SDK

Software Development Kit

Windows 95/98/NT/2000
Version 5.13

hsfc pro dicam pro sensicam

pco.
imaging

Table of Contents

Commands

Camera adjustments and initialization

Starting an exposure and reading out the CCD

Interrogating the status and settings

Reading images and converting

Dialog functions

Recorder- and multibuffer functions

Extended image functions

Extended camera adjustment functions

Logging functions

SDK95/98/NT/2000 - 32Bit
Software Development Kit for SensiCam

This DLL interface is compatible to Windows95/98 and WindowsNT/2000.

Basics

Windows95/98 Camera and PCI Interface Board control is managed on two levels, represented by the 32 Bit DLLs **sen95cam.dll** and the virtual device driver **senmem.vxd**, which providing the interface to the hardware and carrying out the PCI Bus Protocol functions. **sen95cam.dll** contains commands to operate the camera and to transfer the image data from the camera to the PC memory. This user interface consists of the functions which are described below.

WindowsNT/2000 Camera and PCI Interface Board control is managed on two levels, represented by the 32 Bit DLL **senntcam.dll** and the device driver **sennt.sys**, with **sennt.sys** providing the interface to the hardware and carrying out the PCI Bus Protocol functions. **senntcam.dll** contains commands to operate the camera and to transfer the image data from the camera to the PC memory. This user interface consists of the functions which are described below.

Variables Most of the defined functions return an integer variable (int, 32 Bit) indicating whether the function was terminated successfully.
A negative value represents an error message, whereas positive values are warnings , 0 indicates ‘no error occurred’.
If the function creates an error code, you should try to detect the reason for the error. The error code table at the end of this manual might be a helpful tool for this purpose.
The parameters for a function are transferred as integer variables (32 Bit). Tables and file names (strings) are transferred as pointers. To functions returning a value, a pointer of the respective type is handed over. The function writes the return values to the memory space pointed to by the passed pointer. Please ensure enough buffer space is available for this purpose.
If no explicte comment is given, all numbers are decimal and
all hexadecimal numbers have the prefix 0x.

Note: **For exact declaration of each function see header file sencam.h**

Dialog functions The SDK allows any program to send the settings which have been selected by an user to the camera. Alternatively, there is the option to open a dialog window as an interface to the camera.
Additionally, these dialog functions check the various camera types and use the corresponding input fields. We recommend to use these dialog functions whenever possible in order to ensure that new camera types and applications are handled by a new SDK version Thus changes in your source code will be minimized.

General All camera operations are defined by **C**amera **O**peration **C**ode tables (COC) . This tables are built and loaded to the PCI Board with the SET_COC command. RUN_COC starts the execution of the COC resulting in an image transfer from the camera to the PCI Board. STOP_COC stops the execution of the COC.
The PCIBoard contains two buffers to which the images from the camera are transferred. These buffers can be read out to the PC memory using any of the image readout functions i.e. READ_IMAGE_12BIT(). The buffers are organized as FIFO (first image transferred from camera is the first to be read out) and allow the readout from one buffer while transferring an image from the camera to the other buffer.

The installation of the SDK will copy the necessary files to your computer. Please see the readme file which is included in the installation for further details.

This library is compiled with Microsoft Visual C, Version 6.0.
(Other versions are available on request)

Demo Programs On the CDROM you will find two demo programs in the following subdirectories:

...\ sensicpp *.*	for C++ (compiler 6.0)
...\ sensi_c *.*	for C (compiler 6.0)

To run and or compile the programs, please copy the complete directory to your hard disk.

Typical Implementations

A typical implementation of the SensiCam using the SDK would proceed in the following steps:

0. SET_BOARD	Necessary if multi boards are used
1. SET_INIT	Initialization of the camera and of the hardware
2. GET_STATUS	Read the camera type
3. SET_COC	Set the camera adjustments, depending on the camera type
4. RUN_COC	Start an exposure
5. GET_IMAGE_STATUS	Check whether an image is available

in case an image is available:

6. GET_IMAGE_SIZE	Read the resulting image size
7. READ_IMAGE_12BIT	Read the image data
8. STOP_COC	Stop an exposure and reset the image memory

in case of a presentation of a b/w image in VGA:

9. LOAD_OUTPUT_LUT	Load the b/w Look-up-table
10. CONVERT_BUFFER_12TO8	Convert to 8 Bit

in case of an RGB image in VGA:

9. LOAD_COLOR_LUT	Load the color Look-up-table table
10. CONVERT_BUF_12TOCOL	Convert to RGB

ending the program:
11. SET_INIT

The built-in dialog functions open, upon call, a dialog window and carry out an interactive parameter setting. If adjustments were changed, the data is automatically sent to the camera. Calling the functions SET_COC or LOAD_LUT is not necessary in this case. The dialog window has the option to send messages to the main application window to ensure that the main window can follow up any parameter changes.

When opening the dialog it automatically checks the attached camera and chooses the appropriate input fields. A program run with the dialog functions would look like this:

1. SET_INIT	Initiate the camera and the hardware
2. OPEN_DIALOG_CAM	Check and set all camera settings
3. RUN_COC	Start an exposure
4. GET_IMAGE_STATUS	Check if there is an image available

if available, then:

5. GET_IMAGE_SIZE	Read the resulting image size
6. READ_IMAGE_12BIT	Read the image data
7. STOP_COC	Stop the exposure and reset the image memory

in case of a presentation of a b/w image in VGA:

8. OPEN_DIALOG_BW	Read and set the b/w Look-up-table
9. CONVERT_BUFFER_12TO8	Convert to 8 Bit

in case of a presentation of an RGB image in VGA:

8. OPEN_DIALOG_COL	Read and set the color Look-up-table
9. CONVERT_BUF_12TOCOL	Convert to RGB

ending the program:

10. CLOSE_DIALOG_CAM and/or
CLOSE_DIALOG_BW and/or
CLOSE_DIALOG_COL and/or
SET_INIT

Camera adjustments and initialization

int **SET_BOARD** (*int* number)

If you have more than one PCI Interface Board installed in your computer, you can set the board which is addressed by following calls of SDK functions which don't have an own board parameter. The first board starts with number 0. After calling SET_BOARD all following SDK commands will directed to the selected board, until the next SET_BOARD command is called.

Parameters [in]

number	number of PCI Interface Boards to work with range 0 … 4 + flag*256	
flag	0:	with COC reprogramming
	1:	without COC reprogramming

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

There is no need to use this command if only one board is installed in the computer. In this case this board gets automatically the number 0.
If you use more than one board, call this command **first** and then SET_INIT() . You have to repeat this sequence for each board.

Note: The SET_BOARD call cleans the buffers on the board and reprograms the COC. If you do not want the buffers to be cleared and the COC to be reprogrammed, simply add 256D (100H) to the board number value.

int **GET_BOARD** (*int* *boardnr)

This function returns the number of the board, which is currently in use.

Parameters [out]

bordnr	number of PCI Interface Boards

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **SET_INIT** (*int* mode)

This function resets the PCI Interface Board hardware as well as the camera to default values. It checks if the camera is connected and the PCI interface board is installed.

Note: SET_INIT(1) or (2) has to be called before any other function calls (except SET_BOARD) Call SET_INIT(0) to close a selected board and for all boards before closing the application

Parameters [in]

mode	Initialization mode	
	0:	terminate driver, shutdown
	1:	initialize camera, start with standard parameters, without dialog dll's
	2:	initialize camera,start with stored parameters, loading dialog dll's

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

For initialization in mode=1, the standard parameters for exposure time, resolution, etc.) are used, whereas initializing in mode=2 means that the parameter saved most recently in the windows registry is used.

If no Registry Key has been generated or no data can be found under 'HKEY_CURRENT_USER\\Software\\PCO\\Camera-Settings', a new Registry Key with standard parameters is generated when the OPEN_DIALOG_CAM function is called the first time. You should choose mode=2 when using the 'dialog' commands, since these functions also get their parameters from the Registry.

SET_COC for SensiCam

int **SET_COC** (*int* mode, *int* trig, *int* roix1, *int* roix2, *int* roiy1, *int* roiy2, *int* hbin, *int* vbin, *char* *table)

This function generates a COC (**C**amera **O**peration **C**ode) which is loaded into the program memory of the camera. All parameters are checked to ensure that a valid set is generated. If any of the parameters is wrong the function returns WRONGVAL. To get a valid set of parameters TEST_COC() can be used.

