pixelfly

SDK-Description

This document describes the API interface to the pixelfly camera series with pco540 PCI-Board.

Table of Contents

1 General

Important:

Camera and PCI-Board control are managed on two levels, represented by the library **pf_cam.dll** and the driver **pco540.sys**.

This description includes in detail all functions of the upper level of the SDK (pf_cam.dll) and some hints how to use it.
For backward compatibility also a version of the pccam.dll is installed with the driver. This dll does use another calling convention and should be used only with existing software. Therefore this SDK-description must not be used for the pccam.dll although a lot of functions are similarly named.

Most of the SDK functions are declared as *int name().* The returned integer value is one of the SDK error code values. A list of all error codes can be found in the header file pco_err.h a short description of each error code in the header file pco_errt.h. Even though the error code is an output of the function, for brevity the "OUT" section of the description of each function does not include this return value.

Hardware and Library
The pixelfly PCI-Board pco540 does not have a large memory, to store images. The images read out from CCD will be buffered in a small FIFO and then sent directly by Master-DMA transfer (without interaction of the PC-CPU) to the main memory of the PC.

This requires a special memory management for the image buffers. Therefore the image buffers are allocated in kernel memory and the images are copied to the user space buffers during image readout.
In the SDK you will find **Memory Control Functions** to allocate and free image buffers, select one or more buffers for grabbing images (set the buffers in a queue). These buffers can be controlled by using the returned buffer number. Additionally, there are functions to directly write to an application-allocated and controlled memory area.

On the board there are three processors, one (PLUTO) for communication with the PC-CPU via PCI-Bus, one (CIRCE) for handling camera timing and one (ORION) for communication with the external world via the high-side drivers and opto-couplers. You can write your own programs for the last one, to manage your special tasks.

The **General Control Functions** are used to open and close the driver. The driver can manage up to eight boards. So if more than one pco540 Board is installed in the PC, the driver creates a unique handle for the selected board, if opened the first time. This unique handle must be used for all subsequent operations with this board. The driver refuses to connect to a given board if the board was opened before from another process with different access rights.

The **Camera Control Functions** are used to control the connected camera and to get status information from the board and the camera. There are functions to start and stop the camera readout at any time, give trigger commands, write data to and read data from the ORION processor and get status information from the camera head.

Because interaction with the ORION processor could be time critical, a tables of ORION commands for each image buffer can be set. The commands in these tables are executed before respectively after the readout to this buffer is done. I.e. this option can be used to get timestamps for each image.

The SDK is ***not thread safe*** in relation to simultaneous operations such as setting the camera and grabbing images on each camera. Two or more cameras can work in independent threads. So it is not possible to setup two or more threads getting images with different settings and sizes. However threading is possible in case the developer takes care for correct thread synchronization, e.g. one thread changes the settings and a second one grabs the images. In this case the second thread has to stop grabbing till the first one has changed the settings. In principle the order of commands shown in the typical implementation must be met.

Migrating from an older SDK:

First step will be to include the new header files, change error reporting and compile your program with the new library
Second step might be to change the image grabbing to the new buffer handling functions.

2 Library Functions

2.1 General Control Functions

2.1.1 Initializing and close

INITBOARD

int **INITBOARD** (int board, HANDLE *hdriver)

This command initializes the PCI-Controller-Boards 0…7. The communication with the board through the driver is tested and the necessary resources are allocated. If no camera head is connected, this function fails and will return the HEAD_LOST error code.
When calling this function with parameter *hdriver set to NULL, the function opens the driver and returns in *hdriver the file handle of the driver for the selected board.
The file handle of the driver is needed in any library function to communicate with a specific board.
If re-initialization is needed during the process flow, call this function with the filehandle according to the board number. Board numbers start from zero. If only one board is installed board number must be 0.

IN

board		number of the PCI-Controller-Board
	=	0…7
hdriver		pointer to filehandle
*hdriver	=	NULL the driver will be opened and the board initialized
*hdriver	=	filehandle of opened driver the board will be initialized

OUT

*hdriver	=	filehandle of the opened driver

CLOSEBOARD

int **CLOSEBOARD** (HANDLE *hdriver)

This command generates a reset of the PCI-Controller-Board and closes the driver. If the function returns without error, *hdriver is also set to NULL.

IN

hdriver		pointer to filehandle
*hdriver	=	filehandle of opened driver

OUT

*hdriver	=	NULL

CHECK_BOARD_AVAILABILITY

int **CHECK_BOARD_AVAILABILITY**	(int board)

This command can be used to see how many boards are installed in the computer and if the driver has been started successfully

IN

board		number of the PCI-Controller-Board, to check
	=	0…7

OUT

2.2 Camera Control Functions

2.2.1 Start and stop camera

START_CAMERA

int **START_CAMERA** (HANDLE hdriver)

This command starts the camera. A new exposure can be initiated with a hardware or software trigger, depending in which mode the camera is set.

IN

hdriver = filehandle of opened driver

STOP_CAMERA

int **STOP_CAMERA** (HANDLE hdriver)

This command stops the camera. Before setting any of the camera parameters, like binning or gain etc. the camera has to be stopped with STOP_CAMERA. When this command has returned without error then the CCD is cleared and ready for setting new parameters or starting new exposures.

