manual

pco.recorder

excelitas.com

excelitas®

Excelitas PCO GmbH asks you to carefully read and follow the instructions in this document.
For any questions or comments, please feel free to contact us at any time.

address:	Excelitas PCO GmbH Donaupark 11 93309 Kelheim, Germany
phone:	(+49) 9441-2005-0 (+1) 866-662-6653 (+86) 0512-6763-4643
mail:	pco@excelitas.com
web:	www.excelitas.com/pco

pco.recorder manual 3.7.0

Released December 2025

Contents

1 General

This document describes the functionality and usage of the ***pco.recorder***. The ***pco.recorder*** is built on top of the ***SDK*** and forms an ***API*** with a reduced amount of functions to simplify acquiring and retrieving images compared to the standard ***pco.sdk*** functions.

Several ***pco.recorder*** instances can be created, but due to the fact that the API is not thread-safe, the instances have to be handled very carefully in a multithreaded application.

The first chapter (**1**) provides a short introduction on how to work with the ***pco.recorder***.

Chapter **2** features an overview of all available functions, described in detail.

A basic workflow and an example implementation can be found in chapter **3**.

Definition		
SDK	Software Development Kit	A SDK is a collection of libraries, sample projects, and applications for software development.
API	Application Programming Interface	An API is an interface for application programming. It is a set of clearly defined methods of communication between various software components.

1.1 Overview

The basic functionality of the ***pco.recorder API*** is to configure and control the acquisition and storage of a user defined number of images. Therefore, three main acquisition modes are available. The image data can either be stored in computer RAM, image files, or read from the internal camera memory.

The required functions are available via function calls inside the ***PCO_Recorder.dll*** which also requires the ***SC2_Cam.dll*** and, depending on the interface type of the camera, sometimes also interface DLLs (sc2_cl_me4.dll, sc2_clhs.dll …). See the ***SDK manual*** for further information.

Note for file names

"_CamXX" will be added to the filename, where X specifies the camera index (+1) as it has been transferred to the recorder.
For ****.b16*** and single tiff format, ***_yyyy*** will also be added, where ***y*** specifies the number of recorded images.

1.2 Conventions

The following typographic conventions are used in this manual:

Bold	**PCO_RecorderCreate**	Functions, procedures, or modes used in this manual, potentially whith a cross reference to the respective page.
Words in brackets	[run]	Possible values or states of the described functions.
Capitalized words	TRUE	Logical or Boolean values such as TRUE, FALSE, ON, OFF, RISING, FALLING, HIGH, LOW.
Words in arrow brackets	<acq enbl>	Name of hardware input/output signals
Code font	`strGeneral.wSize`	C Example Code
Bold italics	***pco.recorder***	Important terms

1.3 Recorder Modes

The pco.recorder can be used for cameras with or without internal memory. Both camera types can be used in streaming mode where the newest recorded images will directly be sent to the computer. For cameras with internal memory, the images are recorded into the camera RAM either in ring buffer or sequence mode.

There are three different modes and four optional flags to save the captured images(see chapters **2.5** and **2.5.1**):

Value	Type	Description
0x0001	**PCO_RECORDER_MODE_FILE**	***pco.recorder*** will save the recorded images as files on the hard drive.
0x0002	**PCO_RECORDER_MODE_MEMORY**	***pco.recorder*** will save the recorded images in the computer RAM.
0x0003	**PCO_RECORDER_MODE_CAMRAM**	Images will be read from the internal camera memory.

Value	Type	Description
0x1000	**PCO_RECORDER_MODE_USE_DPCORE**	Flag for using DotPhoton Compression.
0x2000	**PCO_RECORDER_USE_USB3_MULTIREQUEST**	***pco.recorder*** will save the recorded images in the computer RAM.
0x8000	**PCO_RECORDER_DOUBLEIMG_SPLIT**	Images will be read from the internal camera memory.
0xC000	**PCO_RECORDER_DOUBLEIMG_SPLIT_SEQUENCE**	Images will be read from the internal camera memory.

Modes	No Flag	DPCORE	MULTI REQUEST	DOUBLE IMAGE	DOUBLE IMAGE SEQUENCE
FILE	✓	✗	✗	✓	✓
MEMORY	✓	✓	✓	✗	✗
CAMRAM	✓	✓	✗	✗	✗

1.4 Typical pco.recorder workflow

The following chapter describes teh typical workflow you should use for building applications with pco.recorder. Complete example implementations can be found in chapter **3**

Before you start the pco.recorder workflow
Perform all camera settings that are necessary for your test setup. Once the pco.recorder object is created, the settings must not be changed anymore.

NOTICE

1 PCO_RecorderCreate

This is the first function that has to be called. Here the user has to transfer all camera handles that should be contolled by the ***pco.recorder*** instance. In this function you define also the acquisition mode (see chapter **1.3**) of the ***pco.recorder*** After the ***pco.recorder*** object is created, only the following ***pco.sdk*** commands are allowed until **PCO_RecorderDelete** is called:

- PCO_GetCameraHealthStatus (see chapter **1.8**)
- PCO_SetDelayExposureTime (see chapter **1.9**)
- PCO_GetDelayExposureTime
- PCO_SetFramerate (see chapter **1.9**)
- PCO_GetFramerate

---

2 PCO_RecorderInit

The **PCO_RecorderCreate** function delivers the maximum number of recordable images (depending on the ***pco.recorder*** type). Considering this upper limit, the ***pco.recorder*** can be initialized with the required number of images that shall be recorded using **PCO_RecorderInit**. In this function you also define the recorder type, which depends on the previously selected acquisition mode (see **2.7.1**)

---

3 PCO_RecorderStartRecord and PCO_RecorderStopRecord

Calling **PCO_RecorderStartRecord** will start the acquisition. For **PCO_RECORDER_MODE_FILE**, **PCO_RECORDER_MODE_MEMORY** with type sequence (see chapter **2.5.1** and **2.7.1**), or **PCO_RECORDER_MODE_CAMRAM** with the camera operating internally in sequence mode, the recording will be stopped automatically.

For **PCO_RECORDER_MODE_MEMORY** with type ring buffer or FIFO (see chapter **2.5.1** and **2.7.1**), or **PCO_RECORDER_MODE_CAMRAM** with the camera operating internally in ring buffer mode, the acquisition has to be stopped manually by calling **PCO_RecorderStopRecord** (see graphic representation in chapter **1.3**). Calling this function during the recording will stop the acquisition.

---

4 PCO_RecorderGetSettings and PCO_RecorderGetStatus

The main settings and the status of the ***pco.recorder*** can be checked using **PCO_RecorderGetSettings** and **PCO_RecorderGetStatus**.

5 PCO_RecorderCopyImage or **PCO_RecorderGetImageAddress**

The recorded images can be accessed either by **PCO_RecorderGetImageAddress** (not available in **PCO_RECORDER_MODE_CAMRAM**), which delivers the address of the required image buffer, or by using **PCO_RecorderCopyImage**, which copies the required image inside a defined region of interest (ROI) in a preallocated buffer.

Note that **PCO_RecorderCopyImage** can be called also when the acquisition is running while **PCO_RecorderGetImageAddress** will be rejected (with an error code) during the recording.

---

6 PCO_RecorderDelete or **PCO_RecorderCleanup**

When the image processing/analysis is finished, **PCO_RecorderDelete** can be called to close the ***pco.recorder*** instance and delete the handle. It is also possible to reset the ***pco.recorder*** with **PCO_RecorderCleanup**. This will reset the data of all image buffers to 0 or, in **PCO_RECORDER_MODE_FILE**, delete all created files, but will not free the resources.

Alternatively, it is also possible to start a new acquisition with **PCO_RecorderStartRecord** which, in **PCO_RECORDER_MODE_MEMORY**, will overwrite the data in the buffers or, in **PCO_RECORDER_MODE_FILE**, overwrite the old files.

For **PCO_RECORDER_MODE_CAMRAM**, the function only affects the internal buffers of the ***pco.recorder***. **PCO_RecorderCleanup** has no effect on the images in the camera memory. A new start acquisition will overwrite the images, just as this would be the case when you start the recording with our standard ***pco.sdk***.

1.5 Running Applications

To allow access to the ***API***, the **PCO_Recorder.dll**, the **SC2_Cam.dll** and possibly additional interface DLLs must reside in the application directory or in the library search path when implicit linkage is used.

The user can also link explicitly. In this case, the DLLs named above can be placed in the application folder or search path.

The files can also be placed in a known folder, but it is necessary to call **LoadLibrary** with the complete path in this case.

1.6 Compiling and Linking

To use the ***API Library*** in an application, the **PCO_Recorder_Export.h** and the **PCO_Recorder_Defines.h** file must be added in addition to the standard header files. The application program must be linked with the appropriate library (32 bit or 64 bit), which can be found in the lib or lib64 folders.

The ***API*** can be called up by linking to the **PCO_Recorder.lib** via the project settings.

Another option is loading the required functions from the **PCO_Recorder.dll** explicitly at runtime with the LoadLibrary function of the ***Windows API***.

1.7 pco.recorder Logging

The ***pco.recorder*** also supports troubleshooting.

If there are problems, you can force the ***pco.recorder*** to write the workflow into a log file by creating a file called **PCO_Recorder.log** in the following directory:

>*systemdisc<:\ProgramData\pco* (On Windows 7/8/10)

Several log levels can be selected. This is done via the **'LOGGING='** parameter in the appropriate **PCO_Recorder_param.ini** file.

For more information about logging with PCO software products, please consult our website regarding the **pco.logging** tool.

1.8 Camera Health Status

Note that the ***pco.recorder*** will not take care of the camera health status internally.

It is recommended to call **PCO_GetCameraHealthStatus** frequently in order to recognize camera internal problems and react to them. This helps to prevent camera hardware from damage.

1.9 Change Frame Rate or Exposure Time

The frame rate is generally limited by readout and exposure times (whichever is larger) and the other way round.

$$framerate \leq \frac{1}{t_{exposure}}$$

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

2 API Function Description

2.1 PCO_RecorderGetVersion

Description This function retrieves the current version information from the ***pco.recorder*** DLL.

Supported camera type(s) All cameras

Prototype

```
void WINAPI PCO_RecorderGetVersion (
    int* iMajor,                    //out
    int* iMinor,                    //out
    int* iPatch,                    //out
    int* iBuild                     //out
);
```

Parameter

Name	Type	Description
iMajor	int*	Pointer to get the major version (can be set to NULL if not relevant)
iMinor	int*	Pointer to get the minor version (can be set to NULL if not relevant)
iPatch	int*	Pointer to get the patch version (can be set to NULL if not relevant)
iBuild	int*	Pointer to get the build number (can be set to NULL if not relevant)

2.2 PCO_RecorderSaveImage

Description This function saves the transferred image as a file at the transferred file path. The file type is implicitly specified by the file extension (e.g. *.tif, *.dcm, *.asc, …). The ***pco.recorder*** supports the same file formats as ***pco.camware***. The image type (bw/color, 8/16 Bit) has to be defined via the filetype parameter. If required, additional metadata can be added to the image using the metadata structure.

Note No recorder instance needs to be created for this function.

Supported camera type(s) All cameras

Prototype

```
int WINAPI PCO_RecorderSaveImage (
    void* pImgBuf,                      //in
    WORD wWidth,                        //in
    WORD wHeight,                       //in
    const char* cFileType,              //in
    bool bIsBitmap,                     //in
    const char* szFilePath,              //in
    bool bOverwrite,                    //in
    PCO_METADATA_STRUCT* strMetadata    //in
);
```

Parameter

Name	Type	Description
pImgBuf	void*	Pointer to the image that should be saved
wWidth	WORD	Width of the image
wHeight	WORD	Height of the image
cFileType	const char*	File type of the input image (see chapter **2.2.1**)
bIsBitmap	bool	Flag to indicate if the input image has bitmap format. ***If you use images from the recorder this is always false***
szFilePath	const char*	File path (including filename and extension) where the file should be saved. The extension defines the file type
bOverwrite	bool	Flag to overwrite an already existing file
strMetadata	PCO_METADATA_STRUCT*	Metadata object containing the additional information that should be saved with the image (can be set to NULL if no metadata should be saved) ***(see pco.sdk manual for struct description)***

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.2.1 File Types

Value	Type	Description
"M_08"	FILESAVE_IMAGE_BW_8	Image is monochrome 8 bit
"M_16"	FILESAVE_IMAGE_BW_16	Image is monochrome 16 bit
"C_08"	FILESAVE_IMAGE_BGR_8	Image is 24 bit color RGB
"C_08_A"	FILESAVE_IMAGE_BGRA_8	Image is 32 bit color RGBA
"C_16"	FILESAVE_IMAGE_BGR_16	Image is 48 bit color RGB

2.3 PCO_RecorderSaveOverlay

Description This function creates a color image out of three transferred monochrome images and saves it at the transferred file path. The file type is implicitly specified by the file extension (e.g. *.tif, *.dcm, *.asc, …). The ***pco.recorder*** supports the same file formats as ***pco.camware***.

Note No recorder instance needs to be created for this function.

Supported camera type(s) All cameras

Prototype

```
int WINAPI PCO_RecorderSaveOverlay    (
    void* pImgBufR,                    //in
    void* pImgBufG,                    //in
    void* pImgBufB,                    //in
    WORD wWidth,                       //in
    WORD wHeight,                      //in
    const char* cFileType,             //in
    const char* szFilePath,             //in
    bool bOverwrite,                   //in
    PCO_METADATA_STRUCT* strMetadata   //in
);
```

Parameter

Name	Type	Description
pImgBufR	void*	Pointer to the image that should be used as red channel
pImgBufG	void*	Pointer to the image that should be used as green channel
pImgBufB	void*	Pointer to the image that should be used as blue channel
wWidth	WORD	Width of the image
wHeight	WORD	Height of the image
cFileType	const char*	File type of the input image (see chapter **2.2.1**)
szFilePath	const char*	File path (including filename and extension) where the file should be saved. The extension defines the file type
bOverwrite	bool	Flag to overwrite an already existing file

Continued on next page

Continued from previous page

Name	Type	Description
strMetadata	PCO_METADATA_STRUCT*	Metadata object containing the additional information that should be saved with the image (can be set to NULL if no metadata should be saved) ***(see pco.sdk manual for struct description)***

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.4 PCO_RecorderResetLib

Description

This function checks if at least one ***pco.recorder*** instance is active. If so, the user is asked via a message box whether they really want to reset. If the silent flag is set, this message box will be omitted and the reset will be completed.

The reset will delete all ***pco.recorder*** instances that are currently active (= created).

Supported camera type(s)

All cameras

Prototype

```
int WINAPI PCO_RecorderResetLib (
    bool bSilent                    //in
);
```

Parameter

Name	Type	Description
bSilent	bool	Flag to decide if the message box should be omitted when a recorder instance is active
		TRUE: If ***pco.recorder*** instances are active, reset will be done anyway, without showing a message box
		FALSE: If ***pco.recorder*** instances are active, the function shows a message box to decide if reset should be done. PCO_ERROR_SDKDLL_ALREADYOPENED will be returned if reset is denied by the user

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.5 PCO_RecorderCreate

Description

This function creates an instance of the ***pco.recorder***. It takes an array of handles to the required cameras as input parameter. If the function succeeds, sdk functions may not be used, except for those listed in chapter **1.4** under (1) **PCO_RecorderCreate**.

The main task of this function is to calculate the maximum recordable number of images for every camera by checking the available memory (RAM, disc or camera internal memory, depending on the recorder mode) and the required distribution of the memory to the single cameras (e.g. camera 1 should get twice as much available memory as camera 2, then the distribution would be [2, 1]).

If the ***pco.recorder*** mode is **PCO_RECORDER_MODE_FILE**, a letter for the drive on which the files will be saved, has to be specified. For the other modes, this parameter is ignored.

Note

- it is also possible to create several recorder instances, but make sure not to use the same camera handles. Otherwise your application could crash
- for **PCO_RECORDER_MODE_FILE**, this function returns a warning if the file name of the specified path already exists
- for **PCO_RECORDER_MODE_CAMRAM**, the memory distribution has no effect. Here `dwImgDistributionArr` can be set to NULL.

Supported camera type(s)

All cameras

Prototype

```
int WINAPI PCO_RecorderCreate (
    HANDLE* phRec,                          //in, out
    HANDLE* phCamArr,                       //in
    const DWORD* dwImgDistributionArr,      //in
    WORD wArrLength,                        //in
    WORD wRecMode,                          //in
    const char* szDrive,                      //in
    DWORD* dwMaxImgCountArr                 //out
);
```

Parameter

Name	Type	Description
phRec	HANDLE*	Pointer to a HANDLE: on Input: HANDLE must be set to NULL on Output: A unique HANDLE to the created ***pco.recorder*** object is returned
phCamArr	HANDLE*	Array of handles to the cameras that should be used by the ***pco.recorder***
dwImgDistributionArr	const DWORD*	Array defining the memory distribution between the used cameras (can be set to NULL for equal distribution or **PCO_RECORDER_MODE_CAMRAM**)
wArrLength	WORD	Length of all transferred arrays and also length of the maxImgCountArray
wRecMode	WORD	Required mode of the ***pco.recorder*** (see chapter **2.5.1**)
szDrive	const char*	Root path name that represents the required drive to save the images to, e.g. **"C"**, **"C:"** for system drive on windows, path to mounted disk e.g. **"/"** or **"/media/<drive_name>"** on linux (only for **PCO_RECORDER_MODE_FILE**, ignored otherwise)

Continued on next page

Continued from previous page

Name	Type	Description
dwMaxImgCountArr	DWORD*	Array to get the maximum available image count for each camera (length must be equal to length of the camera handle array)

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.5.1 Recorder Modes

For further explanations on the different modes of the ***pco.recorder*** and how to use them, please see chapter **1.3**.

1 PCO_RECORDER_MODE_FILE

Value	Type	Description
0x0001	PCO_RECORDER_MODE_FILE	***pco.recorder*** will save the recorded images as files on the hard drive (see chapter **2.7.1** for available types)

2 PCO_RECORDER_MODE_MEMORY

Value	Type	Description
0x0002	PCO_RECORDER_MODE_MEMORY	***pco.recorder*** will save the recorded images in the computer RAM (see chapter **2.7.1** for available types)

3 PCO_RECORDER_MODE_CAMRAM

Value	Type	Description
0x0003	PCO_RECORDER_MODE_CAMRAM	Images will be read from the internal camera memory (see chapter **2.7.1** for available types)

4 PCO_RECORDER_MODE_USE_DPCORE

Value	Type	Description
0x1000	PCO_RECORDER_MODE_USE_DPCORE	Flag for using DotPhoton Compression. Can only be used with CamRam or Memory mode. Is ignored if PCO_RECORDER_MODE_FILE is set

5 PCO_RECORDER_USE_USB3_MULTIREQUEST

Value	Type	Description
0x2000	PCO_RECORDER_USE_USB3_MULTIREQUEST	Flag for enabling the multi image request. Only for usb3 cameras and only with PCO_RECORDER_MODE_MEMORY

2.6 PCO_RecorderDelete

Description

This function deletes the ***pco.recorder*** object. If necessary, it frees all allocated memory and resources. After this function has succeeded, the ***pco.recorder*** handle will be invalid.

The function will be rejected with an error if an acquisition is running.

Supported camera type(s)

All cameras

Prototype

```
int WINAPI PCO_RecorderDelete (
    HANDLE phRec                //in
);
```

Parameter

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.7 PCO_RecorderInit

Description

This function initializes the ***pco.recorder*** according to the required number of images for each camera. It will discard previous initializations.

For **PCO_RECORDER_MODE_MEMORY**, it will allocate the necessary RAM to store the images.

For **PCO_RECORDER_MODE_FILE**, it checks whether files with the same name already exist and, depending on the **wNoOverwrite** flag, either deletes the old files or if the flag is set, renames them. A file is renamed by adding ***(n)*** to the filename, where ***n*** is the lowest number that has not been used yet. This means the higher the numbers in the brackets, the newer the files, but the file without brackets is always the newest/current one.

For **PCO_RECORDER_MODE_CAMRAM**, the function will activate the required segment if necessary and update the maximum image count and the number of recorded images internally. If the camera RAM segment in this function is different to the one that was active during **PCO_RecorderCreate**, it might occur that the required image count is too large, even if the maximum image count from the **PCO_RecorderCreate** function is not reached. So anyway it is recommended to call **PCO_RecorderGetSettings** and **PCO_RecorderGetStatus** to update these parameters after this function. It is also possible to read images directly after **PCO_RecorderInit**, if the selected segment already contains images.

The function will be rejected with an error if an acquisition is running.

Supported camera type(s)

All cameras

Prototype

```
int WINAPI PCO_RecorderInit (
    HANDLE phRec,              //in
    DWORD* dwImgCountArr,      //in
    WORD wArrLength,           //in
    WORD wType,                //in
    WORD wNoOverwrite,         //in
    const char* szFilePath,    //in
    WORD* wRamSegmentArr       //in
);
```

Parameter

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object
dwImgCountArr	DWORD*	Array containing the required image counts for all cameras
wArrLength	WORD	Length of the imgCountArr (must match with the number of cameras)
wType	WORD	Type of the selected pco.recorder mode (functionality depends on ***pco.recorder*** modes) (see chapter **2.7.1**)
wNoOverwrite	WORD	Flag to decide whether existing files should be kept and renamed (files will be deleted if NOT SET) (only for **PCO_RECORDER_MODE_FILE**, ignored otherwise)

Continued on next page

Continued from previous page

Name	Type	Description
szFilePath	const char*	Path (including filename) where the image files should be saved (only for **PCO_RECORDER_MODE_FILE**, ignored otherwise)
wRamSegmentArr	WORD*	Array containing the camera RAM segments (must match with the number of cameras = wArrLength) to be used for acquisition and readout, it can be set to NULL if no RAM segment change is required (only **PCO_RECORDER_MODE_CAMRAM**, ignored otherwise)

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.7.1 Recorder Types

For further explanations on the different modes of the ***pco.recorder*** and how to use them, please see chapter **1.3**.

1 Types for PCO_RECORDER_MODE_FILE

Value	Type	Description
0x0001	PCO_RECORDER_FILE_TIF	Images will be saved on the hard drive in single TIFF format
0x0002	PCO_RECORDER_FILE_MULTITIF	Images will be saved on hard drive in multi TIFF format
0x0003	PCO_RECORDER_FILE_PCORAW	Images will be saved on hard drive in pco-raw format
0x0004	PCO_RECORDER_FILE_B16	Images will be saved on hard drive in b16 format
0x0005	PCO_RECORDER_FILE_DICOM	Images will be saved on hard drive in single dicom format
0x0006	PCO_RECORDER_FILE_MULTIDICOM	Images will be saved on hard drive in multi dicom format

2 Types for PCO_RECORDER_MODE_MEMORY

Value	Type	Description
0x0001	PCO_RECORDER_MEMORY_SEQUENCE	Images will be recorded into the computer memory sequentially, until the required image number is reached
0x0002	PCO_RECORDER_MEMORY_RINGBUF	Images will be recorded into the computer memory in ring buffer mode. After calling stop acquisition, the latest images are in the buffers

Continued on next page

Continued from previous page

Value	Type	Description
0x0003	PCO_RECORDER_MEMORY FIFO	Images will be recorded into the computer memory in fifo mode. If the required image number is reached, the acquisition will wait until first images have been read. Here it is only possible to read out sequentially

3 Types for PCO_RECORDER_MODE_CAMRAM

Value	Type	Description
0x0001	PCO_RECORDER_CAMRAM_SEQUENTIAL	***pco.recorder*** will get the images from the internal camera memory. The readout is optimized for sequential reading. This means that if an image is queried, the readout for the next image in the series will automatically be triggered in parallel
0x0002	PCO_RECORDER_CAMRAM_SINGLE_IMAGE	***pco.recorder*** will get the images from the internal camera memory. The readout is not optimized. Images will be read from the camera when they are queried

2.8 PCO_RecorderCleanup

Description

This function resets the recorded images either for one specific camera or for all cameras (if NULL is transferred as camera handle). For **PCO_RECORDER_MODE_FILE**, reset means that all previously recorded image files will be deleted.

For **PCO_RECORDER_MODE_MEMORY** or **PCO_RECORDER_MODE_CAMRAM**, the image data in the allocated buffers will be reset to 0. The function will not affect the images in the internal camera memory.

The function will be rejected with an error if an acquisition is running.

Supported camera type(s)

All cameras

Prototype

```
int WINAPI PCO_RecorderCleanup (
    HANDLE phRec,                //in
    HANDLE phCam                 //in
);
```

Parameter

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object
phCam	HANDLE	HANDLE to a particular camera (or NULL for all cameras)

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.9 PCO_RecorderGetSettings

Description This function retrieves the current ***pco.recorder*** settings for a specific camera.

Note For **PCO_RECORDER_MODE_CAMRAM**, dwMaxImgCount will be updated if the segment has changed during **PCO_RecorderInit**.

Supported camera type(s) All cameras

Prototype

```
int WINAPI PCO_RecorderGetSettings (
    HANDLE phRec,                   //in
    HANDLE phCam,                   //in
    DWORD* dwRecMode,               //out
    DWORD* dwMaxImgCount,           //out
    DWORD* dwReqImgCount,           //out
    WORD* wWidth,                   //out
    WORD* wHeight,                  //out
    WORD* wMetadataLines            //out
);
```

Parameter

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object
phCam	HANDLE	HANDLE to a particular camera to get the settings from
dwRecMode	DWORD*	Pointer to a DWORD to get the selected mode of the ***pco.recorder*** (High Word is **Recorder Mode**; Low Word is **Recorder Type**) (can be set to NULL if not relevant)
dwMaxImgCount	DWORD*	Pointer to a DWORD to get the maximum number of recordable images for the selected cameras (can be set to NULL if not relevant)
dwReqImgCount	DWORD*	Pointer to a DWORD to get the required number of recordable images for the selected cameras (can be set to NULL if not relevant)
wWidth	WORD*	Pointer to a WORD to get the image width of the camera (can be set to NULL if not relevant)
wHeight	WORD*	Pointer to a WORD to get the image height of the camera (can be set to NULL if not relevant)
wMetadataLines	WORD*	Pointer to a WORD to get the metadata lines added at the end of an image, will be 0 for Metadata Mode OFF (can be set to NULL if not relevant)

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.10 PCO_RecorderStartRecord

Description This function starts the recording either for a specific camera or for all cameras (if NULL is transferred as camera handle).

Supported camera type(s) All cameras

Prototype

```
int WINAPI PCO_RecorderStartRecord (
    HANDLE phRec,                    //in
    HANDLE phCam                     //in
);
```

Parameter

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object
phCam	HANDLE	HANDLE to a particular camera that should be started or NULL if all cameras should be started

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.11 PCO_RecorderStopRecord

Description This function stops the recording either for a specific camera or for all cameras (if NULL is transferred as camera handle).

Supported camera type(s) All cameras

Prototype

```
int WINAPI PCO_RecorderStopRecord (
    HANDLE phRec,                   //in
    HANDLE phCam                    //in
);
```

Parameter

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object
phCam	HANDLE	HANDLE to a particular camera that should be stopped or NULL if all cameras should be stopped

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.12 PCO_RecorderSetAutoExposure

Description This function activates or deactivates the auto exposure functionality for the selected camera or for all cameras (if NULL is transferred as camera handle).

For this functionality, an exposure range has to be selected where the ***pco.recorder*** should be allowed to change the exposure time. Additionally, the transition between exposure time changes can be controlled by a smoothness factor, where 1 means a direct switch to the new exposure time. The higher this value is, the smoother the transition and thus the smoother the adjustment will be (a maximum of 10 is allowed).

The function will be rejected with an error if ***pco.recorder*** is not initialized.

Supported camera type(s) All cameras

Prototype

```
int WINAPI PCO_RecorderSetAutoExposure (
    HANDLE phRec,                   //in
    HANDLE phCam,                   //in
    bool bAutoExpState,             //in
    WORD wSmoothness,               //in
    DWORD dwMinExposure,            //in
    DWORD dwMaxExposure,            //in
    WORD wExpBase                   //in
);
```

Parameter

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object
phCam	HANDLE	HANDLE to a particular camera (or NULL for all cameras)
bAutoExpState	bool	Indicator if auto exposure should be activated
wSmoothness	WORD	Value defining how smooth the transition between exposure times should be (valid are 1 - 10)
dwMinExposure	DWORD	Minimum exposure value that can be used for auto exposure (in expBase units)
dwMaxExposure	DWORD	Maximum exposure value that can be used for auto exposure (in expBase units)
wExpBase	WORD	Exposure unit of the transferred exposure time range (0:ns, 1:us, 2:ms)

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.13 PCO_RecorderSetAutoExpRegions

Description

This function sets the regions of interest for the auto exposure functionality for the selected camera or for all cameras (if NULL is transferred as camera handle).

It is possible to set four different predefined region types (0=balanced, 1=center based, 2=corner based, 3=full) or to define custom regions (=4). Depending on the type a different set of pixel clusters is used to compute the mean values, this can be seen in section 2.13.1. For a custom region up to 7 region-blocks with FIXED size (containing 9 pixel clusters) can be specified using the top left point for each region.

Note

The size of the pixel clusters is fixed, but depends on the overall image size and is treated seperately for width and height:

- for width/height >= 1300 the cluster size is 100
- for 1300 > width/height >= 650 the cluster size is 50
- for 650 > width/height >= 325 the cluster size is 25
- for width/height < 325 the cluster size equal to width/height.

Supported camera type(s)

All cameras

Prototype

```
int WINAPI PCO_RecorderSetAutoExpRegions (
    HANDLE phRec,                          //in
    HANDLE phCam,                          //in
    WORD wRegionType,                      //in
    WORD* wRoiX0Arr,                        //in
    WORD* wRoiY0Arr,                        //in
    WORD wArrLength                        //in
);
```

Parameter

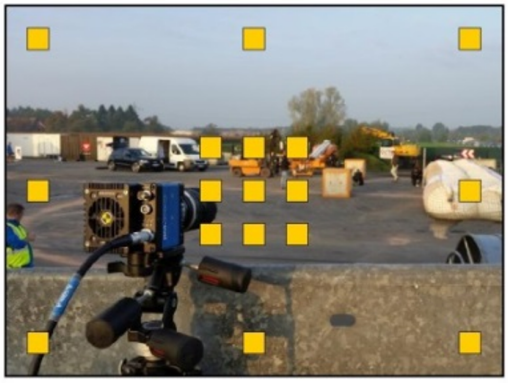
Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object
phCam	HANDLE	HANDLE to a particular camera (or NULL for all cameras)
wRegionType	WORD	Type of the region to be set 0x0000 = balanced 0x0001 = center based 0x0002 = corner based 0x0003 = full 0x0004 = custom
wRoiX0Arr	WORD*	Array of x0 values (starting with 1) defining the left position of the desired regions (only for custom region, set to NULL otherwise)
wRoiY0Arr	WORD*	Array of y0 values (starting with 1) defining the upper position of the desired regions (only for custom region, set to NULL otherwise)
wArrLength	WORD	Length of the ROI arrays (maximum 7) (only for custom region, set to 0 otherwise)

Return value

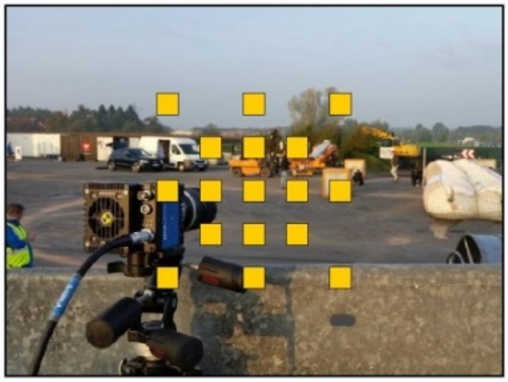
Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.13.1 Region Types

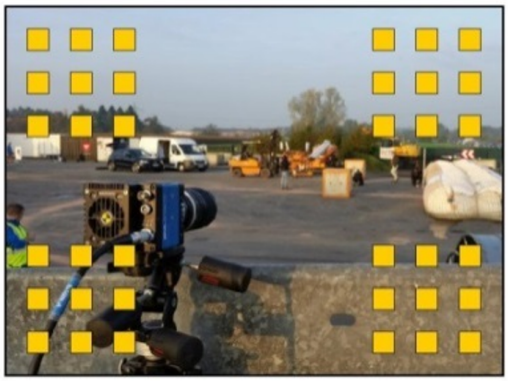
0x0000 **REGION_TYPE_BALANCED**
Measurement fields positioned centrally and in all corners.



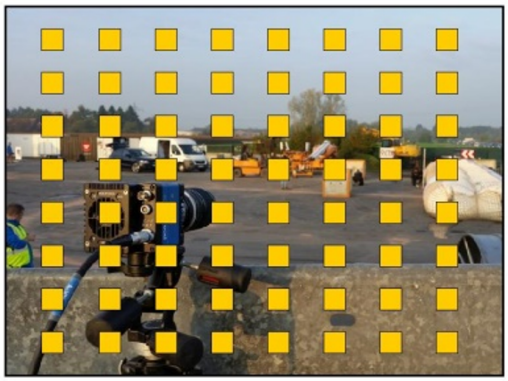
0x0001 **REGION_TYPE_CENTER_BASED**
Measurement fields positioned centrally.



0x0002 **REGION_TYPE_CORNER_BASED**
Measurement fields positioned in all four corners.

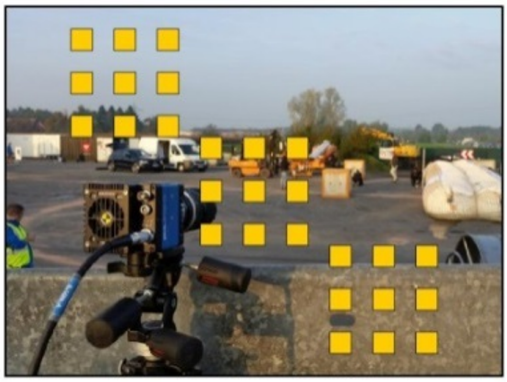


0x0003 **REGION_TYPE_FULL**
Measurement fields across the image.



0x0004 **REGION_TYPE_CUSTOM**
Select up to 7 regions on your own. Values of the example regions:

```
wRoiX0Arr = [301, 901, 1401];
wRoiY0Arr = [101, 601, 1101]
```



2.14 PCO_RecorderSetCompressionParams

Description

This function sets the **PCO_Recorder_CompressionParams** structure in order to enable the **PCO_RecorderCopyImageCompressed** function. The parameters will be used to calculate a LUT according to the extended noise equilibration method published in *TM - Technisches Messen (doi:10.1515/teme-2019-0022)*. This LUT will be used in **PCO_RecorderCopyImageCompressed** to compress the images from 16 bit to 8 bit. The function must be called after **PCO_RecorderInit** and before **PCO_RecorderDelete**.

Supported camera type(s)

All cameras

Prototype

```
int WINAPI PCO_RecorderSetCompressionParams (
    HANDLE phRec,                                          //in
    HANDLE phCam,                                          //in
    PCO_Recorder_CompressionParams* strCompressionParams //in
);
```

Parameter

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object
phCam	HANDLE	HANDLE to a particular camera
strCompressionParams	PCO_Recorder_CompressionParams*	Pointer to struct containing the necessary noise parameters for the compression / equilibration of the specific camera. See chapter **2.14.1** for the details

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.14.1 PCO_Recorder_CompressionParams Structure

Name	Type	Description
dGainK	double	System gain K in DN/e^- (= 1/conversion factor)
dDarkNoise_e	double	Temporal dark noise in electrons (= RMS readout noise)
dDSNU_e	double	DSNU in electrons

Continued on next page

Continued from previous page

Name	Type	Description
dPRNU_pct	double	PRNU in percent
dLightSourceNoise_pct	double	RMS intensity noise of the light source (set to 0 if not known or negligible)

The first four values can be found in the datasheets of the camera. If more precise values are needed for a specific camera, please contact us at PCO: pco@excelitas.com

2.15 PCO_RecorderGetStatus

Description

This function retrieves the current ***pco.recorder*** status for a specific camera.

Note for `dwProcImgCount`:

- for **PCO_RECORDER_MODE_CAMRAM** `dwProcImgCount` is the fill level of the current segment and will be updated if the segment has changed in **PCO_RecorderInit**
- for **PCO_RECORDER_MODE_MEMORY** with type FIFO, `dwProcImgCount` shows the currently available buffers in the FIFO, so an image can only be read if the value is >0.

Supported camera type(s)

All cameras

Prototype

```
int WINAPI PCO_RecorderGetStatus (
    HANDLE phRec,                  //in
    HANDLE phCam,                  //in
    bool* bIsRunning,              //out
    bool* bAutoExpState,            //out
    DWORD* dwLastError,            //out
    DWORD* dwProcImgCount,         //out
    DWORD* dwReqImgCount,          //out
    bool* bBuffersFull,            //out
    bool* bFIFOOverflow,           //out
    DWORD* dwStartTime,            //out
    DWORD* dwStopTime              //out
);
```

Parameter

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object
phCam	HANDLE	HANDLE to a particular camera to get the status from
bIsRunning	bool*	Pointer to a bool to get the running status (can be set to NULL if not relevant)
bAutoExpState	bool*	Pointer to a bool to get the auto exposure status (can be set to NULL if not relevant)
dwLastError	DWORD*	Pointer to a DWORD to get the last error that occurred (can be set to NULL if not relevant)
dwProcImgCount	DWORD*	Pointer to a DWORD to get the number of currently recorded images (can be set to NULL if not relevant)

Continued on next page

Continued from previous page

Name	Type	Description
dwReqImgCount	DWORD*	Pointer to a DWORD to get the required number of images (can be set to NULL if not relevant)
bBuffersFull	bool*	Pointer to a bool to get the indicator if the allocated buffers are all filled (can be set to NULL if not relevant)
bFIFOOverflow	bool*	Pointer to a bool to get the indicator if a FIFO overflow occurred, only relevant in **PCO_RECORDER_MODE_MEMORY** with FIFO type (see chapter **2.7.1**) (can be set to NULL if not relevant)
dwStartTime	DWORD*	Pointer to a DWORD to get the start time in ms of the latest started acquisition (can be set to NULL if not relevant)
dwStopTime	DWORD*	Pointer to a DWORD to get the stop time in ms of the latest finished acquisition (can be set to NULL if not relevant)

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.16 PCO_RecorderGetImageAddress

Description This function retrieves the address of the specified image from the specified camera.

Note

- if the image index exceeds the number of required or recorded images (depending on which value is smaller), the function will return with an error. If **PCO_RECORDER_LATEST_IMAGE** (see chapter **2.16.1**) is set as the image index, the address of the latest image will be transferred
- this function is not available for **PCO_RECORDER_MODE_CAMRAM**.

The function will be rejected with an error if an acquisition is running.

Supported camera type(s) All cameras

Prototype

```
int WINAPI PCO_RecorderGetImageAddress  (
    HANDLE phRec,                       //in
    HANDLE phCam,                       //in
    DWORD dwImgIdx,                     //in
    void** wImgBuf,                     //out
    WORD* wWidth,                       //out
    WORD* wHeight,                      //out
    DWORD* dwImgNumber                  //out
);
```

Parameter

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object
phCam	HANDLE	HANDLE to the required camera
dwImgIdx	DWORD	Index of the required image
wImgBuf	void**	Pointer to a WORD* to get the address of the required image data
wWidth	WORD*	Pointer to a WORD to get the image width of the camera
wHeight	WORD*	Pointer to a WORD to get the image height of the camera
dwImgNumber	DWORD*	Pointer to a DWORD to get the number of the requested image (can be set to NULL if not relevant)

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.16.1 Image Readout

Value	Type	Description
0xFFFFFFFF	PCO_RECORDER_LATEST_IMAGE	***pco.recorder*** will address the latest image

2.17 PCO_RecorderCopyImage

Description

This function copies a defined ROI of the specified image from the specified camera into a preallocated buffer. If the specified image index exceeds the number of required or recorded images (depending on which value is smaller), the function will return an error. If **PCO_RECORDER_LATEST_IMAGE** (see chapter **2.16.1**) is set as the image index, the latest image will be copied.

Make sure that the transferred buffer has always at least the size of the transferred ROI. Since the buffer size will not be checked internally, a buffer which is too small might crash your application.

If the recorder mode is **PCO_RECORDER_MODE_MEMORY** with type ring buffer (see chapter **2.7.1**) and acquisition is running, it is possible that the required image will be overwritten during the copying process. In this case, the resulting data will be unpredictable. Use the function with care during acquisition in this state.

If the mode is **PCO_RECORDER_MODE_FILE** or **PCO_RECORDER_MODE_CAMRAM** and acquisition is running, the function will fail for all indices except **PCO_RECORDER_LATEST_IMAGE** (see chapter **2.16.1**).

Supported camera type(s)

All cameras

Prototype

```
int WINAPI PCO_RecorderCopyImage (
    HANDLE phRec,                          //in
    HANDLE phCam,                          //in
    DWORD dwImgIdx,                        //in
    WORD wRoiX0,                           //in
    WORD wRoiY0,                           //in
    WORD wRoiX1,                           //in
    WORD wRoiY1,                           //in
    void* wImgBuf,                         //out
    DWORD* dwImgNumber,                    //out
    PCO_METADATA_STRUCT* strMetadata,      //out
    PCO_TIMESTAMP_STRUCT* strTimestamp     //out
);
```

Parameter

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object
phCam	HANDLE	HANDLE to a particular camera
dwImgIdx	DWORD	Index of the required image
wRoiX0	WORD	Left horizontal ROI (starting with 1)
wRoiY0	WORD	Upper vertical ROI (starting with 1)
wRoiX1	WORD	Right horizontal ROI (up to image width)
wRoiY1	WORD	Lower vertical ROI (up to image height)
wImgBuf	void*	Pointer to the start address of the buffer the image should be copied to
dwImgNumber	DWORD*	Pointer to a DWORD to get the number of the requested image (can be set to NULL if not relevant)
strMetadata	PCO_METADATA_STRUCT*	Pointer to a PCO_METADATA_STRUCT (see ***pco.sdk*** manual) to get the current metdata of the image if available (can be set to NULL if not relevant)
strTimestamp	PCO_TIMESTAMP_STRUCT*	Pointer to a PCO_TIMESTAMP_STRUCT (see ***pco.sdk*** manual) to get the current binary timestamp information of the image if timestamp is on (can be set to NULL if not relevant)

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.18 PCO_RecorderCopyAverageImage

Description

This function averages a range of images defined by a start and stop index inside a defined ROI and copies the averaged image.

The usage of the function is very similar to **PCO_RecorderCopyImage** (see chapter **2.17**).

Note

The image number, time stamp, and metadata information is not available here.

Supported camera type(s)

All cameras

Prototype

```
int WINAPI PCO_RecorderCopyAverageImage (
    HANDLE phRec,                    //in
    HANDLE phCam,                    //in
    DWORD dwStartIdx,                //in
    DWORD dwStopIdx,                 //in
    WORD wRoiX0,                    //in
    WORD wRoiY0,                    //in
    WORD wRoiX1,                    //in
    WORD wRoiY1,                    //in
    void* wImgBuf                    //out
);
```

Parameter

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object
phCam	HANDLE	HANDLE to a particular camera
dwStartIdx	DWORD	Index of the first image that should be used for averaging
dwStopIdx	DWORD	Index of the last image that should be used for averaging
wRoiX0	WORD	Left horizontal ROI (starting with 1)
wRoiY0	WORD	Upper vertical ROI (starting with 1)
wRoiX1	WORD	Right horizontal ROI (up to image width)
wRoiY1	WORD	Lower vertical ROI (up to image height)
wImgBuf	void*	Pointer to the start address of the buffer the averaged image should be copied to

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.19 PCO_RecorderCopyImageCompressed

Description

This function copies a compressed 8 bit image using a nearly lossless compression method called extended noise equilibration.

The usage of the function is very similar to **PCO_RecorderCopyImage** (see chapter **2.17**)

Since the function returns an 8 bit image, you have to prepare and transfer a BYTE buffer instead of a WORD buffer.

The compressed image you receive by using this function will be viewable directly on screen without the need of decompression. It has a very low and brightness-independent noise level which helps improving e.g. subsequent image analysis tasks.

For more information on the compression method please have a look at this publication:
TM - Technisches Messen (doi:10.1515/teme-2019-0022)

Note

Before calling this function, you have to set the appropriate compression parameters for your camera(s) using **PCO_RecorderSetCompressionParams**, otherwise the function will fail.

Supported camera type(s)

All cameras

Prototype

```
int WINAPI PCO_RecorderCopyImageCompressed (
    HANDLE phRec,                          //in
    HANDLE phCam,                          //in
    DWORD dwImgIdx,                        //in
    WORD wRoiX0,                          //in
    WORD wRoiY0,                          //in
    WORD wRoiX1,                          //in
    WORD wRoiY1,                          //in
    BYTE* bImgBuf,                          //out
    DWORD* dwImgNumber,                    //out
    PCO_METADATA_STRUCT* strMetadata,      //out
    PCO_TIMESTAMP_STRUCT* strTimestamp     //out
);
```

Parameter

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created ***pco.recorder*** object
phCam	HANDLE	HANDLE to a particular camera
dwImgIdx	DWORD	Index of the required image
wRoiX0	WORD	Left horizontal ROI (starting with 1)
wRoiY0	WORD	Upper vertical ROI (starting with 1)
wRoiX1	WORD	Right horizontal ROI (up to image width)
wRoiY1	WORD	Lower vertical ROI (up to image height)
bImgBuf	BYTE*	Pointer to the start address of the buffer the image should be copied to

Continued on next page

Continued from previous page

Name	Type	Description
dwImgNumber	DWORD*	Pointer to a DWORD to get the number of the requested image (can be set to NULL if not relevant)
strMetadata	PCO_METADATA_STRUCT*	Pointer to a PCO_METADATA_STRUCT (see ***pco.sdk*** manual) to get the current metdata of the image if available (can be set to NULL if not relevant)
strTimestamp	PCO_TIMESTAMP_STRUCT*	Pointer to a PCO_TIMESTAMP_STRUCT (see ***pco.sdk*** manual) to get the current binary timestamp information of the image if timestamp is activated (can be set to NULL if not relevant)

Return value

Name	Type	Description
ErrorMessage	int	0 in case of success, Errorcode otherwise.

2.20 PCO_RecorderExportImage

Description Export the image, defined by the transferred index, for the selected camera to the selected file path. Allowed are only raw image formats, i.e. b16, tif, dcm

Supported camera type(s) All cameras

Prototype

```
int WINAPI PCO_RecorderExportImage (
    HANDLE phRec,                    //in
    HANDLE phCam,                    //in
    DWORD dwImgIdx,                  //in
    const char* szFilePath,          //in
    bool bOverwrite                  //in
);
```

Parameter

Name	Type	Description
phRec	HANDLE	Handle to previously created recorder
phCam	HANDLE	Handle to particular camera
dwImgIdx	DWORD	Index of the image that should be read
szFilePath	const char*	File path (including filename and extension) where the file should be saved (File type is automatically detected according to the extension)
bOverwrite	bool	Flag to indicate if the file, when it already exists, should be overwritten

Return value

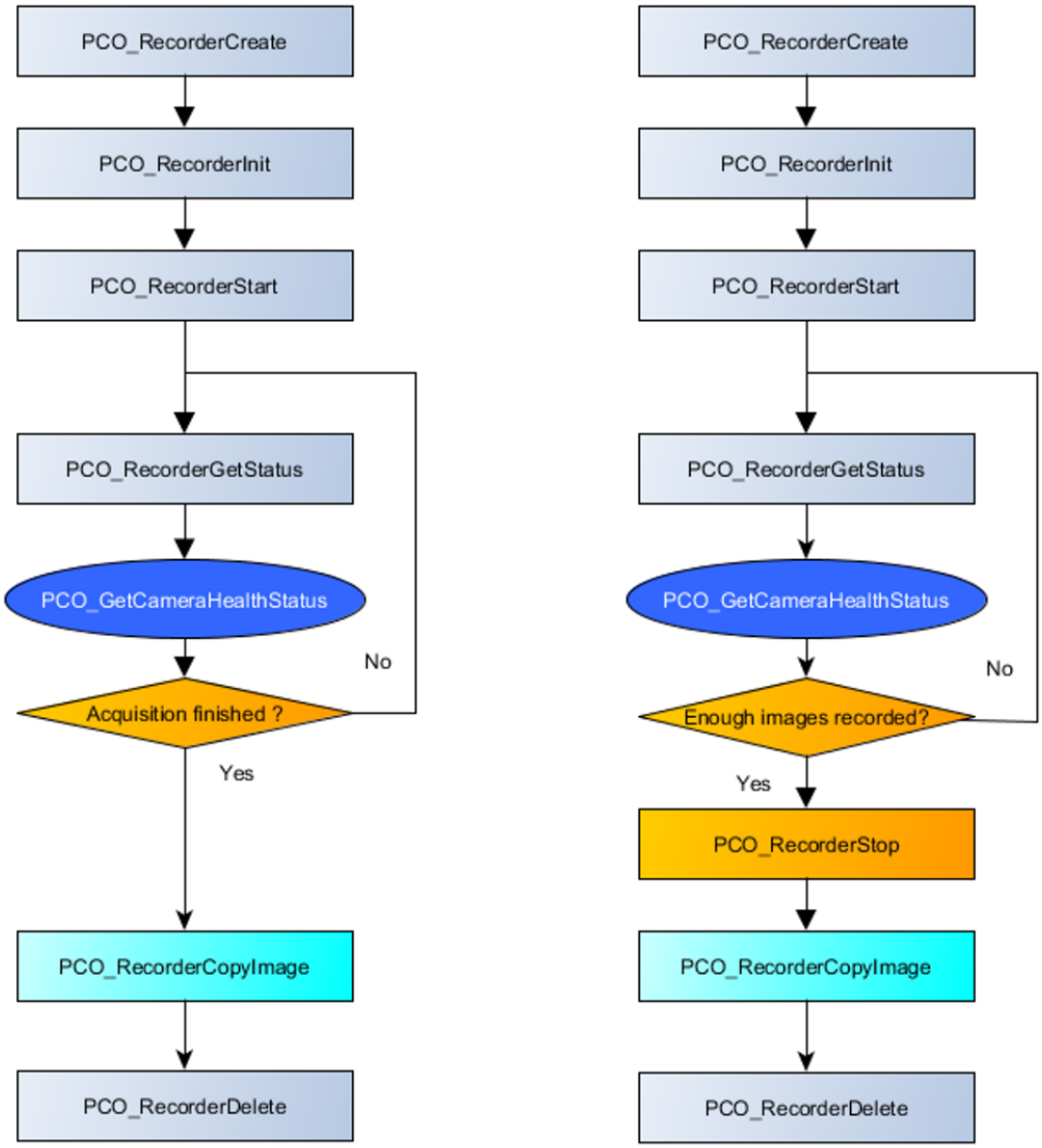
Name	Type	Description
`ErrorMessage`	`int`	0 in case of success, Errorcode otherwise.

3 Typical Implementation

3.1 Basic Workflow

The following flowchart shows two possible basic workflows. The common elements of both are the creation, initialization, and the start of the recording, as well as the final delete of the ***pco.recorder*** object.

The function **PCO_GetCameraHealthStatus** is a standard ***pco.sdk*** function and should be called frequently to prevent the camera from damages (see chapter **1.8**).



The left workflow is similar to the first example in section **3.2**. It uses **PCO_RecorderGetStatus** to wait for the acquisition to finish. This is a default approach for **PCO_RECORDER_MODE_FILE** and **PCO_RECORDER_MODE_MEMORY** with type sequence (see chapter **2.7.1**).

The right diagram shows an approach which is typical for **PCO_RECORDER_MODE_MEMORY** with type ring buffer (see chapter **2.7.1**) or **PCO_RECORDER_MODE_CAMRAM**. Here the number of processed images has to be checked via **PCO_RecorderGetStatus and according** to a defined stop criterion **PCO_RecorderStopRecord** has to be called.

For **PCO_RECORDER_MODE_MEMORY** and **PCO_RECORDER_MODE_FILE** it would also be possible to replace **PCO_RecorderCopyImage** with **PCO_RecorderGetImageAddress**, but since you are working with the internal memory of the ***pco.recorder*** here, you have to be really careful not to cause any application crashes.

3.2 Example Programs

3.2.1 Example for PCO_RECORDER_MODE_MEMORY

```
#include <stdio.h>
#include <tchar.h>
#include <Windows.h>

//SC2 SDK includes
#include "..\..\include\sc2_SDKStructures.h"
#include "..\..\include\sc2_common.h"
#include "..\..\include\sc2_defs.h"
#include "..\..\include\SC2_CamExport.h"
#include "..\..\include\pco_err.h"

//Recorder Includes
#include "..\..\include\PCO_Recorder_Export.h"
#include "..\..\include\PCO_Recorder_Defines.h"

#define CAMCOUNT    1
int _tmain(int argc, _TCHAR* argv[])
{
    int iRet;
    HANDLE hRec = NULL;
    HANDLE hCamArr[CAMCOUNT];
    DWORD imgDistributionArr[CAMCOUNT];
    DWORD maxImgCountArr[CAMCOUNT];
    DWORD reqImgCountArr[CAMCOUNT];

    //Some frequently used parameters for the camera
    DWORD numberOfImages = 10;
    DWORD expTime = 10;
    WORD expBase = TIMEBASE_MS;
    WORD metaSize = 0, metaVersion = 0;

    //Open camera and set to default state
    PCO_OpenStruct camstruct;
    memset(&camstruct, 0, sizeof(camstruct));
    camstruct.wSize = sizeof(PCO_OpenStruct);
    //set scanning mode
    camstruct.wInterfaceType = 0xFFFF;

    hCamArr[0] = 0;
    //open next camera
    iRet = PCO_OpenCameraEx(&hCamArr[0], &camstruct);
    if (iRet != PCO_NOERROR)
    {
        printf("No camera found\n");
        printf("Press <Enter> to end\n");
        iRet = getchar();
        return -1;
    }
```

```
//Make sure recording is off
iRet = PCO_SetRecordingState(hCamArr[0], 0);
//Do some settings
iRet = PCO_SetTimestampMode(hCamArr[0], TIMESTAMP_MODE_OFF);
iRet = PCO_SetMetaDataMode(hCamArr[0], METADATA_MODE_ON,
    &metaSize, &metaVersion);
iRet = PCO_SetBitAlignment(hCamArr[0], BIT_ALIGNMENT_LSB);
//Set Exposure time
iRet = PCO_SetDelayExposureTime(hCamArr[0], 0, expTime,
    2, expBase);
//Arm camera
iRet = PCO_ArmCamera(hCamArr[0]);

//Set image distribution to 1 since only one camera is used
imgDistributionArr[0] = 1;

//Reset Recorder to make sure a no previous instance is running
iRet = PCO_RecorderResetLib(false);

//Create Recorder (mode: memory sequence)
WORD mode = PCO_RECORDER_MODE_MEMORY;
iRet = PCO_RecorderCreate(&hRec, hCamArr, imgDistributionArr,
    CAMCOUNT, mode, 'C', maxImgCountArr);

//Set required images
reqImgCountArr[0] = numberOfImages;
if (reqImgCountArr[0] > maxImgCountArr[0])
    reqImgCountArr[0] = maxImgCountArr[0];

//Init Recorder
iRet = PCO_RecorderInit(hRec, reqImgCountArr, CAMCOUNT,
    PCO_RECORDER_MEMORY_SEQUENCE, 0, NULL, NULL);

//Get image size
WORD imgWidth = 0, imgHeight = 0;
iRet = PCO_RecorderGetSettings(hRec, hCamArr[0], NULL, NULL,
    NULL, &imgWidth, &imgHeight, NULL);

//Start camera
iRet = PCO_RecorderStartRecord(hRec, NULL);

//Wait until acquisition is finished
//(all other parameters are ignored)
bool acquisitionRunning = true;
while (acquisitionRunning)
{
    iRet = PCO_RecorderGetStatus(hRec, hCamArr[0],
        &acquisitionRunning,
        NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);

    DWORD warn = 0, err = 0, status = 0;
    iRet = PCO_GetCameraHealthStatus(hCamArr[0],
        &warn, &err, &status);
    if (err != PCO_NOERROR) //Stop record on health error
        PCO_RecorderStopRecord(hRec, hCamArr[0]);
```

```
        Sleep(100);
    }

    //Allocate memory for one image
    WORD* imgBuffer = NULL;
    imgBuffer = new WORD[(__int64)imgWidth * (__int64)imgHeight];

    //Get number of finally recorded images
    DWORD procImgCount = 0;
    iRet = PCO_RecorderGetStatus(hRec, hCamArr[0], NULL, NULL, NULL,
        &procImgCount, NULL, NULL, NULL, NULL, NULL);

    ////////////////////////////////////////////
    //TODO: Process, Save or analyze the image(s)
    //Here we just read, print image counter and save one tif file
    ////////////////////////////////////////////

    //Get the images and print image counter
    PCO_METADATA_STRUCT metadata;
    metadata.wSize = sizeof(PCO_METADATA_STRUCT);
    bool imageSaved = false;
    DWORD imgNumber = 0;
    for (DWORD i = 0; i < procImgCount; i++)
    {
        iRet = PCO_RecorderCopyImage(hRec, hCamArr[0], i,
            1, 1, imgWidth, imgHeight, imgBuffer,
            &imgNumber, &metadata, NULL);
        if (iRet == PCO_NOERROR)
        {
            printf("Image Number: %lu \n", imgNumber);

            //Save first image as tiff in the binary folder
            //just to have some output
            if (!imageSaved)
            {
                iRet = PCO_RecorderSaveImage(imgBuffer,
                    imgWidth, imgHeight, FILESAVE_IMAGE_BW_16,
                    false, "test.tif", true, &metadata);
                if (iRet == PCO_NOERROR)
                    imageSaved = true;
            }
        }
    }
    delete[] imgBuffer;
    //Delete Recorder
    iRet = PCO_RecorderDelete(hRec);
    //Close camera
    iRet = PCO_CloseCamera(hCamArr[0]);

    return 0;
}
```

3.2.2 Example for PCO_RECORDER_MODE_CAMRAM

```
#include <stdio.h>
#include <tchar.h>
#include <Windows.h>

//SC2 SDK includes
#include "..\..\include\sc2_SDKStructures.h"
#include "..\..\include\sc2_common.h"
#include "..\..\include\sc2_defs.h"
#include "..\..\include\SC2_CamExport.h"
#include "..\..\include\pco_err.h"

//Recorder Includes
#include "..\..\include\PCO_Recorder_Export.h"
#include "..\..\include\PCO_Recorder_Defines.h"

#define CAMCOUNT    1
int _tmain(int argc, _TCHAR* argv[])
{
    int iRet;
    HANDLE hRec = NULL;
    HANDLE hCamArr[CAMCOUNT];
    DWORD imgDistributionArr[CAMCOUNT];
    DWORD maxImgCountArr[CAMCOUNT];
    DWORD reqImgCountArr[CAMCOUNT];

    DWORD procImgCount;
    WORD ramSegment = 1;

    //Some frequently used parameters for the camera
    DWORD numberOfImages = 10;
    DWORD expTime = 100;
    WORD expBase = TIMEBASE_US;
    WORD metaSize = 0, metaVersion = 0;

    //Open camera and set to default state
    PCO_OpenStruct camstruct;
    memset(&camstruct, 0, sizeof(camstruct));
    camstruct.wSize = sizeof(PCO_OpenStruct);
    //set scanning mode
    camstruct.wInterfaceType = 0xFFFF;

    hCamArr[0] = 0;
    //open next camera
    iRet = PCO_OpenCameraEx(&hCamArr[0], &camstruct);
    if (iRet != PCO_NOERROR)
    {
        printf("No camera found\n");
        printf("Press <Enter> to end\n");
        iRet = getchar();
        return -1;
    }
```

```
//Make sure recording is off
iRet = PCO_SetRecordingState(hCamArr[0], 0);
//switch to sequence mode
iRet = PCO_SetRecorderSubmode(hCamArr[0], 1);
//Do some settings
iRet = PCO_SetTimestampMode(hCamArr[0], TIMESTAMP_MODE_OFF);
iRet = PCO_SetMetaDataMode(hCamArr[0], METADATA_MODE_ON,
    &metaSize, &metaVersion);
iRet = PCO_SetBitAlignment(hCamArr[0], BIT_ALIGNMENT_LSB);
//Set Exposure time
iRet = PCO_SetDelayExposureTime(hCamArr[0], 0, expTime,
    2, expBase);
//Arm camera
iRet = PCO_ArmCamera(hCamArr[0]);

//Set image distribution to 1 since only one camera is used
imgDistributionArr[0] = 1;

//Reset Recorder to make sure a no previous instance is running
iRet = PCO_RecorderResetLib(false);

//Create Recorder (mode: cam ram)
WORD mode = PCO_RECORDER_MODE_MEMORY;
iRet = PCO_RecorderCreate(&hRec, hCamArr, imgDistributionArr,
    CAMCOUNT, mode, 'C', maxImgCountArr);

//Set required images
reqImgCountArr[0] = numberOfImages;
if (reqImgCountArr[0] > maxImgCountArr[0])
    reqImgCountArr[0] = maxImgCountArr[0];

//Init Recorder for segment 1 as example, for sequential readout
iRet = PCO_RecorderInit(hRec, reqImgCountArr, CAMCOUNT,
    PCO_RECORDER_CAMRAM_SEQUENTIAL, 0, NULL, &ramSegment);

//Get number of images already in cameras internal memory
iRet = PCO_RecorderGetStatus(hRec, hCamArr[0], NULL, NULL, NULL,
    &procImgCount, NULL, NULL, NULL, NULL, NULL);

//Get width and height to allocate memory
WORD imgWidth = 0, imgHeight = 0;
iRet = PCO_RecorderGetSettings(hRec, hCamArr[0], NULL,
    &maxImgCountArr[0], NULL, &imgWidth, &imgHeight, NULL);

//Allocate memory for image
WORD* imgBuffer = NULL;
imgBuffer = new WORD[(__int64)imgWidth * (__int64)imgHeight];

if (procImgCount > 0)
{
    //If there are already images in the ram segment,
    //you can read them without any previous recording
    // Note: CopyImage is indexed based, so this starts with 0
    iRet = PCO_RecorderCopyImage(hRec, hCamArr[0], 0,
        1, 1, imgWidth, imgHeight, imgBuffer, NULL, NULL, NULL);
```

```
    ////////////////////////////////////////////
    //TODO: Process, Save or analyze the image(s)
    ////////////////////////////////////////////
}

//Start camera
iRet = PCO_RecorderStartRecord(hRec, NULL);

//Wait as long as you want (i.e. for some external event)
int waitTime = 0;
while (waitTime < 10)
{
    //If required you can get a live stream during record
    //(only PCO_RECORDER_LATEST_IMAGE is allowed during record)
    iRet = PCO_RecorderCopyImage(hRec, hCamArr[0],
        PCO_RECORDER_LATEST_IMAGE,
        1, 1, imgWidth, imgHeight, imgBuffer, NULL, NULL, NULL);
    waitTime++;
}
//Stop record
iRet = PCO_RecorderStopRecord(hRec, hCamArr[0]);

//Get number of finally recorded images
iRet = PCO_RecorderGetStatus(hRec, hCamArr[0], NULL, NULL, NULL,
    &procImgCount, NULL, NULL, NULL, NULL, NULL);

////////////////////////////////////////////
//TODO: Process, Save or analyze the image(s)
// Here we just read, print image counter and save one tif file
////////////////////////////////////////////

PCO_METADATA_STRUCT metadata;
metadata.wSize = sizeof(PCO_METADATA_STRUCT);
DWORD imgNumber = 0;
bool imageSaved = false;
//Get the first "numberOfImages" images from
//the cameras internal memory
for (int i = 0; i < (int)numberOfImages; i++)
{
    //Copy the image at index 5 into the buffer
    iRet = PCO_RecorderCopyImage(hRec, hCamArr[0], i,
        1, 1, imgWidth, imgHeight,
        imgBuffer, &imgNumber, &metadata, NULL);
    if (iRet == PCO_NOERROR)
    {
        printf("Image Number: %lu \n", imgNumber);

        //Save first image as tiff in the binary folder
        //just to have some output
        if (!imageSaved)
        {
            iRet = PCO_RecorderSaveImage(imgBuffer,
                imgWidth, imgHeight, FILESAVE_IMAGE_BW_16,
                false,"test.tif", true, &metadata);
```

```
                if (iRet == PCO_NOERROR)
                    imageSaved = true;
            }
        }
    }
    delete[] imgBuffer;
    //Delete Recorder
    iRet = PCO_RecorderDelete(hRec);
    //Close camera
    iRet = PCO_CloseCamera(hCamArr[0]);

    return 0;
}
```

3.2.3 Example for PCO_RECORDER_MODE_FILE

Similar to **PCO_RECORDER_MODE_MEMORY** (chapter **3.2.1**)

4 About Excelitas PCO

Pioneering in Cameras and Optoelectronics (PCO) has been our shared philosophy since our establishment in 1987. Starting with image-intensified cameras, followed by the co-invention of the groundbreaking sCMOS sensor technology, PCO greatly surpassed the imaging performance standards of the day. Acquired by Excelitas in 2021, our PCO camera portfolio continues to forge ahead as a leader in digital imaging innovation across diverse applications such as scientific and industrial research, automotive testing, quality control, and metrology.

With sophisticated mechanical design, extensive software support, and a broad range of accessories, we deliver adaptable solutions for all demands. This adaptability extends to tailor-made firmware and custom image sensors, which allow us to develop highly specialized solutions for all our customers. PCO represents a world-renowned brand of high-performance camera systems that complement Excelitas' expansive range of illumination, optical, and sensor technologies and extend the bounds of our end-to-end photonic solutions capabilities.

Our comprehensive camera portfolio covers the entire spectrum - from deep ultraviolet (DUV) to shortwave infrared (SWIR), from long exposure to high-speed, from line scan to high-resolution area scan. Our camera systems are controlled and processed through an intuitive and powerful software suite addressing an extensive range of platforms and architectures.

pco.®

address:	Excelitas PCO GmbH Donaupark 11 93309 Kelheim, Germany
phone:	(+49) 9441-2005-0 (+1) 866-662-6653 (+86) 0512-6763-4643
mail:	pco@excelitas.com
web:	www.excelitas.com/pco

excelitas®

excelitas.com