Parameters [in]
For exact description of all parameters see notes below

mode	operation mode
trig	trigger and start mode (auto, hw, …)
roix1,roix2	horizontal ROI (Region of Interest)
roiy1,roiy2	vertical ROI (Region of Interest)
hbin	horizontal binning
vbin	vertical binning
table	pointer to zero terminated ASCII string with values for delay and exposure times

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

The following SensiCam camera types are avaiable at the moment:
‘Long Exposure’, ‘Long Exposure QE’, ‘QE Standard’, ‘QE Double Shutter’, ‘Fast Shutter’ and ‘Double Shutter’.
Specific settings can only be made for distinct types as described below.

mode

Set the camera typ, operation mode and analog gain. It is a combination of the following parameters
(**typ**+(**gain***256)+(**submode***65536)) respectively
((**typ**&0xFF) | ((**gain**&0xFF)*<<8) | ((**submode**&0xFF)<<16))
See also the defines in the cam_types.h file

typ

Long Exposure:
The typ ‘Long Exposure’ is for the use with the ‘Long Exposure’ and all ‘QE’ versions of the SensiCam.

typ	0	Long Exposure (M_LONG)	
gain	0	normal analog gain	
	1	extended analog gain	
	3	Low Light Mode[1]	
submode	0	sequential, busy out	(NORMALLONG)
	1	simultaneous, busy out	(VIDEO)
	2	sequential, expos out[2]	(MECHSHUT)
	3	simultaneous, expos out*	(MECHSHUTV)
	8	fast QE, busy out[3]	(QE_FAST)
	9	double QE, busy out[4]	(QE_DOUBLE)

[1] only for cameras SensiCam QE, SensiCam QE Standard and SensiCam QE Double

[2] the TRIG IN BNC plug at the rear of the PCI-Board is used as an output. The Signal on this output follows the exposure time in default mode. Setting additional values in the exposuretime string alters the output signal.
For exact description contact PCO support.

[3] only for cameras SensiCam QE Standard and SensiCam QE Double

[4] only for camera SensiCam QE Double

submode

NORMALLONG:
In the ‘Sequential’ mode delay, exposure and CCD readout are done sequentially, i. e. in chronological order. All possible trigger combinations are allowed.

VIDEO:
The mode ‘Simultaneous’ does not allow a delay setting. Exposure and CCD readout are done simultaneously. The longer duration of either exposure time or readout time determines the maximum achievable repetition rate. For exposure times, which are longer as twice readout time using the mode NORMALLONG is recommended. The only allowed trigger combinations are trig = 0x000, trig = 0x100, trig = 0x200.

MECHSHUT:
BNC-Plug at the PCI-Board is used as an output to monitor exposure time. No trigger settings are possible. Delay, exposure and CCD readout are done sequentially

MECHSHUTV:

BNC-Plug at the PCI-Board is used as an output to monitor exposure time. No trigger settings are possible. Exposure and CCD readout are done simultaneously

QE_FAST:
Sequential mode with possibility to set short exposure times. All trigger combinations are allowed.

QE_DOUBLE:
Two images are taken in a sequence which is started by the external trigger input 'TRIG' on the PCI Interface Board. This sequence can be started by an rising edge (trig = 1) or by a falling edge (trig = 2). The two exposed images are linked together to one data set (one image with double height is transferred, see also GET_IMAGE_SIZE). The interframing time between the two images has to be at least 500ns.

typ Fast Shutter:
The typ Fast Shutter is for the use with the Fast Shutter version of the SensiCam.

typ	1	Fast Shutter (M_FAST)	
gain	0	normal analog gain	
	1	extended analog gain	
submode	0	standard	(NORMALFAST)
	5	cycle	(CYCLE)

submode NORMALFAST:
Single and multiple exposures with delay and exposure times between 100ns and 1ms can be done. On a single trigger the complete time table is started. All possible trigger combinations are allowed.

CYCLE:
In this mode, every exposure is synchronized with an external trigger event. Only external trigger modes are allowed.Every delay-, exposuretime pair can be repeated up to 1000 times. Each event must be released by its own trigger pulse. This means if you set e.g. 20 exposure times, you need 20 trigger events to generate **one** image.

typ

Double Shutter:
The typ Fast Shutter is for the use with the Double Shutter version of the SensiCam.

typ	1	Double Shutter (M_FAST)	
gain	0	normal analog gain	
	1	extended analog gain	
submode	0	standard	(NORMALFAST)
	1	double 200ns	(DOUBLE)
	2	double 1µs	(DOUBLEL)
	5	cycle	(CYCLE)

To be compatible with older versions,
'Double Shutter 200ns' is also typ = 2 and
'Double Shutter 1µs' is also typ = 3.

submode

NORMALFAST:
See Fast Shutter mode.

DOUBLE:
Two images are taken in a sequence which is started by the external trigger input 'TRIG' on the PCI Interface Board. This sequence can be started by an rising edge (trig = 1) or by a falling edge (trig = 2). The two exposed images are linked together to one data set (one image with double height is transferred, see also GET_IMAGE_SIZE). The interframing time between the two images has to be at least 200ns. This short interval time between the two pictures results in an increased blooming effect

DOUBLEL:
Two images are taken in a sequence which is started by the external trigger input 'TRIG' on the PCI Interface Board. This sequence can be started by an rising edge (trig = 1) or by a falling edge (trig = 2). The two exposed images are linked together to one data set (one image with double height is transferred, see also GET_IMAGE_SIZE). The interframing time between the two images has to be at least 1000ns. This submode offers a widely reduced blooming effect compared to submode DOUBLE.

CYCLE
See Fast Shutter mode.

trig

Set the camera trigger mode. When "trig" is set to any external trigger mode delay and exposure times are started with a TTL-trigger signal applied at the external trigger input "TRIG" of the PCI Interface Board.

For SensiCam LongExposure and QE

trig	0x000	auto start, auto frame
	0x001	auto start, frame with external rising edge
	0x002	auto start, frame with external falling edge
	0x100	sequence start with external rising edge [1]
	0x200	sequence start with external falling edge [1]
	0x101	sequence + frame start with external rising edge [1]
	0x202	sequence + frame start with external falling edge [1]

All other combinations are not allowed!

[1] These modes will work only with PCI Interface Boards with revision code 17 and later. Please ask factory.

There are three different modes to trigger the camera:

- **auto start (trig = 0x000, 0x001, 0x002)**
  Each frame will be triggered, either by an internal software trigger or by an external trigger signal.
- **sequence start (trig = 0x100, 0x200)**
  An external trigger signal starts a complete sequence.
- **sequence + frame start (trig = 0x101, 0x202H)**
  The first external trigger starts a sequence, the second trigger starts the first exposure (frame). The following exposures must be triggered, too.

For SensiCam FastShutter and DoubleShutter

trig	0	no external synchronization
	1	external falling edge
	2	external rising edge

All other combinations are not allowed!

roix1, roix2

Set the start and end value for the horizontal Region of Interest (ROI). One unit is 32 pixels. This setting affects the readout of the CCD-Chip. Less data is transferred, but readout time is not changed.

roix1	start value of horizontal ROI	
roix2	end value of horizontal ROI	
	range 1 ... 20	for CCD chip type 640 x 480
	range 1 ... 40	for CCD chip type 1280 x 1024
	range 1 ... 43	for CCD chip type 1376 x 1040

roiy1, roiy2

Set the start and end value for the vertical Region of Interest (ROI). One unit is 32 pixels. This setting affects the readout of the CCD-Chip. Less data is transferred and the readout time is decreased.

roiy1	start value of vertical ROI	
roiy2	end value of vertical ROI	
	range 1 ... 15	for CCD chip type 640 x 480
	range 1 ... 32	for CCD chip type 1280 x 1024
	range 1 ... 33	for CCD chip type 1376 x 1040

Thus the smallest ROI is 32 pixels in square.
To get for example the upper right corner 32*32 pixel the ROI settings should be roix1=1, roix2=1, roiy1=1, roiy2=1.
In the case of the ‘Double Shutter’ camera, the ROI is set for the two half images which are then transferred as one data set of double height.

hbin

Set the horizontal binning. This setting affects the readout of the CCD-Chip. Less data is transferred but the readout time is not changed.

hbin	horizontal binning	
	1	no binning
	other possible values	
	2, 4, 8	

vbin

Set the vertical binning. The maximal vertical binning setting depends on the selected vertical ROI . This setting affects the readout of the CCD-Chip. Less data is transferred and the readout time is decreased.

vbin	vertical binning 1 no binning other possible values SVGA 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 other possible values VGA 2, 4, 8, 15, 16, 30, 32, 60, 64, 120, 128, 240, 256, 480 other possible values SensiCam QE 2, 4, 8, 16

table

Set the delay and the exposure times. The parameter is a pointer to a zero terminated ASCII string. The time-values are separated by comma (,) or space ( ). The array is concluded by the sequence “-1,-1”, so that variable key lengths can be handed over. The characters ‘CR’ (13D) and ‘LF’ (10D) may be used to structure the input string.

table	pointer to string array

a) Long Exposure (NORMALLONG)

string array

```
DELAY ,    EXPOS_WIDTH,
-1    ,    -1
```

The delay and exposure time is in ms with a range from 0 to 1,000,000 for DELAY and from 1 to 1,000,000 for EXPOS_WIDTH, in steps of 1 ms.
Exactly one pair of values is expected.

b) Long Exposure (VIDEO)

string array

```
0     ,    EXPOS_WIDTH,
-1    ,    -1
```

The exposure time is in ms with a range from 1 to 1,000,000 for EXPOS_WIDTH, in steps of 1 ms. Exactly one pair of values is expected.

c) Long Exposure (MECHSHUT)

string array

```
        DELAY ,          EXPOS_WIDTH,
        AV     ,         AV
        START ,          STOP
        -1     ,         -1
```

