user manual

pco.java

1288 EMVA Standard Compliant

pco.

An Excelitas Technologies Brand

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

pco.
An Excelitas Technologies Brand

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

pco.java user manual 2.0.1

Released December 2021

Contents

1 General

The **pco.java** package offers all functions for working with **PCO** cameras that are based on the current **pco.sdk** (see our website). All shared libraries for the communication with the camera and subsequent image processing are included.

- Easy to use de.pco.camera.Camera class.
- Powerful API to **pco.sdk**.
- Image recording and processing with **pco.recorder** (see our website).

1.1 Installation

The project is built using **Apache Maven**.

Maven artifacts on the Maven Central Repository: https://repo1.maven.org/maven2/de/pco/
Parent pom.xml: https://search.maven.org/artifact/de.pco/pco/2.0.0/pom

Group-ID: **de.pco**
Artifact-ID (Maven modules):

- **pco** – Parent pom.xml
- **pco-common** – Common sources for **pco-camera** and **pco-imageio**
- **pco-camera** – Java interface to control the **PCO** cameras
- **pco-imageio** – Java ImageIO plugin for the **PCO** cameras and B16 files
- **pco-example** – Example application

All jars are compiled and tested for at least Java 8.

Add to your pom.xml

```
<dependency>
  <groupId>de.pco</groupId>
  <artifactId>pco-camera</artifactId>
  <version>1.0.0</version>
</dependency>
```

Binaries and sources are also available directly from www.pco.de.

1.2 Basic Usage

```
public static void main(String[] args) throws PcoException
{
  Camera camera1 = new Camera();
  camera1.record(1, ExtendedRecorderType.SEQUENCE);
  ImageData imageData = camera1.getImage(0);
  camera1.close();

  int width = imageData.getWidth();
  int height = imageData.getHeight();
  int[] values = imageData.getData();
  // ... count with the values
}
```

The **pco-imageio** artifact is necessary for getting the `BufferedImage` to be displayed in GUI applications.

```
RawImageReader reader = new RawImageReader();
RawImageInputStream riis = new RawImageInputStream(imageData);
reader.setInput(riis);
BufferedImage image = reader.read(0);
 ... // see also pco-imageio manual
```

2 API Documentation

The Camera class offers the following methods:

- **record()** generates, configures, and starts a new recorder instance.
- **stop()** stops the current recording.
- **close()** closes the current active camera and releases the occupied ressources.
- **getImage()** returns an image from the recorder and its metadata.
- **getImages()** returns all recorded images from the recorder as a list.
- **getExposureTime()** returns the exposure time of the camera.
- **setExposureTime()** sets the exposure time for the camera.
- **waitForFirstImage()** waits for the first available image in the recorder memory.

The Camera class has the following variable:

- **configuration**

The Camera class has the following objects:

- **sdk** offers direct access to all underlying functions of the pco.sdk.
- **recorder** offers direct access to all underlying functions of the pco.recorder.

2.1 Constructors

Description Creates a new Camera object. Checks for the necessary DLLs on the path and loads the native functions. See section **2.6**. Only one Camera object in the application is allowed.

Prototype

```
public Camera() throws PcoException, IllegalStateException
public Camera(Interface iface) throws PcoException, ↩
    IllegalStateException
```

Parameter

Name	Description
`iface`	A particular HW interface to scan the camera for; if not given, these interfaces are scanned automatically: GigE, CamLink ME4, USB3, CLHS

Exception

Name	Description
`PcoException`	See section **2.3**
`IllegalStateException`	Thrown by an attempt to create more Camera objects than the only one.

2.2 Methods

This section describes all methods offered by the Camera class.

2.2.1 Record

Description Creates, configures, and starts a new recorder instance. The entire camera configuration must be set before calling `record()`. The `setExposureTime()` command is the only exception. This function has no effect on the recorder object and can be called up during the recording.

Prototype

```
public void record(long numberOfImages, ExtendedRecorderType type) ↩
    throws PcoException, IllegalArgumentException
```

Parameter

Name	Description
`numberOfImages`	sets the number of images allocated in the driver. The RAM of the PC limits the maximum value.
`type`	In `SEQUENCE` mode, this function is blocking during record. The recorder stops automatically when the `numberOfImages` is reached. In `SEQUENCE_NON_BLOCKING` mode, this function is non-blocking. The status must be checked before reading an image. This mode is used to read images while recording, e.g. thumbnail. In `RING_BUFFER` mode this function is non blocking. The status must be checked before reading an image. Recorder does not stop the recording when the `numberOfImages` is reached. Once this happens, the oldest images are overwritten. In `FIFO` mode, this function is non-blocking. The status must be checked before reading an image. When the `numberOfImages` in the fifo is reached, the following images are dropped until images are read from the fifo.