IN

hdriver = filehandle of opened driver

TRIGGER_CAMERA

int **TRIGGER_CAMERA** (HANDLE hdriver)

This command sends a single trigger in the software trigger mode. When the camera is set to async shutter software triggered mode a single image is exposed and readout from the camera.
When the camera is set to video software triggered mode the camera starts streaming images until the next STOP_CAMERA command is sent.

IN

hdriver = filehandle of opened driver

2.2.2 Set operating parameter

SETMODE

int **SETMODE**	(HANDLE hdriver int mode, int explevel, int exptime int hbin, int vbin, int gain, int offset, int bit_pix, int shift)

This command sets the parameters of the next exposures.
It cannot be called if the camera is running. All parameters are validated and error WRONGVAL is returned, if one of the parameters has an invalid value for the connected camera.
Defines for all possible parameters can be found in the file pccamdef.h

IN

For exact description of all parameters see notes below

hdriver	=	filehandle of opened driver
mode		operation mode
explevel		level at which time auto exposure is stopped
exptime		exposure time of camera
hbin		horizontal binning
vbin		vertical binning
gain		analog gain of camera
offset		analog offset of camera (not supported)
bit_pix		transfer size per pixel
shift		shift parameter for 8Bit transfer

mode

Set the camera operation mode. Not all modes work with all camera types

In video mode a stream of exposures is started with the next trigger.
If exposure time is shorter than readout time the exposure of the actual image is done at the end of the CCD readout of the previous image.
If exposure time is longer than the readout time the actual exposure is directly following the previous exposure.
In all other modes only one exposure is released by a hardware or a software trigger. The exposure time starts directly after the trigger followed by the readout of the CCD.

=	0x10	single asnyc shutter hardware trigger
=	0x11	single asnyc shutter software trigger
=	0x20	double shutter hardware trigger*
=	0x21	double shutter software trigger*
=	0x30	video mode hardware trigger
=	0x31	video mode software trigger
=	0x40	single auto exposure hardware trigger *
=	0x41	single auto exposure software trigger *

*not available with standard cameras. An error is generated if this feature is not supported by your hardware.

explevel

Only available on cameras with auto-exposure capability. Set the level at which the auto exposure mode is stopped. The camera measures the incoming light and stops the exposure if the light exceeds the set exposure level.
Only valid if mode is set to auto exposure (0x40, 0x41).

explevel	= 0…200 step width = 0.5% 200 = 100% = 4095 counts

exptime

Set the exposure time of the camera. In video mode the value represents times in ms. In all other modes the exposure time is in µs.

single async mode (0x10, 0x11)
exptime = 10…10000

newer models of camera support
exptime = 5…65535

video mode (0x30, 0x31)
exptime = 1…10000

newer models of camera support
exptime = 1…65535

hbin

Set the horizontal binning and region of the camera. This setting affects the readout of the CCD-Chip. Less data is transferred but the readout time is not affected. Wide readout includes 8 dark pixel at the beginning of each line.

hbin	= 0x00000 horizontal x1 normal readout = 0x00001 horizontal x2 normal readout = 0x10000 horizontal x1 wide readout = 0x10001 horizontal x2 wide readout

vbin

Set the vertical binning. This setting affects the readout of the CCD-Chip. Less data is transferred and the readout time is decreased.

vbin	= 0 vertical x1 = 1 vertical x2 = 2 vertical x4 (only VGA cameras)

gain

Set the analog gain of the camera.

gain	= 0 low gain = 1 high gain

offset

Setting the analog offset of the cameras is not supported. This value has no effect on the actual offset. Offset should be set to 0.

bit_pix

Set the bit width of the transferred pixels.

bit_pix	= 12 12 bits per pixel , no shift possible. Two bytes with the upper four bits set to zero are sent. Two pixel values are moved with one PCI (32 bit) transfer. = 8 8 bits per pixel, shift possible. 8 bit values are generated with a programmable barrel shifter from the 12 bit A/D values. Four pixel values are moved with one PCI transfer. This reduces the pixel data per image and less resources on the PCI-Bus are needed

shift

Controls how the 12-bit values are converted to 8 bits by shifting. 12 bit values are shifted left, and the MS 8 bits of the result are extracted. Only valid in 8 Bit per pixel mode

shift	= 0	8 bit (D11...D4), no shift, MS 8 bits are used
	= 1	8 bit (D10...D3), pixel value shifted 1 bit left (x2)
	= 2	8 bit (D09…D2), pixel value shifted 2 bits left ((x4)
	= 3	8 bit (D08…D1), pixel value shifted 3 bits left (x8)
	= 4	8 bit (D07...D0), pixel value shifted 4 bits left (x16)
	= 5	8 bit (D06...D0), pixel value shifted 5 bits (x32)

SET_EXPOSURE

int **SET_EXPOSURE**	(HANDLE hdriver int time)

This command is only available with latest board SW-revisions and in mode single async shutter (0x010 or 0x011). It can be called while the camera is running. The exposure time is changed for the next and the following frames.