The delay and exposure time is in ms with a range from 0 to 1,000,000 for DELAY and from 1 to 1,000,000 for EXPOS_WIDTH, in steps of 1 ms.
If both AV values are ‘-1’, the default values are used for start and stop time of the output Signal
If both AV values are ‘0’ the start and stop time of the output signal is calculated according to the START and STOP values given.
Range of START is DELAY*(-1) to EXPOS+STOP-1.
Negative values set the start time of output signal before exposure time start, positive values after exposure time start.
Range of STOP is (DELAY*(-1))+1 to 1,000,000
Negative values set the stop time of output signal before exposure time ends, positive values after exposure time end.

d) Long Exposure (MECHSHUTV)

string array

```
        0      ,         EXPOS_WIDTH,
        AV     ,         AV
        START ,          STOP
        -1     ,         -1
```

The exposure time is in ms with a range from 0 to 1,000,000 for EXPOS_WIDTH, in steps of 1 ms.
If both AV values are ‘-1’, the default values are used for start and stop time of the output Signal
If both AV values are ‘0’ the start and stop time of the output signal is calculated according to the START and STOP values given.
Range of START is 0 to EXPOS+STOP-1.
No negative values are allowed, positive values set the start time of output signal after exposure time start.
Range of STOP is EXPOS*(-1) to 1,000,000
Negative values set the stop time of output signal before exposure time ends, positive values after exposure time end.

e) Long Exposure (QE_FAST)

string array

```
DELAY ,        EXPOS_WIDTH,
-1       ,     -1
```

The delay and exposure time is in ns with a range from 0 to 50,000,000 for DELAY and from 500 to 10,000,000 for EXPOS_WIDTH, in steps of 100 ns (nearest value is selected, get exact time with GET_EXPTIME). Exactly one pair of values is expected.

f) Long Exposure (QE_DOUBLE)

string array

```
-1       ,     -1
```

The exposure times for the two half images are determined by the sequence of the TRIG input signal on the PCI Interface Board. No timevalues are given in this mode

g) Fast Shutter (NORMALFAST)

string array

```
        delay1          , expos_width1,
        delay2          , expos_width2,
                        ...
        delay100        , expos_width100,
        -1              , -1
```

All delay and exposure times are set in ‘ns’ with a range from 0 to 1,000,000, in steps of 100 ns.
Up to 100 pairs of values are possible!

d) Fast Shutter (CYCLE)’

string array

```
        1               , cycle1,
        delay1          , expos_width1,
        1               , cycle2,
        delay2          , expos_width2,
                        ...
        1               , cycle50,
        delay50         , expos_width50,
        -1              , -1
```

All delay and exposure times are set in ‘ns’ with a range from 0 to 1,000,000, in steps of 200 ns.
The cycle value must be in the range of 1 ... 1000.
Up to 50 pairs of values are possible.
Every delay, expos_width must be triggered externally.
Parameter **int trig**: trig = 1 or trig = 2
Delay + expos_width must be ≥ 1µs.

d) Double Shutter

string array

```
        -1      ,       -1
```

The exposure times for the two half images are determined by the sequence of the TRIG input signal on the PCI Interface Board. No timevalues are given in this mode

Examples

Example 1
An ROI of a 640 x 480 sensor with 32 pixels horizontal and 64 pixels vertical in the top right corner has the following settings:

```
int roix1, roix2 = 20, 20;
int roiy1, roiy2 = 1, 2;
```

Example 2
If in addition to the situation in example 1 a horizontal binning of 2 pixels (hbin = 2) and a vertical binning of 16 lines (vbin = 16) is set, the image size is reduced to 16 x 4 pixels.

SET_COC for DiCAM-PRO

int **SET_COC** (*int* mode, *int* trig, *int* roix1, *int* roix2, *int* roiy1, *int* roiy2, *int* hbin, *int* vbin, *char* *table)

This function generates a COC (**C**amera **O**peration **C**ode) which is loaded into the program memory of the camera. All parameters are checked to ensure that a valid set is generated. If any of the parameters is wrong the function returns WRONGVAL. To get a valid set of parameters TEST_COC() can be used.

Parameters [in]

For exact description of all parameters see notes below

mode	operation mode
trig	trigger and start mode (auto, hw, …)
roix1,roix2	horizontal ROI (Region of Interest)
roiy1,roiy2	vertical ROI (Region of Interest)
hbin	horizontal binning
vbin	vertical binning
table	pointer to zero terminated ASCII string with values for DiCAM-PRO specials, delay and exposure times

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

mode

Set the camera typ, operation mode and analog gain. It is a combination of the following parameters
(**typ**+(**gain***256)+(**submode***65536)) respectively
((**typ**&0xFF) | ((**gain**&0xFF)*<<8) | ((**submode**&0xFF)<<16))
See also the defines in the cam_types.h file

typ	5	Dicam Pro (M_DICAM)
gain	0	normal analog gain
	1	extended analog gain
submode	0	single trigger mode
	1	multi trigger mode
	2	double trigger mode

submode DPSINGLE:
A single exposure is released with one trigger event and stored into one frame.

DPMULTI:
Multiple exposures are released with one trigger event and stored into one frame.

DPDOUBLE:
A double exposure with short interframing time is released with one trigger event and stored into two frames.

trig

Trigger setting of Dicam-Pro is done in the string-table, so trig must always be set to zero.

trig	0:	auto start, auto frame
All other values are not allowed!		

roix1, roix2

Set the start and end value for the horizontal Region of Interest. One unit is 32 pixels. This setting affects the readout of the CCD-Chip. Less data is transferred, but readout time is not changed.

roix1	start value of horizontal ROI	
roix2	end value of horizontal ROI	
	range 1 ... 20	for CCD chip type 640 x 480
	range 1 ... 40	for CCD chip type 1280 x 1024

roiy1, roiy2

Set the start and end value for the vertical Region of Interest. One unit is 32 pixels. This setting affects the readout of the CCD-Chip. Less data is transferred and the readout time is decreased.

roiy1	start value of vertical ROI	
roiy2	end value of vertical ROI	
	range 1 ... 15	for CCD chip type 640 x 480
	range 1 ... 32	for CCD chip type 1280 x 1024

Thus the smallest ROI is 32 pixels in square.
In the case of the double trigger mode, the ROI is set for the two half images which are then transferred as one data set of double height.

hbin

Set the horizontal binning. This setting affects the readout of the CCD-Chip. Less data is transferred but the readout time is not changed.

hbin	horizontal binning 1 no binning other possible values 2, 4, 8

vbin

Set the vertical binning. The maximal vertical binning setting depends on the selected vertical ROI . This setting affects the readout of the CCD-Chip. Less data is transferred and the readout time is decreased.

vbin	vertical binning 1 no binning other possible values SVGA 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 other possible values VGA 2, 4, 8, 15, 16, 30, 32, 60, 64, 120, 128, 240, 256, 480

table

Set the special Dicam-Pro values and delay and exposure times. The parameter is a pointer to a zero terminated ASCII string. All values are separated by comma (,) or space ( ). The array is concluded by the sequence “-1,-1”, so that variable key lengths can be handed over. The characters ‘CR’ (13D) and ‘LF’ (10D) may be used to structure the input string.

table	pointer to string array

string array

```
phosphordecay, mcpgain, trigger, loops,
delayhigh1, delaylow1, timehigh1, timelow1,
delayhigh2, delaylow2, timehigh2, timelow2,
delayhigh3, delaylow3, timehigh3, timelow3,
-1, -1
```

phosphordecay	0 … 100 in [ms]	
mcpgain	0 ... 999	
trigger	0	no trigger
	1	extern rising edge
loops	1 ... 256	

mintime	minimum pulse time, depending on the pulser
mindeltime	minimum time between two pulses

If loop is set greater than 1, first delay value must also be greater than mindeltime.

Three DiCAM-PRO modes are defined:

a) DiCAM-PRO single trigger mode
mintime and mindeltime depends on the pulser type, see table below

time and delay setting steps as follows (in ns):
3, 5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100, 120, 140, ... , 1000 in 20ns steps

pulser type	mintime in [ns]	mindeltime in [ns]
HVP3X-3	3 - 10 - 20 - 25 - 30	500
HVP3X-5	5 - 10 - 20 - 25 - 30	500
HVP3X-20	20 - 25 - 30	500
HVP3N	3 - 10 - 15 - 20 - 25	300000
HVP5N	5 - 10 - 15 - 20 – 25	300000
HVP2N	100	500

delayhigh1	0 ... 999999ms
delaylow1	0 ... 999999ns
timehigh1	0 ... 999999ms
timelow1	mintime ... 999999ns

b) DiCAM-PRO multi trigger mode

mintime = 20
mindeltime = 500ns or 300µs (depends on the pulser type)
time and delay settings in 20ns steps.

delayhigh1	0 ... 999ms
delaylow1	0 ... 999980ns
timehigh1	0 ... 999ms
timelow1	mintime ... 999980ns
delayhigh2	0 ... 999ms
delaylow2	mindeltime ... 999980ns
timehigh2	0 ... 999ms
timelow2	mintime ... 999980ns
delayhigh3	0 ... 999ms
delaylow3	mindeltime ... 999980ns
timehigh33	0 ... 999ms
timelow3	mintime ... 999980ns

c) DiCAM-PRO double trigger mode

mintime = 20
mindelpulser = 500ns or 300µs (depends on the pulser type)
time and delay settings in 20ns steps.

delayhigh1	0 ... 10ms
delaylow1	0 ... 999980ns
timehigh1	0 ... 999ms
timelow1	mintime ... 999980ns
delayhigh2	0 ... 10ms
delaylow2	mindeltime ... 999980ns
timehigh2	0 ... 999ms
timelow2	mintime ... 999980ns

Note: The delay1 + delay2 + time1 must be higher than 1000ns.

int **TEST_COC** (*int* mode, *int** trig, *int** roix1, *int** roix2, *int** roiy1, *int** roiy2,
*int** hbin, *int** vbin, *char** table, *int** tablength)
int **TESTCOC** (*int* mode, *int** trig, *int** roix1, *int** roix2, *int** roiy1, *int** roiy2,
*int** hbin, *int** vbin, *char** table, *int** tablength)

Tests all parameters. If the parameters have a valid value, they will be accepted, otherwise the value, next close to the valid one, will be used.

