pco.convert

pco.convert.sdk

SDK–Description

This document describes the API interface to the pco.convert sdk.

Table of contents:

1 General:

This convert SDK description can be used to implement the pco convert routines in proprietary applications, which are used to control pco cameras. It is not possible and prohibited to use the convert routines with third party cameras.

The pco.convert.sdk consists of two parts: The LUT conversion functions (PCO_Conv.DLL) and the dialog functions (PCO_CDlg.DLL).

The conversion functions are used to convert data areas, b/w and color, with a resolution of more than 8 bit per pixel to either b/w data areas with a resolution of 8 bit per pixel or color data areas with a resolution of 24 (32) bit per pixel. The DLL also includes functions to create and fill the various convert objects.

The second part of the API contains the dialog functions. The dialogs are simple GUI dialogs which enable the user to set the parameters of the convert objects. The dialog functions are included in the PCO_CDlg.DLL and are based on some functions of the PCO_Conv.DLL.

In the pco.sdk for pco cameras there exist two samples, which make use of the convert sdk. One is the Test_cvDlg sample and the other is the sc2_demo. Please take a look at those samples in order to 'see' the convert sdk functions in action.

ATTENTION: It is not possible to use the pco_conv.dll without a connected pco.camera. Using this conversion software is restricted to pco cameras only!

1.1 B/W And Pseudo Color Conversion:

The conversion algorithm used in the b/w function is based on the following simple routine:

```
dataout[pos] = lutbw[datain[pos]];
```

where:

- pos is the counter variable
- dataout is the output data area
- dataint is the input data area
- lutbw is a data area of size 2^n containing the LUT, where n = resolution of the input area in bits per pixel

In the pseudocolor function the basic routine to convert to a RGB data area is:

```
val = lutbw[datain[pos]];
dataout[pout + 0] = lutred[val];
dataout[pout + 1] = lutgreen[val];
dataout[pout + 2] = lutblue[val];
```

where:

- pos is the input counter variable
- pout is the output counter variable
- dataout is the output data area
- datain is the input data area
- lutbw is a data area of size 2^n, where n = resolution of the input area in bit per pixel

lutred, lutgreen, lutblue are data areas of size 2^n containing the LUT, where n = resolution of the output area in bit per pixel.

1.2 Color Conversion:

CCD color sensors used in PCO color cameras have filters for the colors red, green and blue. Each pixel has one type of filter, thus originally you do not get full color information for each pixel. Rather each pixel delivers a value with a dynamic range of 12 bits for the color which passes the filter.

All color cameras at PCO work with the Bayer-pattern demosaicking. The color filter pattern of those color CCD can be reduced to a 2x2 matrix. The CCD itself can be seen as a matrix of those 2x2 matrixes. Suppose this color pattern:

R1	G2	G5	B6
G5	B6	G7	B8
R9	G10	R11	G12
G13	B14	G15	B16

The color itself is only an interpretation of the matrix. This interpretation will be done by a so called demosaicking algorithm. The pco_conv.dll works with a special proprietary method.

2 Convert API description:

2.1 PCO_ConvertCreate

Creates a new convert object based on the PCO_SensorInfo structure. The created convert handle will be used during the conversion. Please call PCO_ConvertDelete before the application exits and unloads the convert dll.

a.) Prototype:

```
int PCO_CONVERT_API PCO_ConvertCreate(HANDLE* ph, PCO_SensorInfo *strSensor, int iConvertType)
```

b.) Input parameter:

- HANDLE *ph: Pointer to a handle which will receive the created convert object.
- PCO_SensorInfo*strSensor: Pointer to a sensor information structure. Please do not forget to set the wSize parameter.
- int iConvertType: Variable to determine the conversion type, either b/w, color, pseudo color or color 16.