IN

hdriver	=	filehandle of opened driver
time		exposure time

2.2.3 Get operation mode and size information

GETMODE

int **GETMODE**	(HANDLE hdriver, int *mode, int *explevel, int *exptime, int *hbin, int *vbin, int *gain, int *offset, int *bit_px, int *shift)

Get actual camera settings

IN

hdriver	=	filehandle of opened driver

OUT

*mode	actual mode value
*explevel	actual auto exposure level
*exptime	actual exposure time
*hbin	actual horizontal binning
*vbin	actual vertical binning
*gain	actual analog gain setting
*offset	0
*bit_pix	actual transfer size per pixel
*shift	actual shift parameter for 8Bit transfer

GETSIZES

int **GETSIZES**	(HANDLE hdriver, int *ccdxsize, int *ccdysize, int *actualxsize,int *actualysize, int *bit_pix)

This command returns the size of the CCD, the actual size in pixel and the dynamics.

IN

hdriver	=	filehandle of opened driver

OUT

*ccdxsize	x-resolution of CCD
*ccdysize	y-resolution of CCD
*actualxsize	x-resolution of current image setting
*actualysize	y-resolution of current image setting
*bit_pix	bits per pixel in image

2.2.4 Get common camera information

GETBOARVAL

int **GETBOARDVAL**	(HANDLE hdriver, int pcc_val, void *data)

This command returns the board parameter selected. Definitions for the selection codes can be found in the file “pccamdef.h”. In this file also defines can be found which help to extract information from the return values. Currently all return values are of type DWORD

IN

hdriver	=	filehandle of opened driver
pcc_val	=	selection code
param	=	pointer to memory address

OUT

*param	=	selected parameter

The following values are defined for pcc_val

PCC_VAL_BOARD_INFO 0x00
General information (type, number, features) for the pci-board and camera is bit coded in the return value

bits 0 – 3	0x0000000F	actual board number
bits 4 – 11	0x00000FF0	actual board typ should be 0x00000410
bits 12 – 32	0xFFFFF000	supported features
	0x00001000	420Line Mode supported
	0x00002000	SVGA camera supported
	0x00004000	HVGA camera supported
	0x00008000	IR mode supported
	0x00010000	Double mode supported
	0x00020000	SET_EXPOSURE supported
	0x00040000	VGA2 cameras supported
	0x00080000	QE cameras supported
	0x00100000	Small size board
	0x00200000	exposure time 5 µs supported
	0x10000000	board initialized

PCC_VAL_BOARD_STATUS 0x01
Current Status of the camera (running, camera-head connected)

bits 0	0x00000001	camera running
bits 1	0x00000002	camera stop pending
bits 24	0x08000000	camera head disconnected

PCC_VAL_CCDXSIZE 0x02
Horizontal resolution in pixel of the connected camera-head

PCC_VAL_CCDYSIZE 0x03
Vertical resolution in pixel of the connected camera-head

PCC_VAL_MODE 0x04
Current mode setting
PCC_VAL_EXPTIME 0x05

Current exposure-time setting

PCC_VAL_EXPLEVEL 0x06
Current exposure-level setting

PCC_VAL_BINNING 0x07
Current binning setting (bit-coded in the returned value)

bit 0	0x00000001	0= vertical binning x1 1= vertical binning x2
bit 4	0x00000010	0= horizontal binning x1 1= horizontal binning x2
bit 7	0x00000080	0= No wide readout 1= wide readout

PCC_VAL_AGAIN 0x08
Current analog gain setting

PCC_VAL_BITPIX 0x09
Current bits per pixel setting

PCC_VAL_SHIFT 0x0A
Current shift setting

PCC_VAL_LASTEXP 0x0C
Exposure time of last grabbed image

PCC_VAL_EXTMODE 0x0D
Capabilities of connected camera head

bit 0	0x00000001	double mode supported
bit 8	0x00000100	prisma installed

PCC_VAL_CCDTYPE 0x0E
CCD-Type of connected camera head

bit 0	0x00000001	0= black&white CCD 1= colour CCD

PCC_VAL_LINETIME 0x0F
time in µs, which is needed to read one line

PCC_VAL_TIMEOUT_PROC 0x20
current timeout value for IO to the main processor on board

PCC_VAL_TIMEOUT_DMA 0x21
current timeout value for the DMA-transfers

PCC_VAL_TIMEOUT_HEAD 0x22
current timeout value after which head connection status is checked

PCC_VAL_FRAMETIME 0x40
estimated time for one frame in µs
(error is returned for CCD's for which a calculation is not available)

PCC_VAL_READOUTTIME 0x41
estimated time for readout of the CCD in µs
(error is returned for CCD's for which a calculation is not available)
PCC_VAL_VBIN 0x42

current vertical binning in decimal notation (1,2)

PCC_VAL_HBIN 0x43
current horizontal binning in decimal notation (1,2)

PCC_VAL_WIDE 0x44
current setting for wide horizontal line (0=disabled,1=enabled)

READVERSION

int **READVERSION** (HANDLE hdriver, int typ, char *vers, int length)

This command returns 'length' characters of the version string from the specified typ. There are version strings for each processor, board hardware, head hardware and programmable CPLD's on board. All version strings consist of ASCII-characters.