Parameters [in]

mode,trig,roix1,roix2,roiy1,roiy2,hbin,vbin,table	values to be tested, same as in SET_COC()
tablength	length of the allocated buffer for table

Parameters [out]

mode,trig,roix1,roix2,roiy1,roiy2,hbin,vbin,table	input or corrected values
tablength	length of string built from input values

Return Values

Zero on success.
A negative number indicates failure, returned value is the errorcode.
Positiv numbers as follows:

103	one or more values changed
104	buffer too short for internal built string

Starting an exposure and reading out the CCD

int **RUN_COC** (*int* mode)

Processing of the COC is started.The COC program describes the read out procedure for the CCD as well as the delay and exposure times for capturing a picture.
In continuous mode the COC program is started repeatedly until the STOP_COC command is sent.

Parameters [in]

mode	run mode	
	0	continuous
	4	single

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

In continuous mode immediately after the first call and then after each exposure, a new exposure is started automatically (restart of the COC program) as long as one or both buffers of the PCI interface board are empty. The sequence speed depends on the selected delay and exposure times and the CCD read out time.
In single mode one single exposure is started (start of the COC program).

Note: RUN_COC does **not** transfer images to the image buffers of the PCI Interface Board as long there are all buffers occupied by images! In order to release buffers call READ_IMAGE (releases one buffer) or STOP_COC (releases all buffers).or CLEAR_BOARD_BUFFER (releases one buffer)

Note: single mode should not be called in simultaneous mode, otherwise this causes the camera and DLL to make some processing which will decrease performance.

Example Reading in an image with a non-empty buffer

```
STOP_COC              cleans the memory
RUN_COC               starts an exposure
DO
  GET_IMAGE_STATUS    query until a new picture is available
WHILE
READ_IMAGE            loads an image from the PCI buffer into the
                      PC memory
STOP_COC              terminates the exposure
```

int **STOP_COC** (*int* mode)

This function interrupts a running exposure (execution of the COC program). It can be used as a break option, e.g. in the case of very long delay and exposure times. Additionally, the image buffers of the PCI Interface Board are released. Stored images are lost!
If the camera does a picture transfer, when this command is called, the program is waiting until the CCD chip is cleared! (For a SVGA Chip this delays execution for at least 150ms)

Parameters [in]

mode	stop mode	
	0	the only allowed value

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

Note: After STOP_COC is called the BUSY signal of the PCI Interface Board indicates “busy state, not ready for taking images” until SET_COC is called again. For description of the BUSY signal see the description of the PCI Interface Board.

Interrogating the status and settings

int **GET_STATUS** (*int** cam_type, *int** temp_ele, *int** temp_ccd)

Get status information from the camera and the PCI Interface board and read the temperature of the camera circuits and of the CCD sensor. For values out of range an error is returned.
New CCD-Types can not be recognized with this function, use GET_CAMERA_CCD instead.

Parameters [out]

cam_type	status information, see below
temp_ele	temperature of camera electronic valid range -30 ... +65 °C (-22...+149 °F)
temp_ccd	temperature of CCD-Chip valid range -30 ... +65 °C (-22...+149 °F)

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

cam_type	
D17,D16	gain 00 = normal gain 01 = extended gain normal 02 = extended gain invers
D15, D14	CCD-type 00 = 640 x 480 01 = 640 x 480 10 = 1280 x 1024 or extended CCD’s
D13, D12	Camera-type 00 = LongExposure 01 = FastShutter / DoubleShutter 10 = special version 11 = DiCAM-PRO
D11, D10, D9	SensiCam: version of the camera 000...111 = vers. 1.0,1.5,2.0 ... 3.5 DiCAM-PRO: type of the pulser 000 = HVP2N 001 = HVP5N 011 = HVP5NE 100 = HVP3X, 5ns 101 = HVP3X, 3ns 111 = HVP3X, 20ns
D8	CCD-color 0 = black/white CCD 1 = RGB CCD
D7,D6	if typ is FastShutter/Doubleshutter 01 = FastShutter 02 = DoubleShutter
D5	temperature regulation 0 = regulating 1 = temp locked (-13°C = 10.6 °F)
D4 ... D0	Reserved

int **GET_CAMERA_CCD** (*int* board, *int** ccdtype)

Get the CCD-type of the camera.

Parameters [in]

board	number of PCI board	
	-1	board selected with last SET_BOARD() call
	0 … 3	

Parameters [out]

ccdtype		
	1	VGA black&white
	2	VGA color
	3	SVGA black&white
	4	SVGA color
	17	QE black&white
	23	Standard black&white
	25	Double black&white

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **GET_CAMERA_TYP** (*int* board, *int** camtype)

Get the Camera-type.

Parameters [in]

board	number of PCI board	
	-1	board selected with last SET_BOARD() call
	0 … 3	

Parameters [out]

camtype		
	1	FAST SHUTTER
	2	LONG_EXPOSURE
	5	DICAM-PRO

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **GET_CAMERA_ID** (*int* board, *int** id)

Get the Camera-ID. If available, cameras can be configured for ID's between 0 and 3. This makes it possible to interact with a specified camera even if the cable connection from camera to board is changed.

Parameters [in]

board	number of PCI board -1 board selected with last SET_BOARD() call 0 … 3

Parameters [out]

id	camera id 0 … 3

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **GET_IMAGE_SIZE** (*int** width, *int** height)

Get the actual image size. The image size depends on the binning and ROI settings set by the last call of the SET_COC() command and on the CCD sensor type. In 'Double Shutter' mode, the height of the double image (2 ... 2048) is returned.
This function returns invalid values, if called after a LOAD_USER_COC() command.

Parameters [out]

width	width of image in pixel 1 … 1376
height	height of image in pixel 1 … 2048

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **GET_CCD_SIZE** (*int** ccdsize)
int **GET_CCDSIZE** (*int** ccdsize)

Get the total available pixels of the CCD-Chip.

Parameters [out]

ccdsize	number of pixels of CCD-Chip 307.200 (VGA) 1.310.720 (SVGA) 1.431.040 (QE)

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **GET_IMAGE_STATUS** (*int** stat)

Get the current image and camera status.

Parameters [out]

stat	status of camera	
Bit 0 (2^0):	0	no READ_IMAGE function running
	1	busy, READ_IMAGE_8BIT or READ_IMAGE_12BIT is running
Bit 1 (2^1):	0	image data available in PCI board buffers
	1	no image data available (buffer empty)
Bit 2 (2^2):	0	COC idle, not running
	1	COC is running
Bit 4 (2^4):	0	none or one buffer full
	1	both buffers full

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

Example

A new image is available in the PCI Interface Board buffer if both bit 0 and bit 1 are ‘0’.
Under certain circumstances Bit2 may read a wrong value, therefore validate the first read by calling the function twice.

int **GET_SETTINGS** (*int** mode, *int** trig, *int** roix1, *int** roix2, *int** roiy1, *int** roiy2, *int** hbin, *int** vbin, *int*** table)

Reads the actual camera settings, which have been set using the commands SET_COC or DIALOG_CAM. The returned values have the same format as described for SET_COC.

Parameters [out]

mode,trig,roix1,roix2,roiy1,roiy2,hbin,vbin,table
actual values

Return Values

Zero on success.
A negative number indicates failure, returned value is the errorcode.

Note: Sufficient buffer space has to be available for the string table. Ten pairs of values require approximately 200 bytes.

Note: The values returned by GET_SETTINGS are not valid, .if the current COC was generated by using LOAD_USER_COC,

Note: The pointer to the string table is mistakenly defined as char**. The actual behaviour is as it was defined as char*. In order to keep compatible to older versions of the SDK this mistake is not corrected. Please use the following workaround:

```
char str[200];          // string to get the returned settings
GET_SETTINGS(…, (char**) str);   // only typecast is necessary
```

int **GET_COC_SETTINGS** (*int** mode, *int** trig, *int** roix1, *int** roix2, *int** roiy1, *int** roiy2, *int** hbin, *int** vbin, *int** table, *int* len)

Reads the actual camera settings, which have been set using the commands SET_COC or DIALOG_CAM. The returned values have the same format as described for SET_COC.