Exception

Name	Description
`PcoException`	See section **2.3**
`IllegalArgumentException`	Thrown if numberOfImages <= 4 by recorder type `FIFO` or `RING_BUFFER`

2.2.2 Stop

Description Stops the current recording. In `RING_BUFFER` and `FIFO` mode, this function must be called by the user. In `SEQUENCE` and `SEQUENCE_NON_BLOCKING` mode, this function is automatically called up when the `numberOfImages` is reached.

Prototype

```
public void stop() throws PcoException
```

2.2.3 Close

Description Closes the activated camera and releases the blocked ressources.

Prototype

```
public void close() throws PcoException
```

2.2.4 GetImage

Description Returns an image from the recorder including its metadata.

Prototype

```
public ImageData getImage(int imageNumber) throws PcoException
public ImageData getImage(int imageNumber, Roi roi) throws ↩
    PcoException
```

Parameter

Name	Description
`imageNumber`	Specifies the number of the image to be read. In `SEQUENCE` or `SEQUENCE_NON_BLOCKING` mode, the recorder index matches the `imageNumber`. If `image_number` is set to `0xFFFFFFFF`, the last recorded image is copied. This allows to create a live preview while recording.
`roi`	Sets the region of interest. Only this region of the image is copied to the return value.

Return `ImageData` class represents the image data. The user gets the values in the `int[]` array (by calling `getData()`) where only the range 0-65535 will be used.

The reason is that in the underlying C functions of the **pco.sdk**, the data were recorded as unsigned `short[]`. Since java does not have unsigned data types, **pco.java** wraps these 16-bit values for the user in the more general `int[]` array.

Internally, however, the data are still stored in the `java.nio.ByteBuffer`, 2 bytes for a value, to spare the memory resources. Although not recommended, the user can get access to this internal representation by calling a `getInputStream()` method.

Method	Description
`+getData(): int[]`	Returns the data of the image as `int[]`. The values are in fact unsigned shorts bounded by 0-65535.
`+getDataElementMaxValue(): int`	Returns the maximum value of a (16-bit unsigned `short`) pixel: 65535.
`+getWidth(): int`	Returns the width of the recorded image.
`+getHeight(): int`	Returns the height of the recorded image.
`+isCompressed(): boolean`	Returns always `false` in the current version of **pco.java**, compression to 8-bit proposed for the future releases.
`+getMetadata(): ImageMetadata`	Wraps the image number and the metadata and timestamp structures (see **pco.recorder** manual, `copyImage` method).

Continued on next page

Continued from previous page

Method	Description
`+getInputStream(): InputStream`	Returns an `InputStream` on the backing buffer.

Example

```
camera.record(1, ExtendedRecorderType.SEQUENCE);
ImageData imageData = camera.getImage(0);
int width = imageData.getWidth();     // 2160
int height = imageData.getHeight();   // 2560
int[] data = imageData.getData();
int length = data.length;       // 2160 * 2560

imageData = camera.getImage(0, new Roi(1, 1, 300, 300));
width = imageData.getWidth();     // 300
height = imageData.getHeight();   // 300
data = imageData.getData();
length = data.length;       // 300 * 300
```

2.2.5 GetImages

Description Returns all recorded images from the recorder as a list of `ImageData`.

Prototype

```
public List<ImageData> getImages() throws PcoException
public List<ImageData> getImages(Roi roi) throws PcoException
```

Parameter

Name	Description
`roi`	Sets the region of interest. Only this region of the image is copied to the return value.

Return List of `ImageData` objects, see `getImage()` method (chapter **2.2.4**).

Example

```
camera.record(20, ExtendedRecorderType.SEQUENCE);
List<ImageData> images = camera.getImages();
int listSize = images.size();      // 20

for (ImageData image : images)
{
  double mean =
      org.apache.commons.math3.stat.StatUtils.mean(image.getData());
    System.out.println("Mean: " + mean + " DN");
}
//...
//  Mean: 2147.64 DN
//  Mean: 2144.61 DN
//  ...


images = camera.getImages(new Roi(1, 1, 300, 300));
int width = images.get(0).getWidth();  // 300
int height = images.get(0).getHeight(); // 300
```

2.2.6 GetExposureTime

Description Returns the exposure time of the camera.

Prototype

```
public double getExposureTime() throws PcoException
```

Return Returns the exposure time in seconds.

2.2.7 SetExposureTime

Description Sets the exposure time of the camera.

Prototype

```
public void setExposureTime(double exposureTime) throws PcoException
```

Parameter

Name	Description
`exposureTime`	Must be given as float or integer value in the unit ‘second’. The underlying values for the function `sdk.setDelayExposureTime( 0, Timebase.MS, time, timebase)` will be calculated automatically. The delay time is set to `0`.

Example

```
camera.setExposureTime(0.001);  // 1ms
camera.setExposureTime(1e-3);   // 1ms
```

2.2.8 WaitForFirstImage

Description Waits for the first available image in the recorder memory.
In recorder mode `SEQUENCE_NON_BLOCKING`, `RING_BUFFER`, and `FIFO`, the function `record()` returns immediately. Therefore, this function can be used to wait for images from the camera before calling `getImage(), getImages()`.

Prototype

```
public void waitForFirstImage() throws PcoException
```

2.3 Exception

Each function in the **pco.sdk** and **pco.recorder** can return a 32-bit error code (see section 5 of the pco.sdk manual and the `pco_err.h` file). For convenience of the application programming in Java, these errors are wrapped in subclasses of the `de.pco.common.exceptions.PcoException` class.

```
public class PcoException extends IOException
```

Each possible error in the error level (last three bytes) of the error code presents a particular `PcoException` subclass of the same name as the name of the C define (constant).

```
package de.pco.common.exceptions.application;
...
public class CameratypeException extends PcoException
```

Therefore, the user can prepare the code for treating all the possible errors returned by the underlying **pco.sdk** functions (by catching the `PcoException`) as well as a code for the particular specific errors.

```
try {
  // calls PCO_SetRoi function of the pco.sdk
  // but upper left corner of the roi should be 1,1
  camera.setConfiguration(ConfigurationParameter.ROI,
      new Roi(1,1,300,300));
}
// corresponds to the PCO_ERROR_WRONGVALUE (0xA0???001)
catch (WrongvalueException e1) {
  System.err.print("Roi values wrong.");
}
// corresponds to the returned PCO_ERROR_INVALIDHANDLE (0xA0???002)
catch (InvalidhandleException e2) {
  System.err.print("Camera disconnected.");
}
// corresponds to all the other error codes that could be returned
catch (PcoException e3) {
  System.err.print(e3.getMessage());
}
```

The layer and device level of the returned error code (see section 5 of the pco.sdk manual) are represented by the `LayerEnum` and `DeviceEnum` enumerators which are inherent in all the `PcoException` subclasses thrown (the values of these enumerators are accessible by `getLayer()` and `getDevice()` methods).

There are only 4 possible values in the layer level of the error code (`APPLICATION`, `DRIVER`, `FIRMWARE`, `SDKDLL`). All of them are in **pco.java** the values of the `LayerEnum` enumerator. Opposite to that, there are about 180 possible distinct codes in the error level of the error code resulting in 180 `PcoException` subclasses. Therefore, they are compacted in the java packages (corresponding to C namespaces) according to the layer level of the error code to which these `PcoException` subclasses relate.

package of `PcoException` subclasses	getLayer() returns
de.pco.common.exceptions.application	`LayerEnum.APPLICATION`
de.pco.common.exceptions.common	all the possible layers
de.pco.common.exceptions.driver	`LayerEnum.DRIVER`
de.pco.common.exceptions.firmware	`LayerEnum.FIRMWARE`
de.pco.common.exceptions.sdkdll	`LayerEnum.SDKDLL`

There are about 30 possible values in the device level of the error code enumerating among other the possible drivers (`PCO_ERROR_DRIVER_CAMERALINK`, `PCO_ERROR_DRIVER_USB3`, `PCO_ERROR_DRIVER_GIGE` etc.). All of them are in **pco.java** the values of the `DeviceEnum` enumerator.

For all the errors corresponding to the `PcoException` subclasses and all the constants of the `DeviceEnum` see `pco_err.h`.

There is a special `PcoException` subclass: `de.pco.common.exceptions.Warning`. You are suppose to ignore the warnings and therefore they are never thrown as the exceptions by the methods of **pco.java** even if they are returned by the underlying C functions of the **pco.sdk** and **pco.recorder**. You are able to check whether a warning occured after a method call by calling the `getLastCallWarning()` method of the `Sdk` and `Recorder` classes.

2.4 Variable Configuration

The camera parameters are updated by changing the configuration variable.

```
public Map<ConfigurationParameter, Object> getConfiguration()
    throws PcoException;
public void setConfiguration(Map<ConfigurationParameter,
    Object> conf) throws PcoException;
```

```
Map<ConfigurationParameter, Object> conf =
    new HashMap<ConfigurationParameter, Object>();

conf.put(ConfigurationParameter.EXPOSURE_TIME, 0.001)
conf.put(ConfigurationParameter.ROI, new Roi(1, 1, 512, 512))
conf.put(ConfigurationParameter.TIMESTAMP,
    TimestampMode.BINARY_ASCII));
conf.put(ConfigurationParameter.PIXEL_RATE, 100000000)
conf.put(ConfigurationParameter.TRIGGER, TriggerMode.SOFTWARETRIGGER)
conf.put(ConfigurationParameter.ACQUIRE, AcquireMode.AUTO)
conf.put(ConfigurationParameter.METADATA, OnOffEnum.ON)
conf.put(ConfigurationParameter.BINNING, new Binning(1, 1))

camera.setConfiguration(conf)
```

The variable can only be changed before the `record()` function is called. It is a map with a certain number of entries. Not all possible elements need to be specified. The following sample code only changes the `PIXEL_RATE` and does not affect any other elements of the configuration.

```
camera.setConfiguration(
    ConfigurationParameter.PIXEL_RATE, 286_000_000);

camera.record();
...
```

2.5 Objects

This section describes all objects offered by the Camera class.

2.5.1 sdk

The object `sdk` allows direct access to all underlying functions of the **pco.sdk**.

```
GetTemperatureReturn temperatures = camera.sdk.getTemperature();
/*
public class GetTemperatureReturn
{
    private float sensorTemperature;
    private float cameraTemperature;
    private float powerTemperature;
}
*/
```

The java methods of the `sdk` object wrap precisely the C functions of the **pco.sdk**. Every time when a C function returns more than only one value, there is a particular return class in java present. Not all camera settings are currently covered by the `Camera` class. Special settings have to be set directly by calling the respective `sdk` methods.

2.5.2 rec

The object `rec` offers direct access to all underlying functions of the **pco.recorder**. It is not necessary to call a recorder class method directly. The functions are fully covered by the methods of the `Camera` class.

2.6 DLL Handling

The **pco-camera** module depends on the DLLs of the **pco.sdk** and **pco.recorder**. These DLLs are packed into the `pco-camera-1.0.0.jar`, distributed by Maven. When the user creates a new `Camera`, `Sdk`, or `Recorder` object, the constructor checks at runtime whether the DLLs have been loaded. If not the constructor try to load them from the project root path.

If the DLLs are not found, they get extracted from the `pco-camera-1.0.0.jar` into the project root path and loaded automatically. The user is not required to take any other action in both cases. Note however, that without the DLLs in the working directory, the jar file `pco-camera-1.0.0.jar` on the classpath is necessary in spite of the .class files of the **pco-camera** module only.

3 Example GUI application

pco-example artifact contains an example GUI application. Its purpose is to get the images from the camera, to display them (including the additional metadata from the camera) and to save a particular image into B16 file. It also enables the user to load and display B16 and TIFF files, edit the metadata from and save the file again.

Run the example application (by installed Java) with the mere double-click on `pco-example/pco-example-2.0.0-jar-with-dependencies.jar` or from the console using

```
java -jar pco-example-2.0.0-jar-with-dependencies.jar
```

Alternatively, get the maven **pco-example** artifact by adding to your pom.xml

```
<dependency>
    <groupId>de.pco</groupId>
    <artifactId>pco-example</artifactId>
    <version>2.0.0</version>
</dependency>
```

The application depends both on **pco-camera** and on **pco-imageio** artifacts. Source codes of the application are in the package `de.pco.example`, the main class is `GuiExample`.

Then you can start the example application from your own main method by calling

```
GuiExample.main(null);
```

3.1 User Manual

To open the camera connection click on the **CS** (Camera scanner) button. Select the number of images to be recorded and click on the **Record** button. Then you will be able to switch between the recorded images via the left and right arrow buttons.

On the right-hand side you see a column with the metadata obtained from the camera in addition with the image. You can change the metadata accordingly, e.g. put a commentary in the `TEXT` field.

Save the image and the corresponding metadata into a B16 file via the menu option **File**→**Save**. You can load B16 files and also the 8-bit and 16-bit TIFF files via **File**→**Open**. If these files were created using PCO SW, they also contain the camera metadata and the current example application will display it also.

contact

pco europe
+49 9441 2005 50
info@pco.de
pco.de

pco america
+1 866 678 4566
info@pco-tech.com
pco-tech.com

pco asia
+65 6549 7054
info@pco-imaging.com
pco-imaging.com

pco china
+86 512 67634643
info@pco.cn
pco.cn

APPLICATIONS
Imaging applications for a diverse array of scientific and industrial needs

PCO cameras are used for a broad field of applications within science and industry including high-speed imaging, microscopy and PIV just to name a few. The following reports show a distinguished selection of applications using PCO cameras. Can't find yours? Get in touch with our experts, tell us what you want to achieve and let us help you find the best PCO camera for your application!

for application stories please visit our website

pco.
An Excelitas Technologies Brand