LimaDevice Documentation

Release trunk

A. Parenti

Mar 30, 2023

Contents

Contents:

CHAPTER 1

The limaCameras package

1.1 Introduction

The *limaCameras* package provides a mean for controlling cameras supported by LImA[1], for example Basler cameras.

1.1.1 LimaDevice expected parameters

Some LImA parameters are already available in the LimaDevice base class:

- Camera Type, read-only
- Camera Model, read-only
- Sensor Width [pixel], read-only
- Sensor Height [pixel], read-only
- Image Depth [bytes], read-only
- Trigger Mode
- Exposure Time [ms]
- Latency Time [ms]: the latency time between two successive exposures
- Number of Frames: the number of frames to be acquired (0 means unlimited)
- Image Rotation [degrees]
- Image Flip (X and Y)
- Image Binning (X and Y)
- ROI (X, Y, Width and Height)

More parameters are available in the derived devices.

[1] Lima (**L**ibrary for **Im**age **A**cquisition) is a project for the unified control of 2D detectors.

1.2 Deployment Guidelines

limaCameras will automatically install its dependency LImA.

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.

Please also check the camera specific configuration section, e.g. Basler's *Camera Setup*.

1.3 Expert Contact

- Andrea Parenti <andrea.parenti@xfel.eu>

1.4 How to create a new camera device in limaCameras

After checking-out *limaCameras* and its dependencies, as specified in the DEPENDS file, you can add your a new camera device in *src/limaCameras*, named for example *MyLimaCamera*.

Do not forget to add it to *setup.py*, in the *entry_points* section. This is needed to have it loaded by the Karabo python server.

Once you have done it, you can edit the source file (see *MyLimaCamera.py file* Section).

1.4.1 MyLimaCamera.py file

This method is abstract in *LimaDevice* and must therefore be defined in *MyLimaCamera*:

- **initializeCamera** is the function where the *Camera* and *HwInterface* objects have to be initialized.

There are also two user's hooks in the *LimaDevice* class, which do nothing by default but can be overwritten in the derived class:

- **pollCameraSpecific** is called regularly, and can be used to read parameters from the camera and set them in the *MyLimaCamera* device.
- **reconfigureCameraSpecific** is called upon reconfiguration of the device, and can be used to set camera specific parameters.

This method is defined in the base class but can be redefined in *MyLimaCamera* to do more specific checks:

- **isConnected** shall return *True* if camera is connected.

This is for example the code for a camera using the Simulator plugin (*Lima.Simulator* subpackage):

```
#!/usr/bin/env python

############################################################################
# Author: <andrea.parenti@xfel.eu>
# Created on Jul 21, 2017
# Copyright (C) European XFEL GmbH Hamburg. All rights reserved.
############################################################################

from karabo.bound import (
    KARABO_CLASSINFO, PythonDevice
)

from .lima_device import LimaDevice

import Lima.Simulator

@KARABO_CLASSINFO("MyLimaCamera", "2.1")
class MyLimaCamera(LimaDevice, PythonDevice):
    def __init__(self, configuration):
        # always call PythonDevice constructor first!
        super(MyLimaCamera,self).__init__(configuration)

    @staticmethod
    def expectedParameters(expected):
        # Simulated camera does not have any additional parameters
        pass

    def initializeCamera(self):
        self.log.INFO("Initialize Simulated camera")

        # Creates the camera object (Simulator)
        self.camera = Lima.Simulator.Camera()
        # Creates the HwInterface
        self.interface = Lima.Simulator.Interface(self.camera)
```

CHAPTER 2

LimaBaslerCamera

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

2.1 LimaBaslerCamera expected parameters

2.1.1 Network and stream parameters

- cameraIp: the IP number or hostname of the camera. You can connect to the camera also by using its serial number, eg "sn://21803915". This is a **mandatory** parameter.
- interPacketDelay: the delay between the transmission of each packet (ticks). It should be as small as possible. Try 500 for GigE network, 12586 for 100Mb network.
- frameTransmissionDelay: the delay before image transmission on the Ethernet. Setting different values on synchronous cameras connected to a single server can prevent concurrency issues.
- packetSize: The packet size. Should be as large as possible. Try 8192 if jumbo frames are enabled on the network (MTU=9000), use 1444 otherwise.
- socketBufferSize: the socket buffer size. Should be automatically set by pylon.

2.1.2 TriggerMode options

The trigger modes available in Lima for the Basler camera are four. Here is the explanation of the parameters which will be set by selecting one of those modes.

- IntTrig:
  - TriggerMode = Off
  - ExposureMode = Timed
  - AcquisitionFrameRateEnable = True
- IntTrigMult:

  - TriggerMode = On
  - TriggerSource = Software
  - AcquisitionFrameCount = 1
  - ExposureMode = Timed
- ExtGate
  - TriggerMode = On
  - TriggerSource = Line1
  - AcquisitionFrameRateEnable = False
  - ExposureMode = TriggerWidth
- ExtTrigMult
  - TriggerMode = On
  - TriggerSource = Line1
  - AcquisitionFrameRateEnable = False
  - ExposureMode = Timed

2.1.3 Trigger Activation

Possible options (might not be available for all models) are:

- Rising Edge
- Falling Edge
- AnyEdge
- LevelHigh
- LevelLow

2.1.4 Gain Control

The camera gain will be automatically adjusted by setting *autoGain* to *True*. It can also be manually adjusted by setting *autoGain* to *False* and selecting a value for *gain*, between 0 and 1. 0 corresponds to the minimum *RawGain* value, 1 to the maximum, according to

$$RawGain = RawGainMin + gain \cdot (RawGainMax - RawGainMin)$$

2.1.5 Pixel Format

It can be changed by setting the Image Type property:

- Bpp8 (i.e. 8 bits-per-pixel)
- Bpp10
- Bpp12
- Bpp16

2.1.6 Acquisition Frame Count

This parameter is used to set the number of frames acquired in the multi-frame acquisition mode.

2.1.7 Temperature

The camera temperature (available only on some models).

2.2 Camera Setup

In order to have good performance, especially when multiple cameras are controlled on a single server, you should set the following parameters:

- *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.
- *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).
- *Frame Transmission Delay*: the values should differ for the various cameras controlled by the same control server. This way, they will send the images at slightly different times and concurrency issues will be alleviated.

2.3 Troubleshooting

2.3.1 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 `SCS_XTD10_IMGPI/CAM/BEAMVIEW` corresponds the Beckhoff device `SCS_XTD10_IMGPI/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,
- PylonViewer,
- 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.

2.3.2 The camera is ACQUIRING but *Camera Frame Rate* is 0

If you are in external trigger mode (*ExtTrigMult*, *ExtGate*), it is possible that the camera receives no trigger signal. You can test it by setting the trigger mode to internal (*IntTrig*).

It has been reported in some cases that a camera stops sending images after a while. The reason is under investigation but not understood yet. In those cases is normally enough to stop and re-start the acquisition from the Karabo device. In order to mitigate the issue, a watchdog device (*cameraShaker*) can be deployed, which will automatically restart the acquisition in such cases.

2.3.3 *Camera 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.

2.3.4 *Camera Frame Rate* is not 0, but *Output Frame Rate* is 0

This means that the Karabo device receives data, but some other device is connected to one of its output channels, and is blocking. The list of open connections is available in each output channel. You should check who is connected, try to find out its health state, and restart what needed.

2.3.5 *Output 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.

2.3.6 *Camera 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 error messages like

> [2019/09/19 11:40:13.07e638] 88c2a700 *Camera*_AcqThread::Camera::threadFunction (BaslerCamera.cpp:657)-Error: No image acquired! Error code : 0xhex= e1000014 Error description : The buffer was incompletely grabbed. This can be caused by performance problems of the network hardware used, i.e. network adapter, switch, or ethernet cable. To fix this, try increasing the camera's Inter-Packet Delay in the Transport Layer category to reduce the required bandwidth, and adjust the camera's Packet Size setting to the highest supported frame size.

Please refer to the *Deployment Guidelines* and *Setup* Sections for the needed settings.

2.3.7 Connection to the Camera is Lost During Acquisition

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

CHAPTER 3

Indices and tables

- genindex
- modindex
- search