Parameters [in]

len	size of allocated buffer for table

Parameters [out]

mode,trig,roix1,roix2,roiy1,roiy2,hbin,vbin,table	actual values
tablength	length of string built from input values

Return Values

Zero on success.

A negative number indicates failure, returned value is the errorcode.

Positiv numbers as follows:

104	buffer for builded string too short

Note: Sufficient buffer space has to be available for the string table. Ten pairs of values require approximately 200 bytes.

Note: The values returned by GET_SETTINGS are not valid, .if the current COC was generated by using LOAD_USER_COC,

The following functions return the exact times of the current COC, which was set with function SET_COC.

Readout time is the time, which is necessary to read out the CCD sensor.The readout time depends on the CCD size, vbin, ROI and – in case of special readout procedures – on some other parameters specific for these procedures.

To get the COC time of one image cycle add delay and exposure time to readout time.

float **GET_COCTIME** (*void*)

Return Values

Readout time in µs

float **GET_BELTIME** (*void*)

Return Values

Delay + exposure time in µs

float **GET_EXPTIME** (*void*)

Return Values

Exposure time in µs

float **GET_DELTIME** (*void*)

Return Values

Delay time in µs

Reading images and converting

int **LOAD_OUTPUT_LUT** (*unsigned char** lut)

Copy values of lut into the internal LookUpTable (LUT) memory used by the black&white convert functions of the SDK, which convert the pixel values from 12Bit to 8Bit, eg. READ_IMAGE_8BIT. The size of allocated memory for lut must be at least 4KByte. Only the first 4KByte of lut are copied.

Parameters	**[in]**
lut	pointer to the memory containing a valid LUT

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **LOAD_COLOR_LUT** (*unsigned char** redlut, *unsigned char** greenlut, *unsigned char** bluelut,)

Copy values of redlut, greenlut, bluelut to the corresponding internal LUT memory used by the color convert functions of the SDK which convert the pixel values from 12Bit to 3x8 bit (RGB), eg. CONVERT_BUF_12TOCOL. The size of allocated memory for each lut must be at least 4KByte. Only the first 4KByte of each lut are copied.

Parameters	**[in]**
redlut	pointer to the memory containing a valid LUT for red pixels
greenlut	pointer to the memory containing a validLUT for green pixels
bluelut	pointer to the memory containing av valid LUT for blue pixels

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **LOAD_PSEUDO_COLOR_LUT** (*unsigned char** redlut, *unsigned char** greenlut, *unsigned char** bluelut,)

Copy values of redlut, greenlut, bluelut to the corresponding internal LUT memory used by the pseudo color convert functions of the SDK eg. CONVERT_BUF_12TOCOL. The size of allocated memory for each lut must be at least 256Byte. Only the first 256Byte of each lut are copied.

Parameters	**[in]**
redlut	pointer to the memory containing a valid LUT for red pixels
greenlut	pointer to the memory containing a validLUT for green pixels
bluelut	pointer to the memory containing av valid LUT for blue pixels

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **CONVERT_BUFFER_12TO8** (*int* mode, *int* width, *int* height, *unsigned short** p12_in, *unsigned char** p8_out)

A 16Bit memory area (12 bit pixels) with a size of ‘width’ x ‘height’ in pixels is converted into a 8 Bit memory area (8 Bit pixels) with the use of the internal black&white LUT.

Parameters [in]

mode	convert mode, a combination of the following flags NORMAL 0x0000 FLIP 0x0001 (change lines) MIRROR 0x0008 (change rows)
width	width of input image range 1 ... 1376
height	height of input image range 1 ... 1040
p12_in	pointer to a valid memory region which includes the 12Bit pixel data (16Bit per pixel)
p8_out	pointer to a valid memory region which will receive the output data (8Bit per pixel).

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

If FLIP flag or MIRROR flag is set not only the pixel data is converted but also the total image is flipped or mirrored resp. Combination of both flags (mode = FLIP + MIRROR) is possible which results in an 180° rotated image. In other words: FLIP swaps the lines, MIRROR swaps the columns:

FLIP:	line 0 -> line n-1	MIRROR:	col 0 -> col n-1
	line 1 -> line n-2		col 1 -> col n-2
	…		…
	line n-2 -> line 1		col n-2 -> col 1
	line n-1 -> line 0		col n-1 -> col 0

int **CONVERT_BUFFER_12TOCOL** (*int* mode, *int* width, *int* height,
*unsigned short** p12_in, *unsigned char** p_out)

A 16 bit memory area (12 bit pixel) with dimensions 'width' and 'height' in pixels is converted into a COLOR memory area with 3 colors of 8 Bit each (BGR) using the internal LUT's, which have been loaded with LOAD_COLOR_LUT or LOAD_PSEUDO_COLOR_LUT. The missing intermediate values of the colors red, green and blue are interpolated.

Parameters [in]

mode	convert mode, a combination of the following flags		
	NORMAL	0x0000	(convert to BGR)
	FLIP	0x0001	(change lines)
	32BIT	0x0002	(convert to BGR0)
	MIRROR	0x0008	(change rows)
	PSEUDO	0x0010	(convert via pseudo)
	LOW_AV	0x0020	(low average enable)
width	horizontal size of image		
height	vertical size of image		
p12_in	pointer to a valid memory region which includes the 12Bit pixel data (16Bit per pixel)		
p_out	pointer to a valid memory region which will receive the output data. If 32BIT flag is set, 4x8 bit else 3x8 bit for each pixel must be reserved		

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

If the 32BIT flag is not set, the colors are converted into 3 Byte (24 Bit), whereas , if 32BIT flag is, set the colors are converted into 4 Byte, with the last Byte set to 0.

If FLIP flag or MIRROR flag is set, not only the pixel data is converted but also the total image flipped (mirrored horizontal) or mirrored (mirrored vertical) resp. Combination of both flags (mode = FLIP + MIRROR) is possible which results in an 180° rotated image. For a more detailed description of the MIRROR and FLIP flags see CONVERT_BUFFER_12TO8 (page 34).

If the PSEUDO flag is not set, the convert function uses the COLOR_LUT to convert images from color camera to color images.

If the LOW_AV flag is not set, the resulting color images get higher color resolution and less spatial resolution, otherwise the color images get higher spatial resolution and less color resolution.

If the PSEUDO Flag is set, the convert function uses first the black&white LUT and than the PSEUDO_COLOR_LUT to convert b/w images into pseudocolor images. LOW_AV flag must not be set.

int **READ_IMAGE_8BIT** (*int* mode, *int* width, *int* height, *unsigned char** pointer)

This function reads an image with the selected 'width' and 'height' (in pixels) from the PCI Interface Board buffer, converts the data from 12 bit to 8 bit data using the internal LUT loaded with LOAD_OUTPUT_LUT. The converted image is written into the memory area specified by the pointer If specified by the flags the image is mirrored or flipped.
If the function was successful, the PCI Interface Board buffer containing the image is released and the image can not be read again.
The number of bytes which are read is 'width' x 'height'.
In 'Double Shutter' mode (cf. SET_COC) the two half images are read as one data set of double height when this function is called.

Parameters [in]

mode	convert mode, a combination of the following flags		
	NORMAL	0x0000	
	FLIP	0x0001	(change lines)
	MIRROR	0x0008	(change rows)
width	horizontal size of image		
height	vertical size of image		
pointer	pointer to a valid memory region which will receive the output data (8Bit per pixel).		

Return Values

Zero on success.
Below zero indicates failure, returned value is the errorcode.
100 = No picture is in Board buffer

If FLIP flag or MIRROR flag is set, not only the pixel data is converted but also the total image flipped (mirrored horizontal) or mirrored (mirrored vertical) resp. Combination of both flags (mode = FLIP + MIRROR) is possible which results in an 180° rotated image. For a more detailed description of the MIRROR and FLIP flags see CONVERT_BUFFER_12TO8 (page 34). FLIP and/or MIRROR require additional processing time (approx. 5 ms) compared to 'normal'.

int **READ_IMAGE_12BIT** (*int* mode, *int* width, *int* height, *unsigned short** pointer)

This function reads an image with the selected ‘width’ and ‘height’ (in pixels) from the PCI Interface Board buffer and writes it into the memory area specified by the pointer.

Win95/98: Small Blocks of DMA-transfers are used to write from board to PC-Memory. Simultaneously the data is written to the selected memory area.

WinNT/2000: Direct DMA-transfer to PC-Memopry is used to write data from the board buffer.

If the function was successful, the PCI Interface Board buffer containing the image is released and the image can not be read again.

The number of bytes which are read is ‘width’ * ‘height’ *2.

In ‘Double Shutter’ mode (cf. SET_COC) the two half images are read as one data set of double height when this function is called.

Parameters [in]

mode	convert mode, a combination of the following flags		
	NORMAL	0x0000	
	FLIP	0x0001	(change lines)
	MIRROR	0x0008	(change rows)
width	horizontal size of image		
height	vertical size of image		
pointer	pointer to a valid memory region which will receive the output data (16Bit per pixel).		