IN

hdriver	=	filehandle of opened driver
typ	=	selection
	=	1 PLUTO
	=	2 CIRCE
	=	3 ORION
	=	4 Hardware
	=	5 Head
	=	6 CPLD
vers		pointer to memory address
	=	address of allocated memory
length		bytes to read
	=	size of allocated memory

OUT

*vers	=	version string of selected typ

READTEMPERATURE

int **READTEMPERATURE** (PCC_HANDLE hdriver int *ccd_temp)

This command returns the actual CCD-temperature.
The temperature range is from -55°C to +125°C.

IN

hdriver	=	filehandle of opened driver

OUT

*ccd_temp	= temperature in °C

2.2.5 Orion communication

The following command is for communication with the IO processor on board and can be called at any time.

WRRDORION

int **WRRDORION(**HANDLE hdriver, int cmnd, int *data)

This function writes a command to the ORION-controller and then read the data value answered from the ORION.

IN

hdriver	=	filehandle of opened driver
cmnd		command to send (only the low byte is valid)
*data		data to send (only the low byte is valid)

commands implemented in ORION 2.01:

	=	0x10	rd_portA
	=	0x11	rd_portB
	=	0x13	rd_portD
	=	0x20	rd_portC

OUT

*data	=	data sent back, by the ORION-controller

The driver can call the ORION processor automatically, shortly before and after a DMA transfer is done. With the following functions one can set the commands and data byte, which belongs to every command.
One can send up to 16 commands. If the driver finds a command at the start of the command table, it will catch the data_in byte from the same table position and send it to the ORION processor. After the ORION has finished the command and has written back its data byte, this byte will be stored in the data_back table at the same table position, from where the command is read out. If any command has the value 0x00 or position 16 is reached, the driver will stop sending commands.
Each buffer has its own table, so you can define different commands for each buffer. When allocating a buffer all tables are set to 0x00, so no commands are sent to the ORION processor.

SETORIONINT

int **SETORIONINT(**HANDLE hdriver, int bufnr, int mode, unsigned char *cmnd, int length)

This command writes length bytes to the command or data table for the driver internal ORION call.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER_EX()
mode	=	1 orion data_back post dma
	=	2 orion data_in post dma
	=	3 orion command post dma
	=	4 orion data_back pre dma
	=	5 orion data_in pre dma
	=	6 orion command pret dma
cmnd	=	address of buffer which does hold the command respectively data bytes to set, maximum length of buffer is 16 bytes
length	=	number of bytes to set

GETORIONINT

int **GETORIONINT(**HANDLE hdriver, int bufnr, int mode, unsigned char *cmnd, int length)

This command reads length bytes from the command or data tables for the driver internal ORION call.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER_EX()
mode	=	1 orion data_back post dma
	=	2 orion data_in post dma
	=	3 orion command post dma
	=	4 orion data_back pre dma
	=	5 orion data_in pre dma
	=	6 orion command pre dma
cmnd	=	address of buffer
len	=	length of the buffer

OUT

*cmnd	=	mode =1 data returned from orion post dma
	=	mode =2 data send to orion post dma
	=	mode =3 command send to orion post dma
	=	mode =1 data returned from orion pre dma
	=	mode =2 data send to orion pre dma
	=	mode =3 command send to orion pre dma

2.2.6 Miscellaneous functions

READEEPROM

int **READEEPROM (**HANDLE hdriver, int mode, int adr, unsigned char *data)

This command reads one byte from the EEPROM at the address adr.
Do not call this command while the camera is running!

IN

hdriver	=	filehandle of opened driver
mode	=	0 HEAD-EEPROM
	=	1 CARD-EEPROM
adr	=	address of byte to read (0…255)

OUT

*data	=	returned byte

WRITEEEPROM

int **WRITEEEPROM (**HANDLE hdriver, int mode, int adr, unsigned char data)

This command writes one byte to the EEPROM at the address adr.
Do not call this command while the camera is running!

IN

hdriver	=	filehandle of opened driver
mode	=	0 HEAD-EEPROM
	=	1 CARD-EEPROM
adr	=	address of byte to read (0…255)
data	=	byte to write

SETTIMEOUTS

int **SETTIMEOUTS(**HANDLE hdriver, DWORD dma , DWORD proc, DWORD head)

This command sets timeout values for dma, cardi/o and head connection check.

IN

hdriver	=	filehandle of opened driver
dma	=	timeout in milliseconds for DMA-transfer
proc	=	timeout in milliseconds for cardi/o
head	=	timeout in milliseconds after which head connection status is checked

SET_TIMEOUT_VALUES

int **SET_TIMEOUT_VALUES(**HANDLE hdriver, DWORD *times , int len)

This command sets timeout values for image transfer, dma, cardi/o and head connection check.

IN

hdriver	=	filehandle of opened driver
times	=	address of buffer which hold timeout values
len	=	length of buffer in bytes
times[0]	=	timeout in milliseconds for image requests
times[1]	=	timeout in milliseconds for DMA-transfer
times[2]	=	timeout in milliseconds for cardi/o
times[3]	=	timeout in milliseconds after which head connection status is checked

SETDRIVER_EVENT

int **SETDRIVER_EVENT(**HANDLE hdriver, int mode, HANDLE *hEvent)

This command creates or closes an event handle for driver events.
If the event HANDLE is set to NULL a new event is created as a manual reset event in non-signaled state with default security descriptor and without a name. The internal created event handle will be closed when it is disabled.
If the event HANDLE is not equal to NULL it must be the handle of a valid windows event.
The only event currently defined is the head event.
The head event is set to signalled state when the driver detects that the camera head is connected or disconnected.
Use i.e WaitForSingleObject(*hHeadEvent,TimeOut); to wait and react to these events.