The input data should be filled with the following parameter:

- *ph = NULL
- strSensorInfo: Structure to hold the sensor information. The following structure members are available:
  - WORD wSize = Size of the sensor info structure.
  - WORD wDummy = 0; reserved for future use.
  - int iConversionFactor = 0. Not used.
  - int iDataBits = Bit resolution of the sensor. Set to 12 for SensiCam and PixelFly. Actual camera types have got the information within the camera descriptor (see PCO_GetCameraDescription in pco.sdk).
  - int iSensorInfoBits = bit0 → 0: b/w sensor; bit0 → 1: color sensor
    bit1 → 0: Data is lower bit aligned; bit1 → 1: Data is upper bit aligned.
  - idarkOffset = Dark offset of the camera. PixelFly = 30. SensiCam = 32. pco.1200, pco.dimax = 32. All other types = 100.
  - dwzzDummy0 = 0; reserved for future use.
  - strColorCoeff = color coefficient matrix.
    Set to 1.0 for PixelFly, SensiCam and cameras not covered below:
    da11 = 1.0; da12 = 0.0; da13 = 0.0;
    da21 = 0.0; da22 = 1.0; da23 = 0.0;
    da31 = 0.0; da32 = 0.0; da33 = 1.0;

Set to the following for pco.1200:
da11 = 2.238938052; da12 = -.008849559; da13 = -.699115043;
da21 = -.504424779; da22 = 2.061946901; da23 = -1.088495573;
da13 = -.557522121; da32 = -.32743363; da33 = 1.88495575;
Set to the following for pco.1600, pco.2000 and pco.4000:
da11 = 0.9543; da12 = 0.0608; da13 = -0.0149;
da21 = -0.2081; da22 = 1.3253; da23 = -0.1050;
da31 = 0.0592; da32 = -0.3834; da33 = 1.3459;
For pco.dimax camera call PCO_GetColorCorrectionMatrix.

- int iCamNum = 0, 1, …; Current number of your camera, usually 0.
- HANDLE hCamera = NULL, reserved for future use.
- dwzzDummy1 = 0; reserved for future use.

- iConvertType = Predefined value, see PCO_ConvStructures.h.
  - PCO_BW_CONVERT = Conversion will produce 8bit b/w output.
  - PCO_COLOR_CONVERT = Conversion will produce 24(32)bit RGB output.
  - PCO_PSEUDO_CONVERT = Conversion will produce 24(32)bit pseudo RGB output
  - PCO_COLOR16_CONVERT = Conversion will produce 48bit RGB output.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

2.2 PCO_ConvertDelete

Deletes a previously created convert object. It is mandatory to call this function before closing the application

a.) Prototype:

```
int PCOCONVERT_API PCO_ConvertDelete (HANDLE ph)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously created convert object.

The input data should be filled with the following parameter:

- ph = previously created convert object.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

2.3 PCO_ConvertGet

Gets all the values of a previously created convert object.

a.) Prototype:

```
int PCOCONVERT_API PCO_ConvertGet (HANDLE ph, PCO_Convert* pstrConvert)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously created convert object.
- PCO_Convert* pstrConvert: Pointer to a pco convert structure

The input data should be filled with the following parameter:

- ph = previously created convert object.
- pstrConvert.wSize = size of the pco convert structure

The structure will be filled with the actual convert parameters.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

2.4 PCO_ConvertG(S)etDisplay

Gets / sets the display structure values of a previously created convert object. Use this functions to change the conversion parameters. Usually do a get call, change values and than do a set call.

a.) Prototype:

```
int PCOCONVERT_API PCO_ConvertG(S)etDisplay (HANDLE ph, PCO_Display* pstrDisplay)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously created convert object.
- PCO_Display* pstrDisplay: Pointer to a pco display structure

The input data should be filled with the following parameter for the get command:

- ph = previously created convert object.
- pstrDisplay.wSize = size of the pco display structure

The structure will be filled with the actual convert parameters for the get call.

The structure should be filled with the following parameters for the set command:

- pstrDisplay.wSize = size of the pco display structure
- pstrDisplay.wDummy = 0; reserved for future use.
- pstrDisplay.iScale_maxmax = 2^bit resolution - 1; e.g. 12bit → 4095
- pstrDisplay.iScale_max = max value for conversion; iScale_min … 2^bit res. - 1
- pstrDisplay.iScale_min = min value for conversion; 0 … iScale_max
- pstrDisplay.iColor_temp = color temperature; 1000 … 20000 in K
- pstrDisplay.iColor_tint = -100 … 100; correction value for green in %
- pstrDisplay.iColor_saturation = -100 … 100; color saturation in %
- pstrDisplay.iColor_vibrance = -100 … 100; color vibrance in %
- pstrDisplay.iContrast = -100 … 100; contrast in %
- pstrDisplay.iGamma = -100 … 100; gamma value in %
- pstrDisplay.iSRGB = 0; not used, set to zero
- pstrDisplay.pucLut = NULL; not used; set to NULL
- pstrDisplay.dwzzDummy1 = 0; for future use; set all parameters to zero

Usually only min, max, contrast and gamma will be changed by the application. For color cameras additionally temp, tint, saturation and vibrance will be changed.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

2.5 PCO_ConvertSetBayer

Sets the bayer structure values of a previously created convert object. Use this functions to change the bayer pattern parameters.

a.) Prototype:

```
int PCOCONVERT_API PCO_ConvertSetBayer (HANDLE ph, PCO_Bayer* pstrBayer)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously created convert object.
- PCO_Bayer* pstrBayer: Pointer to a pco bayer structure

The structure should be filled with the following parameters for the set command:

- pstrBayer.wSize = size of the pco bayer structure
- pstrBayer.wDummy = 0; reserved for future use.
- pstrBayer.iKernel = 0 → upper left is red; 1 → upper left is green (red line); 2 → upper left is green (blue line); 3 → upper left is blue; For SensiCam and PixelFly the upper left is always red. For all other cameras, see the camera description. Additionally it depends on the ROI setting of the camera.
- pstrBayer.iColorMode = 0 → bayer pattern.
- pstrBayer.dwzzDummy1 = 0; for future use; set all parameters to zero

For a SensiCam and a PixelFly it is sufficient to call this function once, with iKernel set to 0. For all pco.sdk dependent cameras it is mandatory to call this function with the correct value after changing the ROI, since ROI determines the color of the upper left pixel.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

2.6 PCO_ConvertSetSensorInfo

Sets the PCO_SensorInfo structure for a previously created convert object.

a.) Prototype:

```
int PCO_CONVERT_API PCO_ConvertSetSensorInfo(HANDLE ph, PCO_SensorInfo *strSensor)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously created convert object.
- PCO_SensorInfo*strSensor: Pointer to a sensor information structure. Please do not forget to set the wSize parameter.

The input data should be filled with the following parameter:

- ph = previously created convert object.
- strSensorInfo: Structure to hold the sensor information. The following structure members are available:
  - WORD wSize = Size of the sensor info structure.
  - WORD wDummy = 0; reserved for future use.
  - int iConversionFactor = 0. Not used.
  - int iDataBits = Bit resolution of the sensor. Set to 12 for SensiCam and PixelFly. Actual camera types have got the information within the camera descriptor (see PCO_GetCameraDescription in pco.sdk).
  - int iSensorInfoBits = bit0 → 0: b/w sensor; bit0 → 1: color sensor
    bit1 → 0: Data is lower bit aligned; bit1 → 1: Data is upper bit aligned.
  - idarkOffset = Dark offset of the camera. PixelFly = 30. SensiCam = 32. pco.1200, pco.dimax = 32. All other types = 100.
  - dwzzDummy0 = 0; reserved for future use.
  - strColorCoeff = color coefficient matrix.
    Set to 1.0 for PixelFly, SensiCam and cameras not covered below:
    da11 = 1.0; da12 = 0.0; da13 = 0.0;
    da21 = 0.0; da22 = 1.0; da23 = 0.0;
    da31 = 0.0; da32 = 0.0; da33 = 1.0;

    Set to the following for pco.1200:
    da11 = 2.238938052; da12 = -.008849559; da13 = -.699115043;
    da21 = -.504424779; da22 = 2.061946901; da23 = -1.088495573;
    da13 = -.557522121; da32 = -.32743363; da33 = 1.88495575;
    Set to the following for pco.1600, pco.2000 and pco.4000:
    da11 = 0.9543; da12 = 0.0608; da13 = -0.0149;

da21 = -0.2081; da22 = 1.3253; da23 = -0.1050;
da31 = 0.0592; da32 = -0.3834; da33 = 1.3459;
For pco.dimax camera call PCO_GetColorCorrectionMatrix.

- int iCamNum = 0, 1, …; Current number of your camera, usually 0.
- HANDLE hCamera = NULL, reserved for future use.
- dwzzDummy1 = 0; reserved for future use.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

2.7 PCO_LoadPseudoLut

Loads a pseudo color lookup table to the convert object. This function can be used to load some of the predefined or self created pseudo lookup tables.

a.) Prototype:

```
int PCOCONVERT_API PCO_LoadPseudoLut (HANDLE ph, int format, char* filename)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously created convert object.
- int format: 0 → lt1, 1 → lt2, 2 → lt3, 3 → lt4;
- char* filename: name of the file to load.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

2.8 PCO_Convert16TO...

Converts the camera raw 16 bit data to the desired format. The convert can result in an 8bit b/w format (PCO_Convert16TO8), 24bit RGB b/w format (PCO_Convert16TO24), 24(32)bit RGB color format (PCO_Convert16TOCOL), 24(32)bit RGB pseudo color format (PCO_Convert16TOPSEUDO) or 48bit RGB color format (PCO_Convert16TOCOL16). The output buffer must be big enough to receive the resulting image. Take care that the type of the destination buffer fits the function, which is called.

a.) Prototype:

```
int PCOCONVERT_API PCO_Convert... (HANDLE ph, int imode, int icolormode, int width, int height, word *b16, ...)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously created convert object.
- int imode: Mode parameter.
- int icolmode: Color mode parameter
- int width: Width of the image to convert
- int height: Height of the image to convert
- word *b16: Pointer to the raw image
- …: Pointer to the resulting image; Either a byte* or a word*

The input parameter should be filled with the following values:

- imode = 0 → normal convert; the following flags are available:
  CONVERT_MODE_OUT_FLIPIMAGE → Flips the image horizontally;
  CONVERT_MODE_OUT_MIRRORIMAGE → Mirrors the image vertically;
  CONVERT_MODE_OUT_RGB32 → produce 32bit output (only for 16TOCOL).
- icolormode (16TOCOL, 16TOCOL16 only) = 0 → upper left is red; 1 → upper left is green (red line); 2 → upper left is green (blue line); 3 → upper left is blue; For SensiCam and PixelFly the upper left is always red. For all other cameras, see the camera description. Additionally it depends on the ROI setting of the camera (only for 16TOCOL and 16TOCOL16).
- width and height = width and height of the image.
- word *b16 = Raw data pointer of the image.
- … = byte* with size width*height for PCO_Convert16TO8;
  byte* with size width*height*3 for PCO_Convert16TO24,
  PCO_Convert16TOPSEUDO, PCO_Convert16TOCOL 24bit
  byte* with size width*height*4 for PCO_Convert16TOCOL 32bit
  word* with size width*height*2*3 for PCO_Convert16TOCOL16

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

2.9 PCO_GetWhiteBalance

Calculates the color temperature and tint setting to get a white balanced image.

a.) Prototype:

```
int PCOCONVERT_API PCO_GetWhiteBalance(HANDLE ph, int *color_temp, int *tint, int imode, int width, int height, word *b16, int x_min, int y_min, int x_max, int y_max)
```

b.) Input parameter:

- HANDLE ph: Handle to a previously created convert object.
- int *color_temp: int pointer to receive the color temperature.
- int *tint: int pointer to receive the tint setting.
- int imode: Mode parameter.
- int width: Width of the image to convert
- int height: Height of the image to convert
- word *b16: Pointer to the raw image
- int x_min: x value of the upper left point of a recangle, where the calculation is done.
- int y_min: y value of the upper left point of a recangle, where the calculation is done.
- int x_max: x value of the lower right point of a recangle, where the calculation is done.
- int y_max: y value of the lower right point of a recangle, where the calculation is done.

The x, y settings for the rectangle are zero based. E.g. with a resolution of 1280*1024 the values are x_min=0, y_min=0, x_max = 1279, y_max=1023.

The input parameter should be filled with the following values:

- imode = 0 → normal convert; the following flags are available:
  CONVERT_MODE_OUT_FLIPIMAGE → Flips the image horizontally;
  CONVERT_MODE_OUT_MIRRORIMAGE → Mirrors the image vertically;
  CONVERT_MODE_OUT_RGB32 → produce 32bit output (only for 16TOCOL).
- width and height = width and height of the image.
- word *b16 = Raw data pointer of the image.
- x_min, y_min; x_max, y_max: 0...x_max-1, 0...y_max-1; x_min+1...width-1, y_min+1...height-1

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

2.10 PCO_GetMaxLimit

Calculates the maximum possible value for the min/max setting. Based on a color temperature and the tint a maximum value for min/max setting can be calculated. The max value must not exceed the highest calculated value out of the three color max values (r_max, g_max, b_max). Otherwise you'll get wrong colors, since white R=G=B=255 (for 8bit) is impossible to reach.

a.) Prototype:

```
int PCOCONVERT_API PCO_GetMaxLimit(float *r_max, float *g_max, float *b_max, int color_temp, int tint, int output_bits)
```

b.) Input parameter:

- Float *r_max: Pointer to a float receiving the max red value.
- Float *g_max: Pointer to a float receiving the max green value.
- Float *b_max: Pointer to a float receiving the max blue value.
- int color_temp: Color temperature.
- int tint: Tint setting.
- int output_bits: Bit resolution of the converted image (usually 8).

The input parameter should be filled with the following values:

- color_temp = color temperature; 1000 … 20000 in K
- tint = -100 … 100; correction value for green in %
- output_bits = 8

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

3 Convert dialog API description:

3.1 PCO_OpenConvertDialog

Creates a convert dialog based on a convert object. Please call PCO_CloseConvertDialog before the application exits and unloads the convert dialog dll.

a.) Prototype:

```
int PCO_CONVERT_DIALOG_EXPORTS PCO_OpenConvertDialog(HANDLE * hLutDialog, HWND parent, char *title, int msg_id, HANDLE hlut, int xpos, int ypos)
```

b.) Input parameter:

- HANDLE *hLutDialog: Pointer to a handle to receive the created dialog handle.
- HWND parent: Handle of the parent window.
- char *title: String to set the title of the dialog.
- int msg_id: Message ID which will be sent in case of changes inside the dialog.
- HANDLE hlut: Handle of a previously created convert object, which should be controlled by the dialog.
- int xpos, ypos: x and y position of the upper left corner of the dialog

The input parameter should be filled with the following values:

- hLutDialog = NULL
- parent = Handle of the parent window.
- title = Caption bar title, e.g. “Color Convert Dialog”.
- msg_id = 0 → no message is sent; WM_APP + … → A message with this ID will be sent to the parent in case dialog controls are changed.
- hlut = Handle of the convert object to be controlled by the dialog.
- Xpos, ypos = 0...Screen x,y max-100.

The dialog will send notification messages with a message type identifier in wCommand of a PCO_ConvDlg_Message structure. The pointer to the structure is transferred in LPARAM.

The following parameters are available in the structure:

- WORD wCommand: Command type, which has occurred inside the dialog.
- PCO_Convert *pstrConvert: Pointer to the controlled convert object.
- int iXPos, iYPos: Actual xy position of the upper left corner of the dialog. This values can be used to preserve the position for future sessions.

Parameter wCommand:

- PCO_CNV_DLG_CMD_CLOSING: The dialog has been closed by pressing the 'Close' button in the caption bar. The dialog object will be deleted automatically. Please set the dialog handle to zero. It is not necessary to call PCO_CloseConvertDialog.
- PCO_CNV_DLG_CMD_WHITEBALANCE: The white balancing button has been pressed. Please handle this message and do a whitebalance call and set the parameters to the dialog.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

3.2 PCO_CloseConvertDialog

Closes a convert dialog.

a.) Prototype:

```
int PCO_CONVERT_DIALOG_EXPORTS PCO_CloseConvertDialog(HANDLE hLutDialog)
```

b.) Input parameter:

- HANDLE hLutDialog: Handle of a previously created dialog.

The input parameter should be filled with the following values:

- hLutDialog = Handle of a previously created convert dialog.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

3.3 PCO_GetStatusConvertDialog

Gets the status of a convert dialog.

a.) Prototype:

```
int PCO_CONVERT_DIALOG_EXPORTS PCO_CloseConvertDialog(HANDLE hLutDialog, int *hwnd, int *status)
```

b.) Input parameter:

- HANDLE hLutDialog: Handle of a previously created dialog.
- int *hwnd: Pointer to an integer to receive the window handle of the dialog
- int *status: Pointer to an integer to receive the status of the dialog.

The input parameter should be filled with the following values:

- hLutDialog = Handle of a previously created convert dialog.

The input pointer will be filled with the following values:

- hwnd = Window handle of a previously created convert dialog.
- status = 0 (reserved for future use)

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

3.4 PCO_G(S)etConvertDialog

Gets / Sets the values of a convert dialog based on a previously created convert object.

a.) Prototype:

```
int PCO_CONVERT_DIALOG_EXPORTS PCO_G(S)etConvertDialog(HANDLE hLutDialog, HANDLE hLut)
```

b.) Input parameter:

- HANDLE hLutDialog: Handle of a previously created dialog.
- HANDLE hLut: Handle of a previously created convert object.

The input parameter should be filled with the following values:

- hLutDialog = Handle of a previously created convert dialog.
- Hlut = Handle of a previously created convert object → set new values based on the convert object; NULL → Convert dialog reloads the settings from the convert object it controls.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

3.5 PCO_SetDataToConvertDialog

Sets the converted and raw image data to the convert dialog. This will update the histogram diagrams shown in the dialog. It is not necessary to set the data for each converted image, since the update rate would exceed the realizable display rate. Setting the data all 0,5 seconds is enough.

a.) Prototype:

```
int PCO_CONVERT_DIALOG_EXPORTS PCO_G(S)etConvertDialog(HANDLE hLutDialog, int ixres, int iyres, void *b16_image, void *rgb_image)
```

b.) Input parameter:

- HANDLE hLutDialog: Handle of a previously created dialog.
- int ixres, iyres: Width and height of the image data transferred
- void *b16_image: Pointer to the raw data.
- void *rgb_image: Pointer to the converted data.
- HANDLE hLut: Handle of a previously created convert object.

The input parameter should be filled with the following values:

- hLutDialog = Handle of a previously created convert dialog.
- b16_image = Pointer of the raw data.
- rgb_image= Pointer of the converted image. Set this to the b/w image (8bit) pointer for b/w control dialog and to the color image (24bit RGB) pointer in case of a color control dialog.

c.) Return value:

- int: Error message, 0 in case of success else less than 0: see pco_err.h for more information.

4 Typical Implementation

This typical step by step implementation shows the basic handling:

1. Declarations:
   ```
   PCO_SensorInfo strsensorinf;
   PCO_Display strDisplay;
   ```
2. Set all buffer 'size' parameters to the expected values:
   ```
   strsensorinf.wSize = sizeof(PCO_SensorInfo);
   strDisplay.wSize = sizeof(PCO_Display);
   ```
3. Set the sensor info parameters and create the convert object:
   ```
   PCO_ConvertCreate(&hConvert,(PCO_SensorInfo*)&strsensorinf.wSize,...)
   ```
4. Optionally open a convert dialog:
   ```
   PCO_OpenConvertDialog(&m_hLutDialog, GetSafeHwnd(), "Convert Dialog",
   WM_APP+1011, m_hLut, 410, 252)
   ```
5. Set the min and max value to the desired range and set them to the convert object.
   ```
   PCO_ConvertGetDisplay(hConvert, (PCO_Display*)&strDisplay.wSize,...)
   strDisplay.iScale_min = 200;
   strDisplay.iScale_max = 2000;
   PCO_ConvertSetDisplay(hConvert, (PCO_Display*)&strDisplay.wSize,...)
   ```
6. Do the convert and set the data to the dialog if dialog is open.
   ```
   PCO_Convert16TOCOL(hConvert, 0, 0, 1280, 1024, b16, b8rgb);
   PCO_SetDataToDialog(hLutDialog, 1280, 1024, b16, b8rgb);// in realistic intervalls
   ```
7. Close the optinally opened convert dialog:
   ```
   PCO_CloseConvertDialog(hConvertDialog)
   ```
8. Close the convert object:
   ```
   PCO_ConvertDelete(hLut);
   ```

See the Test_cvDlg sample in the pco.sdk sample folder.

ATTENTION: It is not possible to use the pco_conv.dll without a connected pco.camera. Using this conversion software is restricted to pco cameras only! Missuse and/or reverse engeneering is probhibited and will be prosecuted to the maximum extent of the law.

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

The Cooke Corporation
6930 Metroplex Drive,
Romulus, Michigan 48174, USA
www.cookecorp.com