Return Values

Zero on success.

Below zero indicates failure, returned value is the errorcode.

100 = No picture is in Board buffer

If FLIP flag or MIRROR flag is set, not only the pixel data is converted but also the total image flipped (mirrored horizontal) or mirrored (mirrored vertical) resp. Combination of both flags (mode = FLIP + MIRROR) is possible which results in an 180° rotated image. For a more detailed description of the MIRROR and FLIP flags see CONVERT_BUFFER_12TO8 (page 34). FLIP and/or MIRROR require additional processing time (approx. 5 ms) compared to ‘normal’.

Example

SET_INIT(1)
SET_COC(3, 1, 10, 10, 20, 4, 2, *table)
 ‘Double Shutter’
 ‘roi window horizontal 320 pixels, vertical 320 pixels’
 ‘binning’
 ‘Gives a data set of 80 x 160 pixels per half image’
GET_IMAGE_SIZE(int *width, int *height)
 ‘width=80, height=320’
 ‘In SET_COC ‘Double Shutter’ was selected, therefore it is returned with double height’
READ_IMAGE_SIZE(0, 80, 320, word *pointer)
 ‘The two half images are read as one data set’

int **READ_IMAGE_COL** (*int* mode, *int* width, *int* height, *unsigned char** bcol)

This function reads an image with the selected 'width' and 'height' (in pixels) from the PCI Interface Board buffer and writes it into the memory area specified by the pointer 'bcol'.
READ_IMAGE_12Bit() is used to read the data from the PCI Interface Board. The 12 bit data is then converted into 3 (4) x 8 bit data using the LUT loaded with LOAD_COLOR_LUT.
If the function was successful, the PCI Interface Board buffer containing the image is released and the image can not be read again.
The number of bytes which are read is '3 (4) x width x height', depending on the 32BIT flag
In 'Double Shutter' mode (cf. SET_COC) the two half images are read as one data set of double height when this function is called.

Parameters [in]

mode	convert mode, a combination of the following flags		
	NORMAL	0x0000	(convert to BGR)
	FLIP	0x0001	(change lines)
	32BIT	0x0002	(convert to BGR0)
	MIRROR	0x0008	(change rows)
	PSEUDO	0x0010	(convert via pseudo)
	LOW_AV	0x0020	(low average enable)
width	horizontal size of image		
height	vertical size of image		
bcol	pointer to a valid memory region which will receive the output data. If 32BIT flag is set, 4x 8Bit else 3x 8Bit for each pixel must be reserved		

Return Values

Zero on success.
Below zero indicates failure, returned value is the errorcode.
100 = No picture is in Board buffer

If the 32BIT flag is not set, the colors are converted into 3 Byte (24 Bit), whereas , if 32BIT flag is, set the colors are converted into 4 Byte, with the first Byte set to 0.
If FLIP flag or MIRROR flag is set, not only the pixel data is converted but also the total image flipped (mirrored horizontal) or mirrored (mirrored vertical) resp. Combination of both flags (mode = FLIP + MIRROR) is possible which results in an 180° rotated image. For a more detailed description of the MIRROR and FLIP flags see CONVERT_BUFFER_12TO8 (page 34).If the PSEUDO flag is not set, the convert function uses the COLOR_LUT to convert images from color camera to color images. If the LOW_AV flag is not set, the resulting color images get higher color resolution and less spatial resolution, otherwise the color images get higher spatial resolution and less color resolution.
If the PSEUDO Flag is set, the convert function uses the PSEUDO_COLOR_LUT to convert b/w images into pseudocolor images. LOW_AV flag must not be set.

Dialog functions

int **OPEN_DIALOG_CAM** (*HWND* hwnd, *int* mode, *char** title)

This function opens a dialog window which lets the user change the camera parameters roi, binning, delay and exposure times and trigger settings.
The dialog box is created as a child window of the parent, but has its own thread and its own message queue. Any user inputs result in immediate action: eg. Function SET_COC() is called, the change status flag is set and a message to the parent window is posted.
Input is possible via keyboard or via mouse.
The 'lock' option is disabled when the dialog opens.

Parameters [in]

hwnd	Windows handle of the calling main window
mode	message mode 0 no messages are send 1 a messages is send to the main window after setting new parameters
title	pointer to zero terminated text string, which is written in the title bar of the dialog window. If title is a NULL pointer, standard text is written

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

In mode=1, a message is sent to the calling window each time settings in the dialog window are changed.
This message is defined as Windows API call:
'PostMessage(hwnd,WM_Command,updmsg,0)'
The default value for updmsg is defined in sencam.h as
IDC_UPDATE = 1000 = 0x3E8.
If required this value can be changed as follows
Define the DWORD registry key 'HKEY_CURRENT_USER\\Software\\PCO\\CameraSettings\\Common\\UpdateCommand' and type in the desired value.
All camera dialog parameters are written into the windows registry and are reloaded when the dialog is initialized after an open call. In the case that no registry keys have been created or no key can be found with the name
'HKEY_CURRENT_USER\\Software\\PCO\\CameraSettings',
a new registry with standard parameters is generated.

int **LOCK_DIALOG_CAM** (*int* mode)

This function (mode=1) allows you to lock the input ports of the window. Setting mode=0 unlocks the window. This option might prove to be very helpful in the situation where you want to prevent a change in the image format (change of IMAGE_SIZE) while continuously reading images.
Changing the exposure time is allowed, also toggling in the Info field.

Parameters [in]

mode	lock mode	
	0	lock disabled
	1	lock enabled

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **STATUS_DIALOG_CAM** (*int** hwnd, *int** status)

If the dialog window is open (via OPEN_DIALOG_CAM) the windows handle of the dialog box (*hwnd) is returned. If closed (via CLOSE_DIALOG_CAM) the value ‘0’ is returned. The status flag indicates, if changes in the DIALOG_CAM window occurred since the last call of this function. If changes were made the status is ‘1’, if no change were made the status ‘0’ is returned.

Parameters [out]

hwnd	Windows handle of Dialog	
	0	no Dialog open
status	update status	
	0	no parameters changed since last call
	1	parameters changed since last call

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **CLOSE_DIALOG_CAM** (void)

This function closes the DIALOG_CAM window and writes the parameters into the registry key defined by the key name ‘HKEY_CURRENT_USER\\Software\\PCO\\CameraSettings’.

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **OPEN_DIALOG_BW** (*HWND* hwnd, *int* mode, *char** title)

This function opens a window which allows the interactive setting of the internal BW-LUT, which is used in BW convert functions i.e. READ_IMAGE_8BIT.
The dialog box is created as a child window of the parent, but has its own thread and its own message queue. Any user inputs result in immediate action: I.e.: A new LUT table is calculated, if necessary, the change status flag is set and a message to the parent window is posted.
Input can be done via keyboard or mouse.
The ‘lock’ option is disabled when this dialog opens.

Parameters [in]

hwnd	Windows handle of the calling main window
mode	message mode 0 no messages are send 1 a message is send to the main window after setting new parameters
title	pointer to zero terminated text string, which is shown in the title bar of the dialog window. If NULL pointer is passed, standard text is shown

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

In mode=1, a message is sent to the calling window each time settings in the dialog window are changed.
The sent message is defined as Windows API call:
‘PostMessage(hwnd,WM_Command,updmsg,0)’.
The default value for updmsg is defined in sencam.h as
IDC_UPDATEBW = 1001 = 0x3E9 .
If required this value can be changed as follows
Define the DWORD registry key ‘HKEY_CURRENT_USER\\Software\\PCO\\CameraSettings\\Common\\UpdateCommandBW’ and type in the desired value.
All parameters are written into the windows registry and are reloaded with every call. In the case that no registry key has been created or no data can be found with the key name ‘HKEY_CURRENT_USER\\Software\\PCO\\CameraSettings’, a new registry key with standard parameters is generated when the OPEN_DIALOG_CAM command is given.

int **LOCK_DIALOG_BW** (*int* mode)

This function (mode=1) allows you to lock the input in the BW-Dialog window. Setting mode=0 unlocks the window.

Parameters [in]

mode	lock mode	
	0	lock disabled
	1	lock enabled

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **SET_DIALOG_BW** (*int* bwmin, *int* bwmax, *int* linlog)

Set the values in the dialog box.
Calculate the table values for the internal LUT used for the conversion of 12Bit data to 8Bit data according to the passed parameters, set change status flag and post a message to the parent window.
The parameters bwmin and bwmax define the input range (which is a part of the complete range 0 … 4096) to be converted to the output range (0 ... 256).
Additionally the parameter typ defines a linear (gamma=1) or a logarithmic (gamma=0.45) diagram type for the conversion.

Parameters [in]

bwmin	lower limit of conversion	
	range 0 ... 4094	
bwmax	upper limit of conversion	
	range 1 ... 4095	

bwmax must be greater then bwmin

linlog	diagram type	
	0	linear
	1	logarithmic

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **GET_DIALOG_BW** (int *bwmin, int *bwmax, int *linlog)

Returns the values of the BW dialog box.