IN

hdriver	=	filehandle of opened driver
mode	=	0x00000000 open and enable head event
	=	0x80000000 close and disable head event
*hEvent	=	NULL create event internally
*hEvent	=	HANDLE of an already created event

OUT

* hEvent	=	internal created event

PCC_GET_VERSION

int **PCC_GET_VERSION**(HANDLE hdriver, char *dll, char *sys)

Get version strings for the dll and the driver. The buffer for the strings must have a size of at least 20bytes

IN

hdriver	=	filehandle of opened driver
dll	=	address of buffer for dll version string
sys	=	address of buffer for driver version string

OUT

*dll	=	address of buffer for dll version string
*ys	=	address of buffer for driver version string

2.3 Image and Buffer Control Functions

For backward compatibility the SDK has included some functions from the previous SDK 1.12. These functions are marked as deprecated and should not be used for new projects.

2.3.1 Readout Image

READ_IMAGE

int **READ_IMAGE** (HANDLE hdriver, int mode, int bufsize, void *bufadr, int timeout)

This function reads the next available image from the camera and writes it into the memory area specified by the pointer. In 'Double Shutter' mode (cf. SETMODE) the two images are read as one data set of double height.
Direct DMA-transfer is used to write the data from the board to the memory region.
The value of bufsize and the allocated data array must be equal or larger than the current image size.
If the camera is set to software triggered mode a trigger command is sent to the camera.

IN

hdriver	=	filehandle of opened driver
mode	=	convert mode, a combination of the following flags
		NORMAL 0x0000
		FLIP 0x0001 (change lines)
		MIRROR 0x0008 (change rows)
bufsize	=	size of data array
bufadr	=	pointer to memory address of data array
timeout	=	timeout in ms to wait for image

OUT

*bufadr	=	image data

If FLIP flag or MIRROR flag is set, the total image is flipped (mirrored horizontal) or mirrored (mirrored vertical) resp. Combination of both flags (mode = FLIP + MIRROR) is possible which results in a 180° rotated image. FLIP and/or MIRROR require additional processing time compared to 'normal'

2.3.2 Image buffer control functions

ALLOCATE_BUFFER_EX

```
int ALLOCATE_BUFFER_EX (HANDLE hdriver, int *bufnr, int size,HANDLE *hPicEvent,void**
                        adr);
```

This command attaches a unique buffer number to a range of memory. Memory allocation can be done by this function or an externally allocated memory can be used.
If value of *adr is NULL on input this command allocates a buffer for the camera in memory and does return the allocated address else *adr must be a pointer to a valid data block. This externally allocated memory block is committed and used in further actions. External allocated memory must not be deallocated before FREE_BUFFER was called. Internal allocated memory must be deallocated with the FREE_BUFFER only.
Additionally, it is possible to attach an event to the newly created buffer. If *hEvent is NULL an event will be created inside the SDK-Dll, else *hEvent must be a Handle to a previouslycreated external event. The internal created event is closed when FREE_BUFFER is called and must not be closed with CloseHandle() function. The external event must not be closed before FREE_BUFFER was called
The value of size has to be set to the number of bytes to be allocated.

To allocate a new buffer, the value of bufnr must be set to –1 (*bufnr=-1).
The return value of bufnr must then be used in subsequent calls to the other Buffer functions. If a buffer should be reallocated or attached to a new address *bufnr must be set to its buffer number and *size to the new size.
If the function fails (error is not 0) the return values of bufnr, event and adr are not valid and must not be used.

IN

hdriver	=	filehandle of opened driver
*bufnr	=	-1 for allocating a new buffer
*bufnr	=	image-buffer number returned from previous ALLOCATE_BUFFER_EX, to reallocate with different size
size	=	size of image-buffer in byte
*hPicEvent		= NULL create event internally
*hPicEvent		= HANDLE of an external created event
*adr		= NULL memory will be allocated internally
*adr		= valid address of external allocated memory

OUT

*bufnr	= number of image-buffer
*hPicEvent	= HANDLE of created event
*adr	= address of allocated memory

FREE_BUFFER

int **FREE_BUFFER** (HANDLE hdriver, int bufnr);

Free allocated buffer and all associated resources.
If the buffer was set into the buffer queue and no transfer was done to this buffer, call REMOVE_BUFFER_FROM_LIST first.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER_EX

SET_BUFFER_EVENT

int **SET_BUFFER_EVENT** (HANDLE hdriver, int bufnr, HANDLE *hEvent)

Create and/or attach an event handle for this buffer. The event is set when the image transfer into the buffer is finished or if a error occurred during the transfer. Use i.e WaitForSingleObject(PicEvent,TimeOut); to wait until a transfer is done

