* sPowTemp)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- SHORT* sCCDTemp: Address of a SHORT to get the CCD temperature.
- SHORT* sCamTemp: Address of a SHORT to get the camera temperature.
- SHORT* sPowTemp: Address of a SHORT to get the power supply temperature.

The input pointers will be filled with the following parameters:

- CCD temperature as signed word in °C*10.
- Camera temperature as signed word in °C.
- Power Supply temperature as signed word in °C.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.1.7 PCO_GetCameraName

Request the camera name.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetCameraName(HANDLE ph, char* szCameraName, WORD wSZCameraNameLen)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- char* szCameraName: Address of a char array to get the camera name.
- WORD wSZCameraNameLen: WORD to check the length of the supplied array.

The string szCameraName has to be long enough to get the camera name. Maximum length will be 40 characters including a terminating zero.

The input pointers will be filled with the following parameters:

- Camera name as it is stored inside the camera (e.g. “pco.4000”).

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.1.8 PCO_GetFirmwareInfo

Request firmware version information. This function is not available with all cameras.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetFirmwareInfo(HANDLE ph, WORD wDeviceBlock, PCO_FW_Vers*
                                             pstrFirmWareVersion)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wDeviceBlock: Used to address a block of information; 0 gets the first 10 devices.
- PCO_FW_Vers* pstrFirmWareVersion: Pointer to a PCO_FW_Vers structure, where wDeviceNum will hold the number of devices available.

The input pointers will be filled with the following parameters:

- pstrFirmWareVersion.wDeviceNum: Number of available devices. In case there are more than 10 devices, the user can get the next block of 10 devices, by calling this function with wDeviceBlock set to 1 and so on.
- pstrFirmWareVersion.Device[0…9]: This structure will be filled with the version info. See sc2_sdkstructures.h for more info.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2 Image Sensor

This function group defines the image sensor access to the camera, especially sensor format, ROI, binning, pixel rate, etc.

Overview:

Command:
PCO_GetSensorStruct
PCO_SetSensorStruct
PCO_GetCameraDescription
PCO_GetCameraDescriptionEx
PCO_GetSensorStruct
PCO_SetSensorFormat
PCO_GetSizes (*)
PCO_GetROI
PCO_SetROI
PCO_GetBinning
PCO_SetBinning
PCO_GetPixelrate
PCO_SetPixelrate
PCO_GetConversionFactor
PCO_SetConversionFactor
PCO_GetDoubleImageMode
PCO_SetDoubleImageMode
PCO_GetADCOperation
PCO_SetADCOperation
PCO_GetIRSensitivity
PCO_SetIRSensitivity
PCO_GetCoolingSetpointTemperature
PCO_SetCoolingSetpointTemperature
PCO_GetOffsetMode
PCO_SetOffsetMode
PCO_GetNoiseFilterMode
PCO_SetNoiseFilterMode
PCO_GetHWIOSignalCount (dimax edge only)
PCO_GetHWIOSignalDescriptor (dimax, edge only)
PCO_GetLookuptableInfo (edge only)
PCO_GetActiveLookuptable (edge only)
PCO_SetActiveLookuptable (edge only)

4.2.1 PCO_GetSensorStruct

Get the complete set of the sensor functions settings. Please fill in all wSize parameters, even in embedded structures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetSensorStruct(HANDLE ph, PCO_Sensor* strSensor);
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- PCO_Sensor* strSensor: Address of a PCO_Sensor structure to get the sensor settings.

Please fill in all wSize parameters, even in embedded structures.

The input pointer will be filled with parameters following this function description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.2 PCO_SetSensorStruct

Sets the complete set of timing functions settings. Please fill in all wSize parameters, even in embedded structures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetSensorStruct(HANDLE ph, PCO_Sensor* strSensor);
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- PCO_Sensor* strSensor: Address of a PCO_Sensor structure to set the sensor settings.

Please fill in all wSize parameters, even in embedded structures.

The input pointer has to be filled with parameters following this function description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.3 PCO_GetCameraDescription

Request camera description (sensor type, horizontal / vertical / dynamic resolution/ binning/ delay/ exposure ...). The response message describes the sensor type, the readout hardware and its possible operating range. This set of information can be used to verify the settings before the user calls the PCO_Setxxx – commands. Please fill in all wSize parameters, even in embedded structures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetCameraDescription(HANDLE ph, PCO_Description* strDescription)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- PCO_Description* strDescription: Address of a PCO_Description structure.

Please fill in all wSize parameters, even in embedded structures.

The input structure will be filled with the following parameters:

		Sensor Type	Sensor Sub Type	Hor. Res. standard	Vert. Res. standard	Hor. Res. extended	Vert. Res. extended	Dyn. Res.
		0x####	0x####	0x####	0x####	0x####	0x####	0x####
Max Binn hor	**Binn hor steps**	**Max Binn vert**	**Binn vert steps**	**ROI hor steps**	**ROI vert steps**	**ADCs**	**Pixelrate 1**	**Pixelrate 2**
0x####	0x####	0x####	0x####	0x####	0x####	0x####	0x#### ####	0x#### ####
Pixelrate 3	**Pixelrate 4**	**Convers. Factor 1**	**Convers. Factor 2**	**Convers. Factor 3**	**Convers. Factor 4**	**IR – Sens.**	**Min Del Time (nsec)**	**Max Del Time (msec)**
0x#### ####	0x#### ####	0x####	0x####	0x####	0x####	0x####	0x#### ####	0x#### ####
Min Del Step (nsec)	**Min Exp Time (nsec)**	**Max Exp Time (msec)**	**Min Exp Step (nsec)**	**Min Del Time IR (nsec)**	**Max Del Time IR (msec)**	**Min Exp Time IR (nsec)**	**Max Exp Time IR (msec)**	**Time Table**
0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x####
Double Image Mode	**Min Cooling Setpoint**	**Max Cooling Setpoint**	**Default Cooling Setpoint**	**Power Down Mode**	**Offset Regu-lation**	**Color Pattern**	**Color Pattern Type**	**Reserved 1**
0x####	0x####	0x####	0x####	0x####	0x####	0x####	0x####	0x####
General Caps 1	**Reserved 2**	**Reserved 3**	**Reserved 4**	**Reserved 5**	**Reserved 6**	**Reserved 7**	**Reserved 8**	**Reserved 9**
0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x####

- image sensor type as word, see table “Sensor Type codes” below.
- image sensor sub type as word.
- horizontal resolution standard in pixels (all effective pixels).
- vertical resolution standard in pixels (all effective pixels).
- horizontal resolution extended in pixels (all pixels; dummy + dark + eff.).
- vertical resolution extended in pixels (all pixels; dummy + dark + eff.).
- dynamic resolution in bits/pixel. (i.e. 12, 14 …)
- max. binning value horizontal (allowed values from 1 to max. resolution)
- binning steps horizontal
  1 = linear step (binning from 1 to max i.e. 1,2,3…max is possible)
  0 = binary step (binning from 1 to max i.e. 1,2,4,8,16…max is possible)
- max. binning value vertical (allowed values from 1 to max. resolution)
- binning steps vertical
  1 = linear step (binning from 1 to max i.e. 1,2,3…max is possible)
  0 = binary step (binning from 1 to max i.e. 1,2,4,8,16…max is possible)
- ROI steps horizontal (e.g. 10, => ROI right = 1, 11, 21, 31 ...)
- ROI steps vertical
- ADCs (number of ADCs inside camera; i.e. 1..8)
- pixelrate 1 (long word; frequency in Hz)
- pixelrate 2 (long word; frequency in Hz; if not available, then value = 0)
- pixelrate 3 (long word; frequency in Hz; if not available, then value = 0)
- pixelrate 4 (long word; frequency in Hz; if not available, then value = 0)
- conversion factor 1 (in electron / counts)
  (the value 100 corresponds to 1; i.e. 610 = 6.1 electron/counts)
- conversion factor 2 (in electron / counts; if not available, then value = 0)
  (the value 100 corresponds to 1; i.e. 610 = 6.1 electron/counts)
- conversion factor 3 (in electron / counts; if not available, then value = 0)
  (the value 100 corresponds to 1; i.e. 610 = 6.1 electron/counts)
- conversion factor 4 (in electron / counts; if not available, then value = 0)
  (the value 100 corresponds to 1; i.e. 610 = 6.1 electron/counts)
- IR-sensitivity; sensor can switch to improved IR sensitivity
  (0 = function not supplied; 1 = possible)
- min. delay time in nsec (long word; non IR-sensitivity mode)
- max. delay time in msec (long word; non IR-sensitivity mode)
- min. delay time step in nsec (long word)
  Note: Applies both to non IR-sensitivity mode and IR-sensitivity mode
- min. exposure time in nsec (long word; non IR-sensitivity mode)
- max. exposure time in msec (long word; non IR-sensitivity mode)
- min. exposure time step in nsec (long word)
  Note: Applies both to non IR-sensitivity mode and IR-sensitivity mode
- min. delay time in nsec (long word; IR-sensitivity mode)
- max. delay time in msec (long word; IR-sensitivity mode)
- min. exposure time in nsec (long word; IR-sensitivity mode)
- max. exposure time in msec (long word; IR-sensitivity mode)

(Return values of command "Get Camera Description" continued)

- time table; camera can perform a timetable with several delay/ exposures
  (0 = function not supplied; 1 = possible)
- double image mode; camera can perform a double image with a short interleave time between exposures (0 = function not supplied; 1 = possible)
- min. cooling setpoint (in °C)
  (if all setpoints are 0, then cooling is not available)
- max. cooling setpoint (in °C)
  (if all setpoints are 0, then cooling is not available)
- default cooling setpoint (in °C)
  (if all setpoints are 0, then cooling is not available)
- power down mode; switch sensor into power down mode for reduced dark current (0 = function not supplied; 1 = possible)
- offset regulation; automatic offset regulation with reference Pixels
  (0 = function not supplied; 1 = possible)
- color pattern; four nibbles are desribing the colors of a color chip. (see below)
- color pattern type; 0: RGB Bayer Pattern
- general caps 1; describes special features of the camera, whether they are available
- reserved 1 – 9 (for future use)

Color Pattern description (2x2 matrix):

0	1
2	3

e.g.:

R	G
G	B

The color pattern is declared by four nibbles. Each nibble holds the value of the corresponding color.

Color defines:

- RED = 0x01
- GREEN (RED LINE) = 0x02
- GREEN (BLUE LINE) = 0x03
- BLUE = 0x04

The four nibbles are arranged in the following way:

MSB	3	2	1	0	LSB

For the sample this would result in:

0x04030201 (Nibble3: BLUE, Nibble2: GREENB, Nibble1: GREENR, Nibble0: RED)

The color description is necessary for determining the color of the upper left corner of a color image. The resulting value is a parameter for the demosaicking algorythm.

Sensor Type codes: monochrome sensors:		color sensors:	
Sony ICX285AL	0x0010	Sony ICX285AK	0x0011
Sony ICX263AL	0x0020	Sony ICX263AK	0x0021
Sony ICX274AL	0x0030	Sony ICX274AK	0x0031
Sony ICX407AL	0x0040	Sony ICX407AK	0x0041
Sony ICX414AL	0x0050	Sony ICX414AK	0x0051
Kodak KAI-2000M	0x0110	Kodak KAI-2000CM	0x0111
Kodak KAI-2001M	0x0120	Kodak KAI-2001CM	0x0121
Kodak KAI-4010M	0x0130	Kodak KAI-4010CM	0x0131
Kodak KAI-4011M	0x0132	Kodak KAI-4011CM	0x0133
Kodak KAI-4020M	0x0140	Kodak KAI-4020CM	0x0141
Kodak KAI-4021M	0x0142	Kodak KAI-4021CM	0x0143
Kodak KAI-11000M	0x0150	Kodak KAI-11000CM	0x0151
Kodak KAI-11002M	0x0152	Kodak KAI-11002CM	0x0153
Kodak KAI-16000AXA	0x0160	Kodak KAI-16000CXA	0x0161
Micron MV13 bw	0x1010	Micron MV13 col	0x1011
Fairchild CIS2051 V1 FI	0x2000	-	-
Fairchild CIS2051 V1 BI	0x2010	-	-
Cypress RR V1 bw	0x3000	Cypress RR V1 col	0x3001

Note: This list will be updated with new entries and is available on our webpage ‘www.pco.de’.

General caps description:
Some new firmware features have been implemented after the release of the camera. The general caps dword describes the availability of new functionality.
The **GENERALCAPS1** dword holds the following flags:

Flag name	Value	Short description
NOISE_FILTER	0x00000001	Noise filter is available
HOTPIX_FILTER	0x00000002	Hotpixel correction is available
HOTPIX_ONLY_WITH_NOISE_FILTER	0x00000004	Hotpixel corr. does not work without noise filter
TIMESTAMP_ASCII_ONLY	0x00000008	Time stamp without binary is available
DATAFORMAT2X12	0x00000010	Camlink (1200hs) transfer can be done by 2x12
RECORD_STOP	0x00000020	Record stop event mode is available
HOT_PIXEL_CORRECTION	0x00000040	Hotpixel correction is available
NO_EXTEXPCTRL	0x00000080	External exposure control is not available
NO_TIMESTAMP	0x00000100	Time stamp is not available
NO_ACQUIREMODE	0x00000200	Acquire mode is not available
HW_IO_SIGNAL_DESCRIPTOR	0x40000000	Hardware IO description is available
ENHANCED_DESCRIPTOR_2	0x80000000	Enhanced description 2 is available

See sc2_defs.h for more information.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.4 PCO_GetCameraDescriptionEx

Requests camera description parameters by value. The response message describes the parameter set which is queried by the wType parameter. PCO_DescriptionEx is a generic structure which has to be casted to from the structure queried.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetCameraDescriptionEx(HANDLE ph, PCO_DescriptionEx*
strDescriptionEx, WORD wType)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- PCO_DescriptionEx* strDescriptionEx: Address of a PCO_DescriptionEx structure.
- WORD wType: Word Parameter to hold the number of the descriptor

Please fill in all wSize parameters, even in embedded structures.

The input parameter has to be filled with one of the following parameter:

- x0000 = [standard descriptor]
- x0001 = [descriptor nr. 2]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

Due to the size limitation of the camera descriptor and needs for more additional features, the descriptor has to be extended. This descriptor 2 can be read out by a new command PCO_GetCameraDescriptionEx. PCO_GetCameraDescriptionEx allows further enhancement due to an additional wType parameter, which addresses different descriptors.

Min Per Time (nsec)	Max Per Time (msec)	Min Per Condition (nsec)	Max Number of Exposures	Min Mon. Sig. Offs. (nsec)	Max Mon. Sig. Offs. (nsec)	Min Per Step (nsec)	Start Time Delay (nsec)
0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####
Min Mon Step (nsec)	**Min Del Time Mod (nsec)**	**Max Del Time Mod (msec)**	**Min Del Step Mod (nsec)**	**Min Exp Time Mod (nsec)**	**Max Exp Time Mod (msec)**	**Min Exp Step Mod (nsec)**	**Modulate Caps**
0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####
Reserved 1	**Reserved 2**	**Reserved 3**	**Reserved 4**	**Reserved 5**	**Reserved 6**	**Reserved 7**	**Reserved 8**
0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####
Reserved 9	**Reserved 10**	**Reserved 11**	**Reserved 12**	**Reserved 13**	**Reserved 14**	**Reserved 15**	**Reserved 16**
0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####	0x#### ####

- Min per time as dword; Minimum periodical time in nsec.
- Max per time as dword; Maximum periodical time in msec.
- Min per additional as dword; Minimum periodical time condition: Periodical time – exposure time must not be smaller than 'min per additional'.
- Max number of exposures. Maximum number of possible exposures in one frame.
- Max mon sig offset as dword; Maximum negative monitor signal offset in nsec.
- Min per step as dword; Minimum periodical time step in nsec.
- Start time delay as dword; Constant maximum value for mon signal offset in case of delay = 0 in nsec.
- Min mon step as dword; Minimum monitor step time in nsec.
- Min. delay time in nsec (long word; modulate mode)
- Max. delay time in msec (long word; modulate mode)
- Min. delay time step in nsec (long word)
  Note: Applies to modulate mode
- Min. exposure time in nsec (long word; non IR-sensitivity mode)
- Max. exposure time in msec (long word; non IR-sensitivity mode)
- Min. exposure time step in nsec (long word)
  Note: Applies to modulate mode
- Modulate caps describes the availability of optional functionality
- Array of 24 reserved DWORDs

To introduce the enhanced descriptors some flags inside the GENERALCAPS 1 of the descriptor will be added. Some further flags will be added to the MODULATECAPS.

GENERALCAPS_1

Flag name	Value	Short description
ENHANCED_DESCRIPTOR_2	0x80000000	Further descriptors are available. The function call PCO_GetDescriptionEx will be enabled.
HW_IO_SIGNAL_DESCRIPTOR	0x40000000	Hardware IO Signal description available

MODULATECAPS

Flag name	Value	Short description
MODULATE	0x00000001	Modulate is available

4.2.5 PCO_GetSensorFormat

Get format of sensor. The [standard] format uses only effective pixels, while the [extended] format shows all pixels inclusive effective, dark, reference and dummy.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetSensorFormat(HANDLE ph, WORD* wSensor)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wSensor: Address of a WORD to get the sensor format.

The input pointer will be filled with the following parameter:

- x0000 = [standard]
- x0001 = [extended]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.6 PCO_SetSensorFormat

Set format of sensor. The [standard] format uses only effective pixels, while the [extended] format shows all pixels inclusive effective, dark, reference and dummy.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetSensorFormat(HANDLE ph, WORD wSensor)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wSensor: WORD to set the sensor format.

The input parameter has to be filled with one of the following parameter:

- x0000 = [standard]
- x0001 = [extended]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.7 PCO_GetSizes (*)

Get the actual armed image sizes of the camera. If the user recently changed size influencing values without issuing an ARM, the GetSizes function will return the sizes from the last recording. If no recording occurred, it will return the last ROI settings.

We recommend the following order of commands: SetBinning, SetROI, ARM, GetSizes, AllocateBuffer.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetSizes(HANDLE ph, WORD* wXResAct, WORD* wYResAct, WORD*
wXResMax, WORD* wYResMax)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wXResAct: Address of a WORD to get the actual x resolution.
- WORD* wYResAct: Address of a WORD to get the actual y resolution.
- WORD* wXResMax: Address of a WORD to get the maximum x resolution.
- WORD* wXResMax: Address of a WORD to get the maximum y resolution.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.8 PCO_GetROI

Get ROI (region or area of interest) window. The ROI is equal to or smaller than the absolute image area, which is defined by the settings of **format** and **binning**.

Some sensors have a ROI stepping. See the camera description and check the parameters wRoiHorStepsDESC and/or wRoiVertStepsDESC.

For dual ADC mode the horizontal ROI must be symmetrical. For a pco.dimax the horizontal and vertical ROI must be symmetrical. For a pco.edge the vertical ROI must be symmetrical.

a.) Prototype:

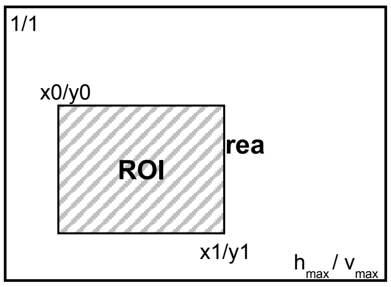
```
SC2_SDK_FUNC int WINAPI PCO_GetROI(HANDLE ph, WORD* wRoiX0, WORD* wRoiY0, WORD*
wRoiX1, WORD* wRoiY1)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wRoiX0: Address of a WORD to get the x0 coordinate of the ROI.
- WORD* wRoiY0: Address of a WORD to get the y0 coordinate of the ROI.
- WORD* wRoiX1: Address of a WORD to get the x1 coordinate of the ROI.
- WORD* wRoiY1: Address of a WORD to get the y1 coordinate of the ROI.

The input pointer will be filled with the following parameters:

- x0, x1, y0, y1: region of interest (in pixels) within the complete image of the sensor (see also figure below).



c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.9 PCO_SetROI

Set ROI (region or area of interest) window. The ROI must be equal to or smaller than the absolute image area, which is defined by the settings of **format** and **binning**. If the binning settings are changed, the user must adapt the ROI, before PCO_ArmCamera is accessed. The binning setting sets the limits for the ROI. For example, a sensor with 1600x1200 and binning 2x2 will result in a maximum ROI of 800x600.

Some sensors have a ROI stepping. See the camera description and check the parameters wRoiHorStepsDESC and/or wRoiVertStepsDESC.

For dual ADC mode the horizontal ROI must be symmetrical. For a pco.dimax the horizontal and vertical ROI must be symmetrical. For a pco.edge the vertical ROI must be symmetrical.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetROI(HANDLE ph, WORD wRoiX0, WORD wRoiY0, WORD wRoiX1,
WORD wRoiY1)
```

b.) Input parameter:

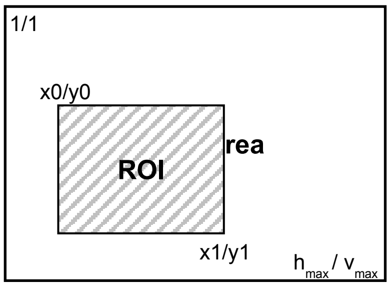
- HANDLE ph: Handle to a previously opened camera device.
- WORD wRoiX0: WORD to set the x0 coordinate of the ROI.
- WORD wRoiY0: WORD to set the y0 coordinate of the ROI.
- WORD wRoiX1: WORD to set the x1 coordinate of the ROI.
- WORD wRoiY1: WORD to set the y1 coordinate of the ROI.

The input parameter must be filled with the following parameters:

Parameter:	• x0, x1, y0, y1: set region of interest (in pixels) within the complete image of the sensor.
Notes:	• valid ROI settings range from 1/1 to h_{max}/v_{max} (h_{max}/v_{max} are dependent from the settings of **format** and **binning**) • values out of range result in a failure response message • the command will be rejected, if Recording State is [run]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes



4.2.10 PCO_GetBinning

Get binning information.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetBinning(HANDLE ph, WORD* wBinHorz, WORD* wBinVert)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wBinHorz: Address of a WORD to get the horizontal binning.
- WORD* wBinVert: Address of a WORD to get the vertical binning.

The input pointer will be filled with the following parameters:

- current binning x (horizontal direction) and binning y (vertical direction).

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.11 PCO_SetBinning

Set binning. If the binning settings are changed, the user must adapt the ROI, before PCO_ArmCamera is accessed. The binning setting sets the limits for the ROI. E.g. a sensor with 1600x1200 and binning 2x2 will result in a maximum ROI of 800x600.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetBinning(HANDLE ph, WORD wBinHorz, WORD wBinVert)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wBinHorz: WORD to set the horizontal binning.
- WORD wBinVert: WORD to set the vertical binning.

The input data have to be filled with the following parameters:

Parameter:	• set binning x (horizontal direction) and binning y (vertical direction)
Notes:	• valid binning settings generally are 1, 2, 4, 8, 16, 32, other values may be possible depending on the camera type • invalid values result in a failure response message • the command will be rejected, if Recording State is [run]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.12 PCO_GetPixelrate

Get pixelrate for reading images from the image sensor.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetPixelRate(HANDLE ph, DWORD* dwPixelRate)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD* dwPixelRate: Address of a DWORD to get the pixel rate.

The input pointer will be filled with the following parameter:

- current pixelrate as long word in Hz.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.13 PCO_SetPixelrate

Set pixelrate.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetPixelRate(HANDLE ph, DWORD dwPixelRate)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD dwPixelRate: DWORD to set the pixel rate.

The input data has to be filled with the following parameter:

Parameter:
- pixelrate to be configured as long word in Hz.

Notes:
- valid values depend on camera type, the adjustable values are defined in the camera description
- invalid values result in a failure response message
- the command will be rejected, if Recording State is [run]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.14 PCO_GetConversionFactor

Get image sensor gain setting.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetConversionFactor(HANDLE ph, WORD* wConvFact)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wConvFact: Address of a WORD to get the conversion factor.

The input pointer will be filled with the following parameter:

- current conversion factor in electrons/count (the variable must be divided by 100 to get the real value)
  i.e. 0x01B3 (hex) = 435 (decimal) = 4.35 electrons/count
  conversion factor must be valid as defined in the camera description

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.15 PCO_SetConversionFactor

Set image sensor gain.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetConversionFactor(HANDLE ph, WORD wConvFact)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wConvFact: WORD to set the conversion factor.

The input data musts be filled with the following parameter:

Parameter:	• conversion factor to be set in electrons/count (the variable must be divided by 100 to get the real value) i.e. 0x01B3 (hex) = 435 (decimal) = 4.35 electrons/count conversion factor must be valid as defined in the camera description
Notes:	• invalid values result in a failure response message • the command will be rejected, if Recording State is [run]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.16 PCO_GetDoubleImageMode

Get double image mode setting.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetDoubleImageMode(HANDLE ph, WORD* wDoubleImage)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wDoubleImage: Address of a WORD to get the double image mode.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

The input pointer will be filled with the following parameter:

- curent mode:
  0x0001 = double image mode ON, 0x0000 = double image mode OFF

4.2.17 PCO_SetDoubleImageMode

Set double image mode - some cameras (defined in the camera description) allow the user to make a double image with two exposures separated by a short interleaving time. A double image is transferred as one frame, that is the two images resulting from the two/double exposures are stitched together as one and are counted as one. Thus the buffer size has to be doubled. The first half of the buffer will be filled with image 'A', the first exposed frame. The second exposure (image 'B') will be transferred to the second half of the buffer.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetDoubleImageMode(HANDLE ph, WORD wDoubleImage)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wDoubleImage: WORD to set the double image mode.

The input data has to be filled with the following parameter:

Parameter:
- mode:
  0x0001 = double image mode ON, 0x0000 = double image mode OFF
- the existence of this option can be checked with the values defined in the camera description

Notes:
- invalid values result in a failure response message
- the command will be rejected, if Recording State is [run]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.18 PCO_GetADCOperation

Get analog-digital-converter (ADC) operation for reading the image sensor data. Pixel data can be read out using one ADC (better linearity), or in parallel using two ADCs (faster). This option is only available for some camera models (defined in the camera description). If the user sets 2ADCs he must center and adapt the ROI to symmetrical values, e.g. pco.1600: x1,y1,x2,y2=701,1,900,500 (100,1,200,500 is not possible).

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetADCOperation(HANDLE ph, WORD* wADCOperation)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wADCOperation: Address of a WORD to get the ADC operation.

The input pointer will be filled with the following parameter:

- current usage:
  0x0001 = 1 ADC or 0x0002 = 2 ADCs are used…

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.19 PCO_SetADCOperation

Set analog-digital-converter (ADC) operation for reading the image sensor data. Pixel data can be read out using one ADC (better linearity) or in parallel using two ADCs (faster). This option is only available for some camera models. If the user sets 2ADCs he must center and adapt the ROI to symmetrical values, e.g. pco.1600: x1,y1,x2,y2=701,1,900,500 (100,1,200,500 is not possible).

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetADCOperation(HANDLE ph, WORD wADCOperation)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wADCOperation: WORD to set the ADC operation.

The input data has to be filled with the following parameter:

- operation to be set:
  0x0001 = 1 ADC or 0x0002 = 2 ADCs should be used…
- the existence of the number of ADCs can be checked with the values defined in the camera description

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.20 PCO_GetIRSensitivity

Get IR sensitivity setting for the image sensor. This option is only available for special camera models with image sensors that have improved IR sensitivity.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetIRSensitivity(HANDLE ph, WORD* wIR)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wIR: Address of a WORD to get the IR sensitivity.

The input pointer will be filled with the following parameter:

- current mode:
  0x0001 = IR sensitivity ON or 0x0000 = IR sensitivity OFF

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.21 PCO_SetIRSensitivity

Set IR sensitivity for the image sensor. This option is only available for special camera models with image sensors that have improved IR sensitivity.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetIRSensitivity(HANDLE ph, WORD wIR)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wIR: WORD to set the IR sensitivity.

The input has to be filled with the following parameter:

- mode to be set:
  0x0001 = IR sensitivity ON or 0x0000 = IR sensitivity OFF
- the existence of this option can be checked with the values defined in the camera description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.22 PCO_GetCoolingSetpointTemperature

Get the temperature setpoint for cooling the image sensor (only available for cooled cameras). If min. cooling setpoint (in °C) and max. cooling setpoint (in °C) are zero, then cooling is not available.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetCoolingSetpointTemperature(HANDLE ph, SHORT* sCoolSet)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- SHORT* sCoolSet: Address of a SHORT to get the setpoint for cooling.

The input pointer will be filled with the following parameter:

Return values:	• current cooling temperature setpoint as signed word in °C units
Notes:	• the actual sensor temperature can be read with the **get temperature** function (see PCO_GetTemperature)

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.23 PCO_SetCoolingSetpointTemperature

Set the temperature setpoint for cooling the image sensor (only available for cooled cameras, the default setpoints are [0] in the camera description). If min. cooling setpoint (in °C) and max. cooling setpoint (in °C) are zero, then cooling is not available.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetCoolingSetpointTemperature(HANDLE ph, SHORT sCoolSet)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- SHORT sCoolSet: SHORT to set the setpoint for cooling.

The input data has to be filled with the following parameter:

Parameter:	• cooling temperature setpoint to be adjusted as signed word in °C units • the range of values can be checked with the values defined in the camera description.
Notes:	• valid range depends on camera type, invalid values result in a failure response message • the actual temperature of the sensor can be read with the **get temperature** command (see PCO_GetTemperature)

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.24 PCO_GetOffsetMode

Get the mode for the offset regulation with reference pixels (see camera manual for further explanations).

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetOffsetMode (HANDLE ph, WORD* wOffsetRegulation)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wOffsetRegulation: Address of a WORD to get the offset mode.

The input pointer will be filled with the following parameter:

- mode:
  0x0000 = [auto] or 0x0001 = [OFF]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.25 PCO_SetOffsetMode

Set the mode for the offset regulation with reference pixels (see the camera manual for further explanations).

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetOffsetMode (HANDLE ph, WORD wOffsetRegulation)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wOffsetRegulation: WORD to set the offset mode.

The input has to be filled with the following parameter:

- mode:
  0x0000 = [auto] or 0x0001 = [OFF]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.26 PCO_GetNoiseFilterMode

Get the actual noise filter mode. See the camera descriptor for availability of this feature.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetNoiseFilterMode (HANDLE ph, WORD* wNoiseFilterMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wNoiseFilterMode: Address of a WORD to get the noisefilter mode.

The input pointer will be filled with the following parameter:

Parameter:	• mode: 0x0000 = [OFF] 0x0001 = [ON] 0x0101 = [ON + Hotpixel correction]
Notes:	• the command will be rejected, if Recording State is [run]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.27 PCO_SetNoiseFilterMode

Sets the actual noise filter mode. See the camera descriptor for availability of this feature.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_ SetNoiseFilterMode (HANDLE ph, WORD wNoiseFilterMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wNoiseFilterMode: WORD to set the noisefilter mode.

The input has to be filled with the following parameter:

Parameter:	• mode: 0x0000 = [OFF] 0x0001 = [ON] 0x0101 = [ON + Hotpixel correction]
Notes:	• the command will be rejected, if Recording State is [run]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.28 PCO_GetHWIOSignalCount (dimax edge only)

Get the number of hardware IO signals, which are available with the camera. To set and get the single signals use PCO_GetHWIOSignal (dimax, edge only) and PCO_SetHWIOSignal (dimax, edge only). This functions is not available with all cameras. Actually it is implemented in the pco.dimax.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetHWIOSignalCount (HANDLE ph, WORD* wNumSignals)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wNumSignals: Address of a WORD to get the number of available hardware I/O signals.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.29 PCO_GetHWIOSignalDescriptor (dimax, edge only)

Get the description of the requested hardware IO signal. To get the number of available hardware IO signals, please call PCO_GetHWIOSignalCount (dimax edge only). To set and get the single signals use PCO_GetHWIOSignal (dimax, edge only) and PCO_SetHWIOSignal (dimax, edge only). This functions is not available with all cameras. Actually it is implemented in the pco.dimax.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetHWIOSignalDescriptor (HANDLE ph, WORD wSignalNum,
PCO_Single_Signal_Desc* pstrSignal)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wSignalNum: Select the signal to query. This parameter must be in the range of available hardware I/O signals.
- PCO_Single_Signal_Desc* pstrSignal: Address of a PCO_Single_Signal_Desc descriptor to get the capabilities of the hardware I/O signal.

The input structure will be filled with the following parameter:

Parameter:

- char strSignalName: Up to four different signal names are available for a hardware terminal. In case more than one name is available, the user can select which output/input signal should be used. See wSelected in PCO_Signal structure.
- wSignalDefinitions: Flags showing signal options:

  - 0x01: Signal can be enabled/disabled
  - 0x02: Signal is a status output
- wSignalTypes: Flags showing which signal type is available:
  - 0x01: TTL
  - 0x02: High Level TTL
  - 0x04: Contact Mode
  - 0x08: RS485 differential
- wSignalPolarity: Flags showing which signal polarity can be selected:
  - 0x01: Low level active
  - 0x02: High Level active
  - 0x04: Rising edge active
  - 0x08: Falling edge active
- wSignalFilter: Flags showing the filter option:
  - 0x01: Filter can be switched off (t > ~65ns)
  - 0x02: Filter can be switched to medium (t > ~1us)
  - 0x04: Filter can be switched to high (t > ~100ms)

Notes:
- the command will be rejected, if Recording State is [run]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.30 PCO_GetLookuptableInfo (edge only)

Get lookup table description. Setting all pointer to NULL but wNumberOfLuts will return the number of available luts in the camera.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetLookupTableInfo(HANDLE ph, WORD wLUTNum, WORD* wNumberOffLuts, char *Description, WORD wDescLen, WORD *wIdentifier, BYTE *bInputWidth, BYTE *bOutputWidth, WORD *wFormat)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wLUTNum: Current number of LUT to query
- WORD *wNumberOfLuts: Number of LUTs which can be queried
- char *Description: Text description, e.g. "HD/SDI 12 to 10"
- WORD wDescLen: Length of the description string buffer
- WORD *wIdentifier: Identifier of the specific lookup table needed to activate with “Set Lookuptable”, ranges from 0...0xFFFF
- BYTE *bInputWidth: Maximal Input in Bits, e.g. 16.
- BYTE *bOutputWidth: Maximal Output in Bits, e.g. 12
- WORD *wFormat: accepted data structures (see defines))

The input structure should be / will be filled with the following parameter:

Parameter:
- WORD wLUTNum: Set this parameter to 0...9.
- *wNumberOfLuts: 0...9; To get only the number of valid LUTs, set all other pointers to zero, e.g. PCO_GetLookupTableInfo(hCam, 0, &wNumberOfLuts, NULL, 0, NULL, NULL, NULL, NULL).
- *Description: String with length < 20 e.g. “sqrt(256*x)”, “sqrt(32*x)”.
- wDescLen: Fill in the length of the Description buffer, which will be set.
- *wIdentifier: 0x0000 – disabled; 0x#### ID of the current LUT, which can be one of the predefined values (e.g. 0x1108, 0x0B08).
- *bInputWidth: usually 16.
- *bOutputWidth:

Notes:
- the command will be rejected, if Recording State is [run]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.31 PCO_GetActiveLookuptable (edge only)

Gets the active lookup table in the camera.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetActiveLookupTable(HANDLE ph, WORD *wIdentifier, WORD *wParameter)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD *wIdentifier: Define loadable LUTs, range 0x0001 to 0xFFFF
  - 0x0000 = lookup table is disabled
  - 0x#### = Identifier of the active lookup table
- WORD *wParameter: Offset: 11 Bit value for fixed offset subtraction before transferring the data via the lookup table.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.2.32 PCO_SetActiveLookuptable (edge only)

Sets the active lookup table in the camera.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetActiveLookupTable(HANDLE ph, WORD *wIdentifier, WORD *wParameter)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD *wIdentifier: Define loadable LUTs, 0x0000 or 0x####
  - 0x0000 = disable lookup table
  - 0x#### = ID of the LUT to activate, see PCO_GetLookupTableInfo for available LUTs.
- WORD *wParameter: Offset: 11 Bit value for fixed offset subtraction before transferring the data via the lookup table.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3 Timing Control

This function group defines timing control access to the camera, especially delay and exposure times, trigger mode, trigger status, etc.

Overview:

Command:
PCO_GetTimingStruct
PCO_SetTimingStruct
PCO_GetDelayExposureTime
PCO_SetDelayExposureTime
PCO_GetDelayExposureTimeTable
PCO_SetDelayExposureTimeTable
PCO_GetTriggerMode
PCO_SetTriggerMode
PCO_ForceTrigger
PCO_GetCameraBusyStatus
PCO_GetPowerDownMode
PCO_SetPowerDownMode
PCO_GetUserPowerDownTime
PCO_SetUserPowerDownTime
PCO_GetExpTrigSignalStatus
PCO_GetCOCRunTime
PCO_GetFPSExposureMode (1200(h)s only)
PCO_SetFPSExposureMode (1200(h)s only)
PCO_GetModulationMode
PCO_SetModulationMode
PCO_GetFrameRate (dimax, edge only)
PCO_SetFrameRate (dimax, edge only)
PCO_GetHWIOSignal (dimax, edge only)
PCO_SetHWIOSignal (dimax, edge only)
PCO_GetImageTiming
PCO_GetCameraSynchMode (dimax only)
PCO_SetCameraSynchMode (dimax only)
PCO_GetFastTimingMode (dimax only)
PCO_SetFastTimingMode (dimax only)
PCO_GetSensorSignalStatus (pco.1400 only)

4.3.1 PCO_GetTimingStruct

Get the complete set of the timing functions settings. Please fill in all wSize parameters, even in embedded structures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetTimingStruct(HANDLE ph, PCO_Timing* strTiming)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- PCO_Timing* strTiming: Address of a PCO_Timing structure to get the timing settings.

Please fill in all wSize parameters, even in embedded structures.

The input pointer will be filled with the parameters following this function description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.2 PCO_SetTimingStruct

Set the complete set of the timing functions settings. Please fill in all wSize parameters, even in embedded structures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetTimingStruct(HANDLE ph, PCO_Timing* strTiming)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- PCO_Timing* strTiming: PCO_Timing structure to set the timing settings.

Please fill in all wSize parameters, even in embedded structures.

The input pointer must be filled with parameters following this function description. If a single exposure/delay pair is to be set, the user must set all of the table members to zero except the first member 0! The table member 0 will hold the value for the single delay/exposure pair.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.3 PCO_GetDelayExposureTime

Get delay / exposure time.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetDelayExposureTime(HANDLE ph, DWORD* dwDelay, DWORD*
dwExposure, WORD* wTimeBaseDelay, WORD* wTimeBaseExposure)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD* dwDelay: Address of a DWORD to get the delay time.
- DWORD* dwExposure: Address of a DWORD to get the exposure time.
- WORD* wTimeBaseDelay: Address of a WORD to get the time base of the delay time.
- WORD* wTimeBaseExposure: Address of a WORD to get the time base of the exposure time.

The input pointers will be filled with the following parameters:

- timebase for delay and exposure times
  - 0x0000 => timebase = [ns] (10-9s)
  - 0x0001 => timebase = [µs] (10-6s)
  - 0x0002 => timebase = [ms] (10-3s)
- delay and exposure time as multiples of timebase units

Note:

- delay and exposure values are multiplied with the configured **timebase** unit values
- the range of possible values can be checked with the values defined in the camera description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.4 PCO_SetDelayExposureTime

Set delay / exposure time. If the recording state is on, it is possible to change the timing without calling PCO_ArmCamera.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetDelayExposureTime(HANDLE ph, DWORD dwDelay, DWORD dwExposure, WORD wTimeBaseDelay, WORD wTimeBaseExposure)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD dwDelay: DWORD to set the delay time.
- DWORD dwExposure: DWORD to set the exposure time.
- WORD wTimeBaseDelay: WORD to set the time base of the delay time.
- WORD wTimeBaseExposure: WORD to set the time base of the exposure time.

The input parameters must be filled with the following parameters:

- timebase for delay and exposure times
  - 0x0000 => timebase = [ns] (10-9s)
  - 0x0001 => timebase = [µs] (10-6s)
  - 0x0002 => timebase = [ms] (10-3s)
- delay and exposure time as multiples of timebase units

Note:

- delay and exposure values are multiplied with the configured **timebase** unit values
- the range of possible values can be checked with the values defined in the camera description.
- can be used to alter the timing in case the recording state is on. In this case it is not necessary to call PCO_ArmCamera. If the recording state is off calling PCO_ArmCamera is mandatory.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.5 PCO_GetDelayExposureTimeTable

Get delay / exposure time table.

General note:

For some camera types it is possible to define a table with delay / exposure times (defined in the camera description). After the exposure is started, the camera will take a series of consecutive images with delay and exposure times, as defined in the table. Therefore, a flexible message format has been defined. The table consists of a maximum of 16 delay / exposure time pairs. If an exposure time entry is set to the value zero, then at execution time this delay / exposure pair is disregarded and the sequence is started automatically with the first valid entry in the table. This results in a sequence of 1 to 16 images with different delay and exposure time settings. External or automatic image triggering is fully functional for every image in the sequence. If the user wants maximum speed (at CCDs overlapping exposure and read out is taken), [auto trigger] should be selected and the sequence should be controlled with the <acq enbl> input.

Note:

- The commands **set delay / exposure time** and **set delay / exposure time table** can only be used alternatively. Using **set delay / exposure time** has the same effect as using the **table** command and setting all but the first delay / exposure entry to zero.

Despite the same parameter set, this function is different to , because the corresponding pointers are used as an array of 16 values each.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetDelayExposureTimeTable(HANDLE ph, DWORD* dwDelay,
DWORD* dwExposure, WORD* wTimeBaseDelay, WORD* wTimeBaseExposure, WORD wCount)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD* dwDelay: Address of a DWORD array to get the delay time.
- DWORD* dwExposure: Address of a DWORD array to get the exposure time.
- WORD* wTimeBaseDelay: Address of a WORD to get the time base of the delay times.
- WORD* wTimeBaseExposure: Address of a WORD to get the time base of the exposure times.
- WORD wCount: WORD to set the length of the array (number of DWORDs, must not be more than 16 DWORDS)

The input pointers will be filled with the following parameters:

- timebase for delay and exposure times
  - 0x0000 => timebase = [ns] (10-9s)
  - 0x0001 => timebase = [µs] (10-6s)
  - 0x0002 => timebase = [ms] (10-3s)
- delay and exposure time as multiples of timebase units

Note:

- Delay and exposure values are multiplied with the configured **timebase** unit values
- The range of possible values can be checked with the values defined in the camera description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.6 PCO_SetDelayExposureTimeTable

Set delay / exposure time.

General note: see **PCO_GetDelayExposureTimeTable**
a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetDelayExposureTimeTable(HANDLE ph, DWORD* dwDelay,
DWORD* dwExposure, WORD wTimeBaseDelay, WORD wTimeBaseExposure, WORD wCount)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD* dwDelay: Address of a DWORD array to set the delay times.
- DWORD* dwExposure: Address of a DWORD array to set the exposure times.
- WORD wTimeBaseDelay: WORD to set the time base of the delay times.
- WORD wTimeBaseExposure: WORD to set the time base of the exposure times.
- WORD wCount: WORD to set the length of the array (number of DWORDs, must not be more than 16 DWORDS)

The input parameters must be filled with the following parameters:

- timebase for delay and exposure times
  - 0x0000 => timebase = [ns] (10-9s)
  - 0x0001 => timebase = [µs] (10-6s)
  - 0x0002 => timebase = [ms] (10-3s)
- delay and exposure time as multiples of timebase units

Note:

- Delay and exposure values are multiplied with the configured **timebase** unit values
- If an exposure value is set to zero, the sequence is repeated from the beginning (first entry)
- If exposure 1 is set to zero, an error is generated
- If all exposure entries are non zero, the sequence consists of 16 images
- The command will be rejected, if Recording State is [run]
- The range of possible values can be checked with the values defined in the camera description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.7 PCO_GetTriggerMode

Get image trigger mode (for further explanation see camera manual).

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetTriggerMode(HANDLE ph, WORD* wTriggerMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wTriggerMode: Address of a WORD to get the trigger mode .

The input pointer will be filled with the following parameters:

- Current trigger mode:
  - 0x0000 = [auto trigger]
    An exposure of a new image is started automatically best possible compared to the readout of an image. If a CCD is used, and images are taken in a sequence, then exposures and sensor readout are started simultaneously. Signals at the trigger input (<exp trig>) are irrelevant.
  - 0x0001 = [software trigger]:
    An exposure can only be started by a **force trigger** command.
  - 0x0002 = [extern exposure & software trigger]:
    A delay / exposure sequence is started at the RISING or FALLING edge (depending on the DIP switch setting) of the trigger input (<exp trig>).
  - 0x0003 = [extern exposure control]:
    The exposure time is defined by the pulse length at the trigger input(<exp trig>). The delay and exposure time values defined by the **set/request delay and exposure** command are ineffective. (Exposure time length control is also possible for double image mode; the exposure time of the second image is given by the readout time of the first image.)

Note: In the [extern exposure & software trigger] and [extern exposure control] modes, it also depends on the selected acquire mode, if a trigger edge at the trigger input (<exp trig>) will be effective or not (*see also 4.5.10*). A software trigger however will always be effective independent of the state of the <acq enbl> input (concerned trigger modes are: [software trigger] and [extern exposure & software trigger].

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.8 PCO_SetTriggerMode

Set image trigger mode.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetTriggerMode(HANDLE ph, WORD wTriggerMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wTriggerMode: WORD to set the trigger mode .

The input data has to be filled with the following parameters:

Parameters:

- trigger mode to be selected:
  - 0x0000 = [auto trigger]
    A new image exposure is automatically started best possible compared to the readout of an image. If a CCD is used and the images are taken in a sequence, then exposures and sensor readout are started simultaneously. Signals at the trigger input (<exp trig>) are irrelevant.
  - 0x0001 = [software trigger]:
    An exposure can only be started by a **force trigger** command.
  - 0x0002 = [extern exposure & software trigger]:
    A delay / exposure sequence is started at the RISING or FALLING edge (depending on the DIP switch setting) of the trigger input (<exp trig>).
  - 0x0003 = [extern exposure control]:
    The exposure time is defined by the pulse length at the trigger input(<exp trig>). The delay and exposure time values defined by the **set/request delay and exposure** command are ineffective. (Exposure time length control is also possible for double image mode; exposure time of the second image is given by the readout time of the first image.)

Notes:

- the command will be rejected, if Recording State is [run]
- In modes [extern exposure & software trigger] and [extern exposure control], it depends also on the selected acquire mode, if a trigger edge at the trigger input (<exp trig>) will be effective or not (*see also 4.5.10*). A software trigger however will always be effective independent of the state of the <acq enbl> input (concerned trigger modes are: [software trigger] and [extern exposure & software trigger].

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.9 PCO_ForceTrigger

This software command starts an exposure if the **trigger mode** is in the [software trigger] (0x0001) state or in the [extern exposure & software trigger] (0x0002) state. If the trigger mode is in the [extern exposure control] (0x0003) state, nothing happens. A ForceTrigger should not be used to generate a distinct timing.

To accept a force trigger command the camera must be recording and ready: (**recording** = [start]) and [not busy]. If a trigger fails it will not trigger future exposures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_ForceTrigger(HANDLE ph, WORD* wTriggered)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wTriggered: Address of a WORD to get the trigger state.

The input pointer will be filled with the following parameters:

- result:
  - 0x0000 = trigger command was unsuccessful because the camera is busy
  - 0x0001 = a new image exposure has been triggered by the command

Note:

- Due to response and processing times, e.g., caused by the interface and/or the operating system on the PC, the delay between command and actual trigger may be several 10 ms up to 100 ms.
- A force trigger command will be effective independent of the selected acquire mode and independent of the state of the <acq enbl> input.
- Triggers are not accumulated or buffered. A trigger will be accepted if the camera is in idle.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.10 PCO_GetCameraBusyStatus

Get camera busy status: a trigger is ignored, if the camera is still busy ([exposure] or [readout]). In case of **force trigger** command, the user may request the camera busy status in order to be able to start a valid **force trigger** command. Please do not use this function for image synchronization. Use either PCO_GetBufferStatus or use the events (recommended, avoids CPU load due to polling).

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetCameraBusyStatus(HANDLE ph, WORD* wCameraBusyState)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wCameraBusyState: Address of a WORD to get the camera busy status.

The input pointer will be filled with the following parameter:

- camera busy status:
  - 0x0000 = camera is [not busy], ready for a new trigger command
  - 0x0001 = camera is [busy], not ready for a new trigger command

Note:

- The busy status is according to the hardware signal <busy> at the <status output> at the power supply unit. Due to response and processing times, e.g., caused by the interface and/or the operating system, the delay between the delivered status and the actual status may be several 10 ms up to 100 ms. If timing is critical, it is strongly recommended that the hardware signal (<busy>) be used.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.11 PCO_GetPowerDownMode

Get mode for CCD or CMOS power down mode (see camera manual).

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetPowerDownMode(HANDLE ph, WORD* wPowerDownMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wPowerDownMode: Address of a WORD to get the power down mode.

The input pointer will be filled with the following parameter:

Return values:
- current mode:
  0x0000 = [auto] or 0x0001 = [user]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.12 PCO_SetPowerDownMode

Set mode for CCD or CMOS power down threshold time control. Power down functions are controllable when **power down mode** = [user] is selected (see camera manual).

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetPowerDownMode(HANDLE ph, WORD wPowerDownMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wPowerDownMode: WORD to set the power down mode.

The input pointer has to be filled with the following parameter:

- current mode:
  0x0000 = [auto] or 0x0001 = [user]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.13 PCO_GetUserPowerDownTime

Get user values for CCD or CMOS power down threshold time (see camera manual).

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetPowerDownMode(HANDLE ph, WORD* wPowerDownTime)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wPowerDownTime: Address of a WORD to get the power down threshold time.

The input pointer will be filled with the following parameter:

- current CCD power down threshold time as multiples of ms (0ms .. 47.9days)

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.14 PCO_SetUserPowerDownTime

Set user values for CCD or CMOS power down threshold time (see camera manual). If the exposure time is greater than the selected Power Down Time, then the CCD or CMOS sensor is switched (electrically) into a special power down mode to reduce dark current effects. If **power down mode** = [user] is selected, the power down threshold time set by this function will become effective. The default Power Down Time is one second.

The value set by this function will become effective if **power down mode** = [user] is selected.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetPowerDownMode(HANDLE ph, WORD wPowerDownTime)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wPowerDownTime: WORD to set the power down threshold time.

The input pointer has to be filled with the following parameter:

- current CCD power down threshold time as multiples of ms (0ms .. 47.9days)

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.15 PCO_GetExpTrigSignalStatus

Get the current status of the <exp trig> user input (one of the <control in> inputs at the rear of pco.power; see Fehler: Referenz nicht gefunden). If the signal level at the <exp trig> input is HIGH and the DIP switch shows _|‾ then the Status is TRUE. If the signal level at the <exp trig> input is HIGH and the DIP switch shows ‾|_ then the Status is FALSE.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetExpTrigSignalStatus(HANDLE ph, WORD* wExpTrgSignal)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wExpTrgSignal: Address of a WORD to get the current status of the <exp trig> user input.

The input pointer will be filled with the following parameter:

- <exp trig> signal status:
  - 0x0000 = [FALSE]
  - 0x0001 = [TRUE]
- the following combinations are possible:

- input signal: HIGH	DIP switch: _|‾	Status: TRUE
- input signal: HIGH	DIP switch: ‾|_	Status: FALSE
- input signal: LOW	DIP switch: _|‾	Status: FALSE
- input signal: LOW	DIP switch: ‾|_	Status: TRUE

Note:

- Due to response and processing times, e.g., caused by the interface and/or the operating system, the delay between the software delivered status and the actual status may be several 10 ms up to 100 ms. If timing is critical, it is strongly recommended that other trigger modes be used.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.16 PCO_GetCOCRunTime

Get and split the 'camera operation code' runtime into two DWORD. One will hold the longer part, in seconds, and the other will hold the shorter part, in nanoseconds. This function can be used to calculate the FPS. The sum of dwTime_s and dwTime_ns covers the delay, exposure and readout time. If external exposure is active, it returns only the readout time.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetCOCRuntime(HANDLE ph, DWORD* dwTime_s, DWORD*
dwTime_ns)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD* dwTime_s: Address of a DWORD to get the coc runtime part in seconds.
- DWORD* dwTime_ns: Address of a DWORD to get the coc runtime part in nanoseconds.

The input pointer will be filled with the following parameter:

- current coc runtime in seconds and nanoseconds (0s .. 136years(maybe enough), 0ns .. 999.999.999ns – all above it will show up in the seconds part)

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.17 PCO_GetFPSExposureMode (1200(h)s only)

The FPS Exposure Mode is available for the pco.1200hs camera model only!

The FPS exposure mode is useful if the user wants to get the maximum exposure time for the maximum frame rate.

The maximum image framerate (FPS = Frames Per Second) depends on the pixelrate, the vertical ROI and the exposure time.

$$FPS = FPS_{max} \approx \frac{\text{Pixelrate}}{\text{Pixels / line} \times n_{lines}} \qquad FPS = \frac{1}{t_{expos}}$$

$$\textit{valid for: } t_{expos} <= 1 / FPS_{max} \qquad \textit{valid for: } t_{expos} > 1 / FPS_{max}$$

where:

Pixels / line: Pixel in one full line, horizontal ROI will not affect this number because always a full line (including dummy pixel) must be read

n_{lines}: Number of lines (vertical ROI)

Note: The formula for FPS_{max} is a rough estimate. Actually, the FPS_{max} will be less due to some overhead time, which depends on the camera and sensor type as well as operating modes.

As can be seen from the formula, the exposure time affects the frame rate, if it gets longer than the frame rate period time. If the camera is in “FPS Exposure Mode” the maximum possible exposure time is automatically set such that $FPS = FPS_{max}$.

Please note, that, if the “FPS Exposure Mode” is on, the “PCO_Set Delay/Exposure Time” or PCO_Set Delay/Exposure Time Table” commands are ignored!

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetFPSExposureMode(HANDLE ph, WORD* wFPSExposureMode,
DWORD* dwFPSExposureTime)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wFPSExposureMode: Address of a WORD to get the FPS-exposure mode.
- DWORD* dwFPSExposureTime: Address of a DWORD to get the optimal exposure time in nanoseconds.

The input pointer will be filled with the following parameters:

- Mode: current mode
  - 0 = FPS Exposure Mode off, exposure time set by "PCO_Set Delay/Exposure Time" or "PCO_Set Delay/Exposure Time Table" command.
  - 1 = FPS Exposure Mode on, exposure time set automatically to 1 / FPS$_{max}$ "PCO_Set Delay/Exposure Time" or "PCO_Set Delay/Exposure Time Table" commands are ignored.

- Exposure time: The exposure time that will be set if "FPS Exposure Mode" is on. The exposure time depends on the current settings of the vertical ROI and the Pixelrate. **The returned time is always expressed in nanoseconds!**

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.18 PCO_SetFPSExposureMode (1200(h)s only)

The FPS Exposure Mode is available for the pco.1200hs camera model only!

See PCO_GetFPSExposureMode (1200(h)s only) for further explanation!

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetFPSExposureMode(HANDLE ph, WORD wFPSExposureMode,
DWORD* dwFPSExposureTime)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wFPSExposureMode: WORD to set the FPS-exposure mode.
- DWORD* dwFPSExposureTime: Address of a DWORD to get the optimal exposure time in nanoseconds.

The input should be filled with the following parameters:

- Mode: current mode
  - 0 = FPS Exposure Mode off, exposure time set by “PCO_Set Delay/Exposure Time” or “PCO_Set Delay/Exposure Time Table” command.
  - 1 = FPS Exposure Mode on, exposure time set automatically to 1 / FPS_{max} ”PCO_Set Delay/Exposure Time” or “PCO_Set Delay/Exposure Time Table” commands are ignored.

- Exposure time: The exposure time that will be set if “FPS Exposure Mode” is on. The exposure time depends on the current settings of the vertical ROI and the Pixelrate. **The returned time is always expressed in nanoseconds!**

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.19 PCO_GetModulationMode

The Modulation Mode is an optional feature which is not available for all camera models. See the descriptors of the camera.

Requests the modulation mode and its' corresponding parameters. The modulation mode is only available with special cameras. Please check the 2nd description for availability.

d.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetModulationMode(HANDLE ph, WORD *wModulationMode, DWORD
*dwPeriodicalTime, WORD *wTimebasePeriodical, DWORD *dwNumberOfExposures, LONG *lMonitorOffset)
```

e.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD *wModulationMode: Address of a WORD to receive the modulation mode.
- DWORD *dwPeriodicalTime: Address of a DWORD to receive the periodical time.
- DWORD *dwTimebasePeriodical: Address of a DWORD to receive the timebase of the periodical time.
- DWORD *dwNumberOfExposures: Address of a DWORD to receive the number of exposures
- LONG *lMonitorOffset: Address of a LONG to receive the monitor offset value.

The input parameters will be filled with one of the following parameters:

- current modulation mode:
  - 0x0000 = [modulation mode off]
  - 0x0001 = [modulation mode on]
- periodical time as a multiple of the timebase unit: The periodical time, delay and exposure time must meet the following condition : $t_p - (t_e + t_d) >$ 'Min Per Condition'
- timebase for periodical time
  - 0x0000 => timebase = [ns]
  - 0x0001 => timebase = [µs]
  - 0x0002 => timebase = [ms]
- number of exposures: number of exposures done for one frame
- monitor signal offset [ns]: controls the offset for the <status out> signal. The possible range is limited in a very special way. See tm in the above timing diagrams. The minimum range is –tstd…0. The negative limit can be enlarged by adding a delay. The maximum negative monitor offset is limited to -20us, no matter how long the delay will be set. The positive limit can be enlarged by longer exposure times than the minimum exposure time. The maximum positive monitor offset is limited to 20us, no matter how long the exposure will be set.

f.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.20 PCO_SetModulationMode

The Modulation Mode is an optional feature which is not available for all camera models. See the descriptors of the camera.

Sets the modulation mode and its' corresponding parameters. The modulation mode is only available with special cameras. Please check the 2nd descripton for availability.

g.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetModulationMode(HANDLE ph, WORD wModulationMode, DWORD dwPeriodicalTime, WORD wTimebasePeriodical, DWORD dwNumberOfExposures, LONG lMonitorOffset)
```

h.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wModulationMode: WORD to hold the modulation mode.
- DWORD dwPeriodicalTime: DWORD to hold the periodical time.
- DWORD dwTimebasePeriodical: DWORD to hold the timebase of the periodical time.
- DWORD dwNumberOfExposures: DWORD to hold the number of exposures
- LONG lMonitorOffset: LONG to hold the monitor offset value.

The input parameter should be filled with one of the following parameters:

- current modulation mode:
  - 0x0000 = [modulation mode off]
  - 0x0001 = [modulation mode on]
- periodical time as a multiple of the timebase unit: The periodical time, delay and exposure time must meet the following condition : t_p - (t_e + t_d) > 'Min Per Condition'
- timebase for periodical time
  - 0x0000 => timebase = [ns]
  - 0x0001 => timebase = [µs]
  - 0x0002 => timebase = [ms]
- number of exposures: number of exposures done for one frame
- monitor signal offset [ns]: controls the offset for the <status out> signal. The possible range is limited in a very special way. See tm in the above timing diagrams. The minimum range is –tstd…0. The negative limit can be enlarged by adding a delay. The maximum negative monitor offset is limited to -20us, no matter how long the delay will be set. The positive limit can be enlarged by longer exposure times than the minimum exposure time. The maximum positive monitor offset is limited to 20us, no matter how long the exposure will be set.

i.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.21 PCO_GetFrameRate (dimax, edge only)

Get framerate / exposure time.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetFrameRate (HANDLE ph, WORD* wFrameRateStatus, DWORD*
dwFrameRate, DWORD* dwFrameRateExposure)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wFrameRateStatus: Address of a WORD to get the frame rate status.
- DWORD* dwFrameRate: Address of a DWORD to get the frame rate.
- DWORD* dwFrameRateExposure: Address of a DWORD to get the exposure time.

The input pointers will be filled with the following parameters:

- Status, where:
  - 0x0000: Settings consistent, all conditions met
  - 0x0001: Framerate trimmed, framerate was limited by readout time
  - 0x0002: Framerate trimmed, framerate was limited by exposure time
  - 0x0004: Exposure time trimmed, exposure time cut to frame time
  - 0x8000: The return values dwFrameRate and dwFrameRateExposure are not yet validated. The values returned are the values which were passed with the most recent call of the PCO_SetFramerate function.
- configured framerate in mHz and exposure time in ns

Note:

- Framerate and exposure time are also affected by the “Set Delay/Exposure Time” command. **It is strongly recommend to use either the “Set Framerate” or the “Set Delay/Exposure Time” command**! The last issued command will determine the timing before calling the ARM command.
- Function is not supported by all cameras, at that moment only by the pco.dimax!

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.22 PCO_SetFrameRate (dimax, edge only)

Set framerate and exposure time. This command is intended to set directly the framerate and the exposure time of the camera. The framerate is limited by the readout time and the exposure time:

$$\text{Framerate} \leq \frac{1}{t_{\text{readout}}} \qquad \text{Framerate} \leq \frac{1}{t_{\text{expos}}}$$

Please note that there are some overhead times, therefore the real values can differ slightly, e.g. the maximum framerate will be a little bit less than 1 / exposure time. The mode parameter of the function call defines, how the function works if these conditions are not met.

The function differs, if the camera is recording (recording state = 1) or if recording is off:

Camera is recording:

The framerate / exposure time is changed immediately. The function recturns the actually configured framerate and exposure time.

Record is off:

The framerate / exposure time is stored. The function does not change the input values for framerate and exposure time. A succeeding "Arm Camera" command (PCO_ArmCamera) validates the input parameters together with other settings, e.g. The status returned indicates, if the input parameters are validated. The following procedure is recommended:

- Set framerate and exposure time using the PCO_SetFrameRate function.
- Do other settings, before or after the PCO_SetFrameRate function.
- Call the PCO_ArmCamera function in order to validate the settings.
- Retrieve the actually set framerate and exposure time using PCO_GetFrameRate.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetFrameRate (HANDLE ph, WORD* wFrameRateStatus, WORD wFramerateMode, DWORD* dwFrameRate, DWORD* dwFrameRateExposure)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wFrameRateStatus: Address of a WORD to set the mode and to get the frame rate status.
- WORD wFrameRateMode: Mode how to set the framerate.
- DWORD dwFrameRate: Address of a DWORD to set and get the frame rate.
- DWORD dwFrameRateExposure: Address of a DWORD to set and get the exposure time.

The input parameter should be filled with one of the following parameters:

- Mode, defines the way of operation when settings are inconsistent, where:
  - 0x0000: auto mode (camera decides which parameter will be trimmed)
  - 0x0001: Framerate has priority, (exposure time will be trimmed)
  - 0x0002: Exposure time has priority, (framerate will be trimmed)
  - 0x0003: Strict, function shall return with error if values are not possible.
- Framerate in mHz (milli!), thus e.g. 1kHz = 1000000
- Exposure time in ns

The input pointer will be filled with the following parameter:

- Status, where:
  - 0x0000: Settings consistent, all conditions met
  - 0x0001: Framerate trimmed, framerate was limited by readout time
  - 0x0002: Framerate trimmed, framerate was limited by exposure time
  - 0x0004: Exposure time trimmed, exposure time cut to frame time
  - 0x8000: The return values dwFrameRate and dwFrameRateExposure are not yet validated. In that case, the values returned are the values passed to the function.

Note:

- Framerate and exposure time are also affected by the “Set Delay/Exposure Time” command. **It is strongly recommend to use either the “Set Framerate” or the “Set Delay/Exposure Time” command**! The last issued command will determine the timing before calling the ARM command.
- Function is not supported by all cameras, at that moment only by the pco.dimax!

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.23 PCO_GetHWIOSignal (dimax, edge only)

Gets the settings of the requested hardware IO signal. To set the single signals use PCO_SetHWIOSignal (dimax, edge only). This functions is not available with all cameras. Actually it is implemented in the pco.dimax.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetHWIOSignal (HANDLE ph, WORD wSignalNum, PCO_Signal* pstrSignal)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wSignalNum: Select the signal to query. This parameter must be in the range of available hardware I/O signals.
- PCO_Signal* pstrSignal: Address of a PCO_Signal structure to get the settings of the hardware I/O signal.

The input structure will be filled with the following parameter:

Parameter:

- wSignalNum: Index of the signal.
- wEnabled: Flags showing enable state of the signal
  - 0x00: Signal is off
  - 0x01: Signal is active
- wType: Flags showing which signal type is selected:
  - 0x01: TTL
  - 0x02: High Level TTL
  - 0x04: Contact Mode
  - 0x08: RS485 differential
- wPolarity: Flags showing which signal polarity is selected:
  - 0x01: High level active
  - 0x02: Low Level active
  - 0x04: Rising edge active
  - 0x08: Falling edge active
- wFilter: Flags showing the filter option which is selected:
  - 0x01: Filter can be switched off (t > ~65ns)
  - 0x02: Filter can be switched to medium (t > ~1us)
  - 0x04: Filter can be switched to high (t > ~100ms)
- wSelected: In case the HWIOSignaldescription shows more than one SignalNames, this parameter can be used to select a different signal, e.g. ‘Status Busy’ or ‘Status Exposure’.

Notes:

- the command will be rejected, if Recording State is [run]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.24 PCO_SetHWIOSignal (dimax, edge only)

Sets the settings of the requested hardware IO signal. To get the single signals use PCO_GetHWIOSignal (dimax, edge only). This functions is not available with all cameras. Actually it is implemented in the pco.dimax.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetHWIOSignal (HANDLE ph, WORD wSignalNum, PCO_Signal*
pstrSignal)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wSignalNum: Select the signal to query. This parameter must be in the range of available hardware I/O signals.
- PCO_Signal* pstrSignal: Address of a PCO_Signal structure to set the settings of the hardware I/O signal.

The input structure should be filled with the following parameter:

Flags must not be combined, thus set only one of the described flags.

Parameter:

- wSignalNum: Index of the signal.
- wEnabled: Flags showing enable state of the signal
  - 0x00: Set Signal off
  - 0x01: Set Signal active
- wType: Flags showing which signal type is selected:
  - 0x01: Set to TTL
  - 0x02: Set to High Level TTL
  - 0x04: Set to Contact Mode
  - 0x08: Set to RS485 differential
- wPolarity: Flags showing which signal polarity is selected:
  - 0x01: Set High level active
  - 0x02: Set Low Level active
  - 0x04: Set Rising edge active
  - 0x08: Set Falling edge active
- wFilter: Flags showing the filter option which is selected:
  - 0x01: Set filter to off (t > ~65ns)
  - 0x02: Set filter to medium (t > ~1us)
  - 0x04: Set filter to high (t > ~100ms)
- wSelected: In case the HWIOSignaldescription shows more than one SignalNames, this parameter can be used to select a different signal, e.g. ‘Status Busy’ or ‘Status Exposure’.

Notes:

- the command will be rejected, if Recording State is [run]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.25 PCO_GetImageTiming

Gets the actual image timing.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetImageTiming (HANDLE ph, PCO_Image_Timing* pstrImageTiming)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wSignalNum: Select the signal to query. This parameter must be in the range of available hardware I/O signals.
- PCO_Image_Timing* pstrImageTiming: Address of a PCO_Image_Timing structure to get the timing of the actual settings.

The input structure will be filled with the following parameter:

Parameter:

- FrameTime_ns: Nanoseconds part of the time to expose and readout one image.
- FrameTime_s: Seconds part of the time to expose and readout one image.
- ExposureTime_ns: Nanoseconds part of the exposure time.
- ExposureTime_s: Seconds part of the exposure time.
- TriggerSystemDelay_ns: System internal minimum trigger delay, till a trigger is recognized and executed by the system.
- TriggerSystemJitter_ns: Maximum possible trigger jitter, which influences the real trigger delay. Real trigger delay=TriggerDelay_ns +/- TriggerSystemJitter
- TriggerDelay_ns: Total trigger delay part in ns, till a trigger is recognized and executed by the system.
- TriggerDelay_ns: Total trigger delay part in s, till a trigger is recognized and executed by the system.

Notes:

- the command will be rejected, if Recording State is [run]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.26 PCO_GetCameraSynchMode (dimax only)

Gets the camera synchronisation mode for a dimax. Dimax cameras can be cascaded in order to synchronize the timing of a camera chain. It is mandatory to set one of the cameras in the chain to master mode. Usually this is the first camera connected to the chain. All output side connected cameras should be set to slave mode. Those cameras will follow the timing of the master camera, thus all timing settings are disabled at the slave cameras.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetCameraSynchMode(HANDLE ph, WORD* wCameraSynchMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wCameraSynchMode: Address of a WORD to get the camera synchronisation mode.

The input pointer will be filled with the following parameter:

Return values:
- current mode:
  0x0000 = [off], 0x0001 = [master] or 0x0002 = [slave]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.27 PCO_SetCameraSynchMode (dimax only)

Sets the camera synchronisation mode for a dimax. Dimax cameras can be cascaded in order to synchronize the timing of a camera chain. It is mandatory to set one of the cameras in the chain to master mode. Usually this is the first camera connected to the chain. All output side connected cameras should be set to slave mode. Those cameras will follow the timing of the master camera, thus all timing settings are disabled at the slave cameras.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetCameraSynchMode(HANDLE ph, WORD wCameraSynchMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wCameraSynchMode: WORD to set the camera synchronisation mode.

The input pointer should be filled with the following parameter:

Return values:
- current mode:
  0x0000 = [off], 0x0001 = [master] or 0x0002 = [slave]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.28 PCO_GetFastTimingMode (dimax only)

Gets the camera fast timing mode for a dimax. To increase the possible exposure time with high frame rates it is possible to enable the 'Fast Timing' mode. This means that the maximum possible exposure time can be longer than in normal mode, while getting stronger offset drops. In case, especially in PIV applications, image quality is less important, but exposure time is, this mode reduces the gap between exposure end and start of the next exposure from ~75uS to 3.5uS.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetFastTimingMode(HANDLE ph, WORD* wFastTimingMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wFastTimingMode: Address of a WORD to get the camera fast timing mode.

The input pointer will be filled with the following parameter:

Return values:
- current mode:
  0x0000 = [off], 0x0001 = [on]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.29 PCO_SetFastTimingMode (dimax only)

Sets the camera fast timing mode for a dimax. To increase the possible exposure time with high frame rates it is possible to enable the 'Fast Timing' mode. This means that the maximum possible exposure time can be longer than in normal mode, while getting stronger offset drops. In case, especially in PIV applications, image quality is less important, but exposure time is, this mode reduces the gap between exposure end and start of the next exposure from ~75uS to 3.5uS.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetFastTimingMode(HANDLE ph, WORD wFastTimingMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wFastTimingMode: WORD to set the camera fast timing mode.

The input pointer should be filled with the following parameter:

Return values:
- current mode:
  0x0000 = [off], 0x0001 = [on]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.3.30 PCO_GetSensorSignalStatus (pco.1400 only)

Gets the signal state of the camera sensor. The command must not be deemed to be a realtime response of the sensor, since the command path adds a system dependent delay. Sending a command and getting the camera response lasts about 2ms (+/- 1ms; for 'simple' commands). In case you need a closer synchronisation use hardware signals.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetSensorSignalStatus(HANDLE ph, DWORD* dwStatus, DWORD*
dwImageCount, DWORD* dwReserved1, DWORD* dwReserved2)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD* dwStatus: DWORD to get the sensor signal states.
- DWORD* dwImageCount: DWORD to get the image count of the last finished image.
- DWORD* dwReserved: for future use.

The input pointer will be filled with the following parameter:

Return values:
- Status (flags):
  Bit0: 0 → camera is not busy (waiting for trigger), 1 → camera is busy
  Bit1: 0 → camera is not idle, 1 → camera is idle (record state is off)
  Bit2: 0 → sensor is not exposing, 1 → sensor is exposing
  Bit3: 0 → sensor is not in readout, 1 → sensor is in readout
  Bit4-31: reserved for future use, set to 0.
- dwImageCount: Image count of the last finished image. This is necessary in case the timing is too close to get signal shifts at the calling application (video mode and exposure = readout).
- dwReserved: for future use, set to 0.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.4 Storage Control

This function group defines storage and camRAM access functions to the camera, especially camRAM size, camRAM segment size and the active camRAM segment.

Overview:

Command:
PCO_GetStorageStruct
PCO_SetStorageStruct
PCO_GetCameraRamSize
PCO_GetCameraRamSegmentSize
PCO_SetCameraRamSegmentSize
PCO_ClearRamSegment
PCO_GetActiveRamSegment
PCO_SetActiveRamSegment

4.4.1 PCO_GetStorageStruct

Get the complete set of the storage functions settings. Please fill in all wSize parameters, even in embedded structures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetStorageStruct(HANDLE ph, PCO_Storage* strStorage)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- PCO_Storage* strStorage: Address of a PCO_Storage structure to get the storage settings.

Please fill in all wSize parameters, even in embedded structures.

The input pointer will be filled with the parameters following this function description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.4.2 PCO_SetStorageStruct

Set the complete set of the storage functions settings. Please fill in all wSize parameters, even in embedded structures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetStorageStruct(HANDLE ph, PCO_Storage* strStorage)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- PCO_Storage* strStorage: Address of a PCO_Storage structure to set the storage settings.

Please fill in all wSize parameters, even in embedded structures.

The input pointer has to be filled with the parameters following this function description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.4.3 PCO_GetCameraRamSize

Get the camera RAM (camRAM) size.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetCameraRamSize(HANDLE ph, DWORD* dwRamSize, WORD*
wPageSize)
```

b.) Input parameters:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD* dwRamSize: Address of a DWORD to get the camRAM size.
- WORD* wPageSize: Address of a WORD to get the CamRAM page.

The input pointer will be filled with the following parameters:

- RAM size: size of the total camera RAM as multiples of pages
- page size: size of one page as multiples of pixels

Note:

- One page is the smallest unit for RAM segmentation as well as for storing images. Segment sizes can only configured as multiples of pages. The size reserved for one image is also calculated as multiples of whole pages. Therefore, there may be some unused RAM memory if the page size is not exactly a multiple of the image size. The number of pages needed for one image depends on the image size (X_{res} x Y_{res}) divided by the pixels per page (page size). Every page size that has been started must be considered, so if 50.6 pages are used for an image 51 pages are actually needed for this image. With this value of 'pages per image',the user can calculate the number of images fitting into the segment.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.4.4 PCO_GetCameraRamSegmentSize

Get camera RAM (camRAM) segment size.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetCameraRamSegmentSize(HANDLE ph, DWORD* dwRamSegSize)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD* dwRamSegSize: Address of a DWORD array to get the segment sizes.

The input pointer will be filled with the following parameter:

- Size of segment 1 .. segment 4 as multiples of RAM pages

Note:

- the sum of all segment sizes must not be larger than the total size of the RAM (as multiples of pages)
- **size** = [0] indicates that the segment will not be used
- using only one segment is possible by assigning the total RAM size to segment 1 and 0x0000 to all other segments.
- The segment number is 1 based, while the array dwRamSegSize is zero based, e.g. ram size of segment 1 is stored in dwRamSegSize[0]!

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.4.5 PCO_SetCameraRamSegmentSize

Set Camera RAM Segment Size. The segment size has to be big enough to hold at least two images.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetCameraRamSegmentSize(HANDLE ph, DWORD* dwRamSegSize)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD* dwRamSegSize: Address of a DWORD array to set the segment sizes.

The input pointer should be filled with the following parameters:

- Size of segment 1 .. segment 4 as multiples of RAM pages

Note:

- the sum of all segment sizes must not be larger than the total size of the RAM (as multiples of pages)
- a single segment size can have the value 0x0000, but the sum of all four segments must be bigger than the size of two images.
- the command will be rejected, if Recording State is [run]
- The segment number is 1 based, while the array dwRamSegSize is zero based, e.g. ram size of segment 1 is stored in dwRamSegSize[0]!
- **This function will result in all segments being cleared. All previously recorded images will be lost!**

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.4.6 PCO_ClearRamSegment

Clear active camera RAM segment, delete all image info and prepare segment for new images.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_ClearRamSegment(HANDLE ph)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.4.7 PCO_GetActiveRamSegment

Get the active camera RAM segment. The active segment is where images are stored.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetActiveRamSegment(HANDLE ph, WORD* wActSeg)
```

b.) Input parameters:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wActSeg: Address of a WORD to get the active segment.

The input pointer will be filled with the following parameter:

- segment number of the currently active segment (valid numbers are 1,2,3,4)

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.4.8 PCO_SetActiveRamSegment

Set the active camera RAM segment. The active segment is where images are stored.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetActiveRamSegment(HANDLE ph, WORD wActSeg)
```

b.) Input parameters:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wActSeg: WORD to set the active segment.

The input pointer should be filled with the following parameter:

- segment number of the active segment (valid numbers are 1,2,3,4)

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5 Recording Control

This function group defines recording control functions, especially recording states and modes.

Overview:

Command:
PCO_GetRecordingStruct
PCO_SetRecordingStruct
PCO_GetStorageMode (Recorder / FIFO buffer)
PCO_SetStorageMode (Recorder / FIFO buffer)
PCO_GetRecorderSubmode (Sequence / Ring buffer)
PCO_SetRecorderSubmode (Sequence / Ring buffer)
PCO_GetRecordingState
PCO_SetRecordingState (*)
PCO_ArmCamera (*)
PCO_GetAcquireMode (Auto / External)
PCO_SetAcquireMode (Auto / External)
PCO_GetAcqEnblSignalStatus
PCO_SetDateTime
PCO_GetTimestampMode
PCO_SetTimestampMode
PCO_GetRecordStopEvent
PCO_SetRecordStopEvent
PCO_StopRecord

4.5.1 PCO_GetRecordingStruct

Get the complete set of the recording function settings. Please fill in all wSize parameters, even in embedded structures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetRecordingStruct(HANDLE ph, PCO_Recording* strRecording)
```

b.) Input parameters:

- HANDLE ph: Handle to a previously opened camera device.
- PCO_Recording* strRecording: Address of a PCO_Recording structure to get the recording settings.

Please fill in all wSize parameters, even in embedded structures.

The input pointer will be filled with the parameters following this function description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.2 PCO_SetRecordingStruct

Set the complete set of the recording functions settings. Please fill in all wSize parameters, even in embedded structures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetRecordingStruct(HANDLE ph, PCO_Recording* strRecording)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- PCO_Recording* strRecording: Address of a PCO_Recording structure to set the recording settings.

Please fill in all wSize parameters, even in embedded structures.

The input pointer must be filled with the parameters following this function description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.3 PCO_GetStorageMode (Recorder / FIFO buffer)

Get storage mode [recorder] or [FIFO buffer].

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetStorageMode(HANDLE ph, WORD* wStorageMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wStorageMode: Address of a WORD to get the active storage mode.

The input pointer will be filled with the following parameter:

- current storage mode (see boxes below):
  - 0x0000 = [recorder] mode
  - 0x0001 = [FIFO buffer] mode

Recorder Mode	**FIFO Buffer mode**
• images are recorded and stored within the internal camera memory (camRAM) • Live View transfers the most recent image to the PC (for viewing / monitoring) • indexed or total image readout after the recording has been stopped	• all images taken are transferred to the PC in chronological order • camera memory (camRAM) is used as a huge FIFO buffer to bypass short data transmission bottlenecks • if buffer overflows, the oldest images are overwritten

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.4 PCO_SetStorageMode (Recorder / FIFO buffer)

Set storage mode [recorder] or [FIFO buffer].

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetStorageMode(HANDLE ph, WORD wStorageMode)
```

b.) Input parameters:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wStorageMode: WORD to set the active storage mode.

The input pointer should be filled with the following parameter:

- current storage mode (see boxes below):
  - 0x0000 = [recorder] mode
  - 0x0001 = [FIFO buffer] mode

Recorder Mode	**FIFO Buffer mode**
• images are recorded and stored within the internal camera memory (camRAM) • Live View transfers the most recent image to the PC (for viewing / monitoring) • indexed or total image readout after the recording has been stopped	• all images taken are transferred to the PC in chronological order • camera memory (camRAM) is used as huge FIFO buffer to bypass short bottlenecks in data transmission • if buffer overflows, the oldest images are overwritten • if Set Recorder = [stop] is sent, recording is stopped and the transfer of the current image to the PC is finished. Images not read are stored within the segment and can be read with the Read Image From Segment command.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.5 PCO_GetRecorderSubmode (Sequence / Ring buffer)

Get recorder submode: [sequence] or [ring buffer] (see explanation boxes below). Recorder submode is only available if the storage mode is set to [recorder].

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetRecorderSubmode(HANDLE ph, WORD* wRecSubmode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wRecSubmode: Address of a WORD to get the active recorder submode.

The input pointer will be filled with the following parameter:

- current recorder submode:
  - 0x0000 = [sequence]
  - 0x0001 = [ring buffer].

recorder submode = [sequence]	**recorder submode =[ring buffer]**
• recording is stopped when the allocated buffer is full	• camera records continuously into ring buffer • if the allocated buffer overflows, the oldest images are overwritten • recording is stopped by software or disabling acquire signal (<acq enbl>)

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.6 PCO_SetRecorderSubmode (Sequence / Ring buffer)

Set recorder submode: [sequence] or [ring buffer] (see explanation boxes below). Recorder submode is only available if the storage mode is set to [recorder].

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetRecorderSubmode(HANDLE ph, WORD wRecSubmode)
```

b.) Input parameters:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wRecSubmode: WORD to set the active recorder submode.

The input data should be filled with the following parameter:

- configured recorder submode:
  - 0x0000 = [sequence]
  - 0x0001 = [ring buffer].

recorder submode = [sequence]	**recorder submode = [ring buffer]**
• recording is stopped when the allocated buffer is full	• camera records continuously into ring buffer • if the allocated buffer overflows, the oldest images are overwritten • recording is stopped by software or disabling acquire signal (<acq enbl>)

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.7 PCO_GetRecordingState

Requests the current **recording state**.

The **recording state** controls the status of the camera. If the **recording status** is [run], images can be started by **exposure trigger** and <acq enbl>. If the **recording status** is [clear]'ed or [stop]'ped, all image readout or exposure sequences are stopped and the sensors (CCD or CMOS) are running in a special idle mode to prevent dark charge accumulation.

The **recording status** has the highest priority compared to functions like <acq enbl> or **exposure trigger**.

The **recording status** is controlled by:

- software command: **set recording status** = [run]

The **recording status** is cleared by:

- powering ON the camera
- software command: **set recording status** = [stop]
- software command: **reset** all settings to default values

During recording the user can read out most recent images, while in 'recorder mode', using the AddBufferEx function. In Fifo mode the user will get recorded images in contiguous order. Once a buffer is provided, a transfer occurs automatically at the next possible interface transfer cycle. This image transfer does not affect CamRam recording. CamRam recording is run independently without the need of applications intervention. The possible frame rate of the CamRam recording is completely different to the interface transfer frame rate! The camera frame rate can be determined by calling the GetCocRuntime. E.g. a pco.1600 produces 29.88fps (dual ADC, 40MHz), which gives 109MB/sec. The firewire can do about 30MB/sec on the whole.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetRecordingState(HANDLE ph, WORD* wRecState)
```

b.) Input parameters:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wRecState: Address of a WORD to get the active recording state.

The input pointer will be filled with the following parameter:

- current recording status:
  - 0x0001 = camera is running, in **recording status** = [run]
  - 0x0000 = camera is idle or [stop]'ped, not ready to take images

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.8 PCO_SetRecordingState (*)

Sets the current **recording status** and waits till the status is valid. If the state can't be set the function will return an error.

The **recording status** controls the status of the camera. If the **recording status** is [run], images are recorded automatically (trigger mode is auto) or can be started by external signals **exposure trigger** and <acq enbl>. If the **recording status** is [clear]'ed or [stop]'ped, all image readout or exposure sequences are stopped and the sensors (CCD or CMOS) are running in a special idle mode to prevent dark charge accumulation.

The **recording status** has the highest priority compared to functions like <acq enbl> or **exposure trigger**.

The **recording status** is controlled by:

- software command: **set recording status** = [run]

The **recording status** is cleared by:

- powering ON the camera
- software command: **set recording status** = [stop]
- software command: **reset** all settings to default values

Notes:

- It is necessary to issue an **arm camera** command before every **set recording status** command in order to ensure that all settings are accepted correctly. Do not change settings between the **arm camera** command and the **set recording status** command.
- If a **set recording status** = [stop] command is sent and the current status is already [stop]'ped, nothing will happen (only warning, error message). If the camera is in [run]'ing state, it will last some time (system delay + last image readout), until the camera is stopped. The system delay depends on the PC and the image readout depends on the image size transferred. The SetRecordingState = [stop] checks for a stable stop state by calling GetRecordingState. Please call PCO_CancelImages to remove pending buffers from the driver.
- If a **set recording status** = [run] command is sent and the current status is already [run], a warning message will be generated
- If a successful **set recording status** = [run] command is sent and recording is started, the images from a previous record to the active segment are lost!
- During a recording session (**recording status** = [run]) it is possible to change the timing by calling PCO_SetDelayExposureTime.

During recording the user can read out most recent images, while in 'recorder mode', using the AddBufferEx function. In Fifo mode the user will get recorded images in contiguous order. Once a buffer is provided, a transfer occurs automatically at the next possible interface transfer cycle. This image transfer does not affect CamRam recording. CamRam recording is run independently without the need of applications intervention. The possible frame rate of the CamRam recording is completely different to the interface transfer frame rate! The camera frame rate can be determined by calling the GetCocRuntime. E.g. a pco.1600 produces 29.88fps (dual ADC, 40MHz), which gives 109MB/sec. The firewire can do about 30MB/sec on the whole.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetRecordingState(HANDLE ph, WORD wRecState)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wRecState: WORD to set the active recording state.

The input pointer will be filled with the following parameter:

- current recording status:
  - 0x0001 = camera is running, in **recording status** = [run]
  - 0x0000 = camera is idle or [stop]'ped, not ready to take images

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.9 PCO_ArmCamera (*)

Arms, i.e. prepares the camera for a consecutive **set recording status** = [run] command. All configurations and settings made up to this moment are accepted and the internal settings of the camera are prepared. Thus the camera is able to start immediately when the **set recording status** = [run] command is performed.

Note: It is required to issue an **arm camera** command before every **set recording state** = [run] command in order to ensure that all settings are accepted correctly. Do not change settings between **arm camera** command and **set recording status** command. It is possible to change the timing by calling PCO_SetDelayExposureTime after the recording state = [run].

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_ArmCamera(HANDLE ph)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.10 PCO_GetAcquireMode (Auto / External)

Get acquire mode: [auto] or [external] (see camera manual for explanation)

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetAcquireMode(HANDLE ph, WORD* wAcquMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wAcquMode: Address of a WORD to get the acquire mode.

The input pointer will be filled with the following parameter:

- current acquire mode:
  - 0x0000 = [auto] - the external <acq enbl> input is ignored
  - 0x0001 = [external] - external signal at the <acq enbl> input controls whether images are stored or not
  - 0x0002 = [external] - the external control input <acq enbl> is a dynamic frame start signal. If this input has got a rising edge TRUE (level depending on the DIP switch), a frame will be started with modulation mode. This is only available with modulation mode enabled (see camera description).

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.11 PCO_SetAcquireMode (Auto / External)

Set acquire mode: [auto] or [external] (see camera manual for explanation).

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetAcquireMode(HANDLE ph, WORD wAcquMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wAcquMode: WORD to set the acquire mode.

The input data should be filled with the following parameter:

- acquire mode to be selected:
  - 0x0000 = [auto] - all images taken are stored
  - 0x0001 = [external] - the external control input <acq enbl> is a static enable signal of images. If this input is TRUE (level depending on the DIP switch), exposure triggers are accepted and images are taken. If this signal is set FALSE, all exposure triggers are ignored and the sensor readout is stopped.
  - 0x0002 = [external] - the external control input <acq enbl> is a dynamic frame start signal. If this input has got a rising edge TRUE (level depending on the DIP switch), a frame will be started with modulation mode. This is only available with modulation mode enabled (see camera description).

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.12 PCO_GetAcqEnblSignalStatus

Get the current status of the <acq enbl> user input (one of the <control in> inputs at the rear of pco.power; see Fehler: Referenz nicht gefunden). If the signal level at the <acq enbl> input is HIGH and the DIP switch shows ⎍ then the Status is TRUE. If the signal level at the <acq enbl> input is HIGH and the DIP switch shows ⎎ then the Status is FALSE. ⎍ or ⎎.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetAcqEnblSignalStatus(HANDLE ph, WORD* wAcquEnableState)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wAcquEnableState: Address of a WORD to get the <acq enbl> user input.

The input pointer will be filled with the following parameter:

- <acq enbl> signal status:
  - 0x0000 = [FALSE]
  - 0x0001 = [TRUE]
- the following combinations are possible:

- input signal: HIGH	DIP switch: ⎍	Status: TRUE
- input signal: HIGH	DIP switch: ⎎	Status: FALSE
- input signal: LOW	DIP switch: ⎍	Status: FALSE
- input signal: LOW	DIP switch: ⎎	Status: TRUE

Note:

- Due to response and processing times e.g. caused by the interface and/or the operating system, the delay between the delivered status and the actual status may be several 10 ms up to 100 ms. If timing is critical it is strongly recommended to use other trigger modes.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.13 PCO_SetDateTime

Set date and time for the **timestamp** function. The date and time is updated automatically, as long as the camera is supplied with power. When powering up the camera, then this command should be done once.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetDateTime(HANDLE ph, BYTE ucDay, BYTE ucMonth, WORD wYear,
WORD wHour, BYTE ucMin, BYTE ucSec)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- BYTE ucDay: BYTE to set the day of month.
- BYTE ucMonth: BYTE to set the month.
- WORD wYear: WORD to set the year.
- WORD wHour: WORD to set the hour.
- BYTE ucMin: BYTE to set the minute.
- BYTE ucSec: BYTE to set the second.

The input data should be filled with the following parameter:

Parameter:
- date: day:month:year binary coded
  example: 21:march:2003 => 0x150307D3
- time: hours:min:sec binary coded
  example: 17h:05min:32sec => 0x00110520

Note:

- [ms] and [µs] values are set to zero, when this command is executed
- this command should be performed, when powering up the camera

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.14 PCO_GetTimestampMode

Get mode of the timestamp function.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetTimestampMode(HANDLE ph, WORD* wTimeStampMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wTimeStampMode: Address of a WORD to get the time stamp mode.

The input pointer will be filled with the following parameter:

- 0x0000 = no stamp in image
- 0x0001 = BCD coded stamp in the first 14 pixel
- 0x0002 = BCD coded stamp in the first 14 pixel + ASCII text
- 0x0003 = ASCII text only (see descriptor for availability)

Note:

- details about modes are explained in the following command **set timestamp mode**.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.15 PCO_SetTimestampMode

Set mode of the timestamp function.
To obtain information about the recording time of images this command can be useful. It writes a continuous image number and date / time information with a resolution of 10 µs direct into the raw image data. The first 14 pixels (top left corner) are used to hold this information. The numbers are coded in BCD with one byte per pixel, which means that every pixel can hold 2 digits. If the pixels have more resolution as 8 bits, then the BCD digits are left bound adjusted and the lower bits are zero. Additionally to this 14 pixels, the information can be written in ASCII text for direct inspection. A 8 by 8 pixel array is used per ASCII digit. The digits are displayed below the BCD coded line.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetTimestampMode(HANDLE ph, WORD wTimeStampMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wTimeStampMode: WORD to set the time stamp mode.

The input data should be filled with the following parameter:

- 0x0000 = no stamp in image
- 0x0001 = BCD coded stamp in the first 14 pixel
- 0x0002 = BCD coded stamp in the first 14 pixel + ASCII text
- 0x0003 = ASCII text only (see descriptor for availability)

Note:

- the image number is set to value = [1], when an **arm** command is performed
- using this command without setting the [date] / [time] results in an error message

Format of BCD coded pixels:

pixel 1	pixel 2	pixel 3	pixel 4	pixel 5	pixel 6	pixel 7
image counter (MSB) (00 … 99)	image counter (00 … 99)	image counter (00 … 99)	image counter (LSB) (00 … 99)	year (MSB) (20)	year (LSB) (03 … 99)	month (01 … 12)
pixel 8	**pixel 9**	**pixel 10**	**pixel 11**	**pixel 12**	**pixel 13**	**pixel 14**
day (01 ... 31)	h (00 … 23)	min (00 … 59)	s (00 … 59)	µs * 10000 (00 … 99)	µs * 100 (00 … 99)	µs (00 … 90)

Format of ASCII text:

• image number:	8 digits	[1 … 99999999]
• date:	9 digits	[01JAN2003 … 31DEZ2099]
• time:	15 digits	[00:00:00.000000 … 23:59:59.999990]

- number, date and time are separated by blanks

Example:
00103822 03JAN2003 17:35:12.376810

image number:	00103822
date:	03 January 2003
time:	17 h, 35 min, 12 s, 376 ms, 810 µs

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.16 PCO_GetRecordStopEvent

This command can be used for setting up the record stop event. After a stop event the camera records the configured number of images and stops after that. The command is useful to record a series of images to see what happens before and after the stop event.

A record stop event can be either a software command or an edge at the <acq enbl> input (at the power unit). The edge detection depends on the DIP switch setting at the power unit. If the DIP switch shows ⎍ then a rising edge is the stop event. If the DIP switch shows ⊔ then a falling edge is the stop event.

The software command is the command “Stop Record” described below.

Use the record stop even function only when Storage Mode = [Recorder] and Recorder Submode = [Ring buffer]!

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetRecordStopEvent(HANDLE ph, WORD* wRecordStopEventMode,
DWORD* dwRecordStopDelayImages)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wRecordStopEventMode: WORD* to get the record stop event mode.
- DWORD* dwRecordStopDelayImages: DWORD* to get the number of images recorded after the record stop event occured.

The input pointer will be filled with the following parameter:

Parameter:

- Mode: record stop event configuration:
  - 0x0000 = no record stop event is accepted
  - 0x0001 = record stop by software command
  - 0x0002 = record stop by edge at the <acq. enbl.> input or by software
- Delay in images: number of images which are taken after the record stop event. If the number of images is taken, record will be stopped automatically.

Note:

- Use the record stop event function only when Storage Mode = [Recorder] and Recorder Submode = [Ring buffer]!
- Due to internal timing issues the actual number of images taken after the event may differ by +/- 1 from the configured number.
- The command is not available for all cameras. Currently it is only implemented for the pco.1200hs. See the descriptor for availability.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.17 PCO_SetRecordStopEvent

This command can be used for setting up the record stop event. After a stop event the camera records the configured number of images and stops after that. The command is useful to record a series of images to see what happens before and after the stop event.

A record stop event can be either a software command or an edge at the <acq enbl> input (at the power unit). The edge detection depends on the DIP switch setting at the power unit. If the DIP switch shows _|‾|_ then a rising edge is the stop event. If the DIP switch shows ‾|_|‾ then a falling edge is the stop event.

The software command is the command "Stop Record" described below.

Use the record stop even function only when Storage Mode = [Recorder] and Recorder Submode = [Ring buffer]!

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetRecordStopEvent(HANDLE ph, WORD wRecordStopEventMode,
DWORD dwRecordStopDelayImages)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wRecordStopEventMode: WORD to set the record stop event mode.
- DWORD dwRecordStopDelayImages: DWORD to set the number of images recorded after the record stop event occured.

The input data should be filled with the following parameter:

Parameter:

- Mode: record stop event configuration:
  - 0x0000 = no record stop event is accepted
  - 0x0001 = record stop by software command
  - 0x0002 = record stop by edge at the <acq. enbl.> input or by software
- Delay in images: number of images which are taken after the record stop event. If the number of images is taken, record will be stopped automatically.

Note:

- Use the record stop event function only when Storage Mode = [Recorder] and Recorder Submode = [Ring buffer]!
- Due to internal timing issues the actual number of images taken after the event may differ by +/- 1 from the configured number.
- The command is not available for all cameras. Currently it is only implemented for the pco.1200hs. See the descriptor for availability.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.5.18 PCO_StopRecord

This command is useful to generate a stop event by software for the record stop event mode. See also PCO_GetRecordStopEvent and PCO_SetRecordStopEvent.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_StopRecord(HANDLE ph, WORD* wReserved0, DWORD *dwReserved1)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wReserved0: Set to zero.
- DWORD dwReserved1: Set to zero

Note:

- Use the record stop event function only when Storage Mode = [Recorder] and Recorder Submode = [Ring buffer]!
- Due to internal timing issues the actual number of images taken after the event may differ by +/- 1 from the configured number.
- The command is not available for all cameras. Currently it is only implemented for the pco.1200hs. See the descriptor for availability.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6 Image Buffer Data

This function group defines image property functions. The readout of images is part of the API-management commands.

Overview:

Command:
PCO_GetImageStruct
PCO_GetSegmentStruct
PCO_GetSegmentImageSettings
PCO_GetNumberOfImagesInSegment
PCO_GetBitAlignment
PCO_SetBitAlignment
PCO_WriteHotPixelList
PCO_ReadHotPixelList
PCO_ClearHotPixelList
PCO_GetHotPixelCorrectionMode
PCO_SetHotPixelCorrectionMode
PCO_GetInterfaceOutputFormat (dimax only)
PCO_SetInterfaceOutputFormat (dimax only)
PCO_PlayImagesFromSegmentHDSDI (dimax only)
PCO_GetPlayPositionHDSDI (dimax only)
PCO_GetMetaDataMode (dimax only)
PCO_SetMetaDataMode (dimax only)

4.6.1 PCO_GetImageStruct

Get the complete set of the image functions settings. Please fill in all wSize parameters, even in embedded structures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetImageStruct(HANDLE ph, PCO_Image* strImage)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- PCO_Image* strImage: Address of a PCO_Image structure to get the image settings.

Please fill in all wSize parameters, even in embedded structures.

The input pointer will be filled with the parameters following this function description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.2 PCO_GetSegmentStruct

Get the complete set of the segment image settings of one segment. This struct is part of the PCO_Image structure. There exist four structures of this type, each for one segment. Please fill in all wSize parameters, even in embedded structures.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetSegmentStruct(HANDLE ph, WORD wSegment, PCO_Segment*
strSegment)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wSegment: WORD to address the segment structure (1,2,3,4) of the desired segment.
- PCO_Segment* strSegment: Address of a PCO_Segment structure to get the segment image settings of the addressed segment.

Please fill in all wSize parameters, even in embedded structures. wSegment is 1 based whereas the structure elements are zero based, e.g. strSegment[0] is the info for segment number 1.

The input pointer will be filled with the parameters following this function description.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.3 PCO_GetSegmentImageSettings

Get the image settings for images stored into one of the four segments.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetSegmentImageSettings(HANDLE ph, WORD wSegment, WORD*
wXRes, WORD* wYRes, WORD* wBinHorz, WORD* wBinVert, WORD* wRoiX0, WORD* wRoiY0,
WORD* wRoiX1, WORD* wRoiY1)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wSegment: WORD to address desired segment (1,2,3,4).
- WORD* wXRes: Address of a WORD to get the x-resolution of the recorded images.
- WORD* wYRes: Address of a WORD to get the y-resolution of the recorded images
- WORD* wBinHorz: Address of a WORD to get the horizontal binning of the recorded images
- WORD* wBinVert: Address of a WORD to get the vertical binning of the recorded images
- WORD* wRoiX0, wRoiY0, wRoiX1, wRoiY1: Address of 4 WORDs to get the ROI.

The input pointer will be filled with the following parameter:

- Segm. = number of segment
- Res. h. = resulting horizontal resolution (sensor resolution, ROI, binning)
- Res. v. = resulting vertical resolution (sensor resolution, ROI, binning)
- Bin. x, y = binning setting for horizontal (x) and vertical (y) direction
- ROI x0, y0, x1, y1 = configured region of interest (ROI, in pixels) within the complete image of the sensor

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.4 PCO_GetNumberOfImagesInSegment

Get the number of valid images within the segment. The operation is slightly different due to the selected storage mode:

In [recorder mode], if recording is not stopped and in [FIFO buffer mode] the number of images is dynamic due to read and write accesses to the camera RAM. If the **camera storage mode** is in [recorder mode] and recording is stopped, the number is fixed.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetNumberOfImagesInSegment(HANDLE ph, WORD wSegment,
DWORD* dwValidImageCnt, DWORD* dwMaxImageCnt)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wSegment: WORD to address the segment structure of the desired segment (1,2,3,4).
- DWORD* dwValidImageCnt: Address of a DWORD to get the valid images of the addressed segment.
- DWORD* dwMaxImageCnt: Address of a DWORD to get the maximum possible images of the addressed segment.

The input data should be and the input pointer will be filled with the parameters following parameters.

- Segm. = segment of the camera RAM which is to be requested
- Valid Num. = number of valid images in the segment.
- Max. Num. = maximum number of images which may be saved to this segment

In [FIFO buffer mode] the ratio of valid number of images to the maximum number of images is a kind of filling level indicator.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.5 PCO_GetBitAlignment

Gets the actual bit alignment of the raw image data.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetBitAlignment(HANDLE ph, WORD* wBitAlignment)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wBitAlignment: Address of a buffer to get to the bit alignment.

The input pointer will be filled with the following parameter:

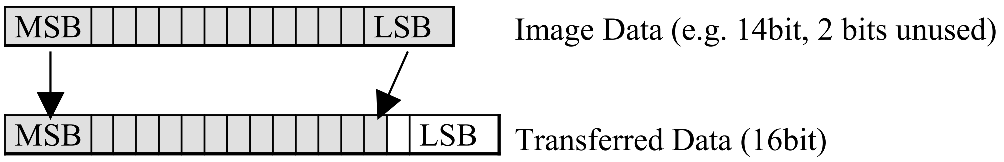
- wBitAlignment:
  - 0x0000 = [MSB aligned]; all raw image data will be aligned to the MSB. This is the default setting.
  - 0x0001 = [LSB aligned]; all raw image data will be aligned to the LSB.

c.) Return value:

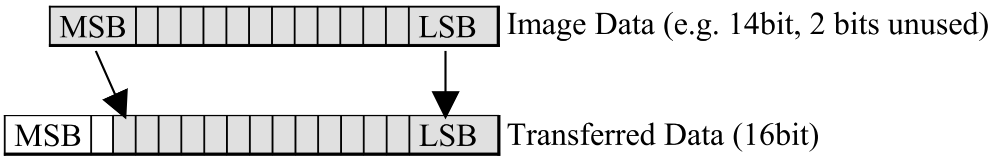
- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

Since the image data is less than a WORD, which is 16 bit, the data can be placed in two reasonable ways. Either you set the LSB of the image data to the LSB of the transferred data or you set the MSB of the image data to the MSB of the transferred data.

Alignment set to 0 – MSB aligned:



Alignment set to 1 – LSB aligned:



4.6.6 PCO_SetBitAlignment

Sets the actual bit alignment of the raw image data. See PCO_GetBitAlignment for further details.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetBitAlignment(HANDLE ph, WORD wBitAlignment)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wBitAlignment: WORD to set to the bit alignment.

Set the following parameter:

- wBitAlignment:
  - 0x0000 = [MSB aligned]; all raw image data will be aligned to the MSB. This is the default setting.
  - 0x0001 = [LSB aligned]; all raw image data will be aligned to the LSB.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.7 PCO_WriteHotPixelList

This command is optional and depends on the hardware and firmware. Check the availability according to the camera descriptor (HOT_PIXEL_CORRECTION). To change the hotpixel list inside the camea, please first call PCO_ReadHotPixelList. Then modify the list and write it back with this command. We recommend doing a backup of the list after readout. An invalid list will break the hotpixel correction!

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_WriteHotPixelList(HANDLE ph, WORD wListNo, WORD wNumValid,
WORD* wHotPixX, WORD* wHotPixY)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wListNo: WORD to address a list (zero based index).
- WORD wNumValid: WORD to tell the number of valid hotpixels
- WORD* wHotPixX: WORD array which holds the x coordinates of a hotpixel list
- WORD* wHotPixY: WORD array which holds the y coordinates of a hotpixel list

Set the following parameter:

- wListNo: Number of the list to modify (0 …)
- wNumValid: Number of valid hotpixels in the list, e.g. 10
- *wHotPixX(Y): Array of words. Each array member holds a x(y)-coordinate of a hotpixel. In our sample with 10 hotpixels, 10 valid x(y)-coordinates have to be inside the array

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.8 PCO_ReadHotPixelList

This command is optional and depends on the hardware and firmware. Check the availability according to the camera descriptor (HOT_PIXEL_CORRECTION). To change the hotpixel list inside the camea, please first call this command. Then modify the list and write it back with PCO_WriteHotPixelList. We recommend doing a backup of the list after readout. An invalid list will break the hotpixel correction!

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_ReadHotPixelList(HANDLE ph, WORD wListNo, WORD wArraySize,
WORD* wNumValid, WORD* wNumMax, WORD* wHotPixX, WORD* wHotPixY)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wListNo: WORD to address a list (zero based index).
- WORD wArraySize: WORD to set the size of the prepared list to receive the hotpixels
- WORD* wNumValid: WORD* to get the number of valid hotpixels
- WORD* wNumMax: WORD* to get the maximum number of hotpixels
- WORD* wHotPixX: WORD array which gets the x coordinates of a hotpixel list
- WORD* wHotPixY: WORD array which gets the y coordinates of a hotpixel list

Set the following parameter:

- wListNo: Number of the list to modify (0 …)
- wArraySize: Number of words which are possible to be filled in the list

The input pointer will be filled with the following parameter:

- *wNumValid: Gets the number of valid hotpixels in the list
- *wNumMax: Gets the maximum possible number of hotpixels
- *wHotPixX(Y): Array of words. Each array member gets a x(y)-coordinate of a hotpixel.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.9 PCO_ClearHotPixelList

This command is optional and depends on the hardware and firmware. Check the availability according to the camera descriptor (HOT_PIXEL_CORRECTION). To change the hotpixel list inside the camea, please first call PCO_ReadHotPixelList. Then modify the list and write it back with PCO_WriteHotPixelList. We recommend doing a backup of the list after readout. An invalid list will break the hotpixel correction! This command clears the list addressed completely. Call with care!

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_ClearHotPixelList(HANDLE ph, WORD wListNo, DWORD dwMagic1, DWORD dwMagic2)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wListNo: WORD to address a list (zero based index).
- DWORD dwMagic1: First DWORD to unlock the access.
- DWORD dwMagic2: Second DWORD to unlock the access.

Set the following parameter:

- wListNo: Number of the list to modify (0 …).
- dwMagic1: Unlock code, set to 0x1000AFFE
- dwMagic2: Unlock code, set to 0x2000ABBA

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.10 PCO_GetHotPixelCorrectionMode

This command is optional and depends on the hardware and firmware. Check the availability according to the camera descriptor (HOT_PIXEL_CORRECTION). Get mode for hotpixel correction.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetHotPixelCorrectionMode (HANDLE ph, WORD*
wHotPixelCorrectionMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wHotPixelCorrectionMode: Address of a WORD to get the hotpixel correction mode.

The input pointer will be filled with the following parameter:

Return values:
- current mode:
  0x0000 = [off] or 0x0001 = [on]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.11 PCO_SetHotPixelCorrectionMode

This command is optional and depends on the hardware and firmware. Check the availability according to the camera descriptor (HOT_PIXEL_CORRECTION). Set mode for hotpixel correction.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetHotPixelCorrectionMode (HANDLE ph, WORD wHotPixelCorrectionMode)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wHotPixelCorrectionMode: WORD to set the hotpixel correction mode.

The input pointer has to be filled with the following parameter:

- current mode:
  0x0000 = [off] or 0x0001 = [on]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.12 PCO_GetInterfaceOutputFormat (dimax only)

Gets the actual interface output format. This is only valid with a dimax with a built in HD/SDI interface. This command can be used to determine the image streaming interface, which is active. If the addressed interface is set to [off], then the standard interface, e.g. GigE or USB, is used to stream the data. If the addressed interface is activated, the standard interface is only for camera control, thus streaming to this interface is disabled.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetInterfaceOutputFormat (HANDLE ph, WORD* wDestInterface, WORD* wFormat, WORD* wReserved1, WORD* wReserved2)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wDestInterface: Address of a WORD to set the interface to query.
- WORD* wFormat: Address of a WORD to get the interface format.
- WORD* wReservedx: Address of a WORD. Reserved; will be filled with zero.

The input pointer wFormat should be filled with the following parameter:

- wDestInterface:
  0x0001 = [HD-SDI]

The input pointer will be filled with the following parameter:

Return values:
- wFormat (HD-SDI, 1080p25):
  0x0000 = [off], 0x0001 = [RGB], 0x0002 = [arbitrary RAW, 2 raw images / frame]

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.13 PCO_SetInterfaceOutputFormat (dimax only)

Sets the actual interface output format. This is only valid with a dimax with a built in HD/SDI interface. This command can be used to set the image streaming interface, which is active. If the addressed interface is set to [off], then the standard interface, e.g. GigE or USB, is used to stream the data. If the addressed interface is activated, the standard interface is only for camera control, thus streaming to this interface is disabled.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetInterfaceOutputFormat (HANDLE ph, WORD wDestInterface, WORD wFormat, WORD wReserved1, WORD wReserved2)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wDestInterface: WORD to set the interface to change.
- WORD wFormat: WORD to set the interface format.
- WORD wReservedx: WORD. Reserved; set it to zero.

The input should be filled with the following parameter:

- wDestInterface:
  0x0001 = [HD-SDI]
- wFormat (HD-SDI, 1080p25):
  0x0000 = [off], 0x0001 = [RGB], 0x0002 = [arbitrary RAW, 2 raw images / frame]
- wReservedx: 0

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.14 PCO_PlayImagesFromSegmentHDSDI (dimax only)

Plays the images recorded to the camera RAM. The command is especially for HD-SDI interface (output only interfaces). This interface does not request images, but it has to be supplied with a continuous data stream.

Note: Command is only valid, if **storage mode** is set to [recorder] and recording to the camera RAM segment is stopped!

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_PlayImagesFromSegmentHDSDI (HANDLE ph, WORD wSegment, WORD wInterface, WORD wMode, WORD wSpeed, DWORD dwRangeLow, DWORD dwRangeHigh, DWORD dwStartPos)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- wSegment = number of segment of the RAM segment to read from
- wInterface = destination interface, RFU. Use 0x0000 for standard interface.
- wMode, where:
  - 0x0000 = Stop, switch datastream off
  - 0x0001 = Play (fast) forward
  - 0x0002 = Play (fast) backward (rewind)
  - 0x0003 = Play slow forward
  - 0x0004 = Play slow backward (rewind)
  - mode & 0x0100 = 0: At the end just repeat the last image (freeze image)
  - mode & 0x0100 = 1: At the end replay sequence from beginning
  - other values reserved for future modes
- wSpeed: Either stepping (fast play mode) or repeat count (slow play mode).
- Range Low = Lowest image number of range to be played
- Range High = Highest image number of range to be played
- Start No. = Start with this image number or leave unchanged (-1)

The play speed is defined by the Speed parameter together with the Mode parameter:

- Fast forward: The play position is **increased** by [Speed], i.e. [Speed – 1] images are leaped.
- Fast rewind: The play position is **decreased** by [Speed], i.e. [Speed – 1] images are leaped.
- Slow forward: The current image is sent [Speed] times before the position is increased
- Slow rewind: The current image is sent [Speed] times before the position is decreased

The play command can also be sent to change parameters (e.g. speed) while a play is active. The new parameters will be changed immediately. It is possible to change parameters like play speed or play direction without changing the current position by setting Start No. to -1 or 0xFFFFFFFFH in the DWORD format.

Some Examples:

Assuming that a record to a segment has been finished and there are ***N*** images in the segment, (use the "Get Number of Images in Segment" command to request the number).

Desired Function	**Range Low**	**Range High**	**Start No. 1)**	**Speed**	**Mode**
Play / Start complete sequence	1	N	1	1	0x0001
Fast Forward (speed x 10)	1	N	1	10	0x0001
Fast Rewind (speed x 10)	1	N	N	10	0x0002
Slow Forward (1/5th in speed)	1	N	N	5	0x0003
Slow Rewind (1/5th in speed)	1	N	N	5	0x0004
Cut out (starting with 1)	$j \geq 1$	$k \leq N$	1	1	0x0001
Cut out (starting with m)	$j \geq 1$	$k \leq N$	1...m...N	1	0x0001
Change Play Speed (to x 20)	1	N	-1	20	0x0001
Change Play Direction (to rewind)	1	N	-1	20	0x0003
Change current Play Position	1	N	$1 \leq p \leq N$	20	0x0001
Display image k as freezed image	1	N	k	0	0x0001
Switch HD/SDI off	0	0	0	0	0x0000

1) -1 means 0xFFFFFFFFH for the Start No. parameter

Notes:

When changing the range and the current image position or the Start No. parameter is out of range, the position will be set to the following positions:

Play forward: Range Low (with replay) or Range High (without replay)

Play reverse: Range High (with replay) or Range Low (without replay)

Effects related to record frame rate and play frame rate:

Please note, that the speed parameter does not depend on the recorded frame rate at all. Speed parameter 1 always means that the recorded images are sent one after another without leaps, as fast as possible for the selected interface and the selected format!

Thus if the record frame rate is 1000 frames/s and the output frame rate defined by the interface and the output format is 50 frames/s, it will result in a play speed which is 20 times slower than the record frame rate. Although the speed parameter is 1, it will appear as a slow motion when played. On the other hand you will have to set the speed parameter to 20, if you want to see the sequence as fast as it really happened.

You will have to care for yourself about these effects!

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.15 PCO_GetPlayPositionHDSDI (dimax only)

When the command “Play Images from Segment” was called, the sequence is started and the response message is sent immediately, whereas it may take seconds or up to minutes, until the sequence transmission is finished.

The “Get Play Position” command requests, at which position the play pointer of the currently started sequence is.

Note: Due to time necessary for communication and processing the command, the actual pointer may be 1 or 2 steps images ahead at the time, when the response is sent completely.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetPlayPositionHDSDI (HANDLE ph, WORD
*wStatus, DWORD  *dwPlayPosition)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wStatus: Address of a WORD to get the hotpixel correction mode.

The input pointer will be filled with the following parameter:

Return values:
- wStatus:
  0x0000: no play active or play has already stopped
  0x0001: play is active, see also play pointer
- dwPlayPosition: Number of the image currently sent to the interface. It is between Range Low and Range High, as set by “Play Images from Segment”. Only valid, when sequence play is still active.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.16 PCO_GetMetaDataMode (dimax only)

This command is optional and depends on the hardware and firmware. Check the availability according to the camera descriptor (METADATA). Gets the mode for meta data. See PCO_GetMetaData (dimax only) for more information.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetMetaDataMode (HANDLE ph, WORD* wMetaDataMode, WORD* wMetaDataSize, WORD* wMetaDataVersion)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD* wMetaDataMode: Address of a WORD to get the meta data mode.
- WORD* wMetaDataSize: Address of a WORD to get the size of the meta data block, which will be added to the image.
- WORD* wMetaDataVersion: Address of a WORD to get the version of the meta data mode.

The input pointer will be filled with the following parameter:

Return values:
- current mode: 0x0000 = [off] or 0x0001 = [on]
- wMetaDataSize: Size of meta data block in additional pixels
- wMetaDataVersion: Version information

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.6.17 PCO_SetMetaDataMode (dimax only)

This command is optional and depends on the hardware and firmware. Check the availability according to the camera descriptor (METADATA). Sets the mode for meta data. See PCO_GetMetaData (dimax only) for more information.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetMetaDataMode (HANDLE ph, WORD wMetaDataMode, WORD* wMetaDataSize, WORD* wMetaDataVersion)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wMetaDataMode: WORD to set the meta data mode.
- WORD* wMetaDataSize: Address of a WORD to get the size of the meta data block, which will be added to the image.
- WORD* wMetaDataVersion: Address of a WORD to get the version of the meta data mode.

The input should be filled with the following parameter:

- current mode:
  0x0000 = [off] or 0x0001 = [on]

The input pointer will be filled with the following parameter:

Return values:
- wMetaDataSize: Size of meta data block in additional pixels
- wMetaDataVersion: Version information

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7 API managament

This function group defines the API management functions, especially open and close of the camera, image buffer handling and image access.

Overview:

Command:
PCO_OpenCamera (* or use PCO_OpenCameraEx)
PCO_OpenCameraEx
PCO_CloseCamera (*)
PCO_AllocateBuffer (*)
PCO_GetBuffer
PCO_FreeBuffer (*)
PCO_AddBufferEx (*)
PCO_AddBuffer (obsolete, please use PCO_AddBufferEx)
PCO_GetBufferStatus
PCO_RemoveBuffer / PCO_CancelImages (*)
PCO_GetImageEx (*)
PCO_GetImage (obsolete, please use PCO_GetImageEx)
PCO_GetPendingBuffer (*)
PCO_CheckDeviceAvailability
PCO_SetTransferParameter
PCO_GetTransferParameter
PCO_CamLinkSetImageParameters (* CamLink and GigE)
PCO_GetInfoString
PCO_SetTimeouts
PCO_GetGigEIPAddress (for GigE only)
PCO_SetGigEIPAddress (for GigE only)
PCO_AddBufferExtern (Only for experienced users!)
PCO_GetMetaData (dimax only)

4.7.1 PCO_OpenCamera (* or use PCO_OpenCameraEx)

Open a camera device and attach it to a handle, which will be returned by the parameter ph. This function scans for the next available camera. If you want to access a distinct camera please use PCO_OpenCameraEx. Due to historical reasons the wCamNum parameter is a don't care.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_OpenCamera(HANDLE* ph, WORD wCamNum)
```

b.) Input parameter:

- HANDLE* ph: HANDLE pointer to receive the opened camera device. This parameter has to be stored for later use with all other function calls.
- WORD wCamNum: Don't care.

The input data should be filled with the following parameter:

- HANDLE* ph:
  - 0 = open new camera.
  - xyz = Handle to a previously opened camera.
- WORD wCamNum:
  - don't care, set to zero.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.2 PCO_OpenCameraEx

Open a camera device with given parameters and attach it to a handle, which will be returned by the parameter ph.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_OpenCameraEx(HANDLE *ph, PCO_OpenStruct* strOpenStruct)
```

b.) Input parameter:

- HANDLE* ph: HANDLE pointer to receive the opened camera device. This parameter has to be stored for later use with all other function calls.
- PCO_OpenStruct* strOpenStruct: Pointer to a previously filled 'open structure'.

Please fill in all wSize parameters, even in embedded structures. Additionally you have to set the strOpenStruct to zero completely, before setting up single parameters.

The input data should be filled with the following parameter:

- HANDLE* ph:
  - 0 = open new camera.
  - xyz = Handle to a previously opened camera.
- WORD wSize: Size of this struct.
- PCO_OpenStruct* strOpenStruct: Structure containing the following parameters:
  - WORD wInterfaceType: Interface number which should be scanned.
    - 1: Firewire,
    - 2: CamLink with Matrox,
    - 3: CamLink with Silicon SW
    - 0xFFFF: The SDK-Dll tries to find a camera at all known interfaces, starts with Firewire (1).
  - WORD wCameraNumber: Desired camera number at the desired interface, starts with 0.
  - WORD wCameraNumAtInterface: Resulting current number of camera at the interface. If in scanning mode, this number might be different to wCameraNumber.
  - WORD wOpenFlags[10]: Additional flags to control the interface. See the SDK-addendum (if available) for the selected interface.
    - [0]: moved on to cameralink interface
    - [1]: moved on to cameralink interface
    - [2]: Bit0: PCO_OPENFLAG_GENERIC_IS_CAMLINK - Set this bit in case of a generic Cameralink interface.
  - DWORD dwOpenFlags[5]: [0]-[4] moved on to cameralink interface.
  - void* wOpenPtr[6]: additional pointers
    - [0]: moved on to cameralink interface strCLOpen.clser_file_name
    - [1]: moved on to cameralink interface strCLOpen.config_file_name
    - [5]: used to open a generic interface dll – this is the name of the dll to open.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.3 PCO_CloseCamera (*)

Close a camera device.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_CloseCamera(HANDLE ph)
```

b.) Input parameter:

- HANDLE ph: HANDLE to close the previously opened camera device.

The input data should be filled with the following parameter:

- HANDLE ph:
  - xyz = Handle to a previously opened camera

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.4 PCO_AllocateBuffer (*)

Allocates a buffer to receive the transferred images. There is a maximum of 8 buffers. This function is needed to create, or to attach buffers for the image transfer. The buffers are attached to the previously opened camera. Using two buffers in an alternating manner is sufficient for most applications. If you use more than one camera, you will get the same buffer numbers 0 and 1 for each camera while allocating e.g. two buffers.

During recording the user can get images with the PCO_AddBuffer function. While waiting for an image the user can poll the buffer status with PCO_GetBufferStatus. Due to the fact that polling is time consuming, it is recommended that the event be used. This event enables the user to work with the WaitFor(Single/Multiple)Objects within a workerthread.

If *sBufNr is set to –1 and wBuf is zero, a new buffer will be created. The user can apply a self allocated buffer to this function call. If *wBuf is not zero and is a valid data block, the SDK-Dll will commit this externally allocated memory block and attaches it to a new buffer number.

Additionally it is possible to attach an event to the newly created buffer. The user can create this event within his own code and set it to hEvent. If *hEvent is zero an event will be created inside the SDK-Dll.

After changing the image size a reallocation should be done, with the valid buffers. In case of the buffer being allocated internally, a buffer with the new size will be set up. If the user has allocated the buffer in his own code, he must call this function to tell the SDK-Dll of the new size.

In case of using FireWire interface and if the user allocates external image buffer, the size of the buffer must to be a multiple of 4096 (0x1000) + 8192 bytes.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_AllocateBuffer(HANDLE ph, SHORT* sBufNr, DWORD dwSize,
WORD** wBuf, HANDLE* hEvent)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- SHORT* sBufNr: Address of a SHORT pointer to get the current number of the buffer.
- DWORD dwSize: DWORD to set the buffer size.
- WORD** wBuf: Address of a WORD* to get the buffer pointer.
- HANDLE* hEvent: Address of a HANDLE to get the event which will be fired in case of a previously arrived image.