Parameters [out]

bwmin	lower limit value of current conversion range
bwmax	upper limit value of current conversion range
typ	typ of diagram 0 linear (gamma=1) 1 logarithmic (gamma=0.45)

Return Values

Zero on success. Negative numbers indicate a failure, the returned value is the errorcode

int **STATUS_DIALOG_BW** (*int** hwnd, *int** status)

If the dialog window is open (via OPEN_DIALOG_BW) the windows handle (*hwnd) is returned. If closed (via CLOSE_DIALOG_BW) the value ‘0’ is returned. Changes in the DIALOG_BW window occurred since the last call of this function set the status to ‘1’. If no change occurred a ‘0’ is returned.

Parameters [out]

hwnd	Window handle of the dialog box if it is open. 0 if dialog box is closed
status	update status 0 if no LUT change occurred since last call 1 if LUT change occurred. The internal change status flag is cleared.

Return Values

Zero on success. A negative numbers indicate a failure, the returned value is the errorcode.

int **CLOSE_DIALOG_BW** (*void*)

This function closes the DIALOG_BW window and writes the parameters into the registry key defined by the key name ‘HKEY_CURRENT_USER\Software\PCO\CameraSettings\LUT’

Return Values

Zero on success. A negative numbers indicate a failure, the returned value is the errorcode.

int **OPEN_DIALOG_COL** (*HWND* hwnd, *int* mode, *char** title)

This function opens a window which allows the interactive setting of the internal COLOR-LUT's, which are used in COLOR convert functions i.e. READ_IMAGE_12BITCOL.
The dialog box is created as a child window of the parent, but has its own thread and its own message queue. Any user inputs result in immediate action: I.e.: A new LUT table is calculated, if necessary, the change status flag is set and a message to the parent window is posted.
Input can be done via keyboard or mouse.
The 'lock' option is disabled when this dialog opens.

Parameters [in]

hwnd	Windows handle of the calling main window	
mode	message mode	
	0	no messages are send
	1	a message is send to the main window after setting new parameters
title	pointer to zero terminated text string, which is shown in the title bar of the dialog window. If NULL pointer is passed, standard text is shown	

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

In mode=1, a message is sent to the calling window each time settings in the dialog window are changed.
The sent message is defined as Windows API call:
'PostMessage(hwnd,WM_Command,updmsg,0)'
The default value for updmsg is defined in sencam.h as
IDC_UPDATECOL = 0x3EA = 1002
If required this value can be changed as follows
Define the DWORD registry key 'HKEY_CURRENT_USER\\ Software\\PCO\\CameraSettings\\Common\\ \UpdateCommandCOL' and type in the desired value.
All parameters are written into the windows registry' and are reloaded with every call. In the case that no registry key has been created or no data can be found with the key name
'HKEY_CURRENT_USER\\Software\\PCO\\CameraSettings',
a new registry key with standard parameters is generated.

int **CLOSE_DIALOG_COL** (*void*)

This function closes the COLOR-Dialog window and saves the parameters to the windows registry key named
'HKEY_CURRENT_USER\Software\PCO\CameraSettings\LUT'

Return Values

Zero on success. A negative numbers indicate a failure, the returned value is the errorcode.

int **LOCK_DIALOG_COL** (*int* mode)

This function (mode=1) allows you to lock the input in the COLOR-Dialog window. Setting mode=0 unlocks the window.

Parameters [in]

mode	lock mode	
	0	lock disabled
	1	lock enabled

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **SET_DIALOG_COL** (*int* redmin, *int* redmax, *int* greenmin, *int* greenmax, *int* bluemin, *int* bluemax, *int* linlog)

Set the values in the COLOR Dialog box.
Calculate the table values for the COLOR LUT's according to the passed parameters, set change status flag and post a message to the parent window.
The parameters min and max define the input range (which is a part of the complete range 0 … 4096) to be converted to the output range (0 … 256).
Additionally the parameter typ defines a linear (gamma=1) or a logarithmic (gamma=0.45) diagram type for the conversion.

Parameters [in]

redmin	lower limit for conversion of red pixel range 0 … 4094
redmax	upper limit for conversion of red pixel range 1 … 4095 must be greater then redmin
greenmin	lower limit for conversion of green pixel range 0 … 4094
greenmax	upper limit for conversion of green pixel range 1 … 4095 must be greater then greenmin
bluemin	lower limit for conversion of blue pixel range 0 … 4094
bluemax	upper limit for conversion of blue pixel range 1 … 4095 must be greater then bluemin
typ	typ of diagram 0 linear (gamma=1) 1 logarithmic (gamma=0.45)

Return Values

Zero on success. Negative numbers indicate a failure, the returned value is the errorcode *int* **GET_DIALOG_COL** (*int** redmin, *int** redmax, *int** greenmin, *int** greenmax, *int** bluemin, *int** bluemax, *int** typ)

Returns the values in the dialog box.

Parameters [out]

redmin	lower limit of current conversion for red pixel
redmax	upper limit of current conversion for red pixel
greenmin	lower limit of current conversion for green pixel
greenmax	upper limit of current conversion for green pixel
bluemin	lower limit of current conversion for blue pixel
bluemax	upper limit of current conversion for blue pixel
typ	typ of diagram 0 linear (gamma=1) 1 logarithmic (gamma=0.45)

Return Values
Zero on success. Negative numbers indicate a failure, the returned value is the errorcode

int **STATUS_DIALOG_COL** (*int** hwnd, *int** status)

If the dialog window is open (via OPEN_DIALOG_BW) the windows handle (*hwnd) is returned. If closed (via CLOSE_DIALOG_BW) the value ‘0’ is returned. Changes in the DIALOG_BW window occurred since the last call of this function set the status to ‘1’. If no change occurred a ‘0’ is returned.

Parameters [out]

hwnd	Window handle of the dialog box if it is open. 0 if dialog box is closed
status	update status 0 if no LUT change occurred since last call 1 if LUT change occurred. The internal change status flag is cleared.

Return Values
Zero on success. A negative numbers indicate a failure, the returned value is the errorcode.

Recorder- and Multibuffer-Functions

int **ALLOC_RECORDER** (*int** count, *int* size)

Windows95/98 This function allocates several blocks of memory of the size defined by ‘size’ for images. These buffers consist of linear continuous buffer areas, data can be written to these buffers from the camera (PCI Interface Board) via DMA.

WindowsNT This function reserves and commits memory. There is no need of continuous buffer areas for DMA transfers in NT.
It is important that you are logged in as ‘Power User’.

Parameters [in]

count	requested number of memory blocks 0 allocate as much memory as possible
size	size of one memory block

Parameters [out]

count	number of received memory blocks

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

The function ‘count’ predefines the desired number and after call of this function the number of actually allocated blocks is returned. To allocate the maximum number of blocks set count to 0. If you set count to another value at most count-1 is returned because one buffer is reserved for internal use. The maximum number of allocated blocks is limited to 1000.

The block size may be defined freely but should be at least the size which is sufficient for storing one image. The size depends on the CCD sensor. To get the actual cameras ccdsize use function GET_CCD_SIZE.
Camera version which can be used in DOUBLE mode deliver images in double height (cf. SET_COC) and therefore require ‘2x ccdsize’ to be allocated.
Generally memory blocks of larger size can be allocated. However, Windows95/98 tends to segments large free memory arrays, making it difficult to allocate very large memory blocks. We recommend to use ‘2x ccdsize’.
After allocating, the memory blocks must be partitioned into buffers of the real image size by using SET_BUFFER_SIZE.

int **FREE_RECORDER** (*void*)
Releases the allocated buffer.

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **SET_BUFFER_SIZE** (*int** bufcount, *int* bufwidth, *int* bufheight)

Windows95/98 Buffers allocated with ALLOC_RECORDER can be partitioned into small buffers for each image. The number of available buffers is returned in ‘bufcount’.

WindowsNT Locks the memory blocks, reserved with ALLOC_RECORDER. Gets physical memory for each block.

The buffer size depends on the settings ‘Binning’ and ‘ROI’ and can be read by calling the function GET_IMAGE_SIZE (width, height).
All DMA functions transfer images of size bufwidth*bufheight from the PCI Interface Board to the selected buffer. The values must match the camera settings to get best performance.

Parameters [in]

bufcount	requested number of buffers
bufwidth	horizontal size of image for buffers
bufheight	vertical size of image for buffers

Parameters [out]

bufcount	number of received buffers

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **GET_BUFFER_ADDR** (*int* bufnum, *int** linaddr, *int** bufwidth, *int** bufheight)

This function delivers the linear 32 bit buffer startaddress from the buffer of number ‘bufnum’. In addition the actual buffersize = bufwidth x bufheight is returned.

Parameters [in]

bufnum	number of buffer from which to get the information

Parameters [out]

linadr	32 bit linear address
bufwidth	horizontal size of image of buffers
bufheight	vertical size of image of buffers

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **STOP_DMA** (*void*)

Stops a running DMA-Transfer.