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER()
*hEvent	=	NULL create event internally
*hEvent	=	HANDLE of an event created outside SDK-Dll

OUT

* hEvent	=	HANDLE of created event

CLEARBUFFER_EVENT

int **CLEARBUFFER_EVENT** (HANDLE hdriver, int bufnr, HANDLE *hEvent)

Detach the event handle from this buffer. If the event was internally created, close the event handle

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER()
*hEvent	=	attached HANDLE

OUT

* hEvent	=	NULL if handle is closed

PCC_RESETEVENT

int **PCC_RESETEVENT** (HANDLE hdriver, int bufnr)

Set the event of the buffer to nonsignalled state.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER()

ALLOCATE_BUFFER (deprecated)

int **ALLOCATE_BUFFER** (HANDLE hdriver, int *bufnr, int *size);

This command allocates a buffer for the camera in memory.
The value of size has to be set to the number of bytesto be allocated.
To allocate a new buffer, the value of bufnr must be set to –1 (*bufnr=-1).
The return value of bufnr must then be used in the calls to subsequent Buffer functions. If a buffer should be reallocated *bufnr must be set to its buffer number and *size to the new size.
If the function fails the return values of size and bufnr are not valid and must not be used.

IN

hdriver	=	filehandle of opened driver
*bufnr	=	-1 for allocating a new buffer
*bufnr	=	image-buffer number returned from previous ALLOCATE_BUFFER, to reallocate with different size
*size		size of image-buffer in byte

OUT

*bufnr	=	number of image-buffer
*size	=	allocated size, which might be greater as the size wanted

MAP_BUFFER (deprecated)

int **MAP_BUFFER** (HANDLE hdriver, int bufnr, int size, int offset, void **linadr)

This command maps the buffer and returns the address.
If size is greater than allocated buffer size an error is returned.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER()
size	=	number of bytes to map
offset	=	0

OUT

*linadr	= address of buffer

UNMAP_BUFFER (deprecated)

int **UNMAP_BUFER**(HANDLE hdriver, int bufnr)

This command unmaps the mapped memory region of the buffer.
Please unmap all mapped buffers before closing the driver.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER()

2.3.3 Image buffer queue handling

The SDK-DLL can manage a queue of 64 buffers. Buffers can be set into the queue at any time and will be filled with image data if available.
A buffer cannot be set to the queue a second time.
If other buffers are already in the list the buffer is set at the end of the queue. If no other buffers are set in the queue the buffer is immediately prepared to read in the data of the next image. If an image transfer is finished the driver changes the buffer status and searches for the next buffer in the queue. If a buffer is found, it is removed from the queue and prepared for the next transfer.
To check whether a transfer to one of the buffers is finished, the buffer status can be polled, or wait until the Event of one of the buffers in the queue is set to to signalled state. When waiting for the buffer events, the buffer status must be checked for errors

ADD_BUFFER_TO_LIST

int **ADD_BUFFER_TO_LIST** (HANDLE hdriver, int bufnr, int size, int offset, int data)

Set the buffer with number bufnr into the buffer queue.
.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER_EX
size	=	number of bytes to transfer
offset	=	0 (not implemented)
data	=	0 (not implemented)

If the number of bytes of the transfer does not match the number of bytes which the camera sends to the PCI-board errors may occur in the status of the buffer.

REMOVE_BUFFER_FROM_LIST

int **REMOVE_BUFFER_FROM_LIST** (HANDLE hdriver, int bufnr);

This command removes the buffer from the buffer queue.
If a transfer is in progress to this buffer, an error is returned.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER_EX

ADD_BUFFER

int **ADD_BUFFER** (HANDLE hdriver, int size, void *adr, HANDLE hEvent, DWORD* Status)

Set an external allocated memory buffer into the buffer queue. The SDK-DLL can manage a queue of 64 buffers. A buffer cannot be set to the queue a second time.
When the image transfer into the buffer is finished or if a error occurred during the transfer,the event is set to signalled state.
Use i.e WaitForSingleObject(*hPicEvent,TimeOut); to wait until a transfer is done. Status must then be checked for errors.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER()
size	=	number of bytes to transfer
adr	=	valid address of memory allocated outside SDK-Dll
hEvent	=	HANDLE of an event
Status	=	address of a DWORD, which will be filled with the success status of the buffer

If the number of bytes of the transfer does not match the number of bytes which the camera sends to the PCI-board errors may occur in the status byte of the buffer.

REMOVE_BUFFER

int **REMOVE_BUFFER** (HANDLE hdriver, void * adr);

This command removes the externally allocated buffer from the queue.
If a transfer is in progress to this buffer, an error is returned.
The status of the buffer is set to removed and the buffer event is set to signalled state.

IN

hdriver	=	filehandle of opened driver
*adr	=	valid address of memory which was set into the buffer queue with an ADD_BUFFER() call.

REMOVE_ALL_BUFFERS_FROM_LIST

int **REMOVE_ALL_BUFFERS_FROM_LIST** (HANDLE hdriver);

This command removes all buffers from the buffer queue.
If a transfer is in progress to one of the buffers, an error is returned.
The status of each removed buffer is set to removed and the buffer event is set to signalled state.

