Lima Documentation

Release 1.9.6

Lima Team

Sep 28, 2020

INSTALLATION

LImA (stands for **L** ibrary for **Im** age **A** cquisition) is a project for the unified control of 2D detectors. It is used in production in ESRF Beamlines and in other places.

The architecture of the library aims at clearly separating hardware specific code from common software configuration and features, like setting standard acquisition parameters (exposure time, external trigger), file saving and image processing.

LImA is a C++ library but the library also comes with a Python binding. A PyTango device server for remote control is provided as well.

We provide Conda binary package for Windows and Linux for some cameras. Check out our Conda channel.

LImA is a very active project and many developments are ongoing and available from GitHub. You can find stable version releases through git branches and tags on Github releases.

If you want to get in touch with the LIMA community, please send an email to lima@esrf.fr. You may also want to subscribe to our mailing list by sending a message to sympa@esrf.fr with `subscribe lima` as subject.

For the latest changes, refers to the `Release Notes`.

Note that this documentation is also available in pdf and epub format.

CHAPTER

ONE

REQUIREMENTS

Some tools and libraries are required to build LImA for either Windows and Linux.

Note: All the dependencies, build or runtime, are available as Conda packages for both Windows and Linux platforms.

1.1 Build dependencies

- A C++ compiler (usually GCC for Linux and Visual Studio for Windows)
  - Visual Studio 2008 for x86 or x64 for python2.7.x
  - Visual Studio 2008 Express for x86 only for python2.7.x
  - Visual Studio 2015 or 2017 for x86 and x64 for python >= 3.5
- CMake >= 3.1

1.2 Python dependencies

LImA is compatible with python 2 and 3.

- numpy >= 1.1
- sip >= 4.19

1.3 Optional dependencies

1.3.1 Saving format dependencies

- TIFF, Tag Image File Format (TIFF), a widely used format for storing image data ;
- zlib, a lossless data-compression library. For Windows, you can download the ESRF binary package zlib-windows and install it under `C:\Program Files` ;
- CBF, a library for accessing Crystallographic Binary Files (CBF files) and Image-supporting CIF (imgCIF) files ;
- HDF5, a data model, library, and file format for storing and managing data ;

- CCfits, CFITSIO, a library for reading and writing data files in FITS (Flexible Image Transport System) data format ;
- LZ4 >= 1.8.2, a lossless compression algorithm ;
- libconfig, a library for processing structured configuration files. For Windows, you can download the ESRF binary package libconfig-windows and install it under `C:\Program Files`.

1.3.2 PyTango server dependencies

- PyTango, the Tango python binding
- libtango, the Tango toolkit

CHAPTER

TWO

BUILD AND INSTALL

2.1 Install binary packages

We provide Conda binary packages for some cameras. This is, by far, the easiest way to get started with LImA! For instance:

```
conda install --channel esrf-bcu lima-camera-basler
```

would install a fully loaded LImA and all its dependencies with the Basler camera plugin and SDK. The camera comes as a python module but is also C++ development package that includes header files and CMake package config files.

If you need the Tango device server for the camera, run:

```
conda install --channel esrf-bcu --channel tango-controls lima-camera-basler-tango
```

Note: The runtime libraries of the camera's SDK are provided as well but some cameras requires drivers or specific setups than needs to be installed manually.

2.2 Build from source

First, you need to get_source. Two methods are provided to build LImA from source:

- using our install script that aims to hide the complexity of CMake;
- using CMake directly for developers who are already acquainted with the tool and need the extra flexibility.

2.2.1 Using scripts

The `install` scripts will run CMake to compile and/or install.

It accepts input arguments (see below) but it also uses a configuration file `scripts/config.txt`. Feel free to update this file for setting a permanent configuration for your own installation.

For Linux:

```
[sudo] install.sh
[--git]
[--install-prefix=<desired installation path>]
[--install-python-prefix=<desired python installation path>]
[options]
```

For Windows:

```
install.bat
[--install-prefix=<desired installation path>]
[--install-python-prefix=<desired python installation path>]
[options]
```

The `--git` (Linux only) option can be used to clone the required submodules as a prerequisite. Otherwise you should install the submodules manually with git commands, for instance:

```
$ git submodule init third-party/Processlib
$ git submodule init camera/basler
$ git submodule init applications/tango/python
$ git submodule update
```

Options are `<camera-name> <saving-format> python pytango-server`:

`<camera-name>` can be a combination of any of the following options:

```
andor|andor3|basler|prosilica|adsc|mythen3|ueye|xh|xspress3|ultra|
xpad|mythen|pco|marccd|pointgrey|imxpad|dexela|merlin|v4l2|
eiger|pixirad|hexitec|aviex|roperscientific|rayonixhs|espia|maxipix|frelon
```

`<saving-format>` can be a combination of any of the following options:

```
cbf|nxs|fits|edfgz|edflz4|tiff|hdf5
```

`python` will install the python module

`pytango-server` will install the PyTango server

For example, to install the Basler camera, use the TIFF output format, the python binding and the TANGO server, one would run:

```
$ sudo install.sh --git --install-prefix=./install --install-python-prefix=./install/
↪python tiff basler python pytango-server
```

2.2.2 Using CMake

Install first the project submodules:

```
git submodule init third-party/Processlib
git submodule init camera/basler
git submodule init applications/tango/python
git submodule update
```

Run `cmake` in the build directory:

```
mkdir build
cd build
cmake ..
   [-G "Visual Studio 15 2017 Win64" | -G "Visual Studio 15 2017" | -G "Unix Makefiles
↪"]
   [-DCMAKE_INSTALL_PREFIX=<desired installation path>]
   [-DPYTHON_SITE_PACKAGES_DIR=<desired python installation path>]
   -DLIMA_ENABLE_TIFF=true
   -DLIMACAMERA_BASLER=true
```

(continues on next page)

(continued from previous page)

```
   -DLIMA_ENABLE_PYTANGO_SERVER=true
   -DLIMA_ENABLE_PYTHON=true
```

Then compile and install:

```
cmake --build
sudo cmake --build --target install
```

2.3 Environment Setup

Warning: If you are using Conda, we advice against setting any environment variables that might affect the Conda environment (e.g. `PATH`, `PYTHONPATH`)as this one of the most common source of troubles.

If the install path for libraries and python modules are not the default, you need to update your environment variables as follow:

For Linux:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<my-custom-install-dir>/Lima/lib
export PYTHONPATH=$PYTHONPATH:<my-custom-install-dir>
```

For Windows:

```
set PATH=%PATH%;<my-custom-install-dir>\Lima\lib
set PYTHONPATH=%PYTHONPATH%;<my-custom-install-dir>
```

or update the system wide variables `PATH` for the libraries and `PYTHONPATH` for python.

CHAPTER

THREE

PYTANGO DEVICE SERVER

3.1 Server setup

As PyTango (Tango for python) server is provided as Python script, you just have to copy the `applications/tango/python` directory wherever you want.

- `camera` directory: contained all camera Tango device specifics so remove all none need script
- `doc` directory: contained plugins camera documentation (exhaustive list of properties, commands and attributes)
- `plugins` directory: contained all plugins device server like:
  - Roi counters
  - Mask...
- `scripts` directory: contained a script use at ESRF to start Lima device server (can also be removed)
- `LimaCCDs.py` file: python script to start Lima device server
- `LimaViewer.py` file: python script to start LimaViewer device server to get image from Lima device server

:: warning: Make sure your environment is properly set for python and library paths, see *Build and Install* for more information.

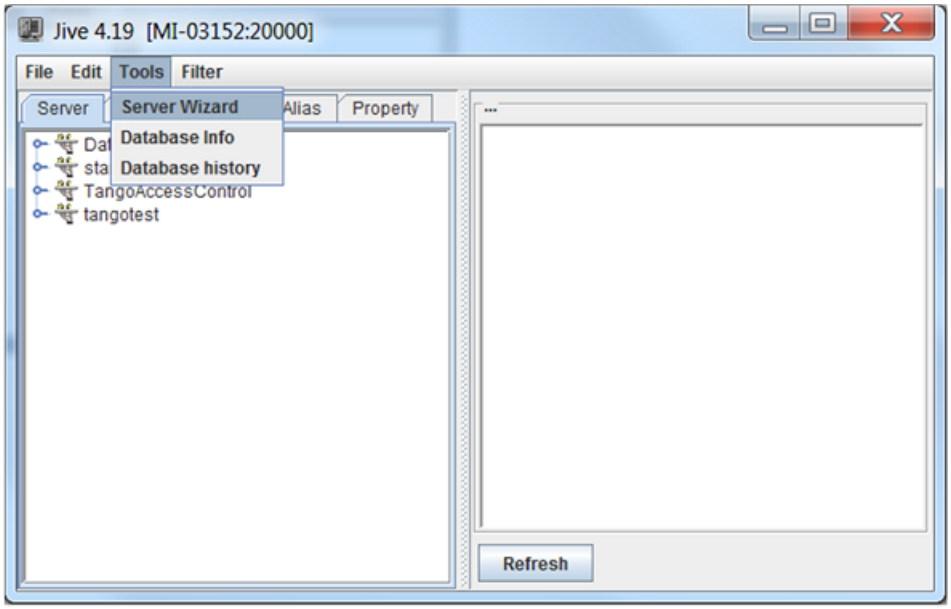
3.2 Example of plugin server setup : Basler detector

This procedure described the way to implement basler camera plugin. It is the same for whole the plugins, only properties may change.

You need to create a device server for Lima and another for the camera plugin. Lima device will use basler device thanks to "LimaCameraType" property. This property corresponds to the name of the camera plugin.

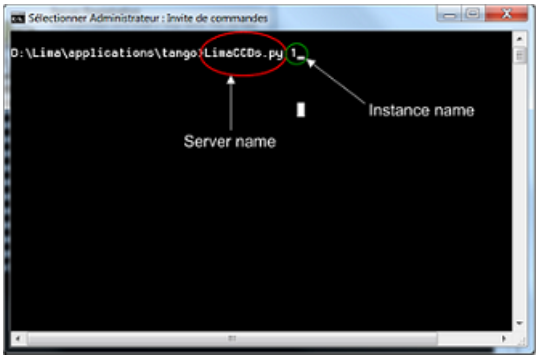
3.2.1 Lima device server

1. Run Jive and select "Tools->Server Wizard" menu. You must enter server and instance names



Click Next. . .

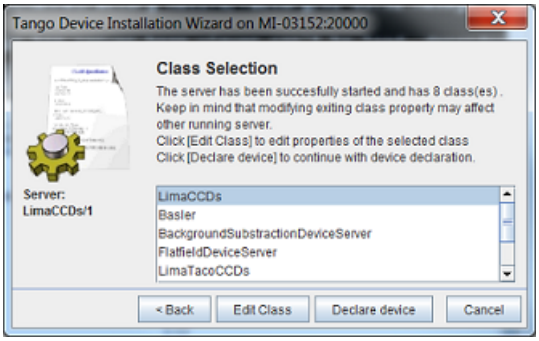
2. Start the Lima device server. Open a terminal and execute the command “server_name instance_name”



Click Next on the “Tango Device Installation Wizard” window

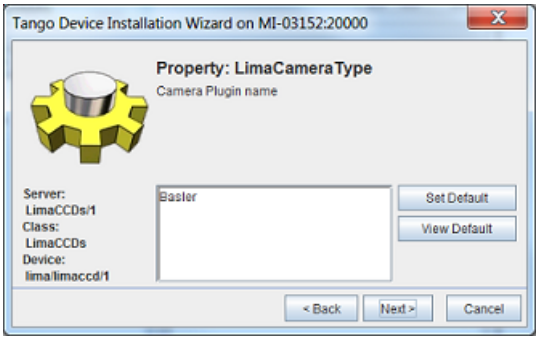
3. Declare a Lima device

   The Lima device server, contained several classes. For Basler camera you need to configure LimaCCDs and Basler classes.

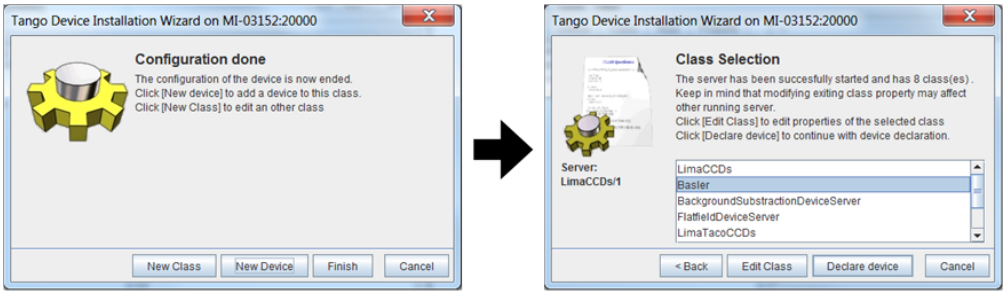


Select “LimaCCDs” class and click “Declare device” button. You must enter the device name with a string as “Domain/Family/member”.

Click Next and configure all the properties. You can let the default property values except for “LimaCameraType”. This property must contain the name of the Camera Plugin “Basler”.

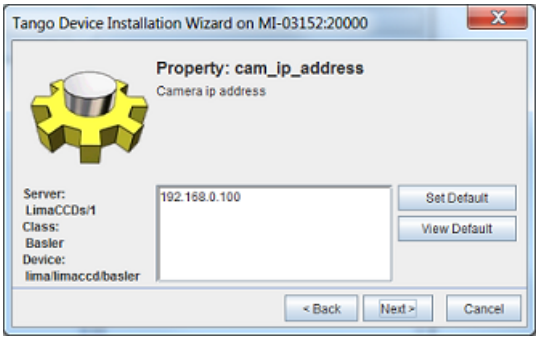


At the end of the configuration, click “New Class” button.

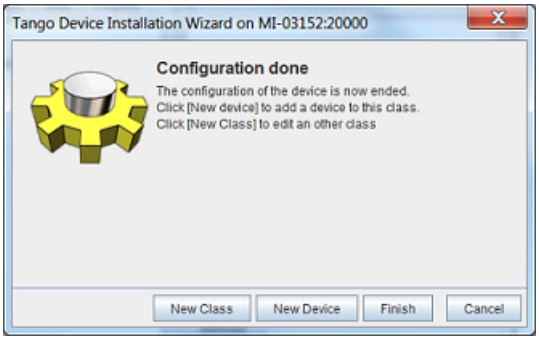


Select “Basler” class and click “Declare device” button. You must enter the device name with a string as “Domain/Family/member”.