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **RUN_DMA** (*int* bufstart, *int* bufend, *int* mode)

This function calls a DMA transfer writing image data of the camera (PCI Interface Board) to the buffer allocated with ALLOC_RECORDER and prepared with SET_BUFFER_SIZE.

Note: The camera must be started with RUN_COC **before** RUN_DMA is started.

At first the command starts a second thread which handles the DMA. Then it runs into a loop in which the Windows messagequeue is released (Peek Message, Translate Message, Dispatch Message).
It remains in the loop until the DMA thread has finished, either after reading the last pictures or if STOP_DMA is called.

In mode = 0 the images are written from ‘bufstart’ to ‘bufend’ and the DMA transfer terminates subsequently.
Mode = 1 is the wrap around mode. Writing starts at the first image again after having written the last image from the previous sequence. The DMA transfer runs continuously and has to be terminated with the function STOP_DMA.

If a DMA transfer is active do not run any other operation!

Parameters [in]

bufstart	number of first buffer, to write images in range 1…bufcount
bufend	number of last buffer range 1…bufcount

bufend must be greater or equal then bufstart

mode	wrapping mode	
	0	single sequence
	1	sequence wrap

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

Note: The three commands RUN_DMA, RUN_DMA_AVG and GET_DMA_STATUS are one unit and **must not be mixed** with the two commands DMA_START_SINGLE and DMA_DONE (also one unit)! Both units can be stopped by the STOP_DMA command.

int **RUN_DMA_AVG** (*int* bufstart, *int* bufend, *int* mode, *int* average)

This function writes averaged image data to the specified buffer. It calls a DMA transfer which adds and averages image data of the camera (PCI Interface Board) on a reserved buffer in computer main memory and writes the resulting data to the buffers allocated with ALLOC_RECORDER and SET_BUFFER_SIZE.

Note: The camera must be started with RUN_COC **before** RUN_DMA_AVG is started.
At first the command starts a second thread which handles the DMA. Then it runs into a loop in which the windows message queue is released (Peek Message, Translate Message, Dispatch Message).
It remains in this loop until the DMA thread has finished, either after reading the last images or if STOP_DMA is called.

In mode = 0 the images are written from 'bufstart' to 'bufend' and the DMA-Transfer is terminated subsequently.
In mode = 1 is the wrap around mode. Writing starts at the first image again after having written the last image from the previous sequence. The DMA transfer runs continuously and has to be terminated calling the function STOP_DMA.

If a DMA transfer is active do not run any other operation!

Parameters [in]

bufstart	number of first buffer, to write images in range 1…bufcount
bufend	number of last buffer range 1…bufcount

bufend must be greater or equal then bufstart

mode	wrapping mode	
	0	single sequence
	1	sequence wrap

average	count of averaged images range 2 … 4096

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **GET_DMA_STATUS** (*int** bufnum)

The variable 'bufnum' points to the image buffer where currently a DMA transfer is performed.

Parameters [out]

bufnum	number of buffer of current DMA-transfer

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **DMA_START_SINGLE** (*int* pics)

The function starts a DMA transfer to one of the buffers and returns to the calling function.
The end of the transfer can be monitored using the function DMA_DONE.

Note: The camera must be started with RUN_COC **before** DMA_START_SINGLE is started.

Parameters [in]	
pics	number of buffer for DMA-Transfer

Return Values
Zero on success.
Below zero indicates failure, returned value is the errorcode.
100 = No picture is in Board buffer

int **DMA_DONE** (*int** pic)

Monitors the status of the actual SINGLE DMA-Transfer.

Note: DMA_DONE must be called **at least** one time! Return code 0 indicates that the DMA transfer was terminated correctly, otherwise the DMA transfer is still active. DMA_DONE must be called until the return code is 0. If the DMA transfer seems to be stuck, call STOP_DMA.

Parameters [out]	
pic	number of buffer of actual DMA-Transfer or 0 if DMA-Transfer is finished.

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

Note: The three commands RUN_DMA, RUN_DMA_AVG and GET_DMA_STATUS are one unit and must **not be mixed** with the two commands DMA_START_SINGLE and DMA_DONE (also one unit)!
Both units can be stopped by the STOP_DMA command.

Extended image functions

int **DMA_IMAGE_START** (*int* board, *int* width, *int* height, *unsigned short** frame, *HANDLE* picevent).

Note: **This command is only available for Windows NT / 2000.**

This command starts a DMA transfer of one image from the board buffer to the PC memory location ‘frame’.
The function always returns immediately. If no image is availabe in the board buffers the return code is 100 and no transfer is started.
When the transfer is done, the event picevent is set. Wait for this event with any of the Windows wait functions i.e. WaitForSingleObject(…).

Parameters [in]

board	number of board
width	number of pixels in each line in the image pic
height	number of lines in the image
frame	pointer to begin of the PC memory area
picevent	HANDLE of event (created with CreateEvent or similar

Return Values
Zero on success.
Below zero indicates failure, returned value is the errorcode.
100 = No picture is in Board buffer

int **CLEAR_BOARD_BUFFER** (*int* board)

This command clears one picture from the board buffer in the selected board.

Note: **This function must not be called if no image is in one of the board buffers**

Parameters [in]

board	number of board

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **BEGIN_WAIT_IMAGE**(*int* board, *HANDLE* picinevent)

Note: **This command is only available for Windows NT / 2000.**

This command enables the board functions to generates an event, if an image is available in the image buffer of the PCI Interface Board. The function returns immediately. When an image is available in the board buffers, the event picinevent is set. Wait for this event with any of the Windows wait functions i.e. WaitForSingleObject(…).

Note: Enabling the event generation increases interrrupt processing time and therefore decreases system performance. If no longer needed **END_WAIT_IMAGE**() should be called.

Parameters	**[in]**
board	number of board
picinevent	HANDLE of event (created with CreateEvent or similar

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

int **END_WAIT_IMAGE**(*int* board)

Note: **This command is only available for Windows NT / 2000.**

This command disables board functions, which have been enabled with **BEGIN_WAIT_IMAGE()** command.

Parameters	**[in]**
board	number of board

Return Values
Zero on success. Nonzero indicates failure, returned value is the errorcode.

Extended camera adjustment functions

By using the following functions you can load extended COC exposure and readout procedures which are not covered by the default readout modes. Therefore you have to build a **C**amera **O**peration **C**ode (COC) table, load it with LOAD_USER_COC and then call LOAD_USER_AOI to set the horizontal AOI.
Please contact factory for more details.

int **LOAD_USER_COC** (*unsigned short** coc_table)

Load **C**amera **O**peration **C**ode (COC).

Parameters [in]

coc_table	pointer to a memory area with 16 Bit COC values

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

The memory area containing the COC is transmitted into the program memory of the camera. The COC must include at least one END code. The transmitted program code (COC) is started calling the RUN_COC command.
Calling LOAD_USER_COC again or calling the SET_COC function destroys the previous COC. The camera must be in idle mode when sending this command.
With the use of an user defined COC you can start complex read out procedures for the CCD chip.

Note: All functions which read and return camera parameters will not work correctly after sending a user defined COC!

int **LOAD_USER_AOI** (int aoixmin, int aoixmax)

Load the horizontal AOI when using a user defined COC.

Parameters [in]

aoixmin	start of horizontal aoi
aoixmax	end of horizontal aoi

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

The aoixmax value must be greater than aoixmin value and aoixmax-aoixmin should not be greater than ccdxsize. The five low order bits are ignored, thus the minimum step size is 32. Depending on the camera typ used a minimum offset value must be added to both input values. Please ask factory for further details.

Logging functions

int **ENABLE_MESSAGE_LOG**(*int* level, *char** filename).

Enables output of messages to the logfile ‘filename’.
If level is not zero the logfile is created and messages are logged to this file. If an empty string (“”) is passed for ‘filename’, the default name senlog_0 is used.
If level is zero the logfile is closed.

Parameters [in]

level	message level, a combination of the following flags	
	ERROR_M	0x0001
	INIT_M	0x0002
	BUFFER_M	0x0004
	PROCESS_M	0x0008
	COC_M	0x0010
	INFO_M	0x0020
filename	path and name of logfile	

Return Values

Zero on success. Nonzero indicates failure, returned value is the errorcode.

Return Codes

Function ok

0	no error, function call successful

Errors

-1	initialization failed; no camera connected
-2	timeout in any function
-3	function call with wrong parameter
-4	cannot locate PCI card or card driver
-5	cannot allocate DMA buffer
-6	reserved
-7	DMA timeout
-8	invalid camera mode
-9	no driver installed
-10	no PCI bios found
-11	device is hold by another process
-12	error in reading or writing data to board
-13	wrong driver function
-14 ...-19	reserved
-20	LOAD_COC error (camera runs program memory)
-21	too many values in COC
-22	CCD temperature or electronics temperature out of range
-23	buffer allocate error
-24	READ_IMAGE error
-25	set/reset buffer flags is failed
-26	buffer is used
-27	call to a windows function is failed
-28	DMA error
-29	cannot open file
-30	registry error
-31	open dialog error
-32	needs newer called vxd or dll

Warnings

100	no image in PCI buffer
101	picture too dark
102	picture too bright
103	one or more values changed
104	buffer for builded string too short

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