IN

hdriver	=	filehandle of opened driver

CANCEL_IMAGES

int **CANCEL_IMAGES** (HANDLE hdriver);

This command removes all buffers from the buffer queue.
If a transfer is in progress to one of the buffers, an error is returned.
The status of each removed buffer is set to removed and the buffer event is set to signalled state.

IN

hdriver	=	filehandle of opened driver

PCC_WAITFORBUFFER

int **PCC_WAITFORBUFFER** (HANDLE hdriver, int nr_of_buffer, PCC_Buflist *bl, int timeout)

This command can wait for the events of one or more buffers. The array of structures PCC_Buflist must be filled with valid buffer numbers. On return each structure is updated with actual buffer status (comp status).

IN

hdriver	=	filehandle of opened driver
nr_of_buffer	=	count of buffers to check
bl	=	pointer to memory area with nr_of_buffer size must be nr_of_buffer*sizeof(PCC_Buflist) bufnr of each structure must be filled with a valid buffer number from ALLOCATE_BUFFER_EX
timeout	=	time to wait in ms

OUT

*bl	=	PCC_Buflists with actual buffer status

2.3.4 Image buffer status information

When the driver queue is used with the image buffer functions, the status of the buffer must be checked either by polling or as part of the event handling for this buffer. The buffer status will show if the image transfer was successful or if an error was encountered.

GETBUFFER_STATUS

int **GETBUFFER_STATUS** (HANDLE hdriver, int bufnr, int mode, int *ptr, int len)

This command returns a given number of status bytes from the internal structure which is allocated for each buffer. The first four DWORDS of this structure are holding useful information.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER_EX
mode	=	0
ptr		pointer to memory address
	=	address of allocated memory
len		bytes to read
	=	4…size of allocated memory

OUT

*ptr = values of internal structure

ptr[0] = comp status flags (compatible to old pccam SDK)

0x00000001	DMA-Write setup and started
0x00000002	DMA-Write finished
0x00000004	Buffer is queued
0x00000008	Buffer was canceled
0x00000010	Buffer has an associated event
0x00000020	Buffer event is set
0x00000040	Buffer is mapped
0x00000080	First Data bytes transfered
0x00000100	Buffer is removed
0x00001000	Error Bit FIFO full
0x00002000	Error Bit Size
0x00004000	Error Bit Io-failure
0x00008000	Error Bit Timeout
0x80000000	Buffer internal allocated

ptr[1]	=	dllstatus flags (compatible to sc2_cam SDK)
	0x80000000	Buffer internal allocated
	0x40000000	Buffer has an associated event
	0x08000000	Buffer is queued
	0x04000000	Buffer io pending
	0x02000000	Buffer remove pending
	0x00800000	Buffer with Orion list pre DMA
	0x00400000	Buffer with Orion list post DMA
	0x00008000	Buffer Event is set
ptr[2]	=	iostatus PCO_NOERROR on Success, else any other PCO_ERROR_... status
ptr[3]	=	image tag the tag is an internal counter, which is increased by one when an buffer is added to the queue. The tag of a buffer is not changed until the buffer is set to the queue again.

GETBUFFER_STATUS_EX

int **GETBUFFER_STATUS** (HANDLE hdriver, int bufnr, DWORD *dllstatus, DWORD *iostatus)

This command returns current dll status and I/O status for the selected buffer.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER_EX
dllstatus		pointer to DWORD
iostatus		pointer to DWORD

OUT

*dllstatus

0x80000000	Buffer internal allocated
0x40000000	Buffer has an associated event
0x08000000	Buffer is queued
0x04000000	Buffer io pending
0x02000000	Buffer remove pending
0x00800000	Buffer with Orion list pre DMA
0x00400000	Buffer with Orion list post DMA
0x00008000	Buffer Event is set

*iostatus = iostatus
PCO_NOERROR on Success, else any other PCO_ERROR_... status

3 Typical Implementation

This typical step by step implementation shows the basic handling.
The camera is set to full resolution, no binning and should grab images with an exposure time of 10ms.
Error handling must be added.
Language C

1. Open the camera

```
int board=0;
HANDLE hdriver=NULL;

err=INITBOARD(board,&hdriver);
```

2. Set camera operating values (exposure time, mode, etc.) and sizes (binning, bitdepth, etc.).

```
int mode=VIDEO_MODE|SW_TRIGGER;
int hbin=0;
int vbin=0;
int gain=0;
int exposure_time=10;

err=SETMODE(hdriver,mode,0,exposure_time
            hbin,vbin,gain,0,12,0);
```

4. Get the sizes and allocate buffer(s)

```
int width,height,ccdxsize,ccdysize,bitpix;
err=GETSIZES(hdriver,&ccdxsize,&ccdysize,
             &width,&height,&bitpix);

int size=width*height*((bitpix+7)/8);
int bufnr=-1;
HANDLE hPicEvent=NULL;
void *adr=NULL;

err=ALLOCATE_BUFFER_EX(hdriver,&bufnr,size,
                       &hPicEvent,&adr);
```

5. Start camera

```
err=START_CAMERA(hdriver);
```

6. Readout directly or

```
err=READ_IMAGE(hdriver,0,size,adr,1000);
```

6. Add your buffer(s) and wait for event

```
err=ADD_BUFFER_TO_LIST(hdriver,bufnr,size,0,0);
err=TRIGGER_CAMERA(hdriver);

DWORD dllstat,drvstat;
DWORD status=WaitForSingleObject(hPicEvent,1000);
if(status==WAIT_OBJECT_0)
 err=GETBUFFER_STATUS_EX(hdriver,bufnr,
                         &dllstat,&drvstat);
 if((dllstat&0x00008000)&&(drvstat==PCO_NOERROR))
 {
//   image ok
 }
```

7. Do a convert and show the image.

Use your own convert routines or use the convert library from PCO (PCO_Conv.dll, see the pco.convert SDK manual for details) to create a displayable (Bitmap) data-array
Display or store the image.

8. Stop the camera.

```
err=STOP_CAMERA(hdriver);
```

9. Free all buffers and close the camera.

```
err=REMOVE_ALL_BUFFERS_FROM_LIST(hdriver);
err=FREE_BUFFER(hdriver,bufnr);
err=CLOSEBOARD(&hdriver);
```

4 Return Codes

The return value of each function is the error code for any error encountered during execution. The error codes are standardized as far as possible. The error codes contain the information of the error layer, the source (microcontrollers, CPLDs, FPGAs) and an error code (error cause). All values are combined by a logical OR operation. Error codes and warnings are always negative values, if read as signed integers, or if read as unsigned word, the MSB is set. Errors have the general format 0x80######, warnings have the format 0xC0######.The error numbers are not unique. Each layer and the common errors have their own error codes. You have to analyze the error in order to get error source. This can easily be done with a call to PCO_GetErrorText.

// e.g.: 0xC0000080 indicates a warning,
// 0x800A3001 is an error inside the SC2-SDK-dll.
// MSB LSB
// XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
// |||| |||| |||| |||| |||| |||| |||| ||||
// |||| |||| |||| |||| |||| ---------- Error or warning code
// |||| |||| |||| |||| ||||
// |||| |||| |||| |||| ---------- Layer code
// |||| |||| |||| ||||
// |||| |||| ---------- Device code
// |||| ||||
// |||---------- reserved for future use
// |||
// ||---------- Common error code flag
// ||
// |---------- Warning indication bit
// |
// ---------- Error indication bit

Error layer:
0x00001000 PCO_ERROR_FIRMWARE // error inside the firmware
0x00002000 PCO_ERROR_DRIVER // error inside the driver
0x00003000 PCO_ERROR_SDKDLL // error inside the SDK-dll
0x00004000 PCO_ERROR_APPLICATION // error inside the application

Error / Warning source:
0x00010000 SC2_ERROR_PCOCAM_POWER_CPLD // error at CPLD in pco.power unit
0x00020000 SC2_ERROR_PCOCAM_HEAD_UP // error at uP of head board in pco.camera
0x00030000 SC2_ERROR_PCOCAM_MAIN_UP // error at uP of main board in pco.camera
0x00040000 SC2_ERROR_PCOCAM_FWIRE_UP // error at uP of firewire board in pco.camera
0x00050000 SC2_ERROR_PCOCAM_MAIN_FPGA // error at FPGA of main board in pco.camera
0x00060000 SC2_ERROR_PCOCAM_HEAD_FPGA // error at FGPA of head board in pco.camera
0x00070000 SC2_ERROR_PCOCAM_MAIN_BOARD // error at main board in pco.camera
0x00080000 SC2_ERROR_PCOCAM_HEAD_CPLD // error at CPLD of head board in pco.camera
0x00090000 SC2_ERROR_SENSOR // error at image sensor (CCD or CMOS)
0x000A0000 SC2_ERROR_SDKDLL // error inside the SDKDLL
0x000B0000 SC2_ERROR_DRIVER // error inside the driver
0x000D0000 SC2_ERROR_POWER // error within power unit
0x00100000 PCO_ERROR_CAMWARE // error in CamWare (also some kind of "device")
0x00110000 PCO_ERROR_CONVERTDLL // error inside the convert dll

Error codes:
Please refer to the file pco_err.h.
Warnings:
Please refer to the file pco_err.h.
In case of successful operation the standard Response Message is returned.
To get detailed error information you can call the function PCO_GetErrorText, which is defined inside the PCO_errt.h header file.

4.1.1 PCO_GetErrorText

Gets a detailed description for an error.
This function is part of the header file pco_errt.h. If you want to use this function include the pco_errt.h header file and define PCO_ERRT_H_CREATE_OBJECT in **one** of your modules.

a.) Prototype:
void PCO_GetErrorText(DWORD dwerr, char* pbuf, DWORD dwlen)

b.) Input parameter:
DWORD dwerr: DWORD which holds the error number.
char* pbuf: Address of the first char of an char array.
DWORD dwlen: DWORD which holds the length of the char array in byte.

pco.
imaging

PCO AG
Donaupark 11
D-93309 Kelheim
fon +49 (0)9441 2005 0
fax +49 (0)9441 2005 20
eMail: info@pco.de
www.pco.de

PCO-TECH Inc.
6930 Metroplex Drive
Romulus, MI 48174
USA
www.pco-tech.com