Click Next and configure all the properties. You can let the default property values except for "cam_ip_adress". This property must contain the IP adress of the Basler camera.



Configuration is now ended, click "Finish"



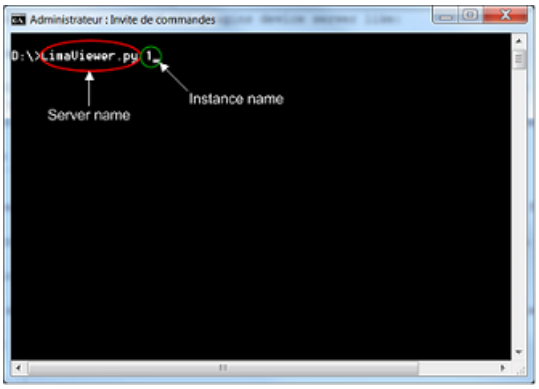
3.2.2 Lima Viewer

To test the Lima device server, you can use the LimaViewer. This is a device server which periodically get the last image from the buffer. It allows the user to check that Lima device server is operational. The procedure below describe how to install and configure the LimaViewer device server.

1. Run Jive and select "Tools->Server Wizard" menu. You must enter server and instance names

Click Next. . .

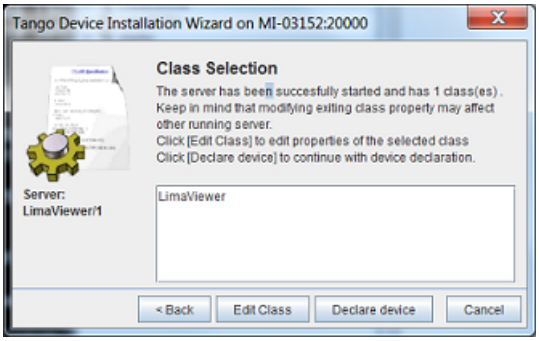
2. Start the LimaViewer device server. Open a terminal and execute the command “server_name instance_name”



Click Next on the “Tango Device Installation Wizard” window

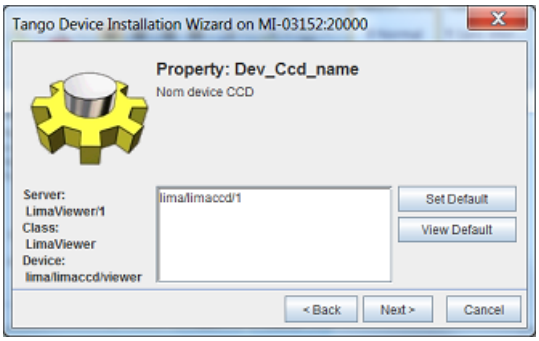
3. Declare a LimaViewer device

Select “LimaViewer” class and click “Declare device” button.

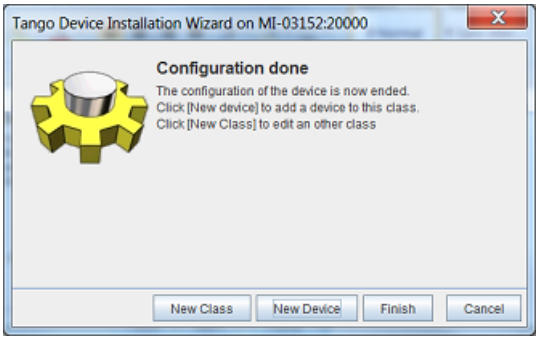


Enter the device name with a string as “Domain/Family/Member”.

Click Next and configure the “Dev_Ccd_name” property. This property corresponds to the name of the Lima device created before.

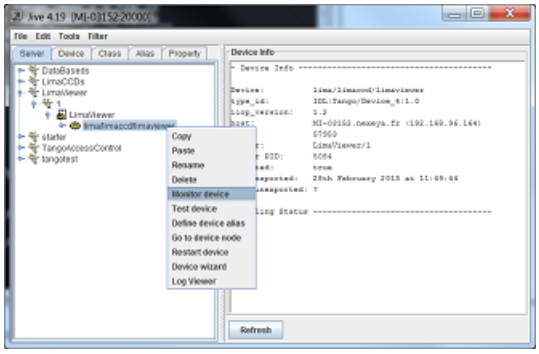


Configuration is now finished, click on “Finish”

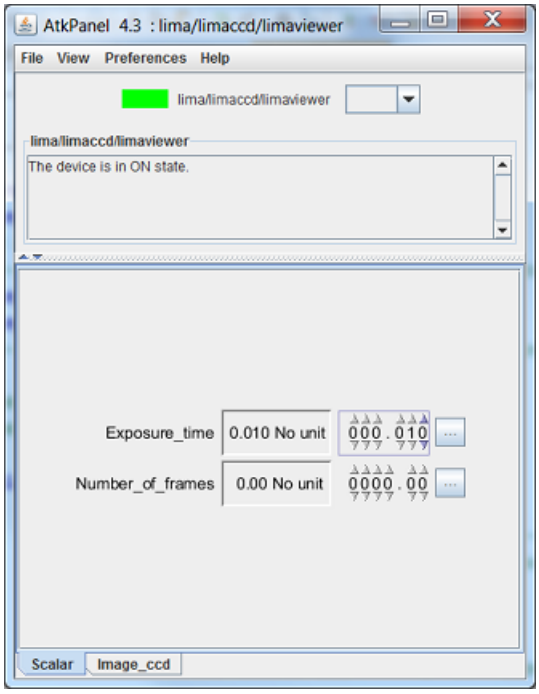


3.2.3 Test LimaCCDs device server with Jive

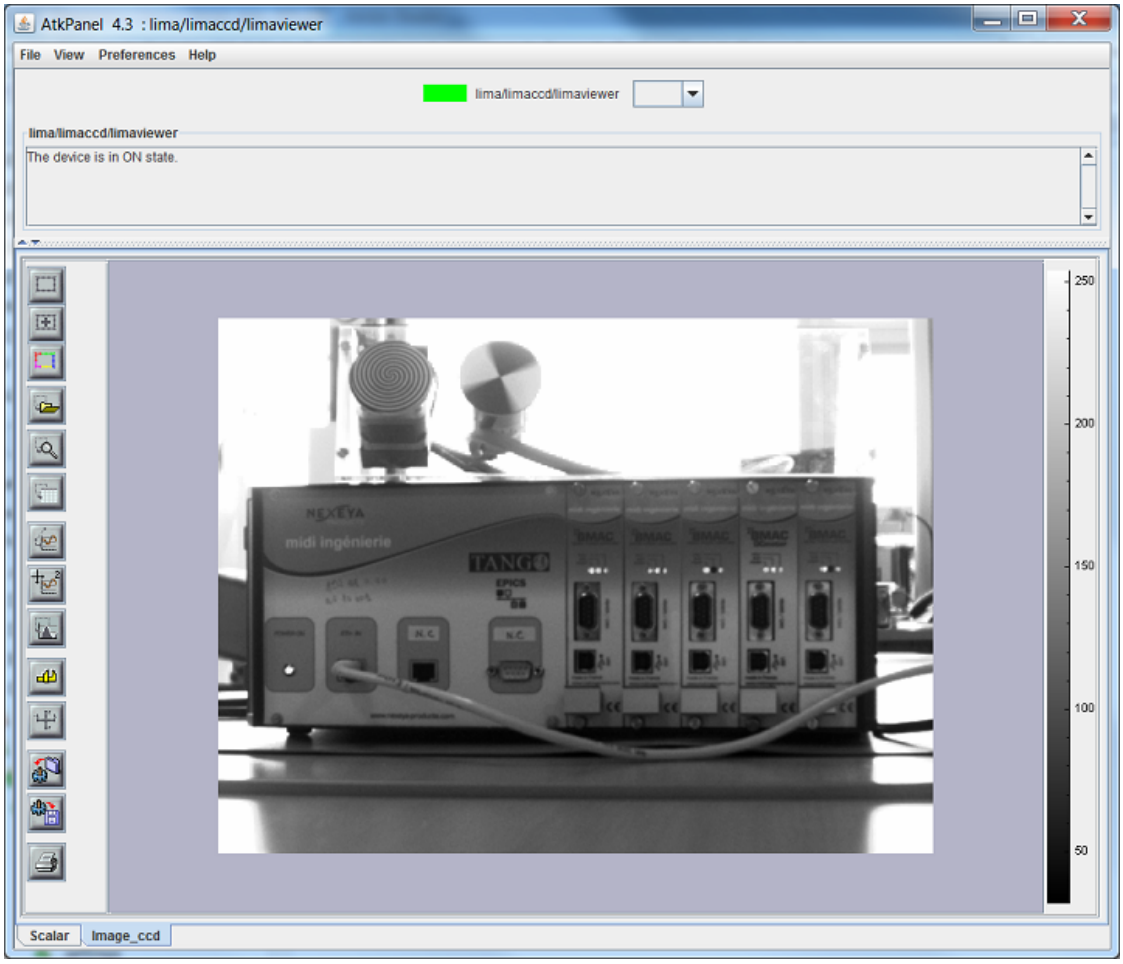
The LimaViewer device appears in the Device tab from Jive. Make a right click on the LimaViewer device server and select “Monitor Device”



AtkPanel is now launched. You can configure exposure time and the number of frames to acquire.



The camera image can be viewed by selecting the “image_ccd” tab



CHAPTER

FOUR

OVERVIEW

This section provides a big picture of LImA.

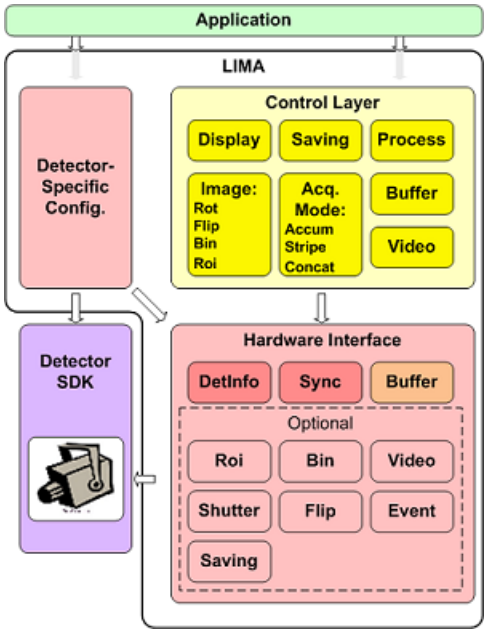


Fig. 1: Fig. 1 LImA Architecture

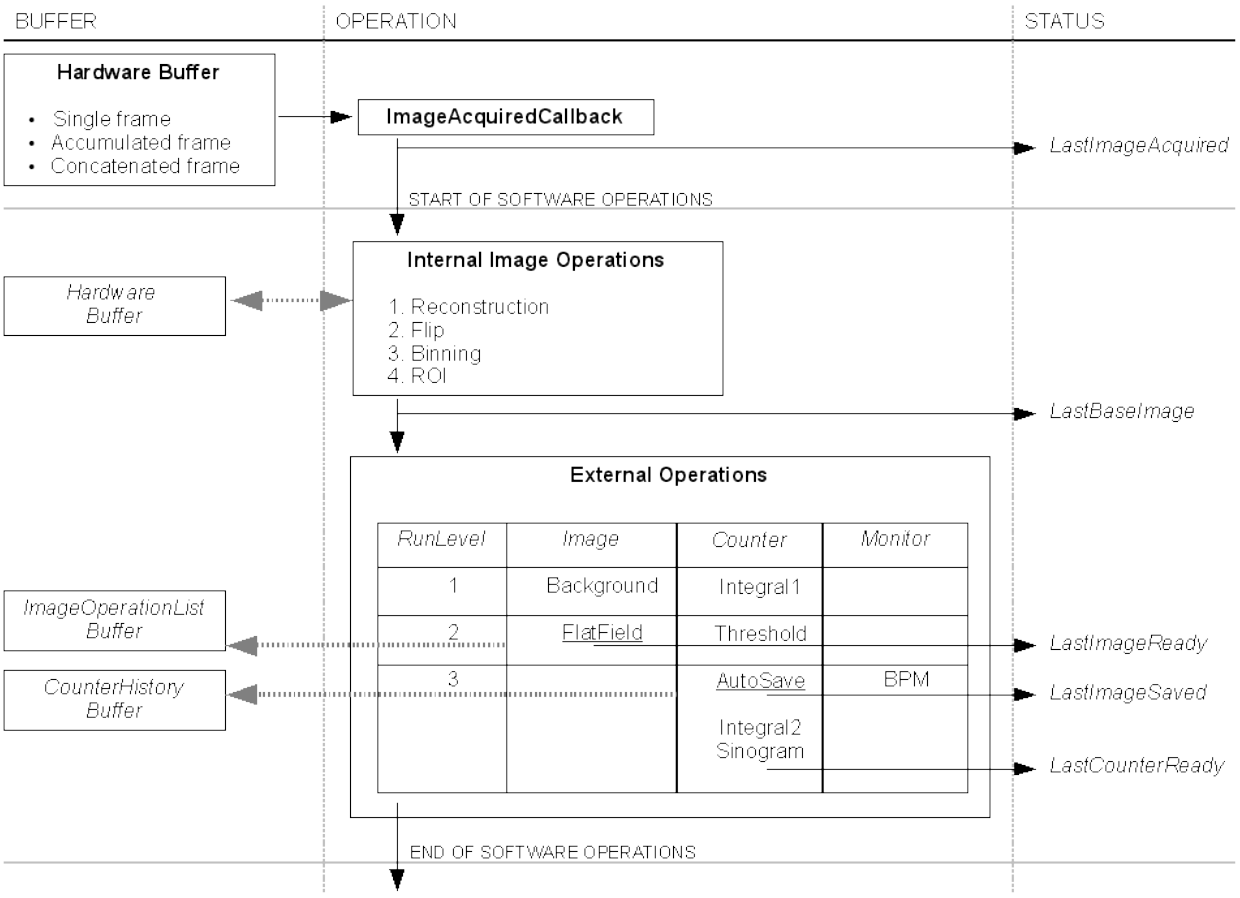


Fig. 2: Fig. 2 LImA Dataflow, Statuses and Events

CHAPTER

FIVE

CONCEPTS

CHAPTER

SIX

TUTORIAL

In this tutorial, we are going to write a program that prepares the camera and run a simple acquisition. We will be using the simulator, but every cameras should work in the same way. The program is in C++, the python binding being similar or simpler.

First some headers needs to be included :

- The `simulator/camera.h` that defines the `Camera` class for this specific cameras
- The `lima/ctcontrol.h` that defines the `CtControl` class which is the main user interface of LImA

If the library and plugin were not installed in the default locations, make sure to adjust the include search paths of your compiler.

