GenICam Camera Documentation

Release trunk

A. Parenti

Mar 03, 2025

Contents

Contents:

CHAPTER 1

The genicamCameras package

1.1 Introduction

The genicamCameras package provides a mean for controlling GenICam compliant cameras[1], and is based on the eBUS SDK from Pleora Technologies Inc.: this is commercial software which can be freely used for cameras based on Pleora hardware (for example: Photonic Science and IMPERX cameras). For all other GenICam compliant cameras (e.g. Basler cameras) the library will still be fully functional but will superimpose a watermark on the images. A licence is then needed in order to get rid of the watermark.

1.1.1 GenicamCamera expected parameters

Some parameters are already available in the GenicamCamera base class. Those are the main ones:

- *Camera Hostname/IP*: the hostname or IP address of the network camera.
- *Stream Packet Size*: it should be set to the maximum value allowed by the network interface, for example 1444 in case of standard MTU size (1500), 8192 in case of "jumbo frames" (MTU = 9000).
- *Stream Inter-Packet Delay*: increasing this parameter will slow down the acquisition, but it will improve its stability, especially when multiple cameras are operated on the same control host. Therefore it is recommended to set it to the maximum value allowed by the sensor size and the desired frame rate.
- *Error Count*, read-only: the number of errors occurred during acquisition.
- *Last Error*, read-only: description of the last error occurred during acquisition.
- *Width* [pixel]: the width of the area of interest.
- *Height* [pixel]: the height of the area of interest.
- *Acquisition Mode*: the type of acquisition (e.g. "Continuous", "SingleFrame", "MultiFrame"). The available options depends on the specific camera.

[1] GenICam is the abbreviation for "Generic Interface for Cameras" and is a generic programming interface for machine vision (industrial) cameras (see e.g. wikipedia).

- *Acquisition Frame Count*: number of frames to acquire in "MultiFrame" acquisition mode.

More parameters are available in the derived devices.

1.2 Deployment Guidelines

genicamCameras will automatically install its dependency eBus.

In order to have the receiving thread executed with real time priority, the line

```
username rtprio 99
```

shall be added to the file `/etc/security/limits.conf` on the control server (may differ depending on the Linux distribution).

The MTU on the network interface used for the camera shall be set - when possible - to 9000, also known as "jumbo frames".

For debugging purposes it can be useful to have `tshark` installed on the control server, and `xctrl` user added to the `wireshark` group.

The control server should have a 10 GbE network interface for sending images to the DAQ, and possibly one for the GUI server.

The `GENICAM_ROOT_V3_0` environmental variable must be defined in order to start **any** C++ device server. It can be done in the deployment by adding

```
environmentvars:
  KARABO: "{{ install_dir }}/karabo"
  GENICAM_ROOT_V3_0: "{{ install_dir }}/karabo/extern/lib/ebus_sdk/5/lib/genicam"
```

to the yaml file for the control server, e.g. `playbooks/host_vars/sa2-xtd6-hirex-con-1`.

1.3 Expert Contact

- Andrea Parenti <andrea.parenti@xfel.eu>
- Wajid Ehsan <wajid.ehsan@xfel.eu>

1.4 Supported Cameras

The package currently supports

- Basler GigE cameras (please be aware that for these you will need eBUS licence)
- IMPERX GigE cameras
- Photon Detector from Bruker
- Photonic Science sCMOS GigE camera

It also provides the *GenicamCamera* device class, a very basic interface to GenICam cameras. You will just be able to connect to the camera and start/stop an acquisition. No camera specific parameters will be avaiable.

Should you need to support another camera, please refer to the following section.

1.5 How to create a new camera device class

Add to the Netbeans project the header and source file, as described in *MyCamera.hh file* and *MyCamera.cc file* Sections.

The first time you compile the project, you will have to do it from Netbeans (both for the 'Debug' and 'Release' configuration), in order to have the Makefiles updated.

1.5.1 MyCamera.hh file

The header file MyCamera.hh is very simple and should look like:

```
#ifndef KARABO_MYCAMERA_HH
#define KARABO_MYCAMERA_HH

#include <karabo/karabo.hpp>
#include <karabo/genicam/GenicamCamera.hh>

/**
 * The main Karabo namespace
 */
namespace karabo {

    class MyCamera : public karabo::genicam::GenicamCamera {

    public:

        KARABO_CLASSINFO(MyCamera, "MyCamera", "2.6")

        static void expectedParameters(karabo::util::Schema& expected);

        MyCamera(const karabo::util::Hash& config);

        virtual ~MyCamera();

    };

} // namespace karabo

#endif // KARABO_MYCAMERA_HH
```

1.5.2 MyCamera.cc file

The source file MyCamera.cc can be also fairly simple. The GenicamCamera will take care of everything, the only part left to you is the mapping of GenICam features to Karabo expected parameters, which you have to code in the expectedParameter function:

```
#include "MyCamera.hh"

using namespace std;
using namespace karabo::genicam;
USING_KARABO_NAMESPACES

namespace karabo {
```

(continues on next page)

(continued from previous page)

```
    KARABO_REGISTER_FOR_CONFIGURATION(BaseDevice, Device<CameraFsm>, GenicamCamera,␣
↪MyCamera)

    MyCamera::MyCamera(const karabo::util::Hash& config) : GenicamCamera(config) {

    }

    MyCamera::~MyCamera() {

    }

    void MyCamera::expectedParameters(Schema& expected) {
        // Fill here with the list of expected parameters
    }

} // namespace karabo
```

1.5.3 How to read/write parameters from/to the camera

Each parameter on the camera you want to have available in the Karabo device, must have a corresponding expected parameter in the Karabo device. The expected parameter must be tagged as 'genicam'. Please have a look at the *Expected parameters* Section for the details.

1.5.4 Expected parameters

For the camera parameters you want to have in the Karabo device, you will need to find out the GenICam 'Feature Name' and 'Type'. One way of doing it is with the help of the eBUSPlayer, which you will find installed in $KARABO/extern/lib/ebus_sdk/4/bin.

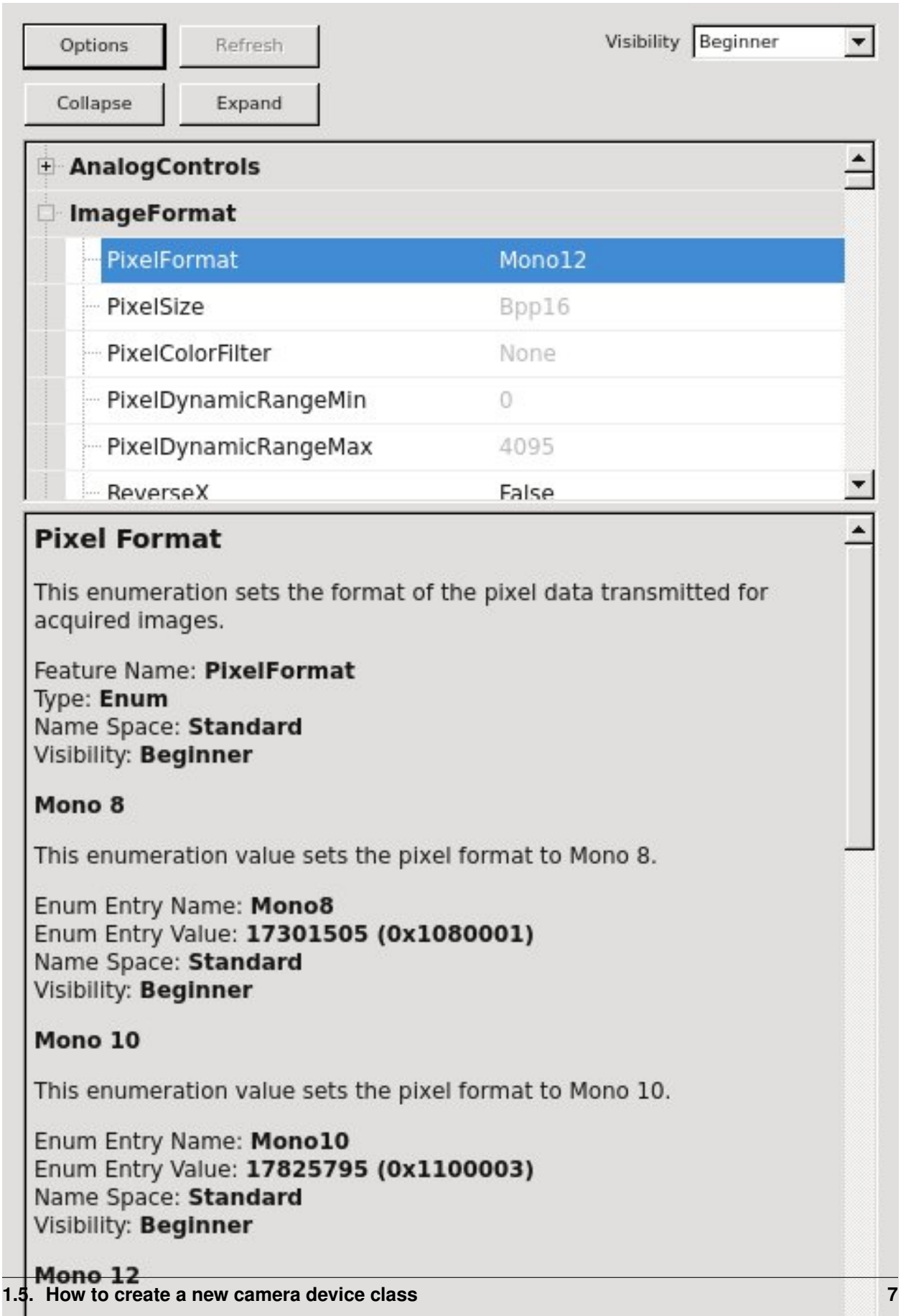
Once you are connected to the camera, if you click on the "Device control" button you will get a list of all available features. By clicking on one, for example 'PixelFormat', you will be able to inspect its details, as shown in Fig. 1.1.

With this information you will be able now to create the corresponding entry in the expectedParameters function:

```
STRING_ELEMENT(expected).key("pixelFormat")
    .alias("PixelFormat")  // The GenICam 'Feature Name'
    .tags("genicam enum poll")
    .displayedName("Pixel Format")
    .description("This enumeration sets the format of the pixel data "
                 "transmitted for acquired images.")
    .assignmentOptional().noDefaultValue()
    .options("Mono8 Mono10 Mono12")
    .reconfigurable()
    .allowedStates(State::STOPPED)
    .commit();
```

The correspondence between GenICam and Karabo types is given in the *Data types* Section.

As already mentioned, all GenICam parameter must have the 'genicam' tag. The description of the other available tags can be found in the *Tags* Section.


Options
Refresh
Visibility
Beginner
Collapse
Expand
AnalogControls
ImageFormat
PixelFormat
Mono12
PixelSize
Bpp16
PixelColorFilter
None
PixelDynamicRangeMin
0
PixelDynamicRangeMax
4095
ReverseX
False
Pixel Format
This enumeration sets the format of the pixel data transmitted for acquired images.
Feature Name: PixelFormat
Type: Enum
Name Space: Standard
Visibility: Beginner
Mono 8
This enumeration value sets the pixel format to Mono 8.
Enum Entry Name: Mono8
Enum Entry Value: 17301505 (0x1080001)
Name Space: Standard
Visibility: Beginner
Mono 10
This enumeration value sets the pixel format to Mono 10.
Enum Entry Name: Mono10
Enum Entry Value: 17825795 (0x1100003)
Name Space: Standard
Visibility: Beginner
Mono 12
This enumeration value sets the pixel format to Mono 12.

Tags

- **'genicam'** tag: Parameters to be read from (written to) the GenICam camera must have the 'genicam' tag.
- **'writeOnConnect'** tag: Parameters having the 'writeOnConnect' flag will be set on the camera when the Karabo device connects to it.
- **'readOnConnect'** tag: Parameters having the 'readOnConnect' flag will be read from the camera when the Karabo device connects to it.
- **'enum'** tag: to be used for the GenICam 'Enum' type, to distinguish it from the 'String' type (since both of them are mapped to STRING_ELEMENT Karabo type).
- **'poll'** tag: Parameters having the 'poll' tag will be polled periodically. The poll interval is a parameter of the base class.

Data types

This is the correspondence between GenICam and Karabo data types:

GenICam Type	Karabo Type
Boolean	BOOL
Integer	INT32
Float	DOUBLE
String	STRING
Enum	STRING (+ 'enum' tag)

1.6 Troubleshooting

1.6.1 The camera is in UNKNOWN state

This means that the Karabo device cannot connect to the camera. You should check that

- the camera is connected to the network,
- and it is powered.

A possible way to verify that the camera is online is to login to the control server and use the `ping` command:

```
ping <camera IP>
```

Camera power can often be controlled via a Beckhoff digital output device, with the same domain name as the camera, but different type and member. For example to the camera `SA1_XTD2_IMGSR/CAM/BEAMVIEW` corresponds the Beckhoff device `SA1_XTD2_IMGSR/DCTRL/CAM_POWER`, which can be used to power on and off the camera.

If the camera is online but still in UNKNOWN state, it is likely that

- another client is already connected to it.

A client can be

- another Karabo device,
- eBUSPlayer,
- ...

If you cannot find out who is keeping the connection busy, you can power cycle the camera and this will kick-out anybody who was connected.

1.6.2 The camera is ACQUIRING but *Frame Rate* is 0

If you are in external trigger mode (the parameter name may differ from camera to camera), it is possible that the camera receives no trigger signal. You can test it by setting the trigger mode to internal.

1.6.3 *Frame Rate* is not 0, but no images are visible in the GUI

Check that in the *Output* node the *hostname* is set correctly. If the control server has a 10 GbE interface dedicated to the GUI server, the IP address of this interface should be set in *output.hostname*.

If this is set correctly, it could be that the GUI server is malfunctioning. In case there is a second GUI server available for the topic, try to switch to that one.

1.6.4 *Frame Rate* is not 0, but the DAQ does not save any data

Check that in the *DAQ Output* node the *hostname* is set to the IP address of the 10 GbE interface dedicated to the DAQ.

1.6.5 *Frame Rate* is Smaller than Expected

If you observe that the acquisition rate is smaller than expected, for example smaller than the trigger rate, the reason could be that the settings are not optimal.

In this case you could find increasing error count in the *Error Count* property, and messages like "TOO_MANY_CONSECUTIVE_RESENDS" or "AUTO_ABORTED" in *Last Error*.

Please refer to the *Deployment Guidelines* and *GenicamCamera expected parameters* Sections for the needed settings.

1.6.6 Connection to the Camera is Lost During Acquisition

Also this could be caused by sub-optimal settings, as described in the *previous* Section.

CHAPTER 2

PhotonicScienceCamera

In addition to expected parameters offered by the *GenicamCamera* class, *PhotonicScienceCamera* features a few more.

2.1 PhotonicScienceCamera expected parameters

The main parameters available for the PhotonicScienceCamera are

- *X Offset* [pixel]: the X offset (left offset) for the area of interest.
- *Y Offset* [pixel]: the Y offset (top offset) for the area of interest.
- *X Binning* [pixel]: the number of horizontal photo-sensitive cells that must be combined (added) together.
- *Y Binning* [pixel]: the number of vertical photo-sensitive cells that must be combined (added) together.
- *Raw Exposure Time*: the exposure time in device-specific unit.
- *Trigger Mode*: trigger mode selector, e.g. “freerun”, “Hardware_rising_edge”.
- *Pixel Format*: the format of the pixel to use during the acquisition, e.g. “Mono8”, “Mono16”.
- *Sensor Width* [pixel], read-only: the effective width of the sensor.
- *Sensor Height* [pixel], read-only: the effective height of the sensor.
- *Max Width* [pixel], read-only: maximum width of the image after binning, decimation or any other function changing the horizontal dimensions.
- *Max Height* [pixel], read-only: maximum height of the image after binning, decimation or any other function changing the vertical dimensions.

More expert-level parameters are available, and should normally not differ from the default set on the device class.

2.2 Troubleshooting

2.2.1 The camera goes regularly to UNKNOWN state

A known issue for Photonic Science cameras connected to the XFEL network, is that at regular time every day (ca 9 AM, 12 PM, 15 PM, . . .) they loose network connection. The reason could not be understood yet. The only know way to recover is by a camera power cycle (see the general *Troubleshooting* Section too).

When possible, it should considered the option of connection the camera to the control server directly, or via a switch, but without connecting it the the XFEL network. This configuration has proven to be unaffected by the disconnection issue.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search