Linux Development Kit
pco.camera API

USB2 USB3

This document describes the development kit API for pco cameras under linux.

Released April 2019 © **PCO AG**

TABLE OF CONTENTS

1 Introduction

This document provides an detailed description of all functions of the PCO Camera Linux API. The API is a class interface and provides the functionality to write own Applications in a Linux environment.

Any C++ compiler can be used for development.

The intention was to provide a simple interface, which can be used with every PCO camera of the pco.camera series independent from the used interface.

The **PCO** Linux Development Kit includes:

- all header files for compilation
- all source code files for building the common and specific classes
- example code
- precompiled libraries, which are used in the examples
- compiled binaries of the examples

1.1 General

The API consist of two classes a communication class which is used to control the camera settings and a grabber class which is used to transfer single or multiple images from the camera to the PC. Because communication with the camera and image transfer depends on the used hardware interface, interface specific classes exist.

For controlling the camera settings a common base class exists. The interface specific classes are subclasses derived from this base class. For image transfers only interface specific classes exist, but main functions use equal function declarations.

Error and warning codes can be found in file pco_err.h.

PCO_NOERROR is defined as value ’0’.

2 Class Documentation

2.1 CPco_com Class Reference

Inherited by CPco_com_usb.

Base Class Functions

Open a communication class and establish a connection with the camera. Communication is done through the Control_Command function.

- virtual DWORD Open_Cam (DWORD num)=0
- virtual DWORD Close_Cam ()
- virtual DWORD Control_Command (void* buf_in, DWORD size_in, void* buf_out, DWORD size_out)=0
- void SetLog (CPco_Log* Log)
- CPco_Log* GetLog ()

General Control and Status

This section contains general functions to control the camera and to request status information about the camera.

- DWORD PCO_SetRecordingState (WORD recstate)
- DWORD PCO_GetRecordingState (WORD* recstate)
- DWORD PCO_ArmCamera ()
- DWORD PCO_GetActualSize (DWORD* width, DWORD* height)
- DWORD PCO_ResetSettingsToDefault ()
- DWORD PCO_SetCameraToCurrentTime ()
- DWORD PCO_SetDateTime (struct tm* strTime)
- DWORD PCO_GetCameraSetup (WORD setup_id, DWORD* setup_flag, WORD* length)
- DWORD PCO_SetCameraSetup (WORD setup_id, DWORD* setup_flag, WORD length)
- DWORD PCO_RebootCamera ()

Camera Description

Functions and comments for Camera description

- DWORD PCO_GetCameraDescriptor (SC2_Camera_Description_Response* description)
- DWORD PCO_GetCameraDescription (SC2_Camera_Description_Response* response)
- DWORD PCO_GetCameraDescription (SC2_Camera_Description_2_Response* response)

General Camera StatusInformation

This section contains general functions to request general information about the camera and the actual camera state.

- DWORD PCO_GetCameraType (WORD* wCamType, DWORD* dwSerialNumber, WORD* wIfType=NULL)
- DWORD PCO_GetCameraName (void* buf, int length)
- DWORD PCO_GetInfo (DWORD typ, void* buf, int length)
- DWORD PCO_GetTemperature (SHORT* CCDTemp, SHORT* CAMTemp, SHORT* ExtTemp)
- DWORD PCO_GetHealthStatus (unsigned int* warnings, unsigned int* errors, unsigned int* status)
- DWORD PCO_GetSensorSignalStatus (DWORD* status, DWORD* imagecount)
- DWORD PCO_GetCameraBusyStatus (WORD* camera_busy)
- DWORD PCO_GetExpTrigSignalStatus (WORD* exptrgsignal)
- DWORD PCO_GetCOCRuntime (DWORD* time_s, DWORD* time_ns)
- DWORD PCO_GetCOCExptime (DWORD* time_s, DWORD* time_ns)
- DWORD PCO_GetImageTiming (SC2_Get_Image_Timing_Response* image_timing)

Timing Control and Status

This section contains functions to change and retrieve actual timing settings Setting of delay and exposure times can be done also when recording state is RUN. When recording state is STOP or for any other settings an additional

PCO_ArmCamera has to be done.

- DWORD PCO_GetTriggerMode (WORD* mode)
- DWORD PCO_SetTriggerMode (WORD mode)
- DWORD PCO_ForceTrigger (WORD* trigger)
- DWORD PCO_GetPixelRate (DWORD* pixelrate)
- DWORD PCO_SetPixelRate (DWORD PixelRate)
- DWORD PCO_GetDelayExposureTime (DWORD* delay, DWORD* expos, WORD* tb_delay, WORD* tb_expos)
- DWORD PCO_SetDelayExposureTime (DWORD delay, DWORD expos, WORD tb_delay, WORD tb_expos)
- DWORD PCO_GetDelayExposure (DWORD* delay, DWORD* expos)
- DWORD PCO_SetDelayExposure (DWORD delay, DWORD expos)
- DWORD PCO_GetTimebase (WORD* delay, WORD* expos)
- DWORD PCO_SetTimebase (WORD delay, WORD expos)
- DWORD PCO_GetFrameRate (WORD* wFrameRateStatus, DWORD* dwFrameRate, DWORD* dwFrameRateExposure)
- DWORD PCO_SetFrameRate (WORD* wFrameRateStatus, WORD wFramerateMode, DWORD* dwFrameRate, DWORD* dwFrameRateExposure)
- DWORD PCO_GetFPSExposureMode (WORD* wFPSExposureMode, DWORD* dwFPSExposureTime)
- DWORD PCO_SetFPSExposureMode (WORD wFPSExposureMode, DWORD* dwFPSExposureTime)

Sensor Control and Status

This section contains functions to change and retrieve actual format settings

- DWORD PCO_GetSensorFormat (WORD* wSensor)
- DWORD PCO_SetSensorFormat (WORD wSensor)
- DWORD PCO_GetROI (WORD* RoiX0, WORD* RoiY0, WORD* RoiX1, WORD* RoiY1)
- DWORD PCO_SetROI (WORD RoiX0, WORD RoiY0, WORD RoiX1, WORD RoiY1)
- DWORD PCO_GetBinning (WORD* BinHorz, WORD* BinVert)
- DWORD PCO_SetBinning (WORD BinHorz, WORD BinVert)
- DWORD PCO_GetADCOperation (WORD* wADCOperation)
- DWORD PCO_SetADCOperation (WORD num)
- DWORD PCO_GetDoubleImageMode (WORD* wDoubleImage)
- DWORD PCO_SetDoubleImageMode (WORD wDoubleImage)
- DWORD PCO_GetNoiseFilterMode (WORD* wNoiseFilterMode)
- DWORD PCO_SetNoiseFilterMode (WORD wNoiseFilterMode)
- DWORD PCO_GetConversionFactor (WORD* wConvFact)
- DWORD PCO_SetConversionFactor (WORD wConvFact)
- DWORD PCO_GetIRSensitivity (WORD* wIR)
- DWORD PCO_SetIRSensitivity (WORD wIR)
- DWORD PCO_GetCoolingSetpointTemperature (SHORT* sCoolSet)
- DWORD PCO_SetCoolingSetpointTemperature (SHORT sCoolSet)
- DWORD PCO_GetCoolingSetpoints (WORD wBlockID, SHORT* sSetPoints, WORD* wValidSetPoints)

Recording Control and Status

This section contains functions to change and retrieve actual recorder settings

- DWORD PCO_GetStorageMode (WORD* wStorageMode)
- DWORD PCO_SetStorageMode (WORD wStorageMode)
- DWORD PCO_GetRecorderSubmode (WORD* wRecSubmode)
- DWORD PCO_SetRecorderSubmode (WORD wRecSubmode)
- DWORD PCO_GetAcquireMode (WORD* wAcquMode)
- DWORD PCO_SetAcquireMode (WORD wAcquMode)
- DWORD PCO_GetAcquireModeEx (WORD* wAcquMode, DWORD* dwNumberImages)
- DWORD PCO_SetAcquireModeEx (WORD wAcquMode, DWORD dwNumberImages)
- DWORD PCO_GetAcqEnblSignalStatus (WORD* wAcquEnableState)

Storage Control and Status

This section contains functions to change and retrieve actual format settings

- DWORD PCO_GetCameraRamSize (DWORD* dwRamSize, WORD* wPageSize)
- DWORD PCO_GetCameraRamSegmentSize (DWORD* dwRamSegSize)
- DWORD PCO_SetCameraRamSegmentSize (DWORD* dwRamSegSize)
- DWORD PCO_GetActiveRamSegment (WORD* wActSeg)
- DWORD PCO_SetActiveRamSegment (WORD wActSeg)
- DWORD PCO_ClearRamSegment ()

- DWORD PCO_GetNumberOfImagesInSegment (WORD wSegment, DWORD* dwValid, DWORD* dwMax)
- DWORD PCO_GetSegmentImageSettings (WORD wSegment, WORD* wRes_hor, WORD* wRes_ver, WORD* wBin_x, WORD* wBin_y, WORD* wRoi_x0, WORD* wRoi_y0, WORD* wRoi_x1, WORD* wRoi_y1)

Image transfer

This section contains functions to invoke image transfers from the camera

- DWORD PCO_ReadImagesFromSegment (WORD wSegment, DWORD dwStartImage, DWORD dwLastImage)
- DWORD PCO_RequestImage ()
- DWORD PCO_RepeatImage ()
- DWORD PCO_CancelImage ()
- DWORD PCO_CancelImageTransfer ()
- DWORD PCO_GetImageTransferMode (WORD* wMode, WORD* wImageWidth, WORD* wImageHeight, WORD* wTxWidth, WORD* wTxHeight, WORD* wTxLineWordCnt, WORD* wParam, WORD* wParamLen)
- DWORD PCO_SetImageTransferMode (WORD wMode, WORD wImageWidth, WORD wImageHeight, WORD wTxWidth, WORD wTxHeight, WORD wTxLineWordCnt, WORD* wParam, WORD wParamLen)

Image data options

This section contains functions to change and retrieve actual settings for the transferred image data

- DWORD PCO_GetLookupableInfo (WORD wLUTNum, WORD* wNumberOfLuts, char* Description, WORD wDescLen, WORD* wIdentifier, BYTE* bInputWidth, BYTE* bOutputWidth, WORD* wFormat)
- DWORD PCO_GetLut (WORD* Identifier, WORD* Parameter)
- DWORD PCO_SetLut (WORD Identifier, WORD Parameter)
- DWORD PCO_GetBitAlignment (WORD* align)
- DWORD PCO_SetBitAlignment (WORD align)
- DWORD PCO_GetTimestampMode (WORD* mode)
- DWORD PCO_SetTimestampMode (WORD mode)
- DWORD PCO_GetHotPixelCorrectionMode (WORD* wHotPixelCorrectionMode)
- DWORD PCO_SetHotPixelCorrectionMode (WORD wHotPixelCorrectionMode)
- DWORD PCO_GetMetadataMode (WORD* wMode, WORD* wMetadataSize, WORD* wMetadataVersion)
- DWORD PCO_SetMetadataMode (WORD wMode, WORD* wMetadataSize, WORD* wMetadataVersion)

Hardware Input/Output options

This section contains functions to change and retrieve actual settings for Input and output connectors

- DWORD PCO_GetHWIOSignalCount (WORD* numSignals)
- DWORD PCO_GetHWIOSignalDescriptor (WORD SignalNum, SC2_Get_HW_IO_Signal_Descriptor_Response* SignalDesc)
- DWORD PCO_GetHWIOSignalDescriptor (WORD SignalNum, char* outbuf, int* size)
- DWORD PCO_GetHWIOSignal (WORD SignalNum, WORD* Enabled, WORD* Type, WORD* Polarity, WORD* FilterSetting, WORD* Selected)
- DWORD PCO_SetHWIOSignal (WORD SignalNum, WORD Enabled, WORD Type, WORD Polarity, WORD FilterSetting, WORD Selected)
- DWORD PCO_GetHWIOSignalTiming (WORD SignalNum, WORD Selection, DWORD* type, DWORD* Parameter)
- DWORD PCO_SetHWIOSignalTiming (WORD SignalNum, WORD Selection, DWORD Parameter)

Enhanced Timing control status

This section contains functions to change and retrieve further timing settings

- DWORD PCO_GetPowerDownMode (WORD* wPowerDownMode)
- DWORD PCO_SetPowerDownMode (WORD wPowerDownMode)
- DWORD PCO_GetUserPowerDownTime (DWORD* dwPdnTime)
- DWORD PCO_SetUserPowerDownTime (DWORD dwPdnTime)
- DWORD PCO_GetDelayExposureTimeTable (DWORD* dwDelay, DWORD* dwExposure, WORD* wTimeBaseDelay, WORD* wTimebaseExposure, WORD wCount)

- DWORD PCO_SetDelayExposureTimeTable (DWORD* dwDelay, DWORD* dwExposure, WORD wTimeBaseDelay, WORD wTimebaseExposure, WORD wCount)
- DWORD PCO_GetModulationMode (WORD* wModulationMode, DWORD* dwPeriodicalTime, WORD* wTimebasePeriodical, DWORD* dwNumberOfExposures, LONG* lMonitorOffset)
- DWORD PCO_SetModulationMode (WORD wModulationMode, DWORD dwPeriodicalTime, WORD wTimebasePeriodical, DWORD dwNumberOfExposures, LONG lMonitorOffset)
- DWORD **PCO_GetCMOSLinetiming** (WORD* wParameter, WORD* wTimebase, DWORD* dwLineTime)
- DWORD **PCO_GetCMOSLinetiming_res** (DWORD* dwMinLineTime, DWORD* dwMaxLineTime, DWORD* dwLineCaps)
- DWORD **PCO_SetCMOSLinetiming** (WORD wParameter, WORD wTimebase, DWORD dwLineTime)
- DWORD **PCO_GetCMOSLineExposureDelay** (DWORD* dwExposureLines, DWORD* dwDelayLines)
- DWORD **PCO_SetCMOSLineExposureDelay** (DWORD dwExposureLines, DWORD dwDelayLines)
- DWORD PCO_GetCameraSynchMode (WORD* wCameraSynchMode)
- DWORD PCO_SetCameraSynchMode (WORD wCameraSynchMode)
- DWORD PCO_GetFastTimingMode (WORD* wFastTimingMode)
- DWORD PCO_SetFastTimingMode (WORD wFastTimingMode)

Enhanced Information

This section contains functions to retrieve further information

- DWORD PCO_GetFirmwareVersion (char* outbuf, int* size)
- DWORD PCO_GetHardwareVersion (char* outbuf, int* size)
- DWORD PCO_GetFirmwareVersion (SC2_Firmware_Versions_Response* response)
- DWORD PCO_GetFirmwareVersionExt (BYTE bNum, SC2_Firmware_Versions_Response* response)

IEEE1394 interface parameters

This section contains functions to change and retrieve parameters for the IEEE1394 interface

- DWORD PCO_GetIEEE1394InterfaceParams (WORD* wMasterNode, WORD* wIsochChannel, WORD* wIsochPacketLen, WORD* wIsochPacketNum)
- DWORD PCO_SetIEEE1394InterfaceParams (WORD wMasterNode, WORD wIsochChannel, WORD wIsochPacketLen, WORD wIsochPacketNum)
- DWORD PCO_GetIEEE1394ISOByteorder (WORD* wMode)
- DWORD PCO_SetIEEE1394ISOByteorder (WORD wMode)

HD/SDI interface parameters and image transfer control

This section contains functions to control output to the HD/SDI interface

- DWORD PCO_GetInterfaceOutputFormat (WORD wInterface, WORD* wFormat)
- DWORD PCO_SetInterfaceOutputFormat (WORD wInterface, WORD wFormat)
- DWORD PCO_GetInterfaceStatus (WORD wInterface, DWORD* dwWarnings, DWORD* dwErrors, DWORD* dwStatus)

HD/SDI interface image transfer control

- DWORD PCO_PlayImagesFromSegment (WORD wSegment, WORD wInterface, WORD wMode, WORD wSpeed, DWORD dwRangeLow, DWORD dwRangeHigh, DWORD dwStartPos)
- DWORD PCO_GetPlayPosition (WORD* wStatus, DWORD* dwPosition)
- DWORD PCO_GetColorSettings (SC2_Get_Color_Settings_Response* ColSetResp)
- DWORD PCO_SetColorSettings (SC2_Set_Color_Settings* SetColSet)
- DWORD PCO_DoWhiteBalance (WORD wMode)
- DWORD PCO_GetWhiteBalanceStatus (WORD* wStatus, WORD* wColorTemp, SHORT* sTint)

Special control status

This section contains functions to change and retrieve special camera settings

- DWORD PCO_InitiateSelftestProcedure (DWORD* dwWarn, DWORD* dwErr)
- DWORD PCO_WriteHotPixelList (WORD wListNo, WORD wNumValid, WORD* wHotPixX, WORD* wHotPixY)

- DWORD PCO_ReadHotPixelList (WORD wListNo, WORD wArraySize, WORD* wNumValid, WORD* wNumMax, WORD* wHotPixX, WORD* wHotPixY)
- DWORD PCO_ClearHotPixelList (WORD wListNo, DWORD dwMagic1, DWORD dwMagic2)
- DWORD PCO_ClearHotPixelList (WORD wListNo)
- DWORD PCO_LoadLookuptable (WORD wIdentifier, WORD wPacketNum, WORD wFormat, WORD wLength, BYTE* bData)
- DWORD PCO_ReadLookuptable (WORD wIdentifier, WORD wPacketNum, WORD* wFormat, WORD* wLength, BYTE* bData, WORD buflen)
- DWORD PCO_GetColorCorrectionMatrix (char* szCCM, WORD* len)
- DWORD PCO_GetBayerMultiplier (WORD* wMode, WORD* wMul)
- DWORD PCO_SetBayerMultiplier (WORD wMode, WORD* wMul)
- DWORD PCO_GetFanControlStatus (WORD* wFanMode, WORD* wFanMin, WORD* wFanMax, WORD* wStepSize, WORD* wSetValue, WORD* wActualValue)
- DWORD PCO_SetFanControlStatus (WORD wFanMode, WORD wSetValue)
- DWORD PCO_GetHWLEDSignal (DWORD* dwParameter)
- DWORD PCO_SetHWLEDSignal (DWORD dwParameter)
- DWORD PCO_GetDSNUAdjustMode (WORD* wMode)
- DWORD PCO_SetDSNUAdjustMode (WORD wMode)
- DWORD PCO_InitDSNUAdjustment (WORD* wMode)
- DWORD PCO_GetCDIMode (WORD* wMode)
- DWORD PCO_SetCDIMode (WORD wMode)
- DWORD PCO_GetPowersaveMode (WORD* wMode, WORD* wDelayMinutes)
- DWORD PCO_SetPowersaveMode (WORD wMode, WORD wDelayMinutes)
- DWORD PCO_GetBatteryStatus (WORD* wBatteryType, WORD* wBatteryLevel, WORD* wPowerStatus)

Class Control Functions

These functions are used to control some internal variables of the class.

- void gettimeouts (PCO_SC2_TIMEOUTS* strTimeouts)
- void Set_Timeouts (void* timetable, DWORD length)
- void Sleep_ms (DWORD time_ms)

2.1.1 Detailed Description

Base interface class.

This is the communication class which includes all functions, which build the commands codes which can then be sent to the pco.cameras. Derived from this class are all interface specific classes. This class includes some common functions and defines the mandatory functions that each subclass has to implement.

2.1.2 Member Function Documentation

2.1.2.1 Open_Cam

pcotag Base Class Functions Opens a connection to a pco.camera This is a virtual function the implementation is in the interface specific class

Prototype:

```
virtual DWORD Open_Cam (
    DWORD num
);
[pure virtual]
```

Parameters:

Name	Type	Description
num	DWORD	Number of the camera to open starting with zero.

Return value:

Error code or PCO_NOERROR on success

Implemented in CPco_com_usb.

2.1.2.2 Close_Cam

Closes a connection to a pco.camera.

Prototype:

```
virtual DWORD Close_Cam (    );
[virtual]
```

Parameters:

No parameter

Return value:

Error code or PCO_NOERROR on success

Reimplemented in CPco_com_usb.

2.1.2.3 Control_Command

The main function to communicate with the pco.camera via **PCO** telegrams.
See sc2_telegram.h for a list of telegram definitions and sc2_command.h for a list of all public commands. Checksum calculation is done in this function, therefore there is no need to pre-calculate it.

Prototype:

```
virtual DWORD Control_Command (
    void*  buf_in,
    DWORD size_in,
    void*  buf_out,
    DWORD size_out
);
[pure virtual]
```

Parameters:

Name	Type	Description
buf_in	void*	Pointer to the buffer where the telegram is stored
size_in	DWORD	Size of the input buffer in bytes
buf_out	void*	Pointer to the buffer where the response gets stored
size_out	DWORD	Size of the output buffer in bytes. If the returned telegram does not fit into the output buffer

Return value:

Error code or PCO_NOERROR on success

Implemented in CPco_com_usb.

2.1.2.4 SetLog

Sets the logging behaviour for the communication class. If this function is not called no logging is performed. Logging might be useful to follow the program flow of the application. Logging class is available through the library libpcolog.

Prototype:

```
void SetLog (
    CPco_Log*  Log
);
```

Parameters:

Name	Type	Description
Log	CPco_Log*	Pointer to a CPco_Log logging class object

Return value:

None

2.1.2.5 GetLog

Returns the currently used logging class object.

Prototype:

```
CPco_Log* GetLog (    );
```

Parameters:

No parameter

Return value:

logging class object or NULL if no logging was set.

2.1.2.6 PCO_SetRecordingState

Sets the current recording state and waits until the status is valid. If the state cannot be set within one second (+ current frametime for state [stop]), the function will return an error.
The recording state controls the run state of the camera. If the Recording State is [run], sensor exposure and readout sequences are started depending on current camera settings (trigger mode,acquire mode, external signals...).
The Recording State has the highest priority compared to functions like <acq enbl> or exposure trigger.
When the Recording State is set to [stop], sensor exposure and readout sequences are stopped. If the camera is currently in [sensor_readout] state, this readout is finished, before camera run state is changed to [sensor_idle]. If the camera is currently in [sensor_exposing] state, the exposure is cancelled and camera run state is changed immediately to [sensor_idle]. In run state [sensor_idle] the camera is running a special idle mode to prevent dark charge accumulation.
If any camera parameter was changed: before setting the Recording State to [run], the function PCO_ArmCamera must be called. This is to ensure that all settings were correctly and are accepted by the camera. If a successful Recording State [run] command is sent and recording is started, the images from a previous record to the active segment are lost.
The recording status has the highest priority compared to functions like <acq enbl> or <exp trig>. The recording state can be [stop]'ped at any time. The recording state can be set to [run] only if the camera was successfully armed before.

Prototype:

```
DWORD PCO_SetRecordingState (
    WORD recstate
);
```

Parameters:

Name	Type	Description
recstate	WORD	Variable to set the active recording state. • 0x0000 = stop camera and wait until recording state = [stop] • 0x0001 = start camera and wait until recording state = [run]

Return value:

Error code or PCO_NOERROR on success

2.1.2.7 PCO_GetRecordingState

Requests the current recording state.
This function returns the current Recording State of the camera. The Recording State can change from [run] to [stop] through:

- Call to function PCO_SetRecordingState [stop]
- PCO_SetStorageMode is [recorder], PCO_SetRecorderSubmode is [sequence] and active segment is full
- PCO_SetStorageMode is [recorder], PCO_SetRecorderSubmode is [ring buffer],
- PCO_SetRecordStopEvent is [on] and the given number of images is recorded.

Prototype:

```
DWORD PCO_GetRecordingState (
    WORD*  recstate
);
```

Parameters:

Name	Type	Description
recstate	WORD*	Pointer to a WORD variable to get the current recording state. • 0x0000 = camera is stopped, recording state [stop] • 0x0001 = camera is running, recording state [run]

Return value:

Error code or PCO_NOERROR on success

2.1.2.8 PCO_ArmCamera

Arms the camera and validates the settings.
This function does arm, this means prepare the camera for a following recording. All configurations and settings made up to this moment are accepted, validated and the internal settings of the camera are prepared. If the arm was successful the camera state is changed to [armed] and the camera is able to start image recording immediately, when recording state is set to [run].
The command will be rejected, if Recording State is [run], see PCO_GetRecordingState.
On power up the camera is in state [not armed] and Recording State [stop]. Camera arm state is changed to [not armed], when settings are changed, with the following exception. Camera arm state is not changed, when settings related to exposure time will be done during Recording State [run].

Prototype:

```
DWORD PCO_ArmCamera (    );
```

Parameters:

No parameter

Return value:

Error code or PCO_NOERROR on success

2.1.2.9 PCO_GetActualSize

Gets the actual armed image size of the camera. This accounts for binning and ROI. 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.

Prototype:

```
DWORD PCO_GetActualSize (
    DWORD*  width,
    DWORD*  height
);
```

Parameters:

Name	Type	Description
width	DWORD*	Pointer to an DWORD to get the current width in pixel
height	DWORD*	Pointer to an DWORD to get the current height in pixel

Return value:

Error code or PCO_NOERROR on success

2.1.2.10 PCO_ResetSettingsToDefault

Resets all camera settings to default values. This function can be used to reset all camera settings to its default values. This function is also executed during a power-up sequence. The camera must be stopped before calling this command. Default settings are slightly different for all cameras.
The following are the default settings: .

Setting:	Default:
Sensor Format	Standard
ROI	Full resolution
Binning	No binning
Pixel Rate	Depending on camera type
Gain	Normal gain (if setting available due to sensor)
Double Image Mode	Off
IR sensitivity	Off (if setting available due to sensor)
Cooling Set point	Depending on camera type
ADC mode	Using one ADC (if setting available
Exposure Time	Depending on camera type (10-20ms)
Delay Time	0ms
Trigger Mode	Auto Trigger
Recording state	Stopped
Memory Segmentation	Total memory allocated to first segment
Storage Mode	Recorder Ring Buffer
Acquire Mode	Auto

Prototype:

```
DWORD PCO_ResetSettingsToDefault (    );
```

Parameters:

No parameter

Return value:

Error code or PCO_NOERROR on success

2.1.2.11 PCO_SetCameraToCurrentTime

Sets the camera time to current system time.
The date and time is updated automatically, as long as the camera is supplied with power. Camera time is used for the timestamp and metadata. When powering up the camera the camera clock is reset and all date and time information is set to zero. Therefore this command or PCO_SetDateTime should be called once. It might be necessary to call the function again in distinct time intervals, because some deviation between PC time and camera time might occur after some time.

Prototype:

```
DWORD PCO_SetCameraToCurrentTime (    );
```

Parameters:

No parameter

Return value:

Error code or PCO_NOERROR on success

2.1.2.12 PCO_SetDateTime

Sets the time and date to the camera. The date and time is updated automatically, as long as the camera is supplied with power. Camera time is used for the timestamp and metadata. When powering up the camera the camera clock is reset and all date and time information is set to zero. Therefore this command or PCO_SetDateTime should be called once. It might be necessary to call the function again in distinct time intervals, because some deviation between PC time and camera time might occur after some time. Note:

- [ms] and [μs] values of the camera clock are set to zero, when this command is executed

Prototype:

```
DWORD PCO_SetDateTime (
    struct tm*  strTime
);
```

Parameters:

Name	Type	Description
strTime	struct tm*	Pointer to a tm structure.

Return value:

Error code or PCO_NOERROR on success

2.1.2.13 PCO_GetCameraSetup

Request the current camera setup.
This function is used to query the current operation mode of the camera. Some cameras can work at different operation modes with different descriptor settings.
pco.edge:
To get the current shutter mode input index setup_id must be set to 0.
current shutter mode is returned in setup_flag[0]

- 0x00000001 = Rolling Shutter
- 0x00000002 = Global Shutter
- 0x00000004 = Global Reset

Prototype:

```
DWORD PCO_GetCameraSetup (
    WORD setup_id,
    DWORD* setup_flag,
    WORD* length
);
```

Parameters:

Name	Type	Description
setup_id	WORD	Identification code for selected setup type.
setup_flag	DWORD*	Pointer to a DWORD array to get the current setup flags. If set to NULL in input only the array length is returned. • On input this variable can be set to NULL, then only array length is filled with correct value. • On output the array is filled with the available information for the selected setup_id
length	WORD*	Pointer to a WORD variable • On input to indicate the length of the Setup_flag array in DWORDs. • On output the length of the setup_flag array in DWORDS

Return value:

Error code or PCO_NOERROR on success

2.1.2.14 PCO_SetCameraSetup

Sets the camera setup structure (see camera specific structures)
pco.edge:
To get the current shutter mode input index setup_id must be set to 0.
current shutter mode is returned in setup_flag[0]

- 0x00000001 = Rolling Shutter
- 0x00000002 = Global Shutter
- 0x00000004 = Global Reset When camera is set to a new shuttermode uit must be reinitialized by calling one of the reboot functions. After rebooting, camera description must be read again see PCO_GetCameraDescription.

Prototype:

```
DWORD PCO_SetCameraSetup (
    WORD setup_id,
    DWORD* setup_flag,
    WORD length
);
```

Parameters:

Name	Type	Description
setup_id	WORD	Identification code for selected setup type.
setup_flag	DWORD*	Flags to be set for the selected setup type.
length	WORD	Number of valid DWORDs in setup_flag array.

Return value:

Error code or PCO_NOERROR on success

2.1.2.15 PCO_RebootCamera

Reboot the camera.

Prototype:

```
DWORD PCO_RebootCamera (    );
```

Parameters:

No parameter

Return value:

Error code or PCO_NOERROR on success

2.1.2.16 PCO_GetCameraDescriptor

Gets the cached camera description data structure. This is a cached value that is retrieved once when Open_Cam is called and with every PCO_GetCameraDescription call. See PCO_GetCameraDescription for a more detailed version of the retrieved camera description structure. Because parameters inside the structure are fixed values, this is the recommended function to work with.

PCO_Description structure

Because different sensors (CCD, CMOS, sCMOS) are used in the different camera models, each camera has its own description. This description should be readout shortly after access to the camera is established. In the description general margins for all sensor related settings and bitfields for available options of the camera are given. This set of information can be used to validate the input parameter for commands, which change camera settings, before they are sent to the camera. The dwGeneralCapsDESC1 and dwGeneralCapsDESC2 bitfields in the PCO_Description structure can be used to see what options are supported from the connected camera. Supported options may vary with different camera types and also between different firmware versions of one camera type.

Prototype:

```
DWORD PCO_GetCameraDescriptor (
    SC2_Camera_Description_Response*  description
);
```

Parameters:

Name	Type	Description
description	SC2_Camera_Description_Response*	Pointer to a PCO description structure. • on output structure is filled with cached camera settings

Return value:

Error code or PCO_NOERROR on success

2.1.2.17 PCO_GetCameraDescription

Read the camera description data structure from the camera.

Prototype:

```
DWORD PCO_GetCameraDescription (
    SC2_Camera_Description_Response*  response
);
```

Parameters:

Name	Type	Description
response	SC2_Camera_Description_Response*	Pointer to a SC2_Camera_Description_Response structure. • on output structure is filled with the camera settings

Return value:

Error code or PCO_NOERROR on success

2.1.2.18 PCO_GetCameraDescription

Gets the additional camera descriptions See sc2_telegram.h for more information.

Prototype:

```
DWORD PCO_GetCameraDescription (
    SC2_Camera_Description_2_Response*  response
);
```

Parameters:

Name	Type	Description
response	SC2_Camera_Description_2_Response*	Pointer to a SC2_Camera_Description_2_Response structure. • on output structure is filled with the camera settings

Return value:

Error code or PCO_NOERROR on success

2.1.2.19 PCO_GetCameraType

Gets the camera type, serial number and interface type.

Prototype:

```
DWORD PCO_GetCameraType (
    WORD*  wCamType,
    DWORD*  dwSerialNumber,
    WORD*  wIfType = NULL
);
```

Parameters:

Name	Type	Description
wCamType	WORD*	Pointer to WORD variable to receive the camera type.
dwSerialNumber	DWORD*	Pointer to DWORD variable to receive the serial number.
wIfType	WORD*	Pointer to WORD variable to receive connected Interface type

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.20 PCO_GetCameraName

Gets the name of the camera.
The string buf 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").

Return values

buf	Null terminated string with camera name

Prototype:

```
DWORD PCO_GetCameraName (
    void*  buf,
    int length
);
```

Parameters:

Name	Type	Description
buf	void*	Pointer to a string to receive the camera name.
length	int	WORD variable which holds the maximum length of the string.

Return value:

Error code

2.1.2.21 PCO_GetInfo

Gets the basic information of the camera.
The string buf has to be long enough to get the information.
The input pointer will be filled with one of the following parameters:

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

Return values

buf	Null terminated string with requested information

Prototype:

```
DWORD PCO_GetInfo (
    DWORD typ,
    void*  buf,
    int length
);
```

Parameters:

Name	Type	Description
typ	DWORD	selector for information • 1: Camera name • 2: Sensor name
buf	void*	Pointer to a string, to receive the requested information
length	int	int variable which holds the length of buf.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.22 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.
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.

Return values

CCDTemp	CCD temperature value.
CAMTemp	Camera temperature value.
ExtTemp	Extended device temperature value (i.E. Power device). Not supported from all cameras.

Prototype:

```
DWORD PCO_GetTemperature (
    SHORT*  CCDTemp,
    SHORT*  CAMTemp,
    SHORT*  ExtTemp
);
```

Parameters:

Name	Type	Description
CCDTemp	SHORT*	Pointer to a SHORT variable, to receive the CCD temp. value.
CAMTemp	SHORT*	Pointer to a SHORT variable, to receive the camera temp. value.
ExtTemp	SHORT*	Pointer to a SHORT variable, to receive the power device temp. value.

Return value:

Error code

2.1.2.23 PCO_GetHealthStatus

Request the current camera health status: warnings, errors.
It is recommended to call this function frequently (e.g. every 5s, or after calling ARM) in order to recognize camera internal problems, like electronics temperature error. This will enable users to prevent the camera hardware from damage.

- Warnings are encoded as bits of a long word. Bit set indicates warning, bit cleared indicates that the corresponding parameter is OK.
- System errors encoded as bits of a long word. Bit set indicates error, bit cleared indicates that the corresponding status is OK.
- System Status encoded as bits of a long word.

Return values

warnings	Actual warning state
errors	Actual error state
status	Actual system state

Prototype:

```
DWORD PCO_GetHealthStatus (
    unsigned int*  warnings,
    unsigned int*  errors,
    unsigned int*  status
);
```

Parameters:

Name	Type	Description
warnings	unsigned int*	Pointer to a DWORD variable, to receive the warning value.
errors	unsigned int*	Pointer to a DWORD variable, to receive the error value.
status	unsigned int*	Pointer to a DWORD variable, to receive the error value.

Return value:

Error code

2.1.2.24 PCO_GetSensorSignalStatus

Gets the signal state of the camera sensor. Edge only!
The signals must not be deemed to be a real time 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 synchronization use hardware signals.

Return values

status	Actual camera state
imagecount	number of last finished image inside camera

Prototype:

```
DWORD PCO_GetSensorSignalStatus (
    DWORD*  status,
    DWORD*  imagecount
);
```

Parameters:

Name	Type	Description
status	DWORD*	DWORD pointer to receive the status flags of the sensor (can be NULL). • Bit0: SIGNAL_STATE_BUSY 0x0001 • Bit1: SIGNAL_STATE_IDLE 0x0002 • Bit2: SIGNAL_STATE_EXP 0x0004 • Bit3: SIGNAL_STATE_READ 0x0008
imagecount	DWORD*	DWORD pointer to receive the # of the last finished image(can be NULL).

Return value:

Error code

2.1.2.25 PCO_GetCameraBusyStatus

Gets the busy state of the camera.
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.
Note: The busy status is according to the hardware signal <busy> at the <status output> at the rear of pco.power or the camera connectors. 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.

Return values

camera_busy	Actual camera busy state • 0x0000 = camera is [not busy], ready for a new trigger command • 0x0001 = camera is [busy], not ready for a new trigger command

Prototype:

```
DWORD PCO_GetCameraBusyStatus (
    WORD* camera_busy
);
```

Parameters:

Name	Type	Description
camera_busy	WORD*	Pointer to a WORD variable to receive the busy state.

Return value:

Error code

2.1.2.26 PCO_GetExpTrigSignalStatus

Get the current status of the <exp trig> user input (one of the <control in> inputs at the rear of pco.power or the camera connectors).
See camera manual for more information about hardware signals.

Return values

exptrgsignal	Actual exposure trigger signal state. • 0x0000 = exposure trigger signal is off • 0x0001 = exposure trigger signal is on

Prototype:

```
DWORD PCO_GetExpTrigSignalStatus (
    WORD* exptrgsignal
);
```

Parameters:

Name	Type	Description
exptrgsignal	WORD*	Pointer to a WORD variable to receive the exposure trigger signal state.

Return value:

Error code.

2.1.2.27 PCO_GetCOCRuntime

Gets the frametime for one image of the camera.
Get and split the 'camera operation code' (COC) 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.

Return values

time_s	Time part in seconds of the COC.
time_ns	Time part in nanoseconds of the COC (range: 0ns-999.999.999ns).

Prototype:

```
DWORD PCO_GetCOCRuntime (
    DWORD*  time_s,
    DWORD*  time_ns
);
```

Parameters:

Name	Type	Description
time_s	DWORD*	Pointer to a DWORD variable to receive the time part in seconds.
time_ns	DWORD*	Pointer to a DWORD variable to receive the time part in nanoseconds.

Return value:

Error code

2.1.2.28 PCO_GetCOCExptime

Gets the actual exposuretime for one image of the camera.
Get and split the actual exposuretime into two DWORD. One will hold the longer part, in seconds, and the other will hold the shorter part, in nanoseconds.

Return values

time_s	Time part in seconds of the exposuretime.
time_ns	Time part in nanoseconds of the exposuretime (range: 0ns-999.999.999ns).

Prototype:

```
DWORD PCO_GetCOCExptime (
    DWORD*  time_s,
    DWORD*  time_ns
);
```

Parameters:

Name	Type	Description
time_s	DWORD*	Pointer to a DWORD variable to receive the time part in seconds.
time_ns	DWORD*	Pointer to a DWORD variable to receive the time part in nanoseconds.

Return value:

Error code

2.1.2.29 PCO_GetImageTiming

Gets the timing of one image, including trigger delay, trigger jitter, etc.
The input structure will be filled with the following parameters:

- 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.

Return values

image_timing	SC2_Get_Image_Timing_Response structure filled with camera settings

Prototype:

```
DWORD PCO_GetImageTiming (
    SC2_Get_Image_Timing_Response*  image_timing
);
```

Parameters:

Name	Type	Description
image_timing	SC2_Get_Image_Timing_Response*	Pointer to a SC2_Get_Image_Timing_Response structure

Return value:

Error code

2.1.2.30 PCO_GetTriggerMode

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

- 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 PCO_GetAcquireMode() (Auto / External)). 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].

Return values

mode	Actual trigger mode

Prototype:

```
DWORD PCO_GetTriggerMode (
    WORD*  mode
);
```

Parameters:

Name	Type	Description
mode	WORD*	Pointer to a WORD variable to receive the triggermode.

Return value:

Error code

2.1.2.31 PCO_SetTriggerMode

Sets the trigger mode of the camera.
See PCO_GetTriggerMode for a detailed explanation.
The command will be rejected, if Recording State is [run].

Prototype:

```
DWORD PCO_SetTriggerMode (
    WORD mode
);
```

Parameters:

Name	Type	Description
mode	WORD	WORD variable to hold the triggermode.

Return value:

Error code

2.1.2.32 PCO_ForceTrigger

Forces a software trigger to the camera.
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.
Result:
Return values

trigger	trigger occurence state • 0x0000 = trigger command was unsuccessful because the camera is busy • 0x0001 = a new image exposure has been triggered by the command

Prototype:

```
DWORD PCO_ForceTrigger (
    WORD* trigger
);
```

Parameters:

Name	Type	Description
trigger	WORD*	Pointer to a WORD variable to receive whether a trigger occurred or not.

Return value:

Error code

2.1.2.33 PCO_GetPixelRate

Gets the pixelrate for reading images from the image sensor.

Return values

pixelrate	Actual pixelrate of image sensor

Prototype:

```
DWORD PCO_GetPixelRate (
    DWORD*  pixelrate
);
```

Parameters:

Name	Type	Description
pixelrate	DWORD*	Pointer to a DWORD variable to receive the pixelrate.

Return value:

Error code

2.1.2.34 PCO_SetPixelRate

Sets the pixelrate of the camera.
For the pco.edge the higher pixelrate needs also execution of PCO_SetTransferParameter() and PCO_SetLut() with appropriate parameters.

Prototype:

```
DWORD PCO_SetPixelRate (
    DWORD PixelRate
);
```

Parameters:

Name	Type	Description
PixelRate	DWORD	DWORD variable to hold the pixelrate.

Return value:

Error code

2.1.2.35 PCO_GetDelayExposureTime

Gets the exposure and delay time and the time bases of the camera.
Timebase:

- 0 -> value is in ns: exp. time of 100 means 0.0000001s.
- 1 -> value is in μs: exp. time of 100 means 0.0001s.
- 2 -> value is in ms: exp. time of 100 means 0.1s.

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.

Prototype:

```
DWORD PCO_GetDelayExposureTime (
    DWORD*  delay,
    DWORD*  expos,
    WORD*  tb_delay,
    WORD*  tb_expos
);
```

Parameters:

Name	Type	Description
delay	DWORD*	Pointer to a DWORD variable to receive the delay time.
expos	DWORD*	Pointer to a DWORD variable to receive the exposure time.
tb_delay	WORD*	Pointer to a WORD variable to receive the delay timebase.
tb_expos	WORD*	Pointer to a WORD variable to receive the exposure timebase.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.36 PCO_SetDelayExposureTime

Sets the exposure and delay time and the time bases of the camera.
If the recording state is on, it is possible to change the timing without calling PCO_ArmCamera().
Timebase:

- 0 -> value is in ns: exp. time of 100 means 0.0000001s.
- 1 -> value is in μs: exp. time of 100 means 0.0001s.
- 2 -> value is in ms: exp. time of 100 means 0.1s.

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.

Prototype:

```
DWORD PCO_SetDelayExposureTime (
    DWORD delay,
    DWORD expos,
    WORD tb_delay,
    WORD tb_expos
);
```

Parameters:

Name	Type	Description
delay	DWORD	DWORD variable to hold the delay time.
expos	DWORD	DWORD variable to hold the exposure time.
tb_delay	WORD	WORD variable to hold the delay timebase.
tb_expos	WORD	WORD variable to hold the exp. timebase.

Return value:

Error code

2.1.2.37 PCO_GetDelayExposure

Gets the exposure and delay time table of the camera. See PCO_SetDelayExposureTime() for a detailed description.

Return values

delay	Actual delay time of camera
expos	Actual exposure time of camera

Prototype:

```
DWORD PCO_GetDelayExposure (
    DWORD* delay,
    DWORD* expos
);
```

Parameters:

Name	Type	Description
delay	DWORD*	Pointer to a DWORD array to receive the delay time.
expos	DWORD*	Pointer to a DWORD array to receive the exposure time.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.38 PCO_SetDelayExposure

Sets the exposure and delay time of the camera without changing the timebase.
If the recording state is on, it is possible to change the timing without calling PCO_ArmCamera.
See PCO_SetDelayExposureTime() for a detailed description.

Prototype:

```
DWORD PCO_SetDelayExposure (
    DWORD delay,
    DWORD expos
);
```

Parameters:

Name	Type	Description
delay	DWORD	DWORD variable to hold the delay time.
expos	DWORD	DWORD variable to hold the exposure time.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.39 PCO_GetTimebase

Gets the exposure and delay time bases of the camera. See PCO_SetDelayExposureTime() for a detailed description.

Prototype:

```
DWORD PCO_GetTimebase (
    WORD* delay,
    WORD* expos
);
```

Parameters:

Name	Type	Description
delay	WORD*	Pointer to WORD variable to receive the delay timebase.
expos	WORD*	Pointer to WORD variable to receive the exposure timebase.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.40 PCO_SetTimebase

Sets the exposure and delay time bases of the camera. See PCO_SetDelayExposureTime() for a detailed description.

Prototype:

```
DWORD PCO_SetTimebase (
    WORD delay,
    WORD expos
);
```

Parameters:

Name	Type	Description
delay	WORD	WORD variable to hold the delay time base.
expos	WORD	WORD variable to hold the exposure time base.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.41 PCO_GetFrameRate

Get frame rate / exposure time.

Note:

- Frame rate 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!

Prototype:

```
DWORD PCO_GetFrameRate (
    WORD*  wFrameRateStatus,
    DWORD*  dwFrameRate,
    DWORD*  dwFrameRateExposure
);
```

Parameters:

Name	Type	Description
wFrameRateStatus	WORD*	Pointer to WORD variable to receive the status • 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 PCO_SetFramerate() function.
dwFrameRate	DWORD*	DWORD variable to receive the actual frame rate in mHz
dwFrameRateExposure	DWORD*	DWORD variable to receive the actual exposure time (in ns)

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.42 PCO_SetFrameRate

Sets the frame rate mode, rate and exposure time.
Set frame rate and exposure time. This command is intended to set directly the frame rate and the exposure time of the camera. The frame rate is limited by the readout time and the exposure time:

$$Framerate \leq \frac{1}{t_{readout}}$$

$$Framerate \leq \frac{1}{t_{expos}}$$

Please note that there are some overhead times, therefore the real values can differ slightly, e.g. the maximum frame rate 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 frame rate / exposure time is changed immediately. The function returns the actually configured frame rate and exposure time.
Record is off: The frame rate / exposure time is stored. The function does not change the input values for frame rate 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 arameters are validated. The following procedure is recommended:

- Set frame rate 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 frame rate and exposure time using PCO_GetFrameRate.

Prototype:

```
DWORD PCO_SetFrameRate (
    WORD*  wFrameRateStatus,
    WORD wFramerateMode,
    DWORD*  dwFrameRate,
    DWORD*  dwFrameRateExposure
);
```

Parameters:

Name	Type	Description
wFrameRateStatus	WORD*	Pointer to WORD variable to receive the status • 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
wFramerateMode	WORD	Pointer to WORD variable to set the frame rate mode • 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.
dwFrameRate	DWORD*	DWORD variable to receive the actual frame rate in mHz (milli!)
dwFrameRateExposure	DWORD*	DWORD variable to receive the actual exposure time (in ns)

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.43 PCO_GetFPSExposureMode

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 frame rate (FPS = Frames Per Second) depends on the pixelrate, the vertical ROI and the exposure time.

Prototype:

```
DWORD PCO_GetFPSExposureMode (
    WORD*  wFPSExposureMode,
    DWORD*  dwFPSExposureTime
);
```

Parameters:

Name	Type	Description
wFPSExposureMode	WORD*	Pointer to a WORD variable to receive the FPS-exposure-mode. • 0: FPS Exposure Mode off, exposure time set by PCO_SetDelay/Exposure Time command • 1: FPS Exposure Mode on, exposure time set automatically to 1 / FPS max. PCO_SetDelay/Exposure Time commands are ignored.
dwFPSExposureTime	DWORD*	Pointer to a DWORD variable to receive the FPS exposure time in nanoseconds.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.44 PCO_SetFPSExposureMode

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 frame rate (FPS = Frames Per Second) depends on the pixelrate, the vertical ROI and the exposure time.

Prototype:

```
DWORD PCO_SetFPSExposureMode (
    WORD wFPSExposureMode,
    DWORD*  dwFPSExposureTime
);
```

Parameters:

Name	Type	Description
wFPSExposureMode	WORD	WORD variable to hold the FPS-exposure-mode. • 0: FPS Exposure Mode off, exposure time set by PCO_SetDelay/Exposure Time command • 1: FPS Exposure Mode on, exposure time set automatically to 1 / FPS max. PCO_SetDelay/Exposure Time commands are ignored.
dwFPSExposureTime	DWORD*	Pointer to a DWORD variable to receive the FPS exposure time in nanoseconds.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.45 PCO_GetSensorFormat

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

- 0x0000 = [standard]
- 0x0001 = [extended]

Prototype:

```
DWORD PCO_GetSensorFormat (
    WORD*  wSensor
);
```

Parameters:

Name	Type	Description
wSensor	WORD*	Pointer to a WORD variable to receive the sensor format.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.46 PCO_SetSensorFormat

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

- 0x0000 = [standard]
- 0x0001 = [extended]

Prototype:

```
DWORD PCO_SetSensorFormat (
    WORD wSensor
);
```

Parameters:

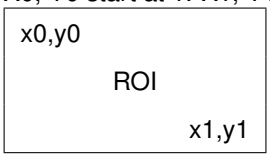
Name	Type	Description
wSensor	WORD	WORD variable which holds the sensor format.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.47 PCO_GetROI

Gets the region of interest of the camera.
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. In case stepping is zero ROI setting other than x0=1, x1=max/bin, y0=1, y1=max/bin it not possible.
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.
X0, Y0 start at 1. X1, Y1 end with max. sensor size.



Prototype:

```
DWORD PCO_GetROI (
    WORD*  RoiX0,
    WORD*  RoiY0,
    WORD*  RoiX1,
    WORD*  RoiY1
);
```

Parameters:

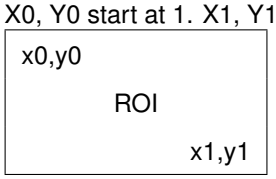
Name	Type	Description
RoiX0	WORD*	Pointer to a WORD variable to receive the x value for the upper left corner.
RoiY0	WORD*	Pointer to a WORD variable to receive the y value for the upper left corner.
RoiX1	WORD*	Pointer to a WORD variable to receive the x value for the lower right corner.
RoiY1	WORD*	Pointer to a WORD variable to receive the y value for the lower right corner.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.48 PCO_SetROI

Sets the region of interest of the camera.
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. In case stepping is zero ROI setting other than x0=1, x1=max/bin, y0=1, y1=max/bin it not possible (max depends on the selected sensor format; bin depends on the current binning settings).
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.
X0, Y0 start at 1. X1, Y1 end with max. sensor size.



Prototype:

```
DWORD PCO_SetROI (
    WORD RoiX0,
    WORD RoiY0,
    WORD RoiX1,
    WORD RoiY1
);
```

Parameters:

Name	Type	Description
RoiX0	WORD	WORD variable to hold the x value for the upper left corner.
RoiY0	WORD	WORD variable to hold the y value for the upper left corner.
RoiX1	WORD	WORD variable to hold the x value for the lower right corner.
RoiY1	WORD	WORD variable to hold the y value for the lower right corner.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.49 PCO_GetBinning

Gets the binning values of the camera.

Prototype:

```
DWORD PCO_GetBinning (
    WORD* BinHorz,
    WORD* BinVert
);
```

Parameters:

Name	Type	Description
BinHorz	WORD*	Pointer to a WORD variable to hold the horizontal binning value.
BinVert	WORD*	Pointer to a WORD variable to hold the vertikal binning value.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.50 PCO_SetBinning

Sets the binning values of the camera.
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.

Prototype:

```
DWORD PCO_SetBinning (
    WORD BinHorz,
    WORD BinVert
);
```

Parameters:

Name	Type	Description
BinHorz	WORD	WORD variable to hold the horizontal binning value.
BinVert	WORD	WORD variable to hold the vertikal binning value.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.51 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).

Prototype:

```
DWORD PCO_GetADCOperation (
    WORD* wADCOperation
);
```

Parameters:

Name	Type	Description
wADCOperation	WORD*	Pointer to a WORD variable to receive the adc operation mode.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.52 PCO_SetADCOperation

Sets the adc operation mode of the camera, if available.
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).
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

Prototype:

```
DWORD PCO_SetADCOperation (
    WORD num
);
```

Parameters:

Name	Type	Description
num	WORD	WORD variable to hold the adc operation mode.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.53 PCO_GetDoubleImageMode

Gets the double image mode of the camera.
Not applicable to all cameras.

Prototype:

```
DWORD PCO_GetDoubleImageMode (
    WORD* wDoubleImage
);
```

Parameters:

Name	Type	Description
wDoubleImage	WORD*	Pointer to a WORD variable to receive the double image mode. • 0x0001 = double image mode ON • 0x0000 = double image mode OFF

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.54 PCO_SetDoubleImageMode

Sets the double image mode of the camera.
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.
Not applicable to all cameras.

Prototype:

```
DWORD PCO_SetDoubleImageMode (
    WORD wDoubleImage
);
```

Parameters:

Name	Type	Description
wDoubleImage	WORD	WORD variable to hold the double image mode. • 0x0001 = double image mode ON • 0x0000 = double image mode OFF

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.55 PCO_GetNoiseFilterMode

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

- 0x0000 = [OFF]
- 0x0001 = [ON]
- 0x0101 = [ON + Hot Pixel correction]

Prototype:

```
DWORD PCO_GetNoiseFilterMode (
    WORD*  wNoiseFilterMode
);
```

Parameters:

Name	Type	Description
wNoiseFilterMode	WORD*	Pointer to a WORD variable to receive the noise filter mode.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.56 PCO_SetNoiseFilterMode

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

- 0x0000 = [OFF]
- 0x0001 = [ON]
- 0x0101 = [ON + Hot Pixel correction]

Prototype:

```
DWORD PCO_SetNoiseFilterMode (
    WORD wNoiseFilterMode
);
```

Parameters:

Name	Type	Description
wNoiseFilterMode	WORD	WORD variable to hold the noise filter mode.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.57 PCO_GetConversionFactor

Get image sensor gain setting.
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

Prototype:

```
DWORD PCO_GetConversionFactor (
    WORD*  wConvFact
);
```

Parameters:

Name	Type	Description
wConvFact	WORD*	Pointer to a WORD variable to receive the conversion factor.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.58 PCO_SetConversionFactor

Set image sensor gain.
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.

Prototype:

```
DWORD PCO_SetConversionFactor (
    WORD wConvFact
);
```

Parameters:

Name	Type	Description
wConvFact	WORD	WORD to set the conversion factor.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.59 PCO_GetIRSensitivity

Gets the IR sensitivity mode of the camera.
This option is only available for special camera models with image sensors that have improved IR sensitivity.

Prototype:

```
DWORD PCO_GetIRSensitivity (
    WORD* wIR
);
```

Parameters:

Name	Type	Description
wIR	WORD*	Pointer to a WORD variable to receive the IR sensitivity mode. • 0x0000 IR sensitivity OFF • 0x0001 IR sensitivity ON

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.60 PCO_SetIRSensitivity

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

Prototype:

```
DWORD PCO_SetIRSensitivity (
    WORD wIR
);
```

Parameters:

Name	Type	Description
wIR	WORD	WORD variable to set the IR sensitivity mode. • 0x0000 IR sensitivity OFF • 0x0001 IR sensitivity ON

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.61 PCO_GetCoolingSetpointTemperature

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

Prototype:

```
DWORD PCO_GetCoolingSetpointTemperature (
    SHORT* sCoolSet
);
```

Parameters:

Name	Type	Description
sCoolSet	SHORT*	Pointer to a SHORT variable to receive the cooling setpoint temperature.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.62 PCO_SetCoolingSetpointTemperature

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

Prototype:

```
DWORD PCO_SetCoolingSetpointTemperature (
    SHORT sCoolSet
);
```

Parameters:

Name	Type	Description
sCoolSet	SHORT	SHORT variable to hold the cooling setpoint temperature in °C units.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.63 PCO_GetCoolingSetpoints

Gets the cooling setpoints of the camera.
This is used when there is no min max range available.

Prototype:

```
DWORD PCO_GetCoolingSetpoints (
    WORD wBlockID,
    SHORT* sSetPoints,
    WORD* wValidSetPoints
);
```

Parameters:

Name	Type	Description
wBlockID	WORD	Number of the block to query (currently 0)
sSetPoints	SHORT*	Pointer to a SHORT array to receive the possible cooling setpoint temperatures.
wValidSetPoints	WORD*	WORD Pointer to set the max number of setpoints to query and to get the valid number of set points inside the camera. In case more than COOLING_SETPOINTS_BLOCKSIZE set points are valid they can be queried by incrementing the wBlockID till wNumSetPoints is reached.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.64 PCO_GetStorageMode

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

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

Prototype:

```
DWORD PCO_GetStorageMode (
    WORD*  wStorageMode
);
```

Parameters:

Name	Type	Description
wStorageMode	WORD*	Pointer to a WORD variable to receive the storage mode. • 0: Recorder • 1: FIFO

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.65 PCO_SetStorageMode

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

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 • 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 ReadImageFrom-Segment command.

Prototype:

```
DWORD PCO_SetStorageMode (
    WORD wStorageMode
);
```

Parameters:

Name	Type	Description
wStorageMode	WORD	WORD variable to hold the storage mode. • 0: Recorder • 1: FIFO

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.66 PCO_GetRecorderSubmode

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

recorder sub mode = [sequence]	recorder sub mode = [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>)

Prototype:

```
DWORD PCO_GetRecorderSubmode (
    WORD*  wRecSubmode
);
```

Parameters:

Name	Type	Description
wRecSubmode	WORD*	Pointer to a WORD variable to receive the recorder sub mode. • 0: Sequence • 1: Ring buffer

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.67 PCO_SetRecorderSubmode

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

recorder sub mode = [sequence]	recorder sub mode = [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>)

Prototype:

```
DWORD PCO_SetRecorderSubmode (
    WORD wRecSubmode
);
```

Parameters:

Name	Type	Description
wRecSubmode	WORD	WORD variable to hold the recorder sub mode. • 0: Sequence • 1: Ring buffer

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.68 PCO_GetAcquireMode

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

- 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).

Prototype:

```
DWORD PCO_GetAcquireMode (
    WORD*  wAcquMode
);
```

Parameters:

Name	Type	Description
wAcquMode	WORD*	Pointer to a WORD variable to receive the acquire mode.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.69 PCO_SetAcquireMode

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

- 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).

Prototype:

```
DWORD PCO_SetAcquireMode (
    WORD wAcquMode
);
```

Parameters:

Name	Type	Description
wAcquMode	WORD	WORD variable to hold the acquire mode.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.70 PCO_GetAcquireModeEx

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

- 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).

Prototype:

```
DWORD PCO_GetAcquireModeEx (
    WORD*  wAcquMode,
    DWORD*  dwNumberImages
);
```

Parameters:

Name	Type	Description
wAcquMode	WORD*	Pointer to a WORD variable to receive the acquire mode.
dwNumberImages	DWORD*	Pointer to a DWORD variable to receive the number of images (for mode sequence).

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.71 PCO_SetAcquireModeEx

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

- 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).

Prototype:

```
DWORD PCO_SetAcquireModeEx (
    WORD wAcquMode,
    DWORD dwNumberImages
);
```

Parameters:

Name	Type	Description
wAcquMode	WORD	WORD variable to hold the acquire mode.
dwNumberImages	DWORD	DWORD variable to hold the number of images (for mode sequence).

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.72 PCO_GetAcqEnblSignalStatus

Get the current status of the <acq enbl> user input (one of the <control in> inputs at the rear of pco.power or the camera). See camera manual for more information.
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.

Prototype:

```
DWORD PCO_GetAcqEnblSignalStatus (
    WORD* wAcquEnableState
);
```

Parameters:

Name	Type	Description
wAcquEnableState	WORD*	Pointer to a WORD variable to receive the acquire enable signal status. • 0x0000 = [FALSE] • 0x0001 = [TRUE]

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.73 PCO_GetCameraRamSize

Gets the ram and page size of the camera.
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 (Xres x Yres) 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.

Prototype:

```
DWORD PCO_GetCameraRamSize (
    DWORD*  dwRamSize,
    WORD*  wPageSize
);
```

Parameters:

Name	Type	Description
dwRamSize	DWORD*	Pointer to a DWORD variable to receive the total camera RAM.
wPageSize	WORD*	Pointer to a DWORD variable to receive the pagesize as a multiple of pixels.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.74 PCO_GetCameraRamSegmentSize

Gets the segment sizes of the camera.
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]!

Prototype:

```
DWORD PCO_GetCameraRamSegmentSize (
    DWORD*  dwRamSegSize
);
```

Parameters:

Name	Type	Description
dwRamSegSize	DWORD*	Pointer to a DWORD array to receive the ramsegmentsizes in pages.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.75 PCO_SetCameraRamSegmentSize

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

Prototype:

```
DWORD PCO_SetCameraRamSegmentSize (
    DWORD*  dwRamSegSize
);
```

Parameters:

Name	Type	Description
dwRamSegSize	DWORD*	Pointer to a DWORD array to receive the ramsegmentsize in pages.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.76 PCO_GetActiveRamSegment

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

Prototype:

```
DWORD PCO_GetActiveRamSegment (
    WORD*  wActSeg
);
```

Parameters:

Name	Type	Description
wActSeg	WORD*	Pointer to a WORD variable to receive the actual segment. (1 - 4)

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.77 PCO_SetActiveRamSegment

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

Prototype:

```
DWORD PCO_SetActiveRamSegment (
    WORD wActSeg
);
```

Parameters:

Name	Type	Description
wActSeg	WORD	WORD variable to hold the actual segment.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.78 PCO_ClearRamSegment

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

Prototype:

```
DWORD PCO_ClearRamSegment (    );
```

Parameters:

No parameter

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.79 PCO_GetNumberOfImagesInSegment

Get the number of valid images within the segment.
This function is not applicable with cameras without internal recorder memory. 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.
In [FIFO buffer] mode the ratio of valid number of images to the maximum number of images is some sort of filling indicator.

Prototype:

```
DWORD PCO_GetNumberOfImagesInSegment (
    WORD wSegment,
    DWORD* dwValid,
    DWORD* dwMax
);
```

Parameters:

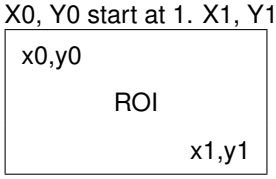
Name	Type	Description
wSegment	WORD	WORD variable that holds the segment to query.
dwValid	DWORD*	Pointer to a DWORD varibale to receive the valid image count.
dwMax	DWORD*	Pointer to a DWORD varibale to receive the max image count which may be saved to this segment.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.80 PCO_GetSegmentImageSettings

Get the image settings for images stored into one of the four segments. This function is not applicable with cameras without internal recorder memory.
Gets the sizes information for one segment.
X0, Y0 start at 1. X1, Y1 end with max. sensor size.



Prototype:

```
DWORD PCO_GetSegmentImageSettings (
    WORD wSegment,
    WORD* wRes_hor,
    WORD* wRes_ver,
    WORD* wBin_x,
    WORD* wBin_y,
    WORD* wRoi_x0,
    WORD* wRoi_y0,
    WORD* wRoi_x1,
    WORD* wRoi_y1
);
```

Parameters:

Name	Type	Description
wSegment	WORD	WORD variable that holds the segment to query.
wRes_hor	WORD*	Pointer to a WORD variable to receive the x resolution of the image in segment
wRes_ver	WORD*	Pointer to a WORD variable to receive the y resolution of the image in segment
wBin_x	WORD*	Pointer to a WORD variable to receive the horizontal binning of the image in segment
wBin_y	WORD*	Pointer to a WORD variable to receive the vertical binning of the image in segment
wRoi_x0	WORD*	Pointer to a WORD variable to receive the left x offset of the image in segment
wRoi_y0	WORD*	Pointer to a WORD variable to receive the upper y offset of the image in segment
wRoi_x1	WORD*	Pointer to a WORD variable to receive the right x offset of the image in segment
wRoi_y1	WORD*	Pointer to a WORD variable to receive the lower y offset of the image in segment

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.81 PCO_ReadImagesFromSegment

Reads the specified images from segment.

Prototype:

```
DWORD PCO_ReadImagesFromSegment (
    WORD wSegment,
    DWORD dwStartImage,
    DWORD dwLastImage
);
```

Parameters:

Name	Type	Description
wSegment	WORD	WORD variable that holds the segment to query.
dwStartImage	DWORD	DWORD variable that holds the first image to receive.
dwLastImage	DWORD	DWORD variable that holds the last image to recieve.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.82 PCO_RequestImage

Requests a single image from the camera.

Prototype:

```
DWORD PCO_RequestImage (    );
```

Parameters:

No parameter

Return value:

Error message, 0 in case of success else less than 0

2.1.2.83 PCO_RepeatImage

Repeats the last image.

Prototype:

```
DWORD PCO_RepeatImage (    );
```

Parameters:

No parameter

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.84 PCO_CancelImage

PCO_CancelImage.
Cancels the image transfer.

Prototype:

```
DWORD PCO_CancelImage (    );
```

Parameters:

No parameter

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.85 PCO_CancelImageTransfer

Cancels the image processing.

Prototype:

```
DWORD PCO_CancelImageTransfer (    );
```

Parameters:

No parameter

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.86 PCO_GetImageTransferMode

Gets the image transfer mode Get the current active transfer mode and the additional parameters for this mode.

Prototype:

```
DWORD PCO_GetImageTransferMode (
    WORD*  wMode,
    WORD*  wImageWidth,
    WORD*  wImageHeight,
    WORD*  wTxWidth,
    WORD*  wTxHeight,
    WORD*  wTxLineWordCnt,
    WORD*  wParam,
    WORD*  wParamLen
);
```

Parameters:

Name	Type	Description
wMode	WORD*	Pointer to WORD variable to receive the image mode. (e.g. full, scaled, cutout etc.)
wImageWidth	WORD*	Pointer to WORD variable to receive the original image width
wImageHeight	WORD*	Pointer to WORD variable to receive the original image height
wTxWidth	WORD*	Pointer to WORD variable to receive the transferred image width
wTxHeight	WORD*	Pointer to WORD variable to receive the transferred image height
wTxLineWordCnt	WORD*	Meaning depends on selected mode
wParam	WORD*	Meaning depends on selected mode
wParamLen	WORD*	Pointer to WORD variable to receive wParam length.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.87 PCO_SetImageTransferMode

Sets the image transfer mode Set the current transfer mode and the additional parameters for this mode This function offers the ability to reduce the amount of image data, which is transferred through the interface. It can be used to offer an quick preview of all images stored in the camera.

Prototype:

```
DWORD PCO_SetImageTransferMode (
    WORD wMode,
    WORD wImageWidth,
    WORD wImageHeight,
    WORD wTxWidth,
    WORD wTxHeight,
    WORD wTxLineWordCnt,
    WORD*  wParam,
    WORD wParamLen
);
```

Parameters:

Name	Type	Description
wMode	WORD	WORD variable to set the image mode. (e.g. full, scaled, cutout etc.)
wImageWidth	WORD	WORD variable to set the original image width
wImageHeight	WORD	WORD variable to set the original image height
wTxWidth	WORD	WORD variable to set the scaled/cutout image width
wTxHeight	WORD	WORD variable to set the scaled/cutout image height
wTxLineWordCnt	WORD	Meaning depends on selected mode
wParam	WORD*	Meaning depends on selected mode
wParamLen	WORD	WORD variable to hold the wParam length.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.88 PCO_GetLookupableInfo

Gets infos about lookup tables in the camera, if available.
Only available with a pco.edge.

Prototype:

```
DWORD PCO_GetLookupableInfo (
    WORD wLUTNum,
    WORD* wNumberOfLuts,
    char* Description,
    WORD wDescLen,
    WORD* wIdentifier,
    BYTE* bInputWidth,
    BYTE* bOutputWidth,
    WORD* wFormat
);
```

Parameters:

Name	Type	Description
wLUTNum	WORD	WORD variable to hold the number of LUT to query.
wNumberOfLuts	WORD*	Pointer to WORD variable to recieve the number of LUTs which can be queried
Description	char*	Pointer to string to recieve the description, e.g. "HD/SDI 12 to 10".
wDescLen	WORD	Pointer to WORD variable to recieve the string length.
wIdentifier	WORD*	Pointer to WORD variable to recieve the loadable LUTs. Range from 0x0001 to 0xFFFF
bInputWidth	BYTE*	Pointer to BYTE variable to recieve the maximum input in bits.
bOutputWidth	BYTE*	Pointer to BYTE variable to recieve the maximum output in bits.
wFormat	WORD*	Pointer to WORD variable to recieve the accepted data structures (see defines!)

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.89 PCO_GetLut

Gets the active lookup table in the camera, if available.
Only available with a pco.edge

Prototype:

```
DWORD PCO_GetLut (
    WORD* Identifier,
    WORD* Parameter
);
```

Parameters:

Name	Type	Description
Identifier	WORD*	Currently active LUT, 0x0000 for no LUT
Parameter	WORD*	Offset: 11 Bit value for fixed offset subtraction before transferring the data via the lookup table

Return value:

Error message, 0 in case of success else less than 0

2.1.2.90 PCO_SetLut

Sets the active lookup table in the camera, if available.
Only available with a pco.edge

Prototype:

```
DWORD PCO_SetLut (
    WORD Identifier,
    WORD Parameter
);
```

Parameters:

Name	Type	Description
Identifier	WORD	define LUT to be activated, 0x0000 for no LUT, see PCO_GetLookupTableInfo() for available LUTs
Parameter	WORD	Offset: 11 Bit value for fixed offset subtraction before transferring the data via the lookup table

Return value:

Error message, 0 in case of success else less than 0

2.1.2.91 PCO_GetBitAlignment

Gets the actual bit alignment of the raw image data.
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.

Prototype:

```
DWORD PCO_GetBitAlignment (
    WORD* align
);
```

Parameters:

Name	Type	Description
align	WORD*	Pointer to a WORD variable to receive the bit alignment.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.92 PCO_SetBitAlignment

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

Prototype:

```
DWORD PCO_SetBitAlignment (
    WORD align
);
```

Parameters:

Name	Type	Description
align	WORD	WORD variable which holds the bit alignment.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.93 PCO_GetTimestampMode

Get mode of the timestamp function.
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)

Prototype:

```
DWORD PCO_GetTimestampMode (
    WORD* mode
);
```

Parameters:

Name	Type	Description
mode	WORD*	Pointer to a WORD variable to receive the time stamp mode. See PCO_SetTimestampMode() for a detailed explanation.

Return value:

Error code

2.1.2.94 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. An 8 by 8 pixel array is used per ASCII digit. The digits are displayed below the BCD coded line.
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) (15...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...99)

Prototype:

```
DWORD PCO_SetTimestampMode (
    WORD mode
);
```

Parameters:

Name	Type	Description
mode	WORD	WORD variable to hold the time stamp mode.

Return value:

Error value or 0 in case of success

2.1.2.95 PCO_GetHotPixelCorrectionMode

Get the Hot Pixel correction mode.
This command is optional and depends on the hardware and firmware. Check the availability according to the camera descriptor (HOT_PIXEL_CORRECTION). Mode:

- 0x0000 = [OFF]
- 0x0001 = [ON]

Prototype:

```
DWORD PCO_GetHotPixelCorrectionMode (
    WORD* wHotPixelCorrectionMode
);
```

Parameters:

Name	Type	Description
wHotPixelCorrectionMode	WORD*	Pointer to a WORD variable to receive the hot pixel correction mode.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.96 PCO_SetHotPixelCorrectionMode

Get the Hot Pixel correction mode.
This command is optional and depends on the hardware and firmware. Check the availability according to the camera descriptor (HOT_PIXEL_CORRECTION). Mode:

- 0x0000 = [OFF]
- 0x0001 = [ON]

Prototype:

```
DWORD PCO_SetHotPixelCorrectionMode (
    WORD wHotPixelCorrectionMode
);
```

Parameters:

Name	Type	Description
wHotPixelCorrectionMode	WORD	Pointer to a WORD variable to receive the hot pixel correction mode.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.97 PCO_GetMetadataMode

Gets the metadata mode.
his 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. When wMode is set to 1, the user is responsible to add further line(s) to the buffers, where the number of lines depends on x-resolution and needed wMetaDataSize.

Prototype:

```
DWORD PCO_GetMetadataMode (
    WORD*  wMode,
    WORD*  wMetadataSize,
    WORD*  wMetadataVersion
);
```

Parameters:

Name	Type	Description
wMode	WORD*	Pointer to a WORD variable receiving the meta data mode. • 0x0000: [OFF] • 0x0001: [ON]
wMetadataSize	WORD*	Pointer to a WORD variable receiving the meta data block size in additional pixels.
wMetadataVersion	WORD*	Pointer to a WORD variable receiving the meta data version information.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.98 PCO_SetMetadataMode

Sets the meta data mode.
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. When wMetaDataMode is set to 1, the user is responsible to add further line(s) to the buffers, where the number of lines depends on x-resolution and needed wMetaDataSize.
This option is only available with pco.dimax

Prototype:

```
DWORD PCO_SetMetadataMode (
    WORD wMode,
    WORD* wMetadataSize,
    WORD* wMetadataVersion
);
```

Parameters:

Name	Type	Description
wMode	WORD	WORD variable to set the meta data mode. • 0x0000: [OFF] • 0x0001: [ON]
wMetadataSize	WORD*	Pointer to a WORD variable receiving the meta data block size in additional pixels.
wMetadataVersion	WORD*	Pointer to a WORD variable receiving the meta data version information.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.99 PCO_GetHWIOSignalCount

Gets the number of available HW signals. Not applicable to all cameras.
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.

Prototype:

```
DWORD PCO_GetHWIOSignalCount (
    WORD* numSignals
);
```

Parameters:

Name	Type	Description
numSignals	WORD*	WORD variable to get the number of signals

Return value:

Error message, 0 in case of success else less than 0

2.1.2.100 PCO_GetHWIOSignalDescriptor

Gets the signal descriptor of the requested hardware IO signal. Not applicable to all cameras.
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.
The output structure has the following parameters:

- wSignalDefinitions: Flags showing signal options:
  - 0x01: Signal can be enabled/disabled
  - 0x02: Signal is a status output
  - 0x10: Signal function 1 has got parameter value
  - 0x20: Signal function 2 has got parameter value
  - 0x40: Signal function 3 has got parameter value
  - 0x80: Signal function 4 has got parameter value
- 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 > ~1 μs)
  - 0x04: Filter can be switched to high (t > ~100ms)

Prototype:

```
DWORD PCO_GetHWIOSignalDescriptor (
    WORD SignalNum,
    SC2_Get_HW_IO_Signal_Descriptor_Response*  SignalDesc
);
```

Parameters:

Name	Type	Description
SignalNum	WORD	WORD variable to query the signal
SignalDesc	SC2_Get_HW_IO_Signal_Descriptor_Response*	Pointer to a SIGNAL_DESC structure to get the signal description

Return value:

Error message, 0 in case of success else less than 0

2.1.2.101 PCO_GetHWIOSignalDescriptor

Gets the signal descriptor of the requested signal number as a string for console output.

Prototype:

```
DWORD PCO_GetHWIOSignalDescriptor (
    WORD SignalNum,
    char*  outbuf,
    int*  size
);
```

Parameters:

Name	Type	Description
SignalNum	WORD	WORD variable to query the signal
outbuf	char*	Pointer to string to hold the signal description.
size	int*	Pointer to size of input string

Return value:

Error message, 0 in case of success else less than 0

2.1.2.102 PCO_GetHWIOSignal

Gets the signal options of the requested signal number.
Gets the settings of the requested hardware IO signal. This functions is not available with all cameras. Actually it is implemented in the pco.dimax.

Prototype:

```
DWORD PCO_GetHWIOSignal (
    WORD SignalNum,
    WORD* Enabled,
    WORD* Type,
    WORD* Polarity,
    WORD* FilterSetting,
    WORD* Selected
);
```

Parameters:

Name	Type	Description
SignalNum	WORD	Index of the signal
Enabled	WORD*	Flags showing enable state of the signal • 0x00: Signal is off • 0x01: Signal is active
Type	WORD*	Flags showing which signal type is selected • 0x01: TTL • 0x02: High Level TTL • 0x04: Contact Mode • 0x08: RS485 differential
Polarity	WORD*	Flags showing which signal polarity is selected • 0x01: High level active • 0x02: Low level active • 0x04: Rising edge active • 0x08: Falling edge active
FilterSetting	WORD*	Flags showing the filter option which is selected • 0x01: Filter can be switched off (t > ~65ns) • 0x02: Filter can be switched to medium (t > ~1 μ) • 0x04: Filter can be switched to high (t > ~100ms)
Selected	WORD*	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'.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.103 PCO_SetHWIOSignal

Sets the signal options of the requested signal number.
Sets the settings of the requested hardware IO signal. This functions is not available with all cameras. Actually it is implemented in the pco.dimax.

Prototype:

```
DWORD PCO_SetHWIOSignal (
    WORD SignalNum,
    WORD Enabled,
    WORD Type,
    WORD Polarity,
    WORD FilterSetting,
    WORD Selected
);
```

Parameters:

Name	Type	Description
SignalNum	WORD	Index of the signal
Enabled	WORD	Flags showing enable state of the signal • 0x00: Signal is off • 0x01: Signal is active
Type	WORD	Flags showing which signal type is selected • 0x01: TTL • 0x02: High Level TTL • 0x04: Contact Mode • 0x08: RS485 differential
Polarity	WORD	Flags showing which signal polarity is selected • 0x01: High level active • 0x02: Low level active • 0x04: Rising edge active • 0x08: Falling edge active
FilterSetting	WORD	Flags showing the filter option which is selected • 0x01: Filter can be switched off (t > ~65ns) • 0x02: Filter can be switched to medium (t > ~1 μs) • 0x04: Filter can be switched to high (t > ~100ms)
Selected	WORD	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'.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.104 PCO_GetPowerDownMode

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

- 0x0000 = [AUTO]
- 0x0001 = [USER]

Prototype:

```
DWORD PCO_GetPowerDownMode (
    WORD*  wPowerDownMode
);
```

Parameters:

Name	Type	Description
wPowerDownMode	WORD*	Pointer to a WORD variable to receive the power down mode.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.105 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).
Mode:

- 0x0000 = [AUTO]
- 0x0001 = [USER]

Prototype:

```
DWORD PCO_SetPowerDownMode (
    WORD wPowerDownMode
);
```

Parameters:

Name	Type	Description
wPowerDownMode	WORD	WORD variable to hold the power down mode.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.106 PCO_GetUserPowerDownTime

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

Prototype:

```
DWORD PCO_GetUserPowerDownTime (
    DWORD*  dwPdnTime
);
```

Parameters:

Name	Type	Description
dwPdnTime	DWORD*	Pointer to a DWORD variable to receive the power down time. Current CCD power down threshold time as multiples of ms (0ms .. 49.7 days)

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.107 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.

Prototype:

```
DWORD PCO_SetUserPowerDownTime (
    DWORD dwPdnTime
);
```

Parameters:

Name	Type	Description
dwPdnTime	DWORD	DWORD variable to set the power down time. Current CCD power down threshold time as multiples of ms (0ms .. 49.7 days)

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.108 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 oautomatic 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 PCO_GetDelayExposureTime() because the corresponding pointers are used as an array of 16 values each.
Timebase:

- 0: ns
- 1: μs;
- 2: ms

Prototype:

```
DWORD PCO_GetDelayExposureTimeTable (
    DWORD*  dwDelay,
    DWORD*  dwExposure,
    WORD*  wTimeBaseDelay,
    WORD*  wTimebaseExposure,
    WORD wCount
);
```

Parameters:

Name	Type	Description
dwDelay	DWORD*	Pointer to a DWORD array to receive the exposure times.
dwExposure	DWORD*	Pointer to a DWORD array to receive the delay times.
wTimeBaseDelay	WORD*	Pointer to a WORD variable to receive the exp. timebase.
wTimebaseExposure	WORD*	Pointer to a WORD variable to receive the del. timebase.
wCount	WORD	Maximum count of delay and exposure pairs, not more than 16 DWORDS.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.109 PCO_SetDelayExposureTimeTable

Sets the exposure and delay time table and the time bases of the camera.
Timebase:

- 0: ns
- 1: μs;
- 2: ms

Prototype:

```
DWORD PCO_SetDelayExposureTimeTable (
    DWORD*  dwDelay,
    DWORD*  dwExposure,
    WORD wTimeBaseDelay,
    WORD wTimebaseExposure,
    WORD wCount
);
```

Parameters:

Name	Type	Description
dwDelay	DWORD*	Pointer to a DWORD array to hold the exposure times.
dwExposure	DWORD*	Pointer to a DWORD array to hold the delay times.
wTimeBaseDelay	WORD	WORD variable to hold the exp. timebase.
wTimebaseExposure	WORD	WORD variable to hold the del. timebase.
wCount	WORD	Count of delay and exposure pairs.

Return value:

Error message, 0 in case of success else less than 0.

See Also

PCO_GetDelayExposureTimeTable

2.1.2.110 PCO_GetModulationMode

Gets the modulation mode and necessary parameters.
The Modulation Mode is an optional feature which is not available for all camera models. See the descriptors of the camera.

- 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 : tp - (te + td) > '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 -20 μs, 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.

Prototype:

```
DWORD PCO_GetModulationMode (
    WORD*  wModulationMode,
    DWORD*  dwPeriodicalTime,
    WORD*  wTimebasePeriodical,
    DWORD*  dwNumberOfExposures,
    LONG*  lMonitorOffset
);
```

Parameters:

Name	Type	Description
wModulationMode	WORD*	Pointer to a WORD variable to receive the modulation mode
dwPeriodicalTime	DWORD*	Pointer to a DWORD variable to receive the periodical time
wTimebasePeriodical	WORD*	Pointer to a WORD variable to receive the time base of pt
dwNumberOfExposures	DWORD*	Pointer to a DWORD variable to receive the number of exposures
lMonitorOffset	LONG*	Pointer to a signed DWORD variable to receive the monitor offset

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.111 PCO_SetModulationMode

Set the modulation mode and necessary parameters.
The Modulation Mode is an optional feature which is not available for all camera models. See the descriptors of the camera.

- 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 : tp - (te + td) > '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 -20 μs, 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.

Prototype:

```
DWORD PCO_SetModulationMode (
    WORD wModulationMode,
    DWORD dwPeriodicalTime,
    WORD wTimebasePeriodical,
    DWORD dwNumberOfExposures,
    LONG lMonitorOffset
);
```

Parameters:

Name	Type	Description
wModulationMode	WORD	WORD variable to hold the modulation mode
dwPeriodicalTime	DWORD	DWORD variable to hold the periodical time
wTimebasePeriodical	WORD	WORD variable to hold the time base of pt
dwNumberOfExposures	DWORD	DWORD variable to hold the number of exposures
lMonitorOffset	LONG	DWORD variable to hold the monitor offset

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.112 PCO_GetCameraSynchMode

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.

Prototype:

```
DWORD PCO_GetCameraSynchMode (
    WORD*  wCameraSynchMode
);
```

Parameters:

Name	Type	Description
wCameraSynchMode	WORD*	Pointer to a WORD variable to receive the synch mode. • 0x0000 = [off] • 0x0001 = [master] • 0x0002 = [slave]

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.113 PCO_SetCameraSynchMode

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.

Prototype:

```
DWORD PCO_SetCameraSynchMode (
    WORD wCameraSynchMode
);
```

Parameters:

Name	Type	Description
wCameraSynchMode	WORD	WORD variable to set the synch mode. • 0x0000 = [off] • 0x0001 = [master] • 0x0002 = [slave]

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.114 PCO_GetFastTimingMode

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 ~75 μS to 3.5 μS.

Prototype:

```
DWORD PCO_GetFastTimingMode (
    WORD*  wFastTimingMode
);
```

Parameters:

Name	Type	Description
wFastTimingMode	WORD*	Pointer to a WORD variable to receive the fast timing mode. • 0x0000 = [OFF] • 0x0001 = [ON]

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.115 PCO_SetFastTimingMode

Set 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 ~75 μS to 3.5 μS.

Prototype:

```
DWORD PCO_SetFastTimingMode (
    WORD wFastTimingMode
);
```

Parameters:

Name	Type	Description
wFastTimingMode	WORD	WORD variable to set the fast timing mode. • 0x0000 = [OFF] • 0x0001 = [ON]

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.116 PCO_GetFirmwareVersion

Gets the firmware versions as string for console output.
If the string size is not large enough, not all firmware strings will be shown.

Prototype:

```
DWORD PCO_GetFirmwareVersion (
    char*  outbuf,
    int*  size
);
```

Parameters:

Name	Type	Description
outbuf	char*	Pointer to string to hold the firmware version output.
size	int*	Pointer int that holds the size of the string.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.117 PCO_GetHardwareVersion

Gets the hardware versions as string for console output.
If the string size is not large enough, not all hardware strings will be shown.

Prototype:

```
DWORD PCO_GetHardwareVersion (
    char*  outbuf,
    int*  size
);
```

Parameters:

Name	Type	Description
outbuf	char*	Pointer to string to hold the firmware version output.
size	int*	Pointer to size of the string.

Return value:

Error message, 0 in case of success else less than 0

2.1.2.118 PCO_GetFirmwareVersion

Gets the firmware versions of first 10 devices in the camera.
Get Firmware version of first 10 devices and the number of installed devices If number of devices exceeds 10 PCO_GetFirmwareVersionExt must be called with increased block counter

Prototype:

```
DWORD PCO_GetFirmwareVersion (
    SC2_Firmware_Versions_Response*  response
);
```

Parameters:

Name	Type	Description
response	SC2_Firmware_Versions_Response*	Pointer to a firmware version structure

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.119 PCO_GetFirmwareVersionExt

Gets the firmware versions of the devices in the camera.
not applicable to all cameras.

Prototype:

```
DWORD PCO_GetFirmwareVersionExt (
    BYTE bNum,
    SC2_Firmware_Versions_Response*  response
);
```

Parameters:

Name	Type	Description
bNum	BYTE	Pointer to a number, which contains the actual blocknumber
response	SC2_Firmware_Versions_Response*	Pointer to a firmware version structure

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.120 PCO_GetIEEE1394InterfaceParams

Gets the IEEE1394 interface parameters.
Gets the FireWire transfer parameters.

Prototype:

```
DWORD PCO_GetIEEE1394InterfaceParams (
    WORD*  wMasterNode,
    WORD*  wIsochChannel,
    WORD*  wIsochPacketLen,
    WORD*  wIsochPacketNum
);
```

Parameters:

Name	Type	Description
wMasterNode	WORD*	Pointer to WORD variable to receive the master node address.
wIsochChannel	WORD*	Pointer to WORD variable to recieve the used ISO channel.
wIsochPacketLen	WORD*	Pointer to WORD variable to receive the ISO packet length.
wIsochPacketNum	WORD*	Pointer to WORD variable to receive the ISO packet count.

Return value:

Error message, 0 in case of success else less than 0.

See Also

PCO_SetIEEE1394InterfaceParams

2.1.2.121 PCO_SetIEEE1394InterfaceParams

PCO_SetIEEE1394InterfaceParams Sets the IEEE1394 interface parameters.
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: Channel numbers are 32-bit encoded and the highest bit equals the lowest channel. (e.g. 0x80000000 = channel 0).
- 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 (Open_Grabber() does it automatically), but in case the user wants to transfer images from more than one camera, the iso channel must be unique for each camera. Only one bit may be set at a time.
In case the user wants to establish a connection, this has to be the first command sent or the camera won't know how to respond to commands.

Prototype:

```
DWORD PCO_SetIEEE1394InterfaceParams (
    WORD wMasterNode,
    WORD wIsochChannel,
    WORD wIsochPacketLen,
    WORD wIsochPacketNum
);
```

Parameters:

Name	Type	Description
wMasterNode	WORD	WORD variable to set the master node address.
wIsochChannel	WORD	WORD variable to set the ISO channel.
wIsochPacketLen	WORD	WORD variable to set the ISO packet length.
wIsochPacketNum	WORD	WORD variable to set the ISO packet count.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.122 PCO_GetIEEE1394ISOByteorder

Gets the IEEE1394 byte order.

Prototype:

```
DWORD PCO_GetIEEE1394ISOByteorder (
    WORD* wMode
);
```

Parameters:

Name	Type	Description
wMode	WORD*	Pointer to WORD variable to receive the IEEE1394 byte order.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.123 PCO_SetIEEE1394ISOByteorder

Sets the IEEE1394 byte order.

Prototype:

```
DWORD PCO_SetIEEE1394ISOByteorder (
    WORD wMode
);
```

Parameters:

Name	Type	Description
wMode	WORD	WORD variable to set the IEEE1394 byte order.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.124 PCO_GetInterfaceOutputFormat

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.

Prototype:

```
DWORD PCO_GetInterfaceOutputFormat (
    WORD wInterface,
    WORD* wFormat
);
```

Parameters:

Name	Type	Description
wInterface	WORD	WORD variable to get the desired interface. • 0: reserved • 1: HD/SDI • 2: DVI
wFormat	WORD*	Pointer to WORD variable to get the interface format • 0: Output is disabled • 1: HD/SDI, 1080p25, RGB • 2: HD/SDI, 1080p25, arbitrary raw mode

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.125 PCO_SetInterfaceOutputFormat

Sets the actual interface output format.
This option is only available with pco.dimax and 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.

Prototype:

```
DWORD PCO_SetInterfaceOutputFormat (
    WORD wInterface,
    WORD wFormat
);
```

Parameters:

Name	Type	Description
wInterface	WORD	WORD variable to set the desired interface. • 0: reserved • 1: HD/SDI • 4: DVI
wFormat	WORD	WORD variable to set the interface format • 0: Output is disabled • 1: HD/SDI, 1080p25, RGB • 2: HD/SDI, 1080p25, arbitrary raw mode

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.126 PCO_GetInterfaceStatus

Get interface status messages. This option is only available with pco.dimax and HD/SDI interface.

Prototype:

```
DWORD PCO_GetInterfaceStatus (
    WORD wInterface,
    DWORD* dwWarnings,
    DWORD* dwErrors,
    DWORD* dwStatus
);
```

Parameters:

Name	Type	Description
wInterface	WORD	WORD variable holding the interface to be queried.
dwWarnings	DWORD*	Pointer to WORD variable to receive the warnings.
dwErrors	DWORD*	Pointer to WORD variable to receive the errors.
dwStatus	DWORD*	Pointer to WORD variable to receive the status.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.127 PCO_PlayImagesFromSegment

Plays the images recorded to the camera RAM.
This option is only available with pco.dimax and HD/SDI interface. **Note:** Command is only valid, if **storage mode** is set to [recorder] and recording to the camera RAM segment is stopped!
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.

Prototype:

```
DWORD PCO_PlayImagesFromSegment (
    WORD wSegment,
    WORD wInterface,
    WORD wMode,
    WORD wSpeed,
    DWORD dwRangeLow,
    DWORD dwRangeHigh,
    DWORD dwStartPos
);
```

Parameters:

Name	Type	Description
wSegment	WORD	WORD variable with the segment to read from
wInterface	WORD	WORD variable to set the interface (0x0001 for HDSDI)
wMode	WORD	WORD variable to set the play mode • 0: Stop play, • 1: Fast forward (step 'wSpeed' images), • 2: Fast rewind (step 'wSpeed' images), • 3: Slow forward (show each image 'wSpeed'-times) • 4: Slow rewind (show each image 'wSpeed'-times) • Additional flags: 0x0100-> - 0: Repeat last image – 1: Repeat sequence
wSpeed	WORD	WORD variable to set the stepping or repeat count
dwRangeLow	DWORD	Lowest image number to be played
dwRangeHigh	DWORD	Highest image number to be played
dwStartPos	DWORD	Set position to image number #, -1: unchanged

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.128 PCO_GetPlayPosition

The "Get Play Position" command requests at which position the play pointer of the currently started sequence is. 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.
Note: Due to time necessary for communication and processing the command, the actual pointer may be 1 or 2 images ahead at the time, when the response is sent completely.

Prototype:

```
DWORD PCO_GetPlayPosition (
    WORD*  wStatus,
    DWORD*  dwPosition
);
```

Parameters:

Name	Type	Description
wStatus	WORD*	WORD variable to get the status of image play state machine. • 0: no play is active, or play has already stopped • 1: play is active, position is valid
dwPosition,:	DWORD*	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.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.129 PCO_GetColorSettings

Gets the color convert settings inside the camera.
This option is only available with pco.dimax and HD-SDI interface.

Prototype:

```
DWORD PCO_GetColorSettings (
    SC2_Get_Color_Settings_Response*  ColSetResp
);
```

Parameters:

Name	Type	Description
ColSetResp	SC2_Get_Color_Settings_Response*	Pointer to a SC2_Set_Color_Settings structure to receive the color set data.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.130 PCO_SetColorSettings

Sets the color convert settings inside the camera.
This option is only available with pco.dimax and HD-SDI interface.

Prototype:

```
DWORD PCO_SetColorSettings (
    SC2_Set_Color_Settings*  SetColSet
);
```

Parameters:

Name	Type	Description
SetColSet	SC2_Set_Color_Settings*	Pointer to a SC2_Set_Color_Settings structure to set the color set data.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.131 PCO_DoWhiteBalance

Starts a white balancing calculation.
This option is only available with pco.dimax and HD-SDI interface.

Prototype:

```
DWORD PCO_DoWhiteBalance (
    WORD wMode
);
```

Parameters:

Name	Type	Description
wMode	WORD	WORD variable to set the meta data mode. Set to 1.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.132 PCO_GetWhiteBalanceStatus

Gets the white balancing status.
This option is only available with pco.dimax and HD-SDI interface.

Prototype:

```
DWORD PCO_GetWhiteBalanceStatus (
    WORD*  wStatus,
    WORD*  wColorTemp,
    SHORT*  sTint
);
```

Parameters:

Name	Type	Description
wStatus	WORD*	Pointer to WORD variable to receive the status.
wColorTemp	WORD*	Pointer to WORD variable to recieve the color temperature.
sTint	SHORT*	Pointer to SHORT variable to recieve the tint.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.133 PCO_InitiateSelftestProcedure

Starts a self test procedure.
See PCO_GetCameraHealthStatus().

Prototype:

```
DWORD PCO_InitiateSelftestProcedure (
    DWORD*  dwWarn,
    DWORD*  dwErr
);
```

Parameters:

Name	Type	Description
dwWarn	DWORD*	Pointer to DWORD variable to receive warnings.
dwErr	DWORD*	Pointer to DWORD variable to receive errors.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.134 PCO_WriteHotPixelList

Writes a hot pixel list to the camera.
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 hot pixel list inside the camera, please call PCO_ReadHotPixelList() first, 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 hot pixel correction!
The x and y coordinates have to be consistent, that means corresponding coordinate pairs must have the same index!

Prototype:

```
DWORD PCO_WriteHotPixelList (
    WORD wListNo,
    WORD wNumValid,
    WORD* wHotPixX,
    WORD* wHotPixY
);
```

Parameters:

Name	Type	Description
wListNo	WORD	WORD variable which holds the number of the list (zero based).
wNumValid	WORD	WORD variable which holds the number of valid members
wHotPixX	WORD*	WORD array which holds the x coordinates of a hotpixel list
wHotPixY	WORD*	WORD array which holds the y coordinates of a hotpixel list

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.135 PCO_ReadHotPixelList

Reads a hot pixel list from the camera.
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 hot pixel list inside the camera, please call this command first, 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 hot pixel correction!

Prototype:

```
DWORD PCO_ReadHotPixelList (
    WORD wListNo,
    WORD wArraySize,
    WORD* wNumValid,
    WORD* wNumMax,
    WORD* wHotPixX,
    WORD* wHotPixY
);
```

Parameters:

Name	Type	Description
wListNo	WORD	WORD variable which holds the number of the list (zero based).
wArraySize	WORD	WORD variable which holds the number of members, which can be transferred to the list
wNumValid	WORD*	Pointer to a WORD variable to receive the number of valid hotpixel.
wNumMax	WORD*	Pointer to a WORD variable to receive the max. possible number of hotpixel.
wHotPixX	WORD*	WORD array which gets the x coordinates of a hotpixel list This ptr can be set to ZERO if only the valid and max number have to be read.
wHotPixY	WORD*	WORD array which gets the y coordinates of a hotpixel list This ptr can be set to ZERO if only the valid and max number have to be read.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.136 PCO_ClearHotPixelList

Clears a hot pixel list in the camera.
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 hot pixel list inside the camera, 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 hot pixel correction! This command clears the list addressed completely. Use with caution!
Set the following parameter:

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

Prototype:

```
DWORD PCO_ClearHotPixelList (
    WORD wListNo,
    DWORD dwMagic1,
    DWORD dwMagic2
);
```

Parameters:

Name	Type	Description
wListNo	WORD	WORD variable which holds the number of the list (zero based).
dwMagic1	DWORD	DWORD variable which holds the unlock code 1.
dwMagic2	DWORD	DWORD variable which holds the unlock code 2.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.137 PCO_ClearHotPixelList

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Prototype:

```
DWORD PCO_ClearHotPixelList (
    WORD wListNo
);
```

Parameters:

Name	Type	Description
wListNo	WORD	

Return value:

2.1.2.138 PCO_LoadLookuptable

Loads a lookup table to the camera, if available.
Only available with a pco.edge.

Prototype:

```
DWORD PCO_LoadLookuptable (
    WORD wIdentifier,
    WORD wPacketNum,
    WORD wFormat,
    WORD wLength,
    BYTE*  bData
);
```

Parameters:

Name	Type	Description
wIdentifier	WORD	WORD variable to hold the LUT to be loaded
wPacketNum	WORD	WORD variable to hold the packet number to load the LUT in several steps.
wFormat	WORD	WORD variable to hold the data structure in bData (see defnines)
wLength	WORD	WORD variable to hold the length of the data structure
bData	BYTE*	Pointer to BYTE to hold the actual data

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.139 PCO_ReadLookuptable

Reads a lookup table from the camera, if available.
Only available with a pco.edge.

Prototype:

```
DWORD PCO_ReadLookuptable (
    WORD wIdentifier,
    WORD wPacketNum,
    WORD*  wFormat,
    WORD*  wLength,
    BYTE*  bData,
    WORD buflen
);
```

Parameters:

Name	Type	Description
wIdentifier	WORD	WORD variable to hold the LUT to be read
wPacketNum	WORD	WORD variable to hold the packet number to read the LUT in several steps.
wFormat	WORD*	Pointer to WORD variable to receive the data structure in bData (see defnines)
wLength	WORD*	Pointer to WORD variable to receive the length of the data structure
bData	BYTE*	Pointer to BYTE array to receive the data
buflen	WORD	WORD variable to hold the length of the BYTE array (bData)

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.140 PCO_GetColorCorrectionMatrix

Gets the color multiplier matrix to normalize the color values of a color camera to 6500k.
Only available with a dimax

Prototype:

```
DWORD PCO_GetColorCorrectionMatrix (
    char*  szCCM,
    WORD*  len
);
```

Parameters:

Name	Type	Description
szCCM	char*	Pointer to a string to hold the values.
len	WORD*	Pointer to a WORD that holds the string length.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.141 PCO_GetBayerMultiplier

Requests the Bayer multipliers.
The Bayer multipliers are used by cameras with color sensor in order to compensate the color response of the sensor and the optical setup. Thus when exposed to white light the R Gr Gb B pixels will ideally show the same amplitude. This option is only available with a pco.dimax

Prototype:

```
DWORD PCO_GetBayerMultiplier (
    WORD*  wMode,
    WORD*  wMul
);
```

Parameters:

Name	Type	Description
wMode	WORD*	• 0x0001: The returned values are changed, but not yet saved. • 0x0002: The returned values are saved.
wMul	WORD*	Pointer to an array of four WORD; Red/GreenRed/GreenBlue/Blue Multiplier: Number from 0 to 3999, where 1000 corresponds to multiplier of 1.0 (leave values unchanged). Element 0 is the same as in the color descriptor. See wColorPatternDESC in PCO_Description.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.142 PCO_SetBayerMultiplier

Sets the Bayer multipliers.
The Bayer multipliers are used by cameras with color sensor in order to compensate the color response of the sensor and the optical setup. Thus when exposed to white light the R Gr Gb B pixels will ideally show the same amplitude. This option is only available with a pco.dimax

Prototype:

```
DWORD PCO_SetBayerMultiplier (
    WORD wMode,
    WORD*  wMul
);
```

Parameters:

Name	Type	Description
wMode	WORD	• 0x0001: Set new values immediately but do not save. • 0x0002: Save values and set immediately.
wMul	WORD*	Pointer to an array of four WORD; Red-GreenRed-GreenBlue-Blue Multiplier: Number from 0 to 3999, where 1000 corresponds to multiplier of 1.0 (leave values unchanged). Element 0 is the same as in the color descriptor. See wColorPatternDESC in PCO_Description.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.143 PCO_GetFanControlStatus

Get parameters for fan operation control.
Only supported from cameras which have GENERALCAPS1_FAN_CONTROL Bit set in the descriptor. Get possible Parameter values which can be set with the PCO_SetFanControlStatus() function in UserMode.
Mode can be set to

- 0x0000 = Automatic: Fan speed is controlled depending on camera temperature.
- 0x0001 = UserMode:

Prototype:

```
DWORD PCO_GetFanControlStatus (
    WORD*  wFanMode,
    WORD*  wFanMin,
    WORD*  wFanMax,
    WORD*  wStepSize,
    WORD*  wSetValue,
    WORD*  wActualValue
);
```

Parameters:

Name	Type	Description
wFanMode	WORD*	Pointer to WORD variable to get the actual fan operation control mode
wFanMin	WORD*	Pointer to WORD variable which holds the minimal setting for fan speed
wFanMax	WORD*	Pointer to WORD variable which holds the maximal setting for fan speed
wStepSize	WORD*	Pointer to WORD variable which holds the steps for fan speed setting
wSetValue	WORD*	Pointer to WORD variable to get the fan speed setting, from last set call
wActualValue	WORD*	Pointer to WORD variable to get the actual fan speed

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.144 PCO_SetFanControlStatus

Set parameters to control Fan operation.
Only supported from cameras which have GENERALCAPS1_FAN_CONTROL Bit set in the descriptor. Mode can be set to

- 0x0000 = Automatic: Fan speed is controlled depending on camera temperature.
- 0x0001 = User:

Prototype:

```
DWORD PCO_SetFanControlStatus (
    WORD wFanMode,
    WORD wSetValue
);
```

Parameters:

Name	Type	Description
wFanMode	WORD	switch fan control mode
wSetValue	WORD	

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.145 PCO_GetHWLEDSignal

Get operating state of the backpanel LEDs.

Prototype:

```
DWORD PCO_GetHWLEDSignal (
    DWORD* dwParameter
);
```

Parameters:

Name	Type	Description
dwParameter	DWORD*	DWORD variable to get operating state of LED's. • 0x00000000 = [OFF] • 0xFFFFFFFF = [ON]

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.146 PCO_SetHWLEDSignal

Switch the operating state of the back panel LEDs.
To ensure, that light of the camera LED's doesn't spoil lowlight level exposures, the state of the LED can be changed. With this command LED's could be switched OFF and on again to get minimal status information of the camera.

Prototype:

```
DWORD PCO_SetHWLEDSignal (
    DWORD dwParameter
);
```

Parameters:

Name	Type	Description
dwParameter	DWORD	DWORD variable to set the Led Signals. • 0x00000000 = [OFF] • 0xFFFFFFFF = [ON]

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.147 PCO_GetDSNUAdjustMode

Gets the camera internal DSNU adjustment mode.
Only available with a dimax.

Prototype:

```
DWORD PCO_GetDSNUAdjustMode (
    WORD* wMode
);
```

Parameters:

Name	Type	Description
wMode	WORD*	• 0: no DSNU correction. • 1: automatic DSNU correction. • 2: manual DSNU correction.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.148 PCO_SetDSNUAdjustMode

Sets the camera internal DSNU adjustment mode.
Only available with a dimax.

Prototype:

```
DWORD PCO_SetDSNUAdjustMode (
    WORD wMode
);
```

Parameters:

Name	Type	Description
wMode	WORD	• 0: no DSNU correction. • 1: automatic DSNU correction. • 2: manual DSNU correction.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.149 PCO_InitDSNUAdjustment

Starts the camera internal DSNU adjustment in case it is set to manual.
Only available with a dimax.

Prototype:

```
DWORD PCO_InitDSNUAdjustment (
    WORD*  wMode
);
```

Parameters:

Name	Type	Description
wMode	WORD*	• 0: no DSNU correction. • 1: automatic DSNU correction. • 2: manual DSNU correction.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.150 PCO_GetCDIMode

Gets the correlated double image mode of the camera.
Only available with a dimax.

Prototype:

```
DWORD PCO_GetCDIMode (
    WORD*  wMode
);
```

Parameters:

Name	Type	Description
wMode	WORD*	Pointer to a WORD variable to receive the correlated double image mode.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.151 PCO_SetCDIMode

Sets the correlated double image mode of the camera.
Only available with a dimax.

Prototype:

```
DWORD PCO_SetCDIMode (
    WORD wMode
);
```

Parameters:

Name	Type	Description
wMode	WORD	WORD variable to set the correlated double image mode.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.152 PCO_GetPowersaveMode

Gets the camera power save mode.
Not applicable to all cameras.

Prototype:

```
DWORD PCO_GetPowersaveMode (
    WORD*  wMode,
    WORD*  wDelayMinutes
);
```

Parameters:

Name	Type	Description
wMode	WORD*	WORD pointer to get the actual power save mode. (0-off,default; 1-on)
wDelayMinutes	WORD*	WORD pointer to get the delay until the camera enters power save mode after main power loss. The actual switching delay is between wDelayMinutes and wDelayMinutes + 1. Possible range is 1 - 60.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.153 PCO_SetPowersaveMode

Sets the camera power save mode.
Not applicable to all cameras.

Prototype:

```
DWORD PCO_SetPowersaveMode (
    WORD wMode,
    WORD wDelayMinutes
);
```

Parameters:

Name	Type	Description
wMode	WORD	WORD to set the actual power save mode. (0-off,default; 1-on)
wDelayMinutes	WORD	WORD to set the delay until the camera enters power save mode after main power loss. The actual switching delay is between wDelayMinutes and wDelayMinutes + 1. Possible range is 1 - 60.

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.154 PCO_GetBatteryStatus

Gets the camera battery status.
Not applicable to all cameras.

Prototype:

```
DWORD PCO_GetBatteryStatus (
    WORD*  wBatteryType,
    WORD*  wBatteryLevel,
    WORD*  wPowerStatus
);
```

Parameters:

Name	Type	Description
wBatteryType	WORD*	WORD pointer to get the battery type. • 0x0000 = no battery mounted • 0x0001 = nickel metal hydride type • 0x0002 = lithium ion type • 0x0003 = lithium iron phosphate type • 0xFFFF = unknown battery type
wBatteryLevel	WORD*	WORD pointer to get the battery level in percent.
wPowerStatus	WORD*	WORD pointer to get the power status. • 0x0001 = power supply is available • 0x0002 = battery mounted and detected • 0x0004 = battery is charged

Return value:

Error message, 0 in case of success else less than 0.

2.1.2.155 gettimeouts

Gets the current timeouts for images and telegrams.

Prototype:

```
void gettimeouts (
    PCO_SC2_TIMEOUTS*  strTimeouts
);
```

Parameters:

Name	Type	Description
strTimeouts	PCO_SC2_TIMEOUTS*	Pointer to a PCO_SC2_TIMEOUTS structure

Return value:

2.1.2.156 Set_Timeouts

Sets the timeouts of the camera communication class.

Prototype:

```
void Set_Timeouts (
    void*  timetable,
    DWORD length
);
```

Parameters:

Name	Type	Description
timetable	void*	Pointer to a DWORD array. First DWORD is the command timeout, second the image timeout, third the transfer timeout.
length	DWORD	Length of the array in bytes, a maximum of 12 bytes are used.

Return value:

2.1.2.157 Sleep_ms

Common sleep function.

Prototype:

```
void Sleep_ms (
    DWORD time_ms
);
```

Parameters:

Name	Type	Description
time_ms	DWORD	Time to sleep in ms

Return value:

None

2.2 CPco_com_usb Class Reference

Inherits CPco_com.

Public Member Functions

- DWORD Open_Cam (DWORD num)
- DWORD Open_Cam_Ext (DWORD num, SC2_OpenStruct* open)
- DWORD Close_Cam ()
- bool IsOpen ()
- DWORD Control_Command (void* buf_in, DWORD size_in, void* buf_out, DWORD size_out)
- bool getCoding ()

Protected Attributes

- sem_t sMutex
- bool gl_Coding

2.2.1 Detailed Description

The CPco_com_usb class, extends CPco_com.

This is the communication class to exchange messages (telegrams) with a pco.camera.

2.2.2 Member Function Documentation

2.2.2.1 Open_Cam

Prototype:

```
DWORD Open_Cam (
    DWORD num
);
[virtual]
```

Parameters:

Name	Type	Description
num	DWORD	

Return value:

Error or 0 on success

Implements CPco_com.

2.2.2.2 Open_Cam_Ext

Prototype:

```
DWORD Open_Cam_Ext (
    DWORD num,
    SC2_OpenStruct*  open
);
```

Parameters:

Name	Type	Description
num	DWORD	
open	SC2_OpenStruct*	

Return value:

Error or 0 on success

2.2.2.3 Close_Cam

Closes a connection to a pco camera.

Prototype:

```
DWORD Close_Cam (    );
[virtual]
```

Parameters:

No parameter

Return value:

Error or 0 on success

Reimplemented from CPco_com.

2.2.2.4 IsOpen

Returns connection status.

Prototype:

```
bool IsOpen (    );
```

Parameters:

No parameter

Return value:

1 if a connection is open, 0 else

2.2.2.5 Control_Command

Prototype:

```
DWORD Control_Command (
    void*  buf_in,
    DWORD size_in,
    void*  buf_out,
    DWORD size_out
);
[virtual]
```

Parameters:

Name	Type	Description
buf_in	void*	desc
size_in	DWORD	desc
buf_out	void*	desc
size_out	DWORD	

Return value:

Error or 0 on success

Implements CPco_com.

2.2.2.6 getCoding

Returns the value of the gl_Coding member variable.

Prototype:

```
bool getCoding (    );
```

Parameters:

No parameter

Return value:

1 if pixelfly coding is active, 0 else

2.3 CPco_grab_usb Class Reference

Public Member Functions

- CPco_grab_usb (CPco_com_usb* camera=NULL)
- DWORD Open_Grabber (int board)
- DWORD Open_Grabber (int board, int initmode)
- DWORD Close_Grabber ()
- bool IsOpen ()
- void SetLog (CPco_Log* elog)

Image Acquire Functions

These functions are used to acquire images from the camera. Synchronous functions do not return until grab and data decoding is finished Asynchronous functions do not return until grab is finished

- DWORD Acquire_Image (void* adr)
- DWORD Acquire_Image (void* adr, int timeout)
- DWORD Get_Image (WORD Segment, DWORD ImageNr, void* adr)
- DWORD Acquire_Image_Async (void* adr)
- DWORD Acquire_Image_Async (void* adr, int timeout)
- DWORD Acquire_Image_Async_wait (void* adr)
- DWORD Acquire_Image_Async_wait (void* adr, int timeout)
- DWORD Acquire_Image_Async (void* adr, int timeout, BOOL waitforimage)

Class Control Functions

These functions are used to control and retrieve some internal variables of the class.

- DWORD Set_Grabber_Timeout (int timeout)
- DWORD Get_Grabber_Timeout (int* timeout)
- DWORD PostArm (int userset=0)
- DWORD Set_Grabber_Size (int width, int height, int bitpix)
- DWORD Set_Grabber_Size (int width, int height)
- DWORD Get_actual_size (unsigned int* width, unsigned int* height, unsigned int* bitpix)
- DWORD Set_DataFormat (DWORD dataformat)
- DWORD Get_DataFormat ()
- void SetBitAlignment (int align)
- int Get_Height ()
- int Get_Width ()
- void Set_Async_Packet_Size (int size)
- int Get_Async_Packet_Size ()

2.3.1 Detailed Description

The CPco_grab_usb class.

This is the (virtual) grabber class for USB. It it responsible for image transfers by allocating and submitting USB transfers. The images are split up into smaller packets for performance reasons and/or other limitations. Libusb-1.0 is used for the transfers. The CPco_grab_usb class provides synchronous and asynchronous acquire functions for image transfers.

Some cameras use special coding for the transferred image data, which is queried from the camera during Open_Grabber call. Decoding of the image data is done during the USB transfers in a special thread, which is setup for each image. Therefore, when using the asynchronous acquire functions for image transfers, only the last transfer should call one of the Asynch_wait functions. All synchronous functions wait until decoding is finished.

2.3.2 Constructor & Destructor Documentation

2.3.2.1 CPco_grab_usb

Constructor for the class. It is possible (though not very useful) to create a class object without passing it a camera class object pointer as parameter.

Prototype:

```
CPco_grab_usb (
    CPco_com_usb* camera = NULL
);
```

Parameters:

Name	Type	Description
camera	**CPco_com_usb***	A CPco_com_usb object with a previously opened pco.camera

Return value:

2.3.3 Member Function Documentation

2.3.3.1 Open_Grabber

Opens the grabber and retrieves the neccessary variables from the camera object.

Prototype:

```
DWORD Open_Grabber (
    int board
);
```

Parameters:

Name	Type	Description
board	int	Set to zero if there is only one camera connected. Open_Cam() on the appropriate class object **must** be called first or this will fail!

Return value:

Errorcode or 0 in case of success.

2.3.3.2 Open_Grabber

Opens the grabber and retrieves the neccessary variables from the camera object.

Prototype:

```
DWORD Open_Grabber (
    int board,
    int initmode
);
```

Parameters:

Name	Type	Description
board	int	Set to zero if there is only one camera connected. Open_Cam() on the appropriate class object **must** be called first or this will fail!
initmode	int	not used

Return value:

Errorcode or 0 in case of success.

2.3.3.3 Close_Grabber

Closes the grabber. This should be done before calling Close_Cam().

Prototype:

```
DWORD Close_Grabber (    );
```

Parameters:

No parameter

Return value:

Errorcode or 0 in case of success.

2.3.3.4 IsOpen

Check if grabber is opened.

Prototype:

```
BOOL IsOpen (    );
```

Parameters:

No parameter

Return value:

openstatus
- true grabber is opened
- false grabber is closed

2.3.3.5 SetLog

Sets the logging behaviour for the grabber class.
If this function is not called no logging is performed. Logging might be useful to follow the program flow of the application.

Prototype:

```
void SetLog (
    CPco_Log*  elog
);
```

Parameters:

Name	Type	Description
elog	CPco_Log*	CPco_Log∗ Pointer to a CPco_Log logging class object

Return value:

2.3.3.6 Acquire_Image

A simple image acquisition function.
This function is synchronous. It does not return until either an image is grabbed and completely transferred to the given address or the timeout has been reached.
Start_Acquire() is called, when Grabber is not already started. Then the function is waiting until a single image is in the grabber buffer. Stop_Acquire() is called when grabber was not started.
Internal timeout setting is used, see Set_Grabber_Timeout.

Prototype:

```
DWORD Acquire_Image (
    void*  adr
);
```

Parameters:

Name	Type	Description
adr	void*	Pointer to address where the image gets stored

Return value:

Errorcode or 0 in case of success.

2.3.3.7 Acquire_Image

A simple image acquisition function. This function is synchronous. It does not return until either an image is grabbed and completely transferred to the given address or the timeout has been reached.
Start_Acquire() is called, when Grabber is not already started. Then the function is waiting until a single image is in the grabber buffer. Stop_Acquire is called when grabber was not started.
Custom timeout setting is used.

Prototype:

```
DWORD Acquire_Image (
    void*  adr,
    int timeout
);
```

Parameters:

Name	Type	Description
adr	void*	Pointer to address where the image gets stored
timeout	int	time to wait for image in ms

Return value:

Errorcode or 0 in case of success.

2.3.3.8 Get_Image

Transfers an image from the recorder buffer of the camera.
This function is synchronous. It does not return until either an image is grabbed and completely transferred to the given address or the timeout has been reached.
Start_Acquire() is called, when Grabber is not already started. Then the function does request an image from the camera and is waiting until this image is in the grabber buffer. Stop_Acquire is called when grabber was not started.
Internal timeout setting is used, see Set_Grabber_Timeout.

Prototype:

```
DWORD Get_Image (
    WORD Segment,
    DWORD ImageNr,
    void*  adr
);
```

Parameters:

Name	Type	Description
Segment	WORD	select segment of recorder buffer
ImageNr	DWORD	select image in recorder buffer
adr	void*	Pointer to address where the image gets stored

Return value:

Error or 0 in case of success

2.3.3.9 Acquire_Image_Async

Setup an image transfer. (Non-Blocking)
Prepare all neccessary data structures to start a (image)-data transfer from the camera over the usb-bus and return immediately. Transfer is cancelled if no image data arrive before timeout run out.
Internal timeout setting is used, see Set_Grabber_Timeout.

Prototype:

```
DWORD Acquire_Image_Async (
    void*  adr
);
```

Parameters:

Name	Type	Description
adr	void*	Pointer to address where the image gets stored

Return value:

Errorcode or 0 in case of success.

2.3.3.10 Acquire_Image_Async

Setup an image transfer. (Non-Blocking)
Prepare all neccessary data structures to start a (image)-data transfer from the camera over the usb-bus and return immediately. Transfer is cancelled if no image data arrive before timeout run out.
Internal timeout setting is used, see Set_Grabber_Timeout.

Prototype:

```
DWORD Acquire_Image_Async (
    void*  adr,
    int timeout
);
```

Parameters:

Name	Type	Description
adr	void*	Pointer to address where the image gets stored
timeout	int	time to wait for image in ms

Return value:

Errorcode or 0 in case of success.

2.3.3.11 Acquire_Image_Async_wait

Setup an image transfer and wait until image is decoded. (Blocking)
Prepare all neccessary data structures to start a (image)-data transfer from the camera over the usb-bus. Wait until image data is sucessfully decoded and transferred to the given address. Transfer is cancelled if no image data arrive before timeout run out. Transfers an image from the camera and waits for decoding to be finished
Internal timeout setting is used, see Set_Grabber_Timeout.

Prototype:

```
DWORD Acquire_Image_Async_wait (
    void*  adr
);
```

Parameters:

Name	Type	Description
adr	void*	Pointer to address where the image gets stored

Return value:

Errorcode or 0 in case of success.

2.3.3.12 Acquire_Image_Async_wait

Setup an image transfer and wait until image is decoded. (Blocking)
Prepare all neccessary data structures to start a (image)-data transfer from the camera over the usb-bus. Wait until image data is sucessfully decoded and transferred to the given address. Transfer is cancelled if no image data arrive before timeout run out. Transfers an image from the camera and waits for decoding to be finished
Custom timeout setting is used.

Prototype:

```
DWORD Acquire_Image_Async_wait (
    void*  adr,
    int timeout
);
```

Parameters:

Name	Type	Description
adr	void*	Pointer to address where the image gets stored
timeout	int	time to wait for image in ms

Return value:

Errorcode or 0 in case of success.

2.3.3.13 Acquire_Image_Async

Asynchronous image transfer function.
Prepare all neccessary data structures to start a (image)-data transfer from the camera over the usb-bus and return immediately. Transfer is cancelled if no image data arrive before timeout run out. Parameter waitforimage is used to determine if this is a Blocking or Non-Blocking function. The default behaviour is to **not** wait for the decoding (if neccessary) to finish. It is recommended to use the overloaded functions! Transfer is cancelled if no image data arrive before timeout run out.
Custom timeout setting is used.

Prototype:

```
DWORD Acquire_Image_Async (
    void*  adr,
    int timeout,
    BOOL waitforimage
);
```

Parameters:

Name	Type	Description
adr	void*	Pointer to address where the image gets stored
timeout	int	time to wait for image in ms
waitforimage	BOOL	If set to TRUE, this function waits until **all** transferred images up to this one are completely decoded. Otherwise set this to false, or use the overloaded member functions.

Return value:

Errorcode or 0 in case of success.

2.3.3.14 Set_Grabber_Timeout

Sets general timeout for all image acquire functions without timeout parameter.

Prototype:

```
DWORD Set_Grabber_Timeout (
    int timeout
);
```

Parameters:

Name	Type	Description
timeout	int	timeout for image acquire in ms

Return value:

Errorcode or 0 in case of success.

2.3.3.15 Get_Grabber_Timeout

Gets the current timeout for all image acquire functions without timeout parameter.

Prototype:

```
DWORD Get_Grabber_Timeout (
    int*  timeout
);
```

Parameters:

Name	Type	Description
timeout	int*	Pointer to integer variable to get the timeout for image acquire in ms

Return value:

Error or 0 in case of success

2.3.3.16 PostArm

Get camera settings and set internal parameters
This function call should be called after Arm_Camera is called and is an overall replacement for the following functions. Parameter userset is used to determine if the grabber parameters are changed (recommended) or not.

Prototype:

```
DWORD PostArm (
    int userset = 0
);
```

Parameters:

Name	Type	Description
userset	int	If set to 0 (default), this function does setup the grabber class correctly for following image transfers. If set to any other value grabber class **must** be setup with the following functions.

Return value:

Errorcode or 0 in case of success.

2.3.3.17 Set_Grabber_Size

Sets the grabber size.
It is extremely important to set this before any images are transferred! If any of the settings are changed that influence the image size Set_Grabber_Size **must** be called again before any images are transferred! If this is not done, memory or segmentation faults will occur!

Prototype:

```
DWORD Set_Grabber_Size (
    int width,
    int height,
    int bitpix
);
```

Parameters:

Name	Type	Description
width	int	width of the picture in pixel
height	int	height of the picture in pixel
bitpix	int	number of bits per pixel. This value is rounded up to a multiple of 8.

Return value:

Errorcode or 0 in case of success.

2.3.3.18 Set_Grabber_Size

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Prototype:

```
DWORD Set_Grabber_Size (
    int width,
    int height
);
```

Parameters:

Name	Type	Description
width	int	width of the picture in pixel
height	int	height of the picture in pixel

Return value:

Errorcode or 0 in case of success.

2.3.3.19 Get_actual_size

Returns the current grabber sizes.

Prototype:

```
int Get_actual_size (
    unsigned int*  width,
    unsigned int*  height,
    unsigned int*  bitpix
);
```

Parameters:

Name	Type	Description
width	unsigned int*	pointer to variable on return current width
height	unsigned int*	pointer to variable on return current height
bitpix	unsigned int*	pointer to variabl on return current bits per pixel

Return value:

Errorcode or 0 in case of success.

2.3.3.20 Set_DataFormat

Set the dataformat for the following image transfers
At the moment this call is only a dummy function without really usefulness.

Prototype:

```
DWORD Set_DataFormat (
    DWORD dataformat
);
```

Parameters:

Name	Type	Description
dataformat	DWORD	New Dataformat

Return value:

Errorcode or 0 in case of success.

2.3.3.21 Get_DataFormat

Returns the current grabber format.

Prototype:

```
DWORD Get_DataFormat (    );
```

Parameters:

No parameter

Return value:

current Dataformat

2.3.3.22 SetBitAlignment

Set BitAlignment parameter to the grabber class, which is needed for correct handling of image data.

Prototype:

```
void SetBitAlignment (
    int align
);
```

Parameters:

Name	Type	Description
align	int	value retrieved from camera after last PCO_ArmCamera() call

Return value:

2.3.3.23 Get_Height

Returns the current grabber height.

Prototype:

```
int Get_Height (    );
```

Parameters:

No parameter

Return value:

Current height.

2.3.3.24 Get_Width

Returns the current grabber width.

Prototype:

```
int Get_Width (    );
```

Parameters:

No parameter

Return value:

Current width.

2.3.3.25 Set_Async_Packet_Size

Set packet size for USB transfers For effective data handling packet size should be set between 256kB and 1MB.

Prototype:

```
void Set_Async_Packet_Size (
    int size
);
```

Parameters:

Name	Type	Description
size	int	value of new packet size

Return value:

2.3.3.26 Get_Async_Packet_Size

Returns the currently used packet size for USB transfers.

Prototype:

```
int Get_Async_Packet_Size (    );
```

Parameters:

No parameter

Return value:

Current packet size.