```
#include <simulator/camera.h>
#include <lima/ctcontrol.h>
```

Then, the camera object is instantiated and the corresponding interface is constructed:

```
// A camera instance
simulator::Camera simu(/* some cameras have specific settings here, e.g. IP address */
↪);

// A hardware interface
simulator::Interface hw(simu);
```

At this point, the code specific to the camera code is over and we can instantiate the *lima::CtControl* object:

```
// The main control object
CtControl ct = lima::CtControl(&hw);
```

lima::CtControl is a class that aggregates many aspects of the configuration and the control of the cameras. Here is a non exhaustive lists of controls:

Control	Description
Acquisition	Controls exposure time, number of frames, trigger mode, etc...
Image	Controls cropping (ROI), binning, rotation and other processing applied either on hardware or by software...
Saving	Controls the file format, compression, metadata...
Shutter	Controls the shutter mode and open and closed times...
Buffer	Controls the number of buffer, the maximum memory to use...

These specific controls are accessible form the main *lima::CtControl* object.

```
// Get the acquisition, saving and image controls
CtAcquisition *acq = ct.acquisition();
CtSaving *save = ct.saving();
CtImage *image = ct.image();
```

All these control objects have member functions to set their parameters, either directly or using a the `Parameter` object, such as `lima::CtSaving::Parameter` (nested class). Here is how we could set the saving properties of our acquisition:

```
save->setDirectory("./data");
save->setPrefix("test_");
save->setSuffix(".edf");
save->setNextNumber(100);
save->setFormat(CtSaving::EDF);
save->setSavingMode(CtSaving::AutoFrame);
save->setFramesPerFile(100);
```

In the same way, image processing can configured to use a 2 x 2 binning:

```
image->setBin(Bin(2, 2));
```

Or acquisition parameters to get 10 frames with a 0.1 sec exposure:

```
acq->setAcqMode(Single);
acq->setAcqExpoTime(0.1);
acq->setAcqNbFrames(10);
```

Once we are happy with our settings, it's time to prepare the acquisition which perform multiple tasks such as buffer allocation, folder creation or applying the camera settings through the camera plugin and SDK.

```
// Prepare acquisition (transfer properties to the camera)
ct.prepareAcq();
```

If the preparation is successful, the acquisition can be started anytime with:

```
// Start acquisition
ct.startAcq();
```

That's all for now, have good fun with LImA!

CHAPTER

SEVEN

SUPPORTED CAMERAS

7.1 Conda packages

The following Conda packages are available from the *esrf-bcu* channel. Some cameras may required to manually install the drivers for the given SDK version.

Camera	Linux	Windows	SDK
Andor	Yes	Yes	linux 2.103 win 2.102
Andor3	Yes		sdk3 3.13
Basler	Yes	Yes	Pylon 5.0 / 5.1
Dexela	Yes		libDexela
Eiger (Dectris)	Yes		SIMPLON 1.8
Frelon	Yes		libEspia 3.10.0
ImXPAD	Yes		n/a
Maxipix	Yes		libEspia 3.10.0
Marccd	Yes		n/a
Merlin	Yes		n/a
Mythen3	Yes		n/a
PCO	Yes	Yes	PCO 1.23
Pilatus	Yes		n/a
Pixirad	Yes		n/a
Pointgrey	Yes		FlyCapture 2.3.3
Prosilica	Yes		PvAPI 1.24
Simulator	Yes	Yes	n/a
SLS Detector / PSI	Yes		SlsDetectorPackage v4
Ueye	Yes		uEye 4.61.0
Ultra	Yes		n/a
V4L2	Yes		v4l2
Xh	Yes		n/a

7.2 Windows Only

7.2.1 Hamamatsu

Introduction

The Hamamatsu Orca flash is digital CMOS camera. It supports USB3 or direct camera link connectivity.

- USB 3.0 -> 30fps
- Cameralink -> 100fps

The Lima plugin controls an Orca camera (**ORCA-Flash4.0 V2, C11440-22CU V2**) under Windows. It is based on the Hamamatsu DCAM-API SDK.

Prerequisite

Host OS is Windows (32 or 64 bits). The driver must be installed on the host system.

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_HAMAMATSU=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialization and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialization

There is nothing specific. The available cameras must first be enumerated. A selected camera can then be inited. (Note that at the moment only one camera will be handled by the pluggin.)

Std capabilities

This plugin has been implemented in respect of the mandatory capabilites but with some limitations according to some programmer's choices. We only provide here extra information for a better understanding of the capabilities of the Orca camera.

- HwDetInfo
- Max image size is : 2048 * 2048
- 16 bit unsigned type is supported
- Pixel size: 6.5µm * 6.5µm
- Detector type: Scientific CMOS sensor FL-400
- HwSync

  Supported trigger types are:

  - IntTrig
  - ExtTrigSingle
  - ExtGate (not yet implemented)

Optional capabilities

- HwBin

  Possible binning values are:

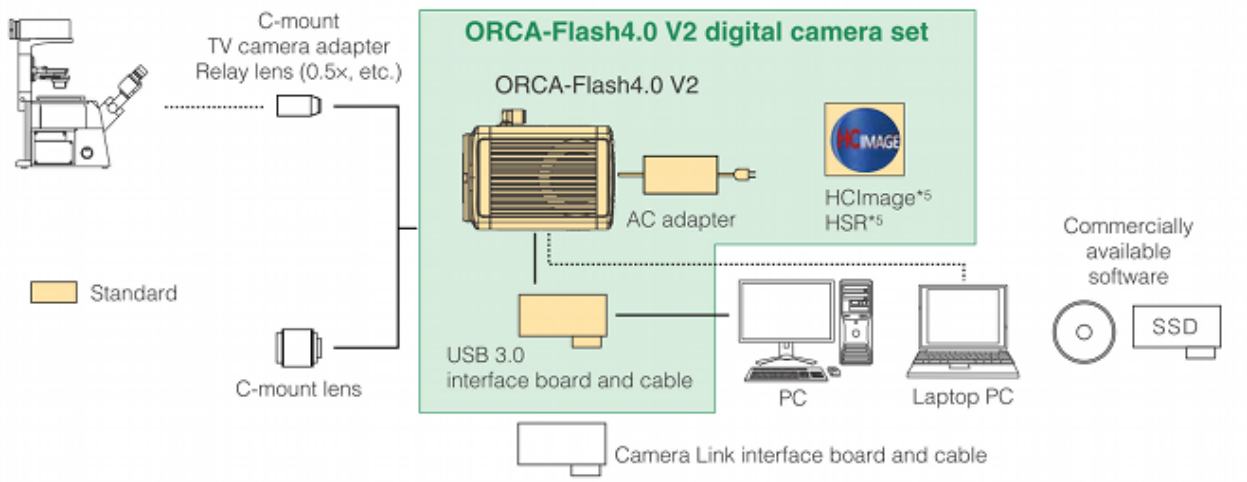
  - 1 * 1
  - 2 * 2
  - 4 * 4
- HwRoi

  The Subarray mode allows defining a rectangle for ROI:

  - X: 0 to 2044
  - Width: 4 to 2048
  - Y: 0 to 2044
  - Heigth: 4 to 2048

- HwShutter
- There is no shutter control available in the DCAM-API SDK.
- Cooling
- There is no cooler sensor access or control to the cooling system via the DCAM-API SDK.
- Cooling management is autonomous and can only be chosen between air or water cooling outside the sdk.
- Readout mode
- Two readout modes are available: SLOW (30fps at full frame) or NORMAL (100fps at full frame).

Configuration



How to use

The following set of functions is used as a wrapper to the DCAM-API SDK. Code can be found in the HamamatsuDCAMSDKHelper.cpp file.

```
dcam_init_open();                       // initialize DCAM-API and get a camera␣
↪handle.
dcamex_setsubarrayrect();               // Initialize the subarray mode (defines a␣
↪ROI -rectangle-)
dcamex_getsubarrayrect();               // Get the current subarray parameters (get␣
↪ROI settings)
dcamex_getimagewidth();                 // Get the width of the image
dcamex_getimageheight();                // Get the height of the image
dcamex_getfeatureinq();                 // Get the settings of a feature (ex:␣
↪exposure time)
dcamex_getbitsperchannel();             // Get the number of bits per channel
```

7.2.2 PCO camera

Introduction

- **PCO camera systems**
- PCO develops specialized fast and sensitive video camera systems, mainly for scientific applications; which covers digital camera systems with high dynamic range, high resolution, high speed and low noise. PCO home page
- **Product overview and technical data of the PCO cameras supported in LIMA**
- **PCO.dimax:** High speed 12 bit CMOS camera with fast image rates of 1469 frames per second (fps) at full resolution of 1920 x 1080 pixel. (tech data pcodimax)

- **PCO.edge:** Extremely low noise sCMOS camera with fast frame rates (100 fps), wide dynamic range (1:27000), high quantum efficiency, high resolution (2560 x 2160) and large field of view. (tech data pcoedge)
- **PCO.2000:** High resolution (2048 x 2048 pixel) and low noise 14bit CCD cooled camera system with internal image memory (camRAM), allows fast image recording with 160 MB/s. The available exposure times range from 500 ns to 49 days. (tech data pco2000)
- **PCO.4000:** High resolution (4008 x 2672 pixel) and low noise 14bit CCD cooled camera system with internal image memory (camRAM), allows fast image recording with 128 MB/s. The available exposure times range from 5 us to 49 days. (tech data)
- **Interface buses**
- **Cameralink:** used by **PCO.dimax** and **PCO.edge**
- **Cameralink HS:** used by **PCO.edge**
- **USB3.0:** used by **PCO.edge**
- **GigE:** used by **PCO.2000** and **PCO.4000**
- **Type of applications**
- Mainly used in scientific applications.
- **OS supported**
- **Win7 Professional** (english) 64 bits SP1.

Prerequisites

- **Required software packages**
  - **download links**
  - PCO and Silicon Software download (login/pw required)
  - VC++ download
  - GSL download
  - python download
  - numpy download
  - PyQt download
  - PyTango download
  - GIT download
  - **md5 checksum and size of packges used (maybe not updated)**

```
Silicon Software Runtime 5.4.4
    f8317c5145bac803f142c51b7c54ba27  RuntimeSetup_with_Applets_v5.4.4_Win64.exe
```

```
pco-sdk 1.20
    eb73ab0495a66c068c408344e20c8ad9  read_me.txt
    69a8f5667b71a8cf206d782e20f526ab  SW_PCOSDKWIN_120.exe
```

```
CAMWARE v403_1
   a9f8b2e465b7702ff727ba349ef327e8     SW_CAMWAREWIN64_403_1.exe
```

```
VC++ compiler
   Microsoft Visual Studio 2008
   Version 9.0.30729.1 SP
   Microsoft .NET Framework
   Version 3.5 SP1

   Installed Edition: Professional
   Microsoft Visual C++ 2008   91605-270-4441125-60040
   Microsoft Visual C++ 2008
```

```
Python
    8d10ff41492919ae93a989aba4963d14  numpy-MKL-1.8.1.win-amd64-py2.7.exe
    5a38820953d38db6210a90e58f85548d  PyTango-8.0.4.win-amd64-py2.7.exe
    b73f8753c76924bc7b75afaa6d304645  python-2.7.6.amd64.msi
```

```
pco edge CLHS / for firmware upgrade to 1.19
    9790828ce5265bab8b89585d8b8e83a9  pco.programmer_edgeHS.exe
    b9266e03a04ac9a9ff835311f0e27d94  pco_clhs_info.exe
    7e2f767684fb4ffaf5a5fac1af0c7679  sc2_clhs.dll
    2ed778785489846fd141f968dca3735b  README.txt
    6bdb7a27b0d7738762c878a33983dada  /FW_pco.edge_CLHS_020_V01_19.ehs
```

```
UTILS
   38ba677d295b4b6c17368bb86b661103  FileZilla_3.22.1_win64-setup_bundled.exe
   0377ccd0a3283617d161f24d080fb105  Git-1.9.0-preview20140217.exe
   3cbd2488210b6e7b3e7fa1baf05022d4  MobaXterm_Setup_7.1.msi
```

- **Enviroment variables**
- **system variables**

```
===> add manually the python path (it is not set by the installation program)
     PATH -> C:\Python26;

===> used for some utility batch files
     PATH -> C:\blissadm\bat;
```

- **user variables**

```
TANGO_HOST -> <host>:20000
```

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_PCO=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Post installation actions

- **enable/disable PCO logs**

```
===> rename file extensions (C:\ProgramData\pco):
     .txt (disabled) / .log (enabled) -----+
                                    camware.log   <---- created by hand
                                   PCO_CDlg.log
                                   PCO_Conv.log
                                    SC2_Cam.log
```

- **Command prompt console (Visual Studio)**

```
> All Programs
  > Microsoft Visual C++ 2008 Express Edition
    > Visual Studio Tools
      > Visual Studio 2008 Command Prompt
```

- **TODO**
- After installing PCO modules *Installation*
- And probably Tango server *PyTango Device Server*

Configuration

- **TODO**

PCO EDGE notes

PC characteristics (used for PCO EDGE at ESRF)

- **PROCESSOR**

```
2x Intel Xeon E5645 Six-Core CPU, 2,40GHz, 80W, Socket LGA1366, 12MB 5,86GT/sec

CPU's: 2x Xeon SixCore E5645 2,40Ghz 12MB 5,86GT/sec
Intel Xeon E5645 Six-Core CPU, 2,40GHz, 80W, Socket LGA1366, 12MB
external cache. 5,86GT/sec QPI speed. 1333MHz memory speed (DDR3 only).
Intel Technologies: Intel Turbo Boost , Intel Hyper-Threading
Technology, Intel Virtualization (VT-x), Intel Trusted Execution,
Enhanced Intel SpeedStep, Intel Demand Based Switching, Execute
Disable Bit.
```

- **RAM**

```
24 GB (6x DDR3-1333 Reg. ECC 4 GB module)
```

- **HD**

```
C:
  WDC WD5003ABYX-01WERA1
  Western Digital 500 GB, 7200 RPM, SATA 2, 300 Mbps

D:
  Adaptec RAID 5405/5405Q with 2 HD of 450 Gb -> RAID0 837 GB
  HUS156045VLS600
  Hitachi 450GB, 15,000RPM SAS / Serial Attached SCSI, 6Gbps
```

- **graphic card**

```
Matrox G200eW
```

- **motherboard**

```
Motherboard Extended ATX format 13,68in x 13in, (34,7cm x 33cm) (W x H);
2 socket LGA 1366-pin. It supports processors Quad-Core Intel Xeon
series 5500; QPI bus system (up to 6.4GT/s); *chipset Intel 5520*;

18 socket DIMM 240 pin, support for up to 288GB memory DDR3
1333/1066/800MHz Registered or 48GB memory DDR3 unbuffered ECC, the real
operating ram speed depends on the processor?s model and number of
installed ram, best performances are achieved through a triple channel
configuration;
```

- **PCI slots**

```
1x PCIe x4 (in x8 slot)
3x PCIe x8
1x PCIe x8 (in x16 slot)
2x PCIe x16
```

PCO EDGE - install instructions for Silicon Software Me4 Board

Check the document **camera/pco/doc/Me4_Installation_Test_e1.pdf** with the requirements and procedure to install the CameraLink grabber card. It is important in order to get the maximum transfer speed required by the PCO EDGE camera.

The boards tested by PCO are:

```
Supermicro X8ST3
GigaByte GA-X58A-UD3R
Intel S5520
Intel DX58SO2
Supermicro X8DTH-iF
```

With the PC described in *PCO EDGE notes* the speed of the CameraLink is about **570 MB/s** (66% of the theoretic max of 860 MB/s).

PCO EDGE - shutter mode (global/rolling)

```
cam.talk("rollingShutter 0")   <--- set shutter mode to GLOBAL

cam.talk("rollingShutter 1")   <--- set shutter mode to ROLLING
```

After the change of the shutter mode, the cam is rebooted and requires about 10s to became ready, meanwhile the acq status is AcqConfig.

The validRanges (exposure and latency time) are updated after the mode change.

7.2.3 Perkin Elmer camera

Intoduction

> "PerkinElmer is a world leader in the design, development, and manufacture of Amorphous Silicon (aSi) Flat Panel Detectors (FPD) designed to perform across a wide range of medical, veterinary, and industrial, Non-Destructive Testing (NDT) applications. Our XRD family of detectors provide superior image resolution, high frame rates up to 30 frames per seconds (fps), energy levels form 20 keV -15 MeV and easy information storage and retrieval."

The detector model we tested (ESRF) is : XRD 1621 CN ES

Prerequisite Windows 7

First, you have to install the Perkinelmer Windows7 SDK to the default path.

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_PERKINELMER=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Camera initialisation

The camera will be initialized by created the `PerkinElmer::Interface` object. The contructor will take care of your detector configuration according to the SDK installation setup done before.

Std capabilities

This plugin has been implement in respect of the mandatory capabilites but with some limitations which are due to the camera and SDK features. We provide here further information for a better understanding of the detector specific capabilities.

- HwDetInfo

  getCurrImageType/getDefImageType(): Bpp16 only.

  setCurrImageType(): this method do not change the image type which is fixed to Bpp16.

- HwSync

  get/setTrigMode(): the supported mode are IntTrig, ExtStartStop, ExtTrigReadout

Optional capabilities

In addition to the standard capabilities, we make the choice to implement some optional capabilities which are supported by the SDK and the I-Kon cameras. A Shutter control, a hardware ROI and a hardware Binning are available.

- HwBin

  Some camera models support binning 4x4, 2x2, 4x2 4x2 and 1x1 and others support only 2x2. Camera type si provided when initing the sdk (_InitDetector()) and only camera of type 15 supports the long range of binning.

Configuration

- Nothing special to do, but read the manual for proper installation.

How to use

This is a python code example for a simple test:

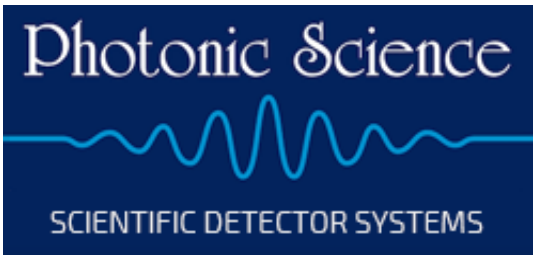
```
from Lima import PerkinElmer
from lima import Core

hwint = PerkinElmer.Interface()
ct = Core.CtControl(hwint)

acq = ct.acquisition()

# set offset and gain calibration, one image 1.0 second exposure
hwint.startAcqOffsetImage(1, 1.0)
hwint.startAcqGainImage(1, 1.0)

# set further hardware configuration
print (hwint.getGain())
hwint.setCorrectionMode(hwint.OffsetAndGain) # or No or OffsetOnly
hwint.setKeepFirstImage(False)

# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# set accumulation mode

acq_pars= acq.getPars()

#0-normal,1-concatenation,2-accumu
acq_pars.acqMode = 2
acq_pars.accMaxExpoTime = 0.05
acq_pars.acqExpoTime =1
acq_pars.acqNbFrames = 1
```

(continues on next page)

(continued from previous page)

```
acq.setPars(acq_pars)
# here we should have 21 accumalated images per frame
print (acq.getAccNbFrames())

# now ask for 2 sec. exposure and 10 frames
acq.setAcqExpoTime(2)
acq.setNbImages(10)

ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)
```

7.2.4 PhotonicScience



Introduction

> "Photonic Science is a high technology independent manufacturer of scientific detector systems covering the range of visible to x-ray and neutron detection. The camera technology offered is wide ranging, from CCD, EMCCD, CMOS to image intensified systems."

The CCD camera 4022 has been tested at ESRF on beamline ID11.

Prerequisite

TODO

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_PHOTONICSCIENCE=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

TODO

Std capabilities

This plugin has been implemented in respect of the mandatory capabilites but with some limitations which are due to the camera and SDK features. We only provide here extra information for a better understanding of the capabilities for Andor cameras.

- HwDetInfo

  TODO

- HwSync

  TODO

Optional capabilities

In addition to the standard capabilities, we make the choice to implement some optional capabilities which are supported by the SDK and the I-Kon cameras. A Shutter control, a hardware ROI and a hardware Binning are available.

- HwShutter

  TODO

- HwRoi

  TODO

- HwBin

  TODO

Configuration

TODO

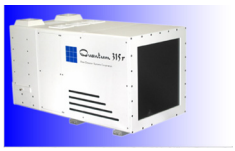
How to use

This is a python code example for a simple test:

```
from Lima import PhotonicScience
from lima import Core

#               camera library path
cam = Xh.Camera('ImageStar4022_v2.5\imagestar4022control.dll')
hwint = Xh.Interface(cam)
ct = Core.CtControl(hwint)

acq = ct.acquisition()

# configure some hw parameters

# set some low level configuration

# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# now ask for 2 sec. exposure and 10 frames
acq.setAcqExpoTime(2)
acq.setNbImages(10)

ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)
```

7.3 Linux Only

7.3.1 ADSC camera



Introduction

ADSC stands for Area Detector System Corporation.

Note: The Lima module has been tested only with the 315r model.

Prerequisite

2 programs have to be running on the ADSC server:

- `ccd_image_gather`
- `det_api_workstation`

Initialisation and Capabilities

In order to help people to understand how the camera plugin has been implemented in LImA this section provide some important information about the developer's choices.

Camera initialisation

Here are the available functions:

- `SetHeaderParameters()`
- `UseStoredDarkImage()`
- `SetImageKind()`
- `SetLastImage()`

Std capabilites

This plugin has been implemented in respect of the mandatory capabilites but with some limitations according to some programmer's choices. We only provide here extra information for a better understanding of the capabilities for the Adsc camera.

- HwDetInfo
  - Max image size is : 3072 * 3072
  - 16 bit unsigned type is supported

- HwSync
  - trigger type supported are: IntTrig

Optional capabilites

- HwBin
  - 1 * 1
  - 2 * 2

Configuration

No specific hardware configuration is needed.

How to use

here is the list of accessible fonctions to configure and use the ADSC detector:

```
void     setHeaderParameters(const std::string& header);
void     setStoredImageDark(bool value);
bool     getStoredImageDark(void);
void     setImageKind(int image_kind);
int      getImageKind(void);
void     setLastImage(int last_image);
int      getLastImage(void);

void                setFileName(const std::string& name);
const std::string&  getFileName(void);
void                setImagePath(const std::string& path);
const std::string&  getImagePath(void);
```

7.3.2 Andor SDK3

Introduction

Andor Technology manufactuer offers a large catalogue of scientific cameras. Covered scientific applications are low ligth imaging, spectroscopy, microscopy, time-resolved and high energy detection. Andor is providing a Software Development Tool (SDK) for both Windows and Linux, supporting different interface buses such as USB, CameraLink and also some specific acquisition PCI board. Unfortunately there was a significant API change between the v2 line of SDK and the brand new v3 of the SDK, and recent cameras are only supported by the v3 SDK, whilst this new SDK is not (yet ?) supporting previously built cameras.

The Lima module has been tested only with these camera models:

- Neo (sCMOS 3-tap, full Camera Link, Linux OS)
- Zyla (5.5 sCMOS, full Camera Link, Linux OS)

Installation & Module configuration

First, you have to install the Andor SDK the default path (/usr/local). For our test we used the SDK for Linux version **V3.3.30004.0** and ran the install script `install_andor` for which option 2 (64b linux) was selected, the default installation is made under `/usr/local/` with:

- `/usr/local/include`, header files
- `/usr/local/lib`, library files
- `/usr/local/andor/bitflow`, files for the frame-grabber driver (including camera firmware/frame grabber configuration)

The Linux SDK 3.3 has shared libraries which has been compiled on recent linux kernel, check first you have the right kernel and libc available by compiling one of the example program available under `examples/console`. Andor3 python module needs at least the lima core module.

The best before using this Lima pluggin with a Andor Neo camera is to test the proper setting of the frame-grabber driver and system configuration by using the two test programs included in the SDK. Those are typically found in `/usr/local/andor/examples/` and are `listdevices` and `image`.

Then, follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_ANDOR3=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Configuration

Connect the camera on both cameralink cables and power on.

How to use

A simple python test programm:

```
from Lima import Andor
from lima import Core

# -------------------+-----------------------------------------------+
#                    |                                               |
#                    v camlink config path                           v camera index
cam = Andor3.Camera('/users/blissadm/local/Andor3/andor/bitflow', 0)
```

(continues on next page)

(continued from previous page)

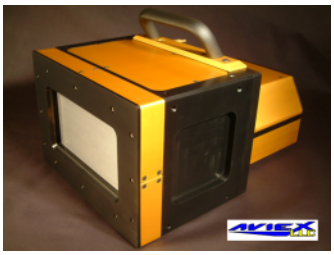
```
hwint = Andor3.Interface(cam)
ct = Core.CtControl(hwint)

acq = ct.acquisition()

# configure some hw parameters
hwint.setTemperatureSP(-30)
hwint.setCooler(True)
.... wait here for cooling

# set some low level configuration

hwint.setCooler(True)
hwint.setTemperatureSP(-55)
hwint.setFanSpeed(cam.Low)
hwint.setAdcGain(cam.b11_low_gain)
hwint.setAdcRate(cam.MHz100)
hwint.setElectronicShutterMode(cam.Rolling)
hwint.setOverlap(False)

# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# set accumulation mode

acq_pars= acq.getPars()

#0-normal,1-concatenation,2-accumu
acq_pars.acqMode = 2
acq_pars.accMaxExpoTime = 0.05
acq_pars.acqExpoTime =1
acq_pars.acqNbFrames = 1

acq.setPars(acq_pars)
# here we should have 21 accumulated images per frame
print acq.getAccNbFrames()

# now ask for 2 sec. exposure and 10 frames
acq.setAcqExpoTime(2)
acq.setNbImages(10)

ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady
```

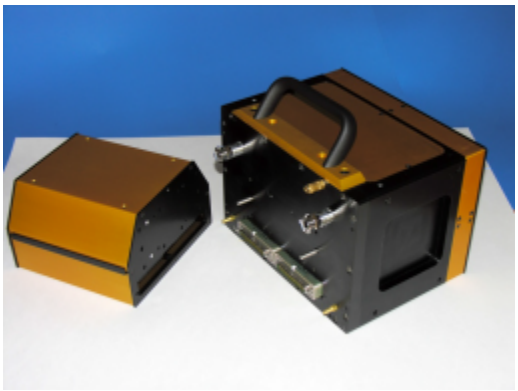
(continues on next page)

(continued from previous page)

```
# read the first image
im0 = ct.ReadImage(0)
```

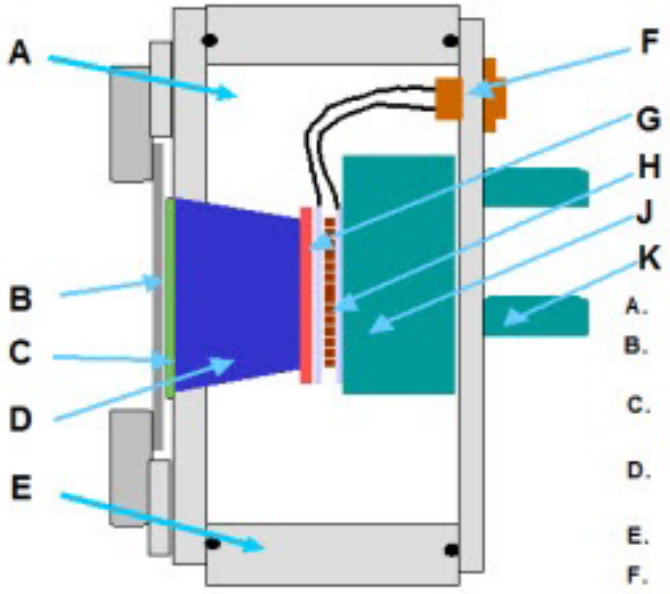
7.3.3 Aviex camera plugin





Intoduction

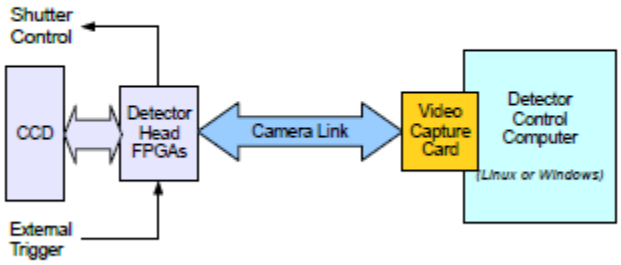
The PCCD-170170 is a large area detector (4096 x 4096) designed for use in WAXS or SAXS experiments in a vacuum environment.



Simplified mechanical layout of the 100 x 100 mm CCD detector head

A. Dry Argon
B. X-ray transparent window
C. GdOS X-ray phosphor screen
D. Fiber-optic coupler assembly
E. Detector housing
F. Hermetic feed-through
G. CCD
H. Thermo-electric cooler
J. Water cooled heat exchanger
K. Coolant connector

Hardware Block Diagram



Video Capture Card - PIXCI E4 *(PCI Express / Camera Link)*

The detector supports full frame, multiframe time-sliced, and streak camera modes of operation.

Used at the SWING beamline of Synchrotron SOLEIL to make timeresolved SAXS measurements together with another WAXS detector.

This Lima plugin controls an Aviex camera under linux.

It is based on the MX beamline control toolkit.

It has been tested at the Synchrotron SOLEIL facility, but has not been installed yet on a Beamline.

Module configuration

First, compile the Mx Library/Driverand and install it in the default path (`/opt/mx/`).

Start the Mx driver with:

```
cd /opt/mx/sbin/
./mx start
```

Then, follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_AVIEX=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

There are 2 parameters to be filled with your Lima client:

- The detector friendly name: can be any string defined by user.
- The detector database file name: this file must contains configuration parameters such as IP adress, port.

Std capabilites

This plugin has been implemented in respect of the mandatory capabilites but with some limitations according to some programmer's choices. We only provide here extra information for a better understanding of the capabilities for the Aviex camera.

- HwDetInfo
- Max image size is : 4096 * 4096
- 16 bit unsigned type is supported
- HwSync trigger type supported are:
  - IntTrig
  - ExtTrigSingle

Optional capabilites

- HwBin
  - 1 * 1
  - 2 * 2
  - 4 * 4
  - 8 * 8
  - Binning above are typical values, but binning is not necessarily square.
- HwRoi

  Not yet implemented

Configuration

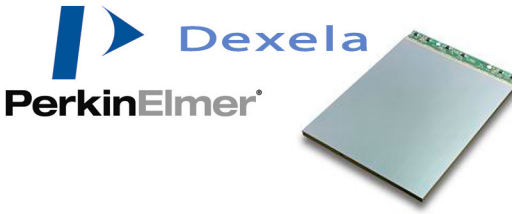
No specific hardware configuration is needed.

How to use

Here is the list of accessible fonctions to configure and use the Aviex detector:

```
//-- Related to Aviex specific features
void getExpMultiplier(double& exp_mult);
void setExpMultiplier(double exp_mult);
void getLatencyTime(double& period_time);
void setLatencyTime(double  period_time);
void getGapMultiplier(double& gap_mult);
void setGapMultiplier(double  gap_mult);
void getMxLibraryVersion(std::string& version);
void getInternalAcqMode(std::string& acq_mode);
//! Available mode : ONESHOT, MULTIFRAME, GEOMETRICAL, MEASURE_DARK, MEASURE_FLOOD_
↪FIELD
void setInternalAcqMode(const std::string& mode);
void getReadoutDelayTime(double& readout_delay);
void setReadoutDelayTime(double readout_delay);
void getReadoutSpeed(bool& readout_speed);
void setReadoutSpeed(bool readout_speed);
void getInitialDelayTime(double& initial_delay);
void setInitialDelayTime(double initial_delay);
//! MASK_CORRECTION_BIT_POSITION                = 0
//! BIAS_CORRECTION_BIT_POSITION                = 1
//! DARK_CORRECTION_BIT_POSITION                = 2
//! FLOOD_CORRECTION_BIT_POSITION               = 3
//! GEOM_CORRECTION_BIT_POSITION                = 12
void setCorrectionFlags(unsigned long);
```

7.3.4 Dexela camera plugin



Introduction

The Dexela detector is a brand product of PerkinElmer. PerkinElmer has recently Acquired Dexela Limited a manufacturer of CMOS flat panel. Nevertheless the Dexela detector SDK still remains not compatible with the other PerkinElmer detector SDK (see perkinelemer plugin) and one need to use this camera plugin instead.

Prerequisite

The Dexela detector model sensor2923 only has been tested at ESRF.

The detector is controlled via an acquisition board: PIXCI(R) E4 PCIExpress Camera Link board (EPIX,Inc.).

You need to install the acquisition card SDK. It was tested with 3.8 version (xclib). You can find them at http://www.epixinc.com/support/files.php .

You also need to install libdexela which is not yet GPL. See detail with mihael.koep@softwareschneiderei.de.

BIOS configuration

You should disable all power saving mode like CSTATE and disable also multiple-threading feature of cpu.

At ESRF, SuperMicro computers have to be configured like this:

- Simultaneous Multi-threading has to be disabled
- C1E support has to be disabled
- Intel CSTATE Tech has to be disabled

Linux kernel configuration

As the PIXCI acquisition card needs a low jitters configuration, you need to change some kernel parameters. To do so, you have to change in grub configuration file (under `/etc/default/grub for debian`) the `GRUB_CMDLINE_LINUX_DEFAULT` by adding theses options:

```
pcie_aspm=off
intel_idle.max_cstate=0
processor.max_cstate=0
idle=poll
mce=ignore_ce
ipmi_si.force_kipmi=0
nmi_watchdog=0
noht
```

(continues on next page)

(continued from previous page)

```
nosoftlockup
isolcpus=0
```

the whole line should look something like this:

```
GRUB_CMDLINE_LINUX_DEFAULT="ipv6.disable=1 quiet pcie_aspm=off intel_idle.max_
↪cstate=0 processor.max_cstate=0 idle=poll mce=ignore_ce ipmi_si.force_kipmi=0 nmi_
↪watchdog=0 noht nosoftlockup isolcpus=0"
```

You also have to uninstall or disable the irqbalance process. On Debian you can simply type:

```
sudo apt-get purge irqbalance
```

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_DEXELA=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialization and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialization

The camera will be initialized within the `DexelaInterface` object. The parameter to pass to `DexelaInterface()` constructor is the fill path need for the acquisition card. This file is generated by xcap software provided by PIXCI. you can find some example in the config directory.

Std capabilities

This plugin has been implemented in respect of the mandatory capabilites but with limitations according due to the detector specific features and with some programmer's choices. We do not explain here the standard Lima capabilites but you can find in this section the useful information on the Dexela specfic features.

- HwDetInfo

  The Dexela detector as a pixel size of 74.8e-6 m (74.8 um) and the image data type is fixed to 16bpp (bit per pixel).

- HwSync

  The supported trigger modes are IntTrig, IntTrigMult, ExtTrigMult and ExtGate.

The exposure time range is 0.0116 (1/86) to 120 seconds.

The latency time is not manage.

Optional capabilities

In addition to the standard capabilities, we make the choice to implement some optional capabilities in order to have an improved simulation.

- HwShutter

  There is no shutter capability.

- HwRoi

  There is no hardware capability, but Lima provides the sofware Roi as well.

- HwBin

  The supported hardware binning factors are 1x1, 2x2, and 4x4.

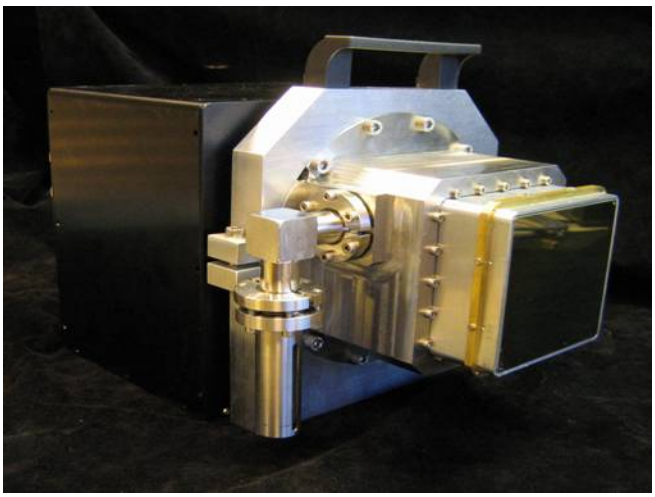
How to use

The LimaCCDs tango server provides a complete interface to the dexela plugin so feel free to test.

For a quick test one can use python, is this a short code example:

```
from Lima import Dexela
from lima import Core
import time

hwint = Dexela.Interface('./sensor2923.fmt')
ct = Core.CtControl(hwint)

acq = ct.acquisition()

# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/tmp/'
pars.prefix='testdexela_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)


# now ask for 2 sec. exposure and 10 frames
acq.setAcqExpoTime(2)
acq.setNbImages(10)

ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)
```

7.3.5 Frelon camera



Introduction

The FReLoN camera is a 14 bit dynamic CCD camera, with a 2048*2048 pixel chip. This camera has been developped by the awesome people with the 'Analog and Transient Electronic' ESRF group.

Prerequisite

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_FRELON=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

The Frelon plugin provides a helper class `FrelonAcq` which manages the initialisation sequence with the camera and interface object. An Espia board channel number should be set as the initialisation parameter (default is 0).

```
frelon = Frelon.FrelonAcq(int(espia_dev_nb))
return frelon.getGlobalControl()
```

Std capabilites

This plugin has been implemented in respect of the mandatory capabilites but with limitations according due to the detector specific features and with some programmer's choices. We do not explain here the standard Lima capabilites but you can find in this section the useful information on the Dexela specfic features.

- HwDetInfo

  TODO

- HwSync

  TODO

Optional capabilites

In addition to the standard capabilities, we make the choice to implement some optional capabilities in order to have an improved simulation.

- HwShutter

  TODO

- HwRoi

  TODO

- HwBin

  TODO

Configuration

The main configuration will consist in providing the correct `DexelaConfig.cfg` file to the detector API. The file has to be provided by the manufacturer with a second file like `sensor2923.fmt`. The `.fmt` file contains some calibration data.

How to use

The LimaCCDs tango server provides a complete interface to the dexela plugin so feel free to test.

For a quick test one can use python, this is a short example code:

```
from Lima import Frelon
from lima import Core
import time

FrelonAcq  = Frelon.FrelonAcq(int(espia_dev_nb))
control = FrelonAcq.getGlobalControl()

acq = control.acquisition()

# setting new file parameters and autosaving mode
saving=control.saving()

pars=saving.getParameters()
pars.directory='/tmp/'
```

(continues on next page)

(continued from previous page)

```
pars.prefix='testfrelon_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)


# now ask for 2 sec. exposure and 10 frames
acq.setAcqExpoTime(2)
acq.setNbImages(10)

acq.prepareAcq()
acq.startAcq()

# wait for last image (#9) ready
lastimg = control.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(1)
  lastimg = control.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = control.ReadImage(0)
```

7.3.6 Maxipix

Intoduction

MAXIPIX is a high spatial resolution (small pixels), high frame rate, photon-counting pixel detector developed by ESRF. MAXIPIX is based on MEDIPIX2/TIMEPIX readout ASICs developed by CERN and the MEDIPIX2 collaboration. The active detector element consists of a hybrid pixel circuit glued on a chipboard and connected to it with microwire connections. The hybrid pixel circuit consists itself of a pixelated semiconductor sensor connected to one or several readout ASICs by individual micro solder bumps on each pixel. Various module formats are available and may implement either MEDIPIX2 or TIMEPIX ASICs. Both ASICs have identical pixel geometries but different characteristics as regards principally the lowest energy threshold, the discriminator range, and the available detection modes.

We provide today Maxipix 5x1, 4x1 and 1x1 formats based on both TIMEPIX and MEDIPIX2 ASICs.

Beamlines are equiped with the detector, Espia card and a specific computer running centOS 5 x86_64.

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_MAXIPIX=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

The camera will be initialized within the `Maxipix::Camera` class. Camera contructor aims to load the configuration and calibration data to the detector backend electronic (Priam card).

There are so many hardware parameters you can set, but refer to the maxipix documentation for a good pratice.

> set/getSignalLevel() set/getReadLevel() set/getTriggerLevel() set/getShutterLevel() set/getReadyMode() set/getGateMode() set/getFillMode() set/getEnergy()

Std capabilites

This plugin has been implemented in respect of the mandatory capabilites but with some limitations which are due to the camera. We only provide here extra information for a better understanding of the capabilities for Maxipix cameras.

- HwDetInfo

  getCurrImageType/getDefImageType(): always 16bpp.

  setCurrImageType(): this method do not change the image type which is fixed to 16bpp.

- HwSync

  get/setTrigMode(): supported modes are IntTrig, IntTrigMult,ExtTrigSingle, ExtTrigMult and ExtGate.

Optional capabilites

In addition to the standard capabilities, we make the choice to implement some optional capabilities which are supported by this detector. A Shutter control.

- HwShutter

  setMode(): only ShutterAuto and ShutterManual modes are supported.

Configuration

Only provided configuration files (`.cfg` and `.bpc`) must be used for your detector, you must not change those files. Each detector has its own set of files. Please contact ESRF Detector group for help.

How to use

This is a python code example of a simple acquisition:

```
from Lima.Maxipix import Maxipix
from lima import Core

#-------------------------------------------------------------------------+
#                                          config name (.cfg file)        |
#-------------------------+                                               |
#     config path         |                                               |
#-------------------- +   |                                               |
#    espia channel    |   |                                               |
#                     v   v                                               v
cam = Maxipix.Camera(0, '/users/blissadm/local/maxipix/calib/tpxatl25', 'tpxatl25X')

hwint = Maxipix.Interface(cam)
ct = Core.CtControl(hwint)

acq = ct.acquisition()

# set some low level configuration
# see maxipix documentationf for more information
hwint.setEnergyThreshold(10.0)
hwint.setFillMode(cam.DISPATCH)
hwint.setShutterLevel(cam.HIGH_RISE)


# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# set accumulation mode

acq_pars= acq.getPars()

#0-normal,1-concatenation,2-accumu
acq_pars.acqMode = 2
acq_pars.accMaxExpoTime = 0.05
acq_pars.acqExpoTime =1
acq_pars.acqNbFrames = 1

acq.setPars(acq_pars)
# here we should have 21 accumalated images per frame
print acq.getAccNbFrames()
```

(continues on next page)

(continued from previous page)

```
# now ask for 2 sec. exposure and 10 frames
acq.setAcqExpoTime(2)
acq.setNbImages(10)

ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)
```

7.3.7 DECTRIS EIGER

Introduction

The EIGER 1M is a high performance X-Ray detector system. It is made of two subsystems: a detector and a control server. The control server is driven using an HTTP RESTful interface.

A C++ API for LImA has been developed at Synchrotron SOLEIL.

Prerequisite

Some dependencies need to be installed:

- libcurl
- liblz4
- libzmq
- libjsoncpp

to install all dependencies on debian like system, use this command:

```
$ sudo apt-get install libcurl4-gnutls-dev liblz4-dev libzmq3-dev libjsoncpp-dev
```

Installation and Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_EIGER=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialization

Initialization is performed automatically within the Eigercamera object. By default the stream will be use to retrieved images unless hardware saving is activated (CtSaving::setManagedMode(CtSaving::Hardware))

Std capabilities

- HwDetInfo

Capability	1M Value	4M Value	9M Value	16M Value
Maximum image size	1030 * 1065	2070 * 2167	3110 * 3269	4150 * 4371
Pixel depth	12 bits	12 bits	12 bits	12 bits
Pixel size	75µm * 75µm	75µm * 75µm	75µm * 75µm	75µm * 75µm
Maximum frame rate	3000Hz	750Hz	238Hz	133Hz

- HwSync

  Supported trigger types are:

- IntTrig
- IntTrigMult
- ExtTrigSingle
- ExtTrigMult
- ExtGate
- There is no hardware support for binning or roi.
- There is no shutter control.

Optional capabilities

- **Cooling**
- The detector uses liquid cooling.
- The API allows accessing the temperature and humidity as read-only values.

At the moment, the specific device supports the control of the following features of the Eiger Dectris API.
(Extended description can be found in the Eiger API user manual from Dectris).

- **Photon energy**: This should be set to the incoming beam energy. Actually it's an helper which set the threshold
- **Threshold energy**: This parameter will set the camera detection threshold. This should be set between 50 to 60 % of the incoming beam energy.
- **Auto Summation** (if activate image depth is 32 and, if not image depth is 16)
- **HwSaving**: This detector can directly generate hd5f, if this feature is used. Internally Lima control the file writer Eiger module. This capability can be activated though the control part with CtSaving object with setManagedMode method.
- **Countrate correction**
- **Efficiency correction**
- **Flatfield correction**
- **LZ4 Compression**
- **Virtual pixel correction**
- **Pixelmask**

Configuration

- Device configuration

  The default values of the following properties must be updated in the specific device to meet your system configuration.

Property name	Description	Default value
DetectorIP	Defines the IP address of the Eiger control server (ex: 192.168.10.1)	127.0.0.1

How to use

This is a python code of a simple acquisition:

```
from Lima import Eiger
from lima import Core

#------------------+
#                  |
#                  v ip adress or hostname
cam = Eiger.Camera(lid32eiger1)

hwint = Eiger.Interface(cam)
ct = Core.CtControl(hwint)

acq = ct.acquisition()

# set hardware configuration
# refer to the Dectris Eiger documentation for more information
cam.setCountrateCorrection(False)
cam.setFlatfieldCorrection(True)
cam.setAutoSummation(False)
cam.setEfficiencyCorrection(True)
cam.setVirtualPixelCorrection(True)
cam.setPixelMask(True)

# read some parameters
print (cam.getTemperature())
print (cam.getHumidity())


# set energy threshold in KeV
cam.seThresholdEnery(16.0)
cam.setPhotonEnergy(16.0)

# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)
```

(continues on next page)

(continued from previous page)

```
# set accumulation mode

acq_pars= acq.getPars()

# now ask for 10 msec exposure and 10 frames
acq.setAcqExpoTime(0.01)
acq.setNbImages(10)

ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)
```

7.3.8 Dectris Mythen camera

Introduction

Server for the control of a Mythen detector.

Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_MYTHEN=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Installation

Configuration

7.3.9 Dectris Mythen3

Intoduction

Server for the control of a Mythen detector.

Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_MYTHEN=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Testing

Here is a simple python test program:

```
import time
from Lima import Mythen3
from Lima import Core
import time

camera = Mythen3.Camera("160.103.146.190", 1031, False)
interface = Mythen3.Interface(camera)
control = Core.CtControl(interface)

# check its OK
print camera.getDetectorType()
print camera.getDetectorModel()
print camera.getVersion()

nframes=10
acqtime=2.0
# setting new file parameters and autosaving mode
saving=control.saving()
saving.setDirectory("/buffer/dubble281/mythen")
saving.setFramesPerFile(nframes)
saving.setFormat(Core.CtSaving.HDF5)
saving.setPrefix("mythen3_")
saving.setSuffix(".hdf")
saving.setSavingMode(Core.CtSaving.AutoFrame)
saving.setOverwritePolicy(Core.CtSaving.Overwrite)

# do acquisition
acq = control.acquisition()
acq.setAcqExpoTime(acqtime)
acq.setAcqNbFrames(nframes)

control.prepareAcq()
control.startAcq()
time.sleep(25)
```

7.3.10 Dectris Pilatus

Intoduction

The PILATUS detector (pixel apparatus for the SLS) is a novel type of a x-ray detector, which has been developed at the Paul Scherrer Institut (PSI) for the Swiss Light Source (SLS). PILATUS detectors are two-dimensional hybrid pixel array detectors, which operate in single-photon counting mode. A hybrid pixel that features single photon counting, comprises a preamplifier, a comparator and a counter. The preamplifier enforces the charge generated in the sensor by the incoming x-ray; the comparator produces a digital signal if the incoming charge exceeds a predefined threshold and thus, together with the counter, one obtains a complete digital storage and read-out of the number of detected x-rays per pixel without any read-out noise or dark current!

PILATUS detectors feature several advantages compared to current state-of-the-art CCD and imaging plate detectors. The main features include: no readout noise, superior signal-to-noise ratio, read-out time of 5 ms, a dynamic range of 20bit, high detective quantum efficiency and the possibility to suppress fluorescence by a energy threshold that is set individually for each pixel. A more complete comparison is given in Table 1. The short readout and fast framing time allow to take diffraction data in continuous mode without opening and closing the shutter for each frame (see Fig. 1). For a comparison on the response to x-rays of integrating and single photon counting detectors see Fig. 2.

Because of the specified properties, PILATUS detectors are superiour to state-of-the-art CCD and imaging plate detectors for various x-ray detection experiments. Major improvements can be expected for time-resolved experiments, for the study of weak diffraction phenomena (e.g. diffuse scattering), for accurate measurements of Bragg intensities, for resonant scattering experiments,...

Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_PILATUS=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Installation

On Pilatus PC, create **as root** a ramdisk of 8GB which will be used by Lima dserver as temporary buffer:

- edit file `/etc/fstab` and add the following line:

```
none                    /lima_data tmpfs    size=8g,mode=0777       0 0
```

- make the directory:

```
mkdir /lima_data
```

- and finally mount the ramdisk:

```
mount -a
```

- For Pilatus3, edit file `~det/p2_det/config/cam_data/camera.def` and add thoses two lines:
  - camera_wide = WIDTH_OF_THE_DETECTOR
  - camera_high = HEIGHT_OF_THE_DETECTOR

Start the system

- Log on the detector pc as *det* user start tvx/camserver:

```
cd p2_det
./runtvx
```

- when tvx has finished initializing camserver just type *quit* in tvx window
- Log on the detector pc as *an other user* or *det*

```
cd WHERE_YOU_HAVE_INSTALL_PILATUS_TANGO_SERVER
TANGO_HOST=Host:Port python LimaCCD.py instance_name
```

If the cameserver window notice a connection, seams to work ;)

How to use

This is a python code example for a simple test:

```
from Lima import Pilatus
from Lima import Core

cam = Pilatus.Camera()
hwint = Pilatus.Interface(cam)
ct = Core.CtControl(hwint)

acq = ct.acquisition()

# set some low level configuration
cam.setThresholdGain(1)
cam.setFillMode(True)
cam.setEnergy(16.0)
cam.setHardwareTriggerDelay(0)
cam.setNbExposurePerFrame(1)

# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# now ask for 2 sec. exposure and 10 frames
```

(continues on next page)

(continued from previous page)

```
acq.setAcqExpoTime(2)
acq.setAcqNbFrames(10)

ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)
```

7.3.11 Finger Lakes Instrumentation Microline camera plugin

camera/fli/doc/microline.png

Introduction

FLI supplies cameras to more than 50 countries for life science imaging, veterinary radiology, astronomy, forensics, transmission electron microscopy, and a wide range of other applications. Our on-site staff includes a talented group of mechanical, electrical, and software engineers. FLI provides a two Software Development Tool (SDK) for both Windows and Linux.

The Lima module as been tested only with this cameras models:

- IKon-M and IKon-L (USB interface, Linux OS debian 6)
- IKon-L (USB interface, Windows XP - 32bits)

Prerequisites

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_FLI=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

The camera will be initialized within the `AndorCamera` object. The AndorCamera contructor sets the camera with default parameