user manual

pco.recorder manual

pco.

In case of any questions or comments, please contact us at PCO.

telephone	+49 (0) 9441 2005 50
fax	+49 (0) 9441 2005 20
email	info@pco.de
postal address	PCO AG Donaupark 11 93309 Kelheim, Germany

Released September 2017 © PCO AG

pco.recorder User Manual V1.00 © PCO AG, Germany

TABLE OF CONTENTS

1. GENERAL

This document describes the functionality and usage of the PCO_Recorder. This module is suitable for all types of pco cameras, as long as they are operated in streaming mode. The recorder is built on top of the SDK and forms an API with reduced amount of functions to simplify acquiring and retrieving images compared to the raw pco_sdk functions.

Due to the fact that the API is not thread save, only one instance of the recorder can be created. The first chapter provides a short introduction on how to work with the recorder. An overview of all available functions, described in detail, is shown in **API FUNCTION DESCRIPTION**.

A basic workflow and an example implementation can be found in **TYPICAL IMPLEMENTATION**.

Definition:

SDK	Software Development Kit	SDK is a collection of libraries, sample projects and applications to develop software.
API	Application Programming Interface	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 base functionality of the API is to configure and control the acquisition and storage of a user defined number of images. Therefore two main storage modes can be selected. The image data can either be stored in computer RAM (sequence or ring buffer mode) or in image files of type *.b16, *.tif (single or multi tiff) or *.pcoraw.

The required functions are available through 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 and where to find those DLLs.

Note for file mode: "_CamXX" will be added to the filename, where X specifies the number of the camera as it has been transferred to the recorder. For *.b16 and single-TIFF format also _yyyy will be added, where y specifies the number of recorded images.

1.2 CONVENTIONS

The following typographic conventions are used in this manual:

Bold e.g. PCO_RecorderCreate	Functions, procedures or modes used in this manual, with a cross-reference to the appropriate page
[words in brackets]: [run]	Possible values or states of the described functions
ALL CAPITAL WORDS: TRUE	Logical or boolean values such as TRUE, FALSE, ON, OFF, RISING, FALLING, HIGH, LOW
<words in arrows>: <acq enbl>	Names of hardware input / output signals
`Font Courier New: strGeneral.wSize = sizeof(strGeneral)`	C example code
bold italics	Important terms

1.3 BUILDING APPLICATIONS

At first perform all necessary camera settings for your test setup, because once the recorder is created, the settings must not be changed anymore.
The first function that has to be called is **PCO_RecorderCreate**. Here the user has to transfer all required camera handles to the recorder. ***After the recorder has been created the transferred handles must not be used outside of the recorder until*** **PCO_RecorderDelete** ***was called.*** Otherwise the behaviour will be undefined. The **PCO_RecorderCreate** function delivers the maximum number of recordable images (depending on the recorder type). Considering this upper limit, the recorder can be initialized with the required number of images that shall be recorded using **PCO_RecorderInit**. Calling **PCO_RecorderStartRecord** will start the acquisition. For record-to-file or record-to-memory in sequence mode, the recording will be stopped automatically. For record-to-memory in ring buffer mode the acquisition has to be stopped manually by calling **PCO_RecorderStopRecord**. Calling this function during record will stop the acquisition anyway.

The recorded images can be accessed either by **PCO_RecorderGetImageAddress** which delivers the address of the required image buffer, or by using **PCO_RecorderCopyImage** which copies a defined region of interest (ROI) of the required image in a preallocated buffer. Note that **PCO_RecorderCopyImage** can also be called when acquisition is running whereas **PCO_RecorderGetImageAddress** will be rejected with error during record.

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

When the image processing/analysis has finished, **PCO_RecorderFree** will free all resources. After that either a new initialization with a new number of required images can be performed, or **PCO_RecorderDelete** can be called to close the recorder and delete the handle. Furthermore it is possible to reset the recorder with **PCO_RecorderCleanup**, this will reset the data of all image buffers to 0 or, in file mode, delete all created files, but will not free the resources.

1.4 RUNNING APPLICATIONS

To allow access to the API, the PCO_Recorder.dll, the SC2_Cam.dll and maybe additional interface DLLs must reside in the application directory or in the library search path in case 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 you will have to call LoadLibrary with the complete path then.

1.5 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 to the appropriate library (32 bit or 64 bit) which can be found in the win32 or x64 folders. The API can be invoked by linking to the PCO_Recorder.lib through project settings. Another option is to load the required functions from the PCO_Recorder.dll explicitly at runtime using the LoadLibrary function out of the Windows-API.

1.6 PCO_RECORDER LOGGING

The recorder also supports troubleshooting. If there were problems, you can force the 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 using the 'LOGGING=' parameter in the appropriate **PCO_Recorder_param.ini** file.

2. API FUNCTION DESCRIPTION

2.1 PCO_RecorderCreate

Description:

This function creates the one and only instance of the recorder. It takes an array of handles to the required cameras as input parameter. If the function succeeds, these handles may not be used outside of the recorder until **PCO_RecorderDelete** is called. The main task of this function is to calculate the maximum recordable number of images for every camera by checking the available memory (RAM or hard disk depending on the recorder mode) and the required distribution of the memory assigned to the single cameras (e.g camera 1 should get twice as much available memory as camera 2, than the distribution would be [2, 1]). If the recorder is used in record-to-file mode, also the file path (complete path including the file name) and the file type have to be specified, otherwise those parameters are ignored.

Note: For file mode this function returns a warning, if the file name of the specified path already exists.
The function will be rejected with error if a recorder was already created.

Supported camera type:

All cameras

Prototype:

```
int WINAPI PCO_RecorderCreate (
    HANDLE* phRec,                          //in, out
    HANDLE* phCamArr,                       //in
    const DWORD* dwImgDistributionArr,      //in
    WORD wArrLenght,                        //in
    DWORD dwRecMode,                        //in
    const char* szFilePath,                 //in
    WORD wFileType,                         //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 recorder object is returned
phCamArr	HANDLE*	Pointer to an array of handles to the cameras that should be used by the recorder
dwImgDistributionArr	const DWORD*	Pointer to an array defining the memory distribution between the used cameras (can be set to NULL for equal distribution)
wArrLenght	WORD	Length of all transferred arrays
dwRecMode	DWORD	Required mode of the recorder (see **2.1.1**)
szFilePath	const char*	Path (including the filename) where the image files should be saved (will be ignored if record-to-memory is selected)
wFileType	WORD	Required file type for saving the images (see **2.1.2**) (will be ignored if record-to-memory is selected)

dwMaxImgCountArr	DWORD*	Pointer to an array to get the maximum available image count for each camera (length must be equal to length of the camera handle array)

Return value:

int ErrorMessage	0 in case of success, PCO_WARNING_SDKDLL_RECORDER_FILES_EXIST if the required file already exists, error code otherwise

2.1.1 Recorder Modes

Value	Type	Description
0x00000001	PCO_RECORDER_MODE_FILE	Recorder will save the recorded images as files on disk
0x00000002	PCO_RECORDER_MODE_MEMORY_SEQUENCE	Recorder will save the recorded images in RAM, acquisition stops automatically if required number of images is reached (allocated buffers will not be overwritten)
0x00000004	PCO_RECORDER_MODE_MEMORY_RINGBUF	Recorder will save the recorded images in RAM, acquisition ***will not stop*** automatically if required number of images is reached (allocated buffers will be overwritten instead)

2.1.2 File Types

Value	Type	Description
0x0001	PCO_RECORDER_FILE_TIF	Recorder will save the recorded images in single-TIFF format
0x0002	PCO_RECORDER_FILE_MULTITIF	Recorder will save the recorded images in multi-TIFF format
0x0004	PCO_RECORDER_FILE_PCORAW	Recorder will save the recorded images in pcoraw format
0x0008	PCO_RECORDER_FILE_B16	Recorder will save the recorded images in b16 format

2.2 PCO_RecorderDelete

Description:

This function deletes the recorder object. If necessary, it frees all allocated memory and resources. After this function has succeeded, it is possible to create a new recorder object.
The old recorder handle will be invalid.

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

Supported camera type:

All cameras

Prototype:

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

Parameter:

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created recorder object

Return value:

int ErrorMessage	0 in case of success, error code otherwise

2.3 PCO_RecorderInit

Description:

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

For record-to-memory it will allocated the RAM necessary to store the images. For record-to-file, it checks whether files with the same name are already existent 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 contracted yet.

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

Supported camera type:

All cameras

Prototype:

```
int WINAPI PCO_RecorderInit (
    HANDLE phRec,                    //in
    DWORD* dwImgCountArr,            //in
    WORD wArrLength,                 //in
    WORD wNoOverwrite                //in
);
```

Parameter:

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created recorder object
dwImgCountArr	DWORD*	Pointer to an array for required image counts for all cameras (at least 4 images per camera are required)
wArrLength	WORD	Length of the transferred array
wNoOverwrite	WORD	Flag to decide if existing files should be kept. If not set existing files will be deleted (will be ignored, if record-to-memory is selected)

Return value:

int ErrorMessage	0 in case of success, error code otherwise

2.4 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). Reset means for record-to-file, that all previously recorded image files will be deleted. For record-to-memory the image data in the allocated buffers will be reset to 0.

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

Supported camera type:

All cameras

Prototype:

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

Parameter:

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created recorder object
phCam	HANDLE	HANDLE to the camera that should be reset or NULL if all cameras should be reset

Return value:

int ErrorMessage	0 in case of success, error code otherwise

2.5 PCO_RecorderFree

Description:

This function frees all allocated memory and resources either for a specific camera or for all cameras (if NULL is transferred as camera handle). After this function has succeeded a new initialization of the recorder is necessary in order to acquire images.

Note for record-to-file mode: in contrast to **PCO_RecorderCleanup** this function ***will not delete*** previously recorded image files.

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

Supported camera type:

All cameras

Prototype:

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

Parameter:

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created recorder object
phCam	HANDLE	HANDLE to the camera that should be freed or NULL if all cameras should be freed

Return value:

int ErrorMessage	0 in case of success, error code otherwise

2.6 PCO_RecorderGetSettings

Description:

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

Supported camera type:

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
);
```

Parameter:

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created recorder object
phCam	HANDLE	HANDLE to the camera the settings should be retrieved
dwRecmode	DWORD*	Pointer to a DWORD to get the recorder mode (can be set to NULL if not relevant)
dwMaxImgCount	DWORD*	Pointer to a DWORD to get the maximum number of images (can be set to NULL if not relevant)
dwReqImgCount	DWORD*	Pointer to a DWORD to get the required number of images (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)

Return value:

int ErrorMessage	0 in case of success, error code otherwise

2.7 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:

All cameras

Prototype:

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

Parameter:

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

Return value:

int ErrorMessage	0 in case of success, error code otherwise

2.8 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:

All cameras

Prototype:

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

Parameter:

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

Return value:

int ErrorMessage	0 in case of success, error code otherwise

2.9 PCO_RecorderGetStatus

Description:

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

Supported camera type:

All cameras

Prototype:

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

Parameter:

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created recorder object
phCam	HANDLE	HANDLE to the camera the status should be retrieved
bIsRunning	bool*	Pointer to a bool to get the running 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)
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 at least the required number of images has already been recorded (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:

int ErrorMessage	0 in case of success, error code otherwise

2.10 PCO_RecorderGetImageSize

Description:

This function retrieves the image size of a specified camera.

Supported camera type:

All cameras

Prototype:

```
int WINAPI PCO_RecorderGetImageSize(
    HANDLE phRec,                   //in
    HANDLE phCam,                   //in
    WORD* wWidth,                   //out
    WORD* wHeight                   //out
);
```

Parameter:

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created recorder object
phCam	HANDLE	HANDLE to the camera the image size should be retrieved
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

Return value:

int ErrorMessage	0 in case of success, error code otherwise

2.11 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 **2.11.1**) is set as image index, the address of latest image will be transferred. Function will be rejected with error if the acquisition is running.

Supported camera type:

All cameras

Prototype:

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

Parameter:

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created recorder object
phCam	HANDLE	HANDLE to the camera the image should be retrieved from
dwImgIdx	DWORD	Index of the required image
wImgBuf	WORD**	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

Return value:

int ErrorMessage	0 in case of success, error code otherwise

2.11.1 Image Readout

Value	Type	Description
0xFFFFFFFF	PCO_RECORDER_LATEST_IMAGE	Recorder will address latest recorded image

2.12 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 with an error. If **PCO_RECORDER_LATEST_IMAGE** (see **2.11.1**) is set as image index, the latest image will be copied.

If the recorder mode is record-to-memory in conjunction with ring buffer mode and acquisition is running, it is possible that the required image will be overwritten during copy. In this case the resulting data will be unpredictable. So be careful using the function during this state.
If the mode is record-to-file and acquisition is running, the function will fail for all indices except **PCO_RECORDER_LATEST_IMAGE** (see **2.11.1**).

Supported camera type:

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
    WORD* wImgBuf                   //out
);
```

Parameter:

Name	Type	Description
phRec	HANDLE	HANDLE to a previously created recorder object
phCam	HANDLE	HANDLE to the camera the image should be copied from
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	WORD*	Pointer to the start address of the buffer the image should be copied to

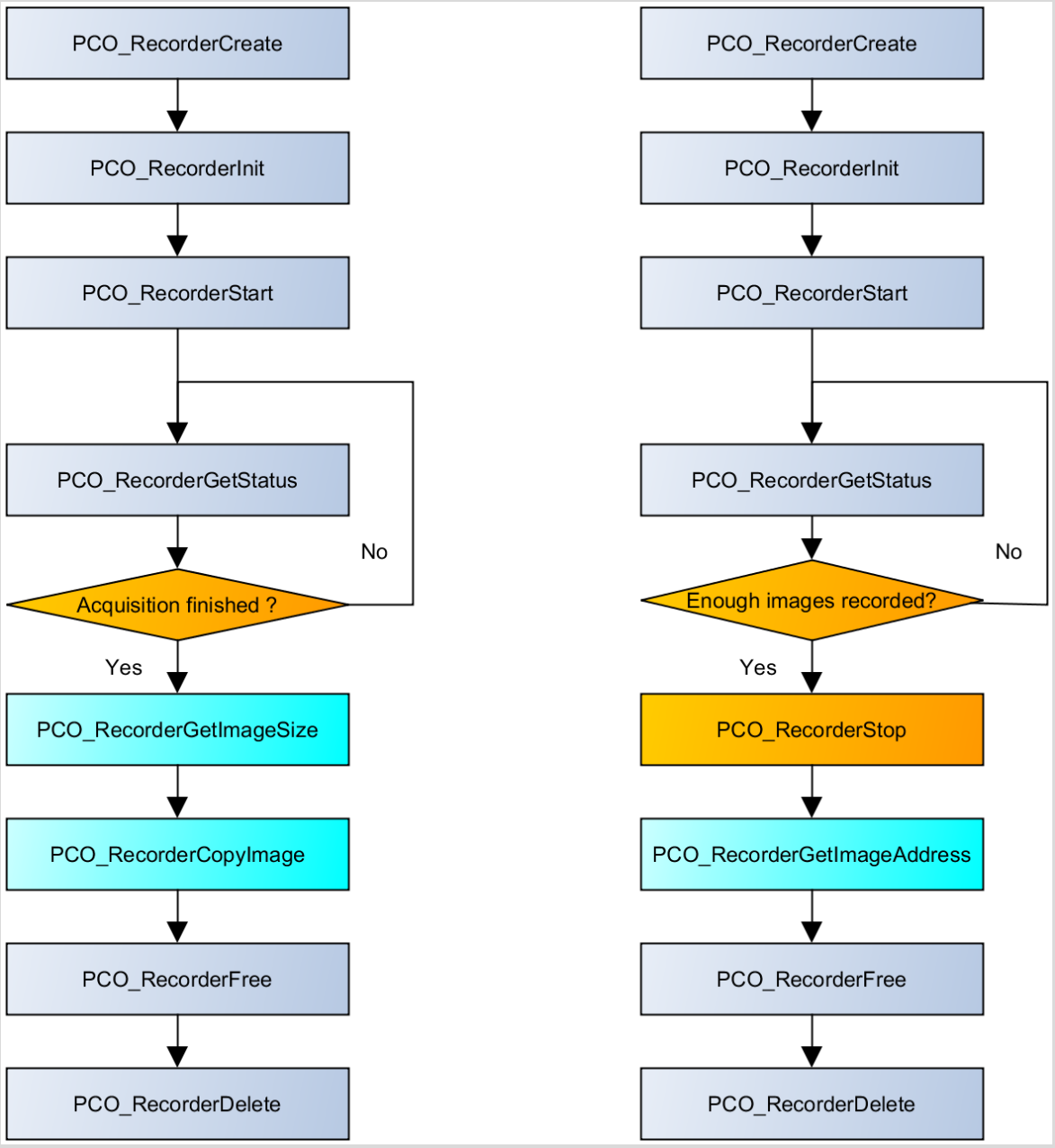
Return value:

int ErrorMessage	0 in case of success, error code otherwise

3. TYPICAL IMPLEMENTATION

3.1 Basic Workflow

The following flowchart shows two possibilities of basic workflows. The common elements of both are the creation, initialization and the start of the recording, as well as the freeing of the resources and the final deletion of the recorder.



The left workflow is similar to the example in section 3.2. It uses **PCO_RecorderGetStatus** to wait for the acquisition to finish. This would be the default approach for record-to-file and record-to-memory sequence mode. The right diagram shows an approach which is typical for the record-to-memory ring buffer mode. Here the number of processed images has to be checked using **PCO_RecorderGetStatus** and according to a defined stop criterion **PCO_RecorderStopRecord** is called.
It mainly depends on the intention of the software, wether **PCO_RecorderGetImageAddress** or **PCO_RecorderCopyImage** in conjunction with **PCO_RecorderGetImageSize** should be used and if the whole image should be processed or if a certain ROI has to be applied.

3.2 Example Program

```
#include <stdio.h>
#include <tchar.h>
#include <Windows.h>
//Recorder Includes
#include "..\include\PCO_Recorder_Export.h"
#include "..\include\PCO_Recorder_Defines.h"
//SC2 SDK includes
#include "..\include\sc2_SDKStructures.h"
#include "..\include\SC2_CamExport.h"
#include "..\include\pco_errt.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];
    bool acquisitionRunning;
    WORD* imgBuffer = NULL;
    WORD imgWidth = 0, imgHeight = 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;
    }
    //set camera to default state, recording off, armed
    iRet = PCO_SetRecordingState(hCamArr[0], 0);
    iRet = PCO_ResetSettingsToDefault(hCamArr[0]);
    iRet = PCO_ArmCamera(hCamArr[0]);

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

    //Create Recorder (mode: memory sequence)
    iRet = PCO_RecorderCreate(&hRec, hCamArr, imgDistributionArr, CAMCOUNT,
                              PCO_RECORDER_MODE_MEMORY_SEQUENCE, NULL, 0, maxImgCountArr);

    //Set required images
    reqImgCountArr[0] = 10;
    if (reqImgCountArr[0] > maxImgCountArr[0])
        reqImgCountArr[0] = maxImgCountArr[0];

    //Init Recorder
    iRet = PCO_RecorderInit(hRec, reqImgCountArr, CAMCOUNT);
    //Start camera
    iRet = PCO_RecorderStartRecord(hRec, NULL);
```

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

        Sleep(100);
    }
    //Allocate memory for image
    iRet = PCO_RecorderGetImageSize(hRec, hCamArr[0], &imgWidth, &imgHeight);
    imgBuffer = new WORD[imgWidth * imgHeight];

    //Copy the image at index 5 into the buffer
    iRet = PCO_RecorderCopyImage(hRec, hCamArr[0], 5, 1, 1, imgWidth, imgHeight, imgBuffer);

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

    delete[] imgBuffer;
    //Delete Recorder
    iRet = PCO_RecorderDelete(hRec);
    //Close camera
    iRet = PCO_CloseCamera(hCamArr[0]);

    return 0;
}
```

ABOUT PCO

In 1987, PCO was founded with the objective to develop and to produce specialized, fast and sensitive video camera systems, mainly for scientific applications. Meanwhile the product range of PCO cameras covers digital camera systems with high dynamic range, high resolution, high speed and low noise, which are sold in the scientific and industrial market all over the world.

Currently PCO is one of the leading manufacturers of scientific cameras. Worldwide representatives, together with our own sales department and technical support assure that we keep in touch with our customers and their needs. The current wide range of specialized camera systems is the result of technical challenge and product specific know-how. A design according to advanced techniques, a high standard of production and strict quality controls guarantee a reliable operation of the cameras. Our own developments in conjunction with an excellent contact to leading manufacturers of image sensors ensure our access to state-of-the-art CCD- and CMOS-technology for our cameras.

Since 2001, PCO is located in its own facility building in Kelheim at the shore of the beautiful and international river Danube. Here in the county Bavaria, which is well known for its excellent support and conditions for high technology companies, we share the benefits of the simple access to high performance products and services in the surrounding area.

Kelheim itself is a historical town, first documented in 866. The small city is founded at the confluence of the Danube and the Altmühl, which has been converted into the Rhine-Main-Danube bypass channel for water transport. Located in Danube-valley, it is the heart of a beautiful river and forest covered lime plateau landscape. It´s landmark, the Hall of Liberation, was built by Ludwig I. in 1863 on the Mount Michael and is visible from all over the city and valley. The beautiful Danube-Gorge, which is protected as natural monument since 1840, is located between Kelheim and the famous abbey Weltenburg.

pco.