The input data should be filled with the following parameters:

- *sBufNr = -1 to allocate a new buffer, 0 … 7, to change a previously allocated buffer.
- dwSize = size of the buffer in byte (normally: Xres * Yres * 2).
- **wBuf = must be the address of a WORD*.
- *hEvent = 0 to create a new event in the SDK-dll, or valid event created outside the SDK-dll.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.5 PCO_GetBuffer

Gets the data pointer and the event which are assigned to the buffer number passed in.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetBuffer(HANDLE ph, SHORT* sBufNr, WORD** wBuf, HANDLE*
hEvent)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- SHORT* sBufNr: Address of a SHORT pointer to get the current number of the buffer.
- WORD** wBuf: Address of a WORD* to get the buffer pointer.
- HANDLE* hEvent: Address of a HANDLE to get the event which will be fired in case of a previously arrived image.

The input data should be filled with the following parameters:

- *sBufNr = 0 … 7, to get data about an already allocated buffer
- **wBuf = must be the address of a WORD*.
- *hEvent = must be the address of a handle.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.6 PCO_FreeBuffer (*)

Frees a previously allocated buffer.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_FreeBuffer(HANDLE ph, SHORT sBufNr)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- SHORT sBufNr: SHORT to set the number of the buffer to be freed.

The input data should be filled with the following parameter:

- sBufNr = 0 … 7, to free a previously allocated buffer

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.7 PCO_AddBufferEx (*)

Adds a buffer to the driver queue. This function returns immediately. If the desired image is transferred to the buffer the buffer event will be fired. The user can start a thread, which can wait for the event of the buffer (WaitFor(Single/Multiple)Objects). This function can be used to view images while the recording is enabled (the user must set dw1stImage=dwLastImage=0).

To read out previously recorded images with recording disabled, the user can call PCO_GetImageEx. Nevertheless you can use this function to read out single images while the camera is not in recording state, by setting dw1stImage=dwLastImage=x, where x is a valid image number (1…max available).

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_AddBuffer(HANDLE ph, DWORD dw1stImage, DWORD dwLastImage,
SHORT sBufNr, WORD wXRes, WORD wYRes, WORD wBitPerPixel)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD dw1stImage: Set dw1stImage=dwLastImage=0 during record for actual image
- DWORD dwLastImage: Set dw1stImage=dwLastImage=x after record for desired image
- SHORT sBufNr: SHORT to set the buffer number to fill with.
- WORD wXRes: x-Resolution.
- WORD wYRes: y-Resolution.
- WORD wBitPerPixel: BitResolution of one Pixel (e.g. 14bit).

The input data should be filled with the following parameter:

- dw1stImage = set to 0 for live view mode("live view" transfers the most recent image to the PC for viewing / monitoring)
  - 0 = live view mode. x = set to the same value as dwLastImage. Has to be a valid image number (see PCO_GetNumberOfImagesInSegment, 1…max available).
- dwLastImage = set to 0 in preview mode.
  - 0 = live view mode. x = set to the same value as dw1stImage. Has to be a valid image number (see PCO_GetNumberOfImagesInSegment, 1…max available).
- sBufNr: 0 … 7:.number of desired buffer. A buffer can be reused after the event is fired.
- wXRes: Actual x-Resolution of the image which should be transferred.
- wYRes: Actual y-Resolution of the image which should be transferred.
- WBitPerPixel: BitResolution of the image which should be transferred.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.8 PCO_AddBuffer (obsolete, please use PCO_AddBufferEx)

See PCO_AddBufferEx (*).

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_AddBuffer(HANDLE ph, DWORD dw1stImage, DWORD dwLastImage,
SHORT sBufNr)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD dw1stImage: -
- DWORD dwLastImage: -.
- SHORT sBufNr: SHORT to set the buffer number to fill with.

The input data should be filled with the following parameter:

- dw1stImage = set to 0 for live view mode ("live view" transfers the most recent image to the PC for viewing / monitoring)
  - 0 = live view mode.
  - x = set to the same value as dwLastImage. Has to be a valid image number (see PCO_GetNumberOfImagesInSegment, 1…max available).
- dwLastImage = set to 0 in preview mode.
  - 0 = live view mode.
  - x = set to the same value as dw1stImage. Has to be a valid image number (see PCO_GetNumberOfImagesInSegment, 1…max available).
- sBufNr:
  - 0 … 7:.number of desired buffer.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.9 PCO_GetBufferStatus

Get the buffer status of a previously 'allocated' and 'added' buffer. This can be used to poll the status, while waiting for an image during recording. After an image has been transferred it is recommended to check the buffer status according to driver errors, which might occur during the image transfer.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetBufferStatus(HANDLE ph, SHORT sBufNr, DWORD* dwStatusDll,
DWORD* dwStatusDrv)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- SHORT sBufNr: SHORT to set the buffer number to fill with.
- DWORD* dwStatusDll: Address of a DWORD to get the status inside the SDK dll.
- DWORD* dwStatusDrv: Address of a DWORD to get the status inside the driver.

The input data will be filled with the following parameters:

- sBufNr: 0 … 7:.number of desired buffer.
- DWORD* dwStatusDll:
  - 0x80000000 = buffer is allocated.
  - 0x40000000 = buffer event created inside the SDK dll.
  - 0x20000000 = buffer is allocated externally.
  - 0x00008000 = buffer event is set.
- DWORD* dwStatusDrv:
  - PCO_NOERROR = no error occurred during transfer.
  - other than PCO_NOERROR = see Error codes.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.10 PCO_RemoveBuffer / PCO_CancelImages (*)

Removes any buffer from the driver queue. Stops pending buffers while the camera is recording and you want to stop recording and remove the buffers. It is mandatory to call PCO_CancelImages. Usually it should be called after setting recording state to zero. PCO_RemoveBuffer and PCO_CancelImages have exact the same functionality.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_RemoveBuffer(HANDLE ph)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.11 PCO_GetImageEx (*)

Gets previously recorded images from the camera. This function does not return until the desired number of image(s) are transferred to the buffer. You can get more than one image from the camera with this function call, but you have to take care about the size of the receiving buffer. If you like to view images while the recording is enabled, you should use PCO_AddBufferEx. Nevertheless you can use this function to get images while recording. In this case set 1stImage and LastImage to zero. The function will return if an image is completely transferred to the memory.

If the user is going to call this function during record he must be aware about the fact that there is a fixed timeout for this function. If the camera does not send an image after 5 seconds while getting one image this function will return with an error.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetImage(HANDLE ph, WORD wSegment, DWORD dw1stImage,
DWORD dwLastImage, SHORT sBufNr, WORD wXRes, WORD wYRes, WORD wBitPerPixel)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wSegment: WORD to set the desired segment.
- DWORD dw1stImage: DWORD to set the first image to be retrieved.
- DWORD dwLastImage: DWORD to set the last image to be retrieved.
- SHORT sBufNr: SHORT to set the buffer number to fill with.
- WORD wXRes: x-Resolution.
- WORD wYRes: y-Resolution.
- WORD wBitPerPixel: BitResolution of one Pixel (e.g. 14bit).

The input data should be filled with the following parameter:

- wSegment: set to a segment number 1,2,3 or 4
- dw1stImage (1 based index: 1…max available):
- x = has to be a valid image number. If you try to access image e.g. 1 set this value to 1. (see PCO_GetNumberOfImagesInSegment).
- dwLastImage (1 based index: dw1stImage…max available):
- x = has to be a valid image number. If you try to access image e.g. 1 set this value to 1. (see PCO_GetNumberOfImagesInSegment).
- sBufNr:
- 0 … 7:.number of desired buffer.
- wXRes: Actual x-Resolution of the image which should be transferred.
- wYRes: Actual y-Resolution of the image which should be transferred.
- WBitPerPixel: BitResolution of the image which should be transferred

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.12 PCO_GetImage (obsolete, please use PCO_GetImageEx)

See PCO_GetImageEx (*)

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetImage(HANDLE ph, WORD wSegment, DWORD dw1stImage,
DWORD dwLastImage, SHORT sBufNr)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wSegment: WORD to set the desired segment.
- DWORD dw1stImage: DWORD to set the first image to be retrieved.
- DWORD dwLastImage: DWORD to set the last image to be retrieved.
- SHORT sBufNr: SHORT to set the buffer number to fill with.

The input data should be filled with the following parameter:

- wSegment: set to a segment number 1,2,3 or 4
- dw1stImage (1 based index: 1…max available):
  - x = has to be a valid image number. If you try to access image e.g. 1 set this value to 1. (see PCO_GetNumberOfImagesInSegment).
- dwLastImage (1 based index: dw1stImage…max available):
  - x = has to be a valid image number. If you try to access image e.g. 1 set this value to 1. (see PCO_GetNumberOfImagesInSegment).
- sBufNr:
  - 0 … 7:.number of desired buffer.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.13 PCO_GetPendingBuffer (*)

Get the number of pending buffers while the camera is recording and you want to stop recording and to remove the buffers. If there are pending buffers you should call PCO_CancelImages. We recommed this before you stop recording with PCO_SetRecordingState 0.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetPendingBuffer(HANDLE ph, int* icount)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- int* icount: Address of an int to get the number of pending buffers.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.14 PCO_CheckDeviceAvailability

In case of a busreset you can check whether the device is still valid.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_CheckDeviceAvailability(HANDLE ph, WORD wNum)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wNum: WORD to set the camera to check. Same number is with PCO_OpenCamera.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.15 PCO_SetTransferParameter

Change the transfer parameters of the transfer media. E.g. in case of FireWire this can be used to deal with more than one camera while the user wants to transfer data from all connected cameras at the same time. If the users transfers only from one camera at a time this function does not have to be called.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetTransferParameter(HANDLE ph, void* buffer, int ilen)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- void* buffer: Address of a buffer to set to the transfer parameter settings.
- int ilen: Length of the buffer in bytes.

The input data should be filled with the following parameters:

FireWire Interface:

The user can instantiate a structure _PCO1394_ISO_PARAMS, which is defined in SC2_SDKAddendum.h.

- bandwidth_bytes: set to a bandwidth size which is a fraction of 4096 / (num of cameras). e.g. 2 cameras connected: bandwidth_bytes = 2048.
- speed_of_isotransfer: 1,2,4, whereas 1 is 100MBit/s, 2=200 and 4=400; default is 4.
- number_of_isochannel: manual 0…7, must be ored with 0x100; auto -1
- number_of_isobuffers: 16…256; default is 128
- byte_per_isoframe: set to the same value as bandwidth_bytes.

Remarks for number_of_isochannel: Usually it is not necessary to change this parameter, but in case the user wants to transfer images from more than one camera, the iso channel must be unique for each camera. To control the channel number set this to 0x100 for the first camera, 0x101 for the second and so on. 0x100 sets the driver to keep up with the number supplied.

Cameralink Interface:

The user can instantiate a structure _PCO_SC2_CL_TRANSFER_PARAMS, which is defined in SC2_SDKAddendum.h.

- baudrate: sets the baudrate of the cameralink serial port interface. Valid values are: 9600, 19200, 38400, 57600 and 115200.
- ClockFrequency: sets the clockrate of the cameralink interface. See SC2_SDKAddendum.h for valid parameters.
- CCline: sets the usage of CC1-CC4 lines, use parameter returned by the GetTransferParameter command.
- DataFormat: sets the Testpattern and Datasplitter switch, use parameter returned by the GetTransferParameter command.
- Transmit: sets the transmitting format, 0-single images, 1-continous image transfer.

In case an edge is controlled, the user has to call this function with appropriate parameters. With pixelclock = 286MHz and an x-resolution >= 1920 the user must switch to dataformat PCO_CL_DATAFORMAT_5x12 (switches to 12bit, whereas upper bits are moved to LSB) or PCO_CL_DATAFORMAT_5x12L (switches to 12bit, whereas all data above 255 is calculated by a square-root LUT. The driver tries to recalculate to 16bit values.).
For all other cases (pixelclock = 95.3MHz or x-resolution < 1920) the user must switch to dataformat PCO_CL_DATAFORMAT_5x16.

USB Interface:

The user can instantiate a structure _PCO_USB_TRANSFER_PARAMS, which is defined in SC2_SDKAddendum.h.

- MaxNumUsb: sets the packet size.
- ClockFrequency: sets the clockrate of the cameralink interface, use parameter returned by the GetTransferParameter command.
- Transmit: sets the transmitting format, 0-single images, 1-continous image transfer, use parameter returned by the GetTransferParameter command.
- UsbConfig: sets the transfer mode of the USB interface, 0-bulk image, 1-iso_image
- Img12Bit: sets the transferred pixel size. This parameter can be set to 12 in order to improve transfer rate, but with a loss of bit resolution. 0-14bit, 1-12bit.

GigE Interface:

The user can instantiate a structure _PCO_GIGE_TRANSFER_PARAMS, which is defined in SC2_SDKAddendum.h.

- dwPacketDelay: sets the delay between two stream packets in us, default 2000, range 0-8000.
- dwResendPercent: sets the procentual part of lost packages per image, which will be retransferred, default 30. In case more packages got lost, the complete image will be retransferred till it times out and produces an error.
- dwFlags: sets single flags, Bit0: enable packet resend, Bit1: Enable burst mode, Bit2: Enable max speed mode, Bit3: Enable Camera debug mode. Bit4: Transfer bandwidth distribution: 0-same bandwidth for all cameras, 1-active camera gets whole bandwidth.. Bit5-7: reserved, set to zero.
- dwDataFormat: sets the data format of the transferred data. See SC2_SDKAddendum.h for valid parameters.
- dwCameraIPAddress: current IP Address of the camera, use parameter returned by the GetTransferParameter command.
- DwUDPImgPcktSize: size of an UDP image packet, use parameter returned by the GetTransferParameter command.
- Ui64MACAddress: MAC address of camera, use parameter returned by the GetTransferParameter command.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.16 PCO_GetTransferParameter

Get the transfer parameters of the transfer media. See PCO_SetTransferParameter for more information.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetTransferParameter(HANDLE ph, void* buffer, int ilen)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- void* buffer: Address of a buffer to set to the transfer parameter settings.
- int ilen: Length of the buffer in bytes.

The input data will be filled with interface dependent parameters:

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.17 PCO_CamLinkSetImageParameters (* CamLink and GigE)

Set the image parameters for the image buffer transfer inside the CamLink and GigE interface. While using CamLink or GigE this function must be called, before the user tries to get images from the camera and the sizes have changed. With all other interfaces this is a dummy call.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_CamLinkSetImageParameters(HANDLE ph, WORD wxres, WORD wyres)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- WORD wxres: Actual x resolution of the image to be transferred.
- WORD wyres: Actual y resolution of the image to be transferred.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.18 PCO_GetInfoString

This call can be used to read some information about the camera, e.g. firmware versions.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetInfoString(HANDLE ph, DWORD dwinfoflags, char *buf_in, WORD
size_in)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- DWORD dwinfoflags: Not used, now. Set to ZERO!
- char* buf: String pointer to receive the info string.
- WORD wsize_in: WORD variable which holds the maximum length of the string.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.19 PCO_SetTimeouts

This call sets the driver timeouts to new values. Usually there is no need to change these values. In case there are timeout errors while calling e.g. PCO_ArmCamera, the user might increase the command timeout.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_SetTimeouts(HANDLE ph, void *buf_in, unsigned int size_in)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- void* buf: Pointer to hold the address of the first element of a unsigned int array.
- unsigned int size_in: Variable which sets the number of valid values accessable by the pointer.

The input data should be filled with the following parameter:

- buf_in: Array of unsigned int values, whereas:
  - buf_in[0]: Command timeout – A command will be aborted after x ms if there is no response from the camera.
  - buf_in[1]: Image timeout – An image request will be aborted after x ms if there is no response from the camera. This is valid for the PCO_GetImage command.
  - buf_in[2]: Transfer timeout – The 1394 driver will close the allocated isochronous channel after x ms if there is no image transfer from the camera. The camera link interface will remove all occupied resources after x ms.
- size_in: Number of valid values, usually three. Set this to two for the camera link interfaces.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.20 PCO_GetGigEIPAddress (for GigE only)

Gets the fixed IP address of the camera, which is stored inside the camera internal EEProm. To get the actual IP address (in case a DHCP server is used), please use PCO_GetTransferParameter.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetGigEIPAddress(HANDLE ph, BYTE *BField0, BYTE *BField1, BYTE
*BField2, BYTE *BField3)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- BYTE* BField0: Address of a BYTE to get the upper part of the IP address.
- BYTE* BField1: Address of a BYTE to get the upper mid part of the IP address.
- BYTE* BField2: Address of a BYTE to get the lower mid part of the IP address.
- BYTE* BField3: Address of a BYTE to get the lower part of the IP address.
- int ilen: Length of the buffer in bytes.

The input data will be filled with the following parameters (e.g. 192.168.144.20):

- BField0: 10...254, e.g. 192
- BField1: 1...254, e.g. 168
- BField2: 1...254, e.g. 144
- BField3: 1...254, e.g. 20

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.21 PCO_SetGigEIPAddress (for GigE only)

Sets the fixed IP address of the camera, which is stored inside the camera internal EEProm. Setting the actual IP address (in case a DHCP server is used) is not possible. See PCO_GetTransferParameter.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetGigEIPAddress(HANDLE ph, BYTE *BField0, BYTE *BField1, BYTE
*BField2, BYTE *BField3)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- BYTE BField0: BYTE to set the upper part of the IP address.
- BYTE BField1: BYTE to set the upper mid part of the IP address.
- BYTE BField2: BYTE to set the lower mid part of the IP address.
- BYTE BField3: BYTE to set the lower part of the IP address.

The input data should be filled with the following parameters (e.g. 192.168.144.20):

- BField0: 10...254, e.g. 192
- BField1: 1...254, e.g. 168
- BField2: 1...254, e.g. 144
- BField3: 1...254, e.g. 20

Remarks: After issuing this command the camera must be restarted in order to work with the new IP address.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.22 PCO_AddBufferExtern (Only for experienced users!)

WRONG USE OF THIS FUNCTION MAY CRASH YOUR SYSTEM. PCO IS NOT RESPONSIBLE FOR DAMAGES DUE TO IMPROPER USE OF THIS FUNCTION.

Adds a buffer to the driver queue. This function returns immediately. If the desired image is transferred to the buffer the buffer event will be fired. The user can start a thread, which can wait for the event of the buffer (WaitFor(Single/Multiple)Objects). This function can be used to view images while the recording is enabled (the user must set dw1stImage=dwLastImage=0).

To read out previously recorded images with recording disabled, the user can call PCO_GetImageEx. Nevertheless you can use this function to read out single images while the camera is not in recording state, by setting dw1stImage=dwLastImage=x, where x is a valid image number (1…max available).

ATTENTION: Use this function only in case you're absolutely sure what you do! The SDK dll does not check the parameters for validity. Improper use might harm your system!

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_AddBufferExtern(HANDLE ph, HANDLE hEvent, DWORD dw1stImage,
DWORD dwLastImage, DWORD dwSynch, void* pBuf, DWORD dwLen, DWORD* dwStatus)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- HANDLE hEvent: Handle to an externally allocated event.
- DWORD dw1stImage: Set dw1stImage=dwLastImage=0 during record for actual image
- DWORD dwLastImage: Set dw1stImage=dwLastImage=x after record for desired image
- DWORD dwSynch: Synchronization paremeter, usually 0.
- void *pBuf: Pointer to the buffer to receive the transferred image.
- DWORD dwLen: Length of the buffer.
- DWORD *dwStatus: Driver status.

The input data should be filled with the following parameter:

- hEvent = externally created event used to signal an occurred transfer.
- dw1stImage = set to 0 for live view mode("live view" transfers the most recent image to the PC for viewing / monitoring)
  - 0 = live view mode. x = set to the same value as dwLastImage. Has to be a valid image number (see PCO_GetNumberOfImagesInSegment, 1…max available).
- dwLastImage = set to 0 in preview mode.
  - 0 = live view mode. x = set to the same value as dw1stImage. Has to be a valid image number (see PCO_GetNumberOfImagesInSegment, 1…max available).
- dwSynch: set to 0.
- pBuf: Address of the first buffer element to which the image should be transferred.
- dwLen: Length of the buffer in bytes.
- dwStatus: Address of a DWORD to receive the buffer status.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

4.7.23 PCO_GetMetaData (dimax only)

This command is optional and depends on the hardware and firmware. Check the availability according to the camera descriptor (METADATA). Gets the image attached meta data. Additional image information will be added in case the meta data mode is enabled. The user has to take care for sufficient buffer size in case buffers are allocated externally. Use PCO_GetMetaDataMode (dimax only) to get the actual size in pixels, which will be valid in the added lines. E.g. if the size is 140 pixels and the x-resolution is 1200, one line will be added to the image. In case the size is 140 and the x-resolution is 80, two lines will be added to the image. The function will only work with buffers allocated with PCO_AllocateBuffer.

The data will be transferred as one byte per pixel. So the bytes have to be unpacked from 16bit format to 8bit format. This function does this automatically.

To use this function, please add sc2_common.h include file to your project. sc2_common.h has to be included before sc2_camexport.h. See sc2_common.h for more information about the structure members.

a.) Prototype:

```
SC2_SDK_FUNC int WINAPI PCO_GetMetaData (HANDLE ph, SHORT sBufNr, PCO_METADATA_STRUCT *pMetaData, DWORD dwReserved1, DWORD dwReserved2)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously opened camera device.
- SHORT sBufNr: Number of an allocated buffer.
- PCO_METADATA_STRUCT* wMetaData: Address of a PCO_METADATA_STRUCT to get the meta data.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see Error / Warning Codes

5 Typical Implementation

This typical step by step implementation shows the basic handling:

1. Declarations:

PCO_General	strGeneral;
PCO_CameraType	strCamType;
PCO_Sensor	strSensor;
PCO_Description	strDescription;
PCO_Timing	strTiming;
PCO_Storage	strStorage;
PCO_Recording	strRecording;

2. Set all buffer 'size' parameters to the expected values:
```
strGeneral.wSize = sizeof(strGeneral);
strGeneral.strCamType.wSize = sizeof(strGeneral.strCamType);
strCamType.wSize = sizeof(strCamType);
strSensor.wSize = sizeof(strSensor);
strSensor.strDescription.wSize = sizeof(strSensor.strDescription);
strSensor.strDescription2.wSize = sizeof(strSensor.strDescription2);
strDescription.wSize = sizeof(strDescription);
strTiming.wSize = sizeof(strTiming);
strStorage.wSize = sizeof(strStorage);
strRecording.wSize = sizeof(strRecording);
```
3. Open the camera and fill the structures:
```
PCO_OPENCAMERA(&hCam, iBoardNumber)
PCO_GETGENERAL(hCam, &strGeneral)
PCO_GETCAMERATYPE(hCam, &strCamType)
PCO_GETSENSORSTRUCT(hCam, &strSensor)
PCO_GETCAMERADESCRIPTION(hCam, &strDescription)
PCO_GETTIMINGSTRUCT(hCam, &strTiming)
PCO_GETRECORDINGSTRUCT(hCam, &strRecording)
```
4. Set camera settings (exposure, modes, etc.) and sizes (binning, ROI, etc.).
5. Arm the camera.
6. Get the sizes and allocate a buffer:
   PCO_GETSIZES(hCam, &actualsizex, &actualsizey, &ccdsizex, &ccdsizey)
   PCO_ALLOCATEBUFFER(hCam, &bufferNr, actualsizex * actualsizey * sizeof(WORD), &data, &hEvent)
   In case of CamLink and GigE interface: PCO_CamLinkSetImageParameters(actualsizex, actualsizey)
   PCO_ArmCamera(hCam)
7. Set the recording state to 'Recording' and add your buffer(s).
```
PCO_SetRecordingState(hCam, 0x0001);
PCO_AddBufferEx(hCam, 0, 0, bufferNr, actualsizex, actualsizey, bitres);
```
8. Access your images with the pointer got by AllocateBuffer.
   Do a convert and show the image.
9. Stop the camera.
```
PCO_SetRecordingState(hCam, 0x0000);
PCO_CancelImages(hCam);
```
10. After recording, if you like to read out some images from the CamRAM you can use:
    PCO_GetNumberOfImagesInSegment(hCam, wActSeg, &dwValidImageCnt, &dwMaxImageCnt);
    PCO_GetImageEx(hCam, wActSeg, dw1stImage, dwLastImage, bufferNr, actualsizex, actualsizey, bitres)
11. Free all buffers and close the camera.
```
PCO_FreeBuffer(hCamera, sBufNr);
PCO_CloseCamera(hCamera);
```

Here is a short code listing (symbolical calls), which shows a typical image acquire loop:

Do all necessary settings, including an ARM, before running the loop:

```
 OpenCamera()
 Change settings according to your needs.
 ArmCamera()
 GetSizes(...)
 AllocateBuffer(...)
```

Start recording and run the loop:

```
 SetRecordingstate(1)  // activates the image recording and transfers images to camram
 Addbuffer(0)   // adds a buffer to the driver queue for image transfer to the pc
 Addbuffer(1)
 do
 {
   int err = WaitForMultipleObjects(..., 500);  // wait for a buffer to get ready
   if(err == TIMEOUT)
   {
     // Error handling, e.g.: count errors and break in case errcount > 5
   }
   if(err == WAIT_OBJECT_0)
   {
     GetBufferStatus(0)
     // handle buffer 0
     Addbuffer(0)
   }
   if(err == WAIT_OBJECT_1)
   {
     GetBufferStatus(1)
     // handle buffer 1
     Addbuffer(1)
   }
 }while(some condition);
 SetRecordingstate(0)  // stops recording images
 CancelImages()  // removes pending buffers from the drivers queue
```

Short code discussion:

SetRecordingstate enables recording of images, depending on the trigger mode. If trigger mode is 0 (auto) and aquire mode is 0 (auto) images are transferred automatically to the camram.

Addbuffer moves a buffer to the driver (set 1stimage=lastimage=0 during record is on), in order to transfer the most recent recorded image to the pc. Two buffers are enough to transfer images with maximum possible performance (depending on the interface).

GetBufferStatus gives further information about error states.

WaitForMultipleObjects waits for an event to fire. If an event fires the return value shows, which buffer has got ready. Now you can process the data. Add the buffer to the driver, after you've finished processing.

SetRecordingstate to zero stops recording. The image sensor inside the camera is read out completely and set to idle.

CancelImages must be called to remove all pending buffers from the driver queue. This clears the driver and prepares for further image transfers.

To see those calls 'in action' please take a look at the console sample, which is part of the SDK-installation.

Some pitfalls:

- wSize is not set. DO NOT FORGET TO SET ALL wSize PARAMETERS.
- Segment index is zero: The segment parameter is 1 based, whereas all structure reflections are zero based, e.g. dwRamSegSize[0] is the size of segment 1.
- The user calls PCO_GetImageEx with dw1stImage number 0. If the user wants to access the first image inside the camera, set the image parameter to 1. Access to the camera is 1 based!
- The minimum segment size has to be at least two images.

Some thoughts about fps:
Q: I need 100 images per second with an exposure time of 100ms.
A: Sorry, but this is not possible. An exposure time of 100ms results in a maximum of 10 fps. Usually those 10 fps are still not possible, since the image has to be read out of the image sensor. There are two different modes while reading out images from an image sensor: Either video mode, where an exposure occurs during the readout of the last exposed image, or asynchronous mode, where an exposure is followed by the readout. With video mode the fps is based on exposure time or readout, depending on what is longer. In asynch mode fps is based on the sum of exposure time and reaout. The camera decides what mode gives best fps in auto mode. In triggered mode (either ext. or software) it runs in asynch mode.

6 Error / Warning Codes

The error codes are standardized as far as possible. The error codes contain the information of the error layer, the source (micocontrollers, CPLDs, FPGAs) and an error code (error cause). All values are combined by a logical OR operation. Error codes and warnings are always negative values, if read as signed integers, or if read as unsigned word, the MSB is set. Errors have the general format 0x80######, warnings have the format 0xC0######.
The error numbers are not unique. Each layer and the common errors have its own error codes. You have to analyze the error in order to get error source. This can easily be done with a call to PCO_GetErrorText.

```
// e.g.: 0xC0000080 indicates a warning,
//       0x800A3001 is an error inside the SC2-SDK-dll.
// MSB                                  LSB
// XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
// |||| |||| |||| |||| |||| |||| |||| ||||
// |||| |||| |||| |||| |||| --------------- Error or warning code
// |||| |||| |||| |||| ||||
// |||| |||| |||| |||| -------------------- Layer code
// |||| |||| |||| ||||
// |||| |||| ------------------------------ Device code
// |||| ||||
// |||------------------------------------- reserved for future use
// |||
// ||-------------------------------------- Common error code flag
// ||
// |--------------------------------------- Warning indication bit
// |
// ---------------------------------------- Error indication bit
```

Error layer:

```
0x00001000 PCO_ERROR_FIRMWARE          // error inside the firmware
0x00002000 PCO_ERROR_DRIVER            // error inside the driver
0x00003000 PCO_ERROR_SDKDLL            // error inside the SDK-dll
0x00004000 PCO_ERROR_APPLICATION       // error inside the application
```

Error / Warning source:

```
0x00010000 SC2_ERROR_PCOCAM_POWER_CPLD // error at CPLD in pco.power unit
0x00020000 SC2_ERROR_PCOCAM_HEAD_UP    // error at uP of head board in pco.camera
0x00030000 SC2_ERROR_PCOCAM_MAIN_UP    // error at uP of main board in pco.camera
0x00040000 SC2_ERROR_PCOCAM_FWIRE_UP   // error at uP of firewire board in pco.camera
0x00050000 SC2_ERROR_PCOCAM_MAIN_FPGA  // error at FPGA of main board in pco.camera
0x00060000 SC2_ERROR_PCOCAM_HEAD_FPGA  // error at FGPA of head board in pco.camera
0x00070000 SC2_ERROR_PCOCAM_MAIN_BOARD // error at main board in pco.camera
0x00080000 SC2_ERROR_PCOCAM_HEAD_CPLD  // error at CPLD of head board in pco.camera
0x00090000 SC2_ERROR_SENSOR            // error at image sensor (CCD or CMOS)
0x000A0000 SC2_ERROR_SDKDLL            // error inside the SDKDLL
0x000B0000 SC2_ERROR_DRIVER            // error inside the driver
0x000D0000 SC2_ERROR_POWER             // error within power unit

0x00100000 PCO_ERROR_CAMWARE           // error in CamWare (also some kind of "device")
0x00110000 PCO_ERROR_CONVERTDLL        // error inside the convert dll
```

Error codes:

Please take a look at the file pco_err.h.

Warnings:

Please take a look at the file pco_err.h.

In case of successful operation the standard Response Message is returned.
To get detailed error information you can call the function PCO_GetErrorText, which is defined inside the PCO_errt.h header file.

6.1.1 PCO_GetErrorText

Gets a detailed description for an error.

This function is part of the header file pco_errt.h. If you want to use this function include the pco_errt.h header file and define PCO_ERRT_H_CREATE_OBJECT in **one** of your modules.

a.) Prototype:

```
void PCO_GetErrorText(DWORD dwerr, char* pbuf, DWORD dwlen)
```

b.) Input parameter:

- DWORD dwerr: DWORD which holds the error number.
- char* pbuf: Address of the first char of an char array.
- DWORD dwlen: DWORD which holds the length of the char array in byte.

7 SDK - Function Matrix

		pco.1200	pco.dimax pco.dimax.HD	pco.edge	pco.1400 pco.pixelfly.usb	pco.1300	pco.1600 pco.2000 pco.4000
4.1	General Control/Status						
4.1.1	PCO_GetGeneral	x	x	x	x	x	x
4.1.2	PCO_GetCameraType	x	x	x	x	x	x
4.1.3	PCO_GetCameraHealth-Status	x	x	x	x	x	x
4.1.4	PCO_ResetSettings-ToDefault	x			x	x	
4.1.5	PCO_InitiateSelftest-Procedure						
4.1.6	PCO_GetTemperature	x	x	x	x	x	x
4.1.7	PCO_GetCameraName	x	x	x	x	x	x
4.1.8	PCO_GetFirmwareInfo	x	x				
4.2	Image Sensor						
4.2.3	PCO_GetCameraDescription	x	x	x	x	x	x
4.2.4	PCO_GetCamera-DescriptionEx						desc.
4.2.5	PCO_GetSensorFormat	x	x	x	x	x	x
4.2.6	PCO_SetSensorFormat	x	x	x	x	x	x
4.2.7	PCO_GetSizes						
4.2.8	PCO_GetROI	x	x	x	x	x	x
4.2.9	PCO_SetROI	x	x	x	x	x	x
4.2.10	PCO_GetBinning	x	desc		x	x	x
4.2.11	PCO_SetBinning	x	desc		x	x	x
4.2.12	PCO_GetPixelRate	x	desc	desc	x	x	x
4.2.13	PCO_SetPixelRate	x	desc	desc	x	x	x
4.2.14	PCO_GetConversionFactor	x	desc		x	x	x
4.2.15	PCO_SetConversionFactor	x	desc		x	x	x
4.2.16	PCO_GetDoubleImageMode	x	desc		x	x	x
4.2.17	PCO_SetDoubleImageMode	x	desc		x	x	x
4.2.18	PCO_GetADCOperation						x
4.2.19	PCO_SetADCOperation				x	x	x
4.2.20	PCO_GetIRSensitivity				x	x	desc.
4.2.21	PCO_SetIRSensitivity				x	x	desc.
4.2.22	PCO_GetCoolingSetpoint-Temperature					x	x
4.2.23	PCO_SetCoolingSetpoint-Temperature					x	x
4.2.24	PCO_GetOffsetMode						
4.2.25	PCO_SetOffsetMode				x	x	
4.2.26	PCO_GetNoiseFilterMode	x	x				
4.2.27	PCO_SetNoiseFilterMode	x	x				
4.2.28	PCO_GetHWIOSignal-Count		x	x			
4.2.29	PCO_GetHWIOSignal-Descriptor		x	x			
4.2.30	PCO_GetLookupTableInfo			x			
4.2.31	PCO_GetActiveLookupTable			x			
4.2.32	PCO_SetActiveLookupTable			x			

		pco.1200	pco.dimax pco.dimax.HD	pco.edge	pco.1400 pco.pixelfly usb	pco.1300	pco.1600 pco.2000 pco.4000
4.3	Timing Control						
4.3.3	PCO_GetDelayExposure-Time	x	x	x			x
4.3.4	PCO_SetDelayExposure-Time	x	x	x			x
4.3.5	PCO_GetDelayExposure-TimeTable						
4.3.6	PCO_SetDelayExposure-TimeTable						
4.3.7	PCO_GetTriggerMode	x	x	x	x	x	x
4.3.8	PCO_SetTriggerMode	x	x	x	x	x	x
4.3.9	PCO_ForceTrigger	x	x	x	x	x	x
4.3.10	PCO_GetCameraBusyStatus			x			x
4.3.11	PCO_GetPowerDownMode						
4.3.12	PCO_SetPowerDownMode						
4.3.13	PCO_GetUserPower-DownTime						
4.3.14	PCO_SetUserPower-DownTime						
4.3.15	PCO_GetExpTrigSignal-Status				x	x	
4.3.16	PCO_GetCOCRuntime	x	x	x			x
4.3.17	PCO_GetFPSExposureMode	x					
4.3.18	PCO_SetFPSExposureMode	x					
4.3.19	PCO_GetModulationMode						desc.
4.3.20	PCO_SetModulationMode						desc.
4.3.21	PCO_GetFrameRate		x	x			
4.3.22	PCO_SetFrameRate		x	x			
4.3.23	PCO_GetHWIOSignal		x	x			
4.3.24	PCO_SetHWIOSignal		x	x			
4.3.25	PCO_GetImageTiming						
4.3.26	PCO_GetCameraSynchMode		x				
4.3.27	PCO_SetCameraSynchMode		x				
4.3.28	PCO_GetFastTimingMode		x (FW dep)				
4.3.29	PCO_SetFastTimingMode		x (FW dep)				
4.3.30	PCO_GetSensorSignalStatus						
4.4	Storage Control						
4.4.3	PCO_GetCameraRamSize	x	x		x	x	x
4.4.4	PCO_GetCameraRam-SegmentSize	x	x		x	x	x
4.4.5	PCO_SetCameraRam-SegmentSize	x	x		x	x	x
4.4.6	PCO_ClearRamSegment	x	x		x	x	x
4.4.7	PCO_GetActiveRamSegment	x	x				x
4.4.8	PCO_SetActiveRamSegment	x	x				x

		pco.1200	pco.dimax pco.dimax.HD	pco.edge	pco.1400 pco.pixelfly usb	pco.1300	pco.1600 pco.2000 pco.4000
4.5	Recording Control						
4.5.3	PCO_GetStorageMode	x	x		x	x	x
4.5.4	PCO_SetStorageMode	x	x		x	x	x
4.5.5	PCO_GetRecorderSubmode	x	x		x	x	x
4.5.6	PCO_SetRecorderSubmode	x	x		x	x	x
4.5.7	PCO_GetRecordingState	x	x	x	x	x	x
4.5.8	PCO_SetRecordingState	x	x	x	x	x	x
4.5.9	PCO_ArmCamera	x	x	x	x	x	x
4.5.10	PCO_GetAcquireMode	x	x	tbd	x	x	x
4.5.11	PCO_SetAcquireMode	x	x	tbd	x	x	x
4.5.12	PCO_GetAcqEnbl-SignalStatus		tbd	tbd			x
4.5.13	PCO_SetDateTime	x	x	x	x	x	x
4.5.14	PCO_GetTimestampMode	x	x	x	x	x	x
4.5.15	PCO_SetTimestampMode	x	x	x	x	x	x
4.5.16	PCO_GetRecordStopEvent	x	x				
4.5.17	PCO_SetRecordStopEvent	x	x				
4.5.18	PCO_StopRecord	x	x				
4.6	Image Buffer Data						
4.6.3	PCO_GetSegmentImage-Settings	x	x		x	x	x
4.6.4	PCO_GetNumberOf-ImagesInSegment	x	x		x	x	x
4.6.5	PCO_GetBitAlignment	x	x		x	x	x
4.6.6	PCO_SetBitAlignment	x	x		x	x	x
4.6.7	PCO_WriteHotPixelList	x	x	x	x	x	
4.6.8	PCO_ReadHotPixelList	x	x	x	x	x	
4.6.9	PCO_ClearHotPixelList	x	x	x	x	x	
4.6.10	PCO_GetHotPixel-CorrectionMode	x	x	x			
4.6.11	PCO_SetHotPixel-CorrectionMode	x	x	x			
4.6.12	PCO_GetInterfaceOutput-Format		x	x			
4.6.13	PCO_SetInterfaceOutput-Format		x	x			
4.6.14	PCO_PlayImagesFromSeg-mentHDSDI						
4.6.15	PCO_GetPlayPosition-HDSDI						
4.6.16	PCO_GetMetaDataMode		x (FW dep)				
4.6.17	PCO_SetMetaDataMode		x (FW dep)				

		pco.1200	pco.dimax pco.dimax.HD	pco.edge	pco.1400 pco.pixelfly usb	pco.1300	pco.1600 pco.2000 pco.4000
4.7	API Management						
4.7.1	PCO_OpenCamera	x	x	x	x	x	x
4.7.2	PCO_OpenCameraEx	x	x	x	x	x	x
4.7.3	PCO_CloseCamera	x	x	x	x	x	x
4.7.4	PCO_AllocateBuffer	x	x	x	x	x	x
4.7.5	PCO_GetBuffer	x	x	x	x	x	x
4.7.6	PCO_FreeBuffer	x	x	x	x	x	x
4.7.7	PCO_AddBufferEx	x	x	x	x	x	x
4.7.8	PCO_AddBuffer	x	x	x	x	x	x
4.7.9	PCO_GetBufferStatus	x	x	x	x	x	x
4.7.10	PCO_CancelImages PCO_RemoveBuffer	x	x	x	x	x	x
4.7.11	PCO_GetImageEx	x	x	x	x	x	x
4.7.12	PCO_GetImage	x	x	x	x	x	x
4.7.13	PCO_GetPendingBuffer	x	x	x	x	x	x
4.7.14	PCO_CheckDevice- Availability	x	x	x	x	x	x
4.7.15	PCO_SetTransferParameter	interface	interface	mandatory	interface	interface	interface
4.7.16	PCO_GetTransferParameter	interface	interface	mandatory	interface	interface	interface
4.7.17	PCO_CamLinkSet- ImageParameters	x	x	x			x
4.7.18	PCO_GetInfoString	x	x	x	x	x	x
4.7.19	PCO_SetTimeouts	x	x	x	x	x	x
4.7.20	PCO_GetGigEIPAddress	interface	interface				interface
4.7.21	PCO_SetGigEIPAddress	interface	interface				interface
4.7.22	PCO_AddBufferExtern	x	x	x	x	x	x
4.7.23	PCO_GetMetaData		x				

- x: Function is available
- desc.: See descriptor for availability
- interface: Depends on interface

pco.
imaging

PCO AG
Donaupark 11
D-93309 Kelheim
fon +49 (0)9441 2005 0
fax +49 (0)9441 2005 20
eMail: info@pco.de
www.pco.de

The Cooke Corporation
6930 Metroplex Drive,
Romulus, Michigan 48174, USA
www.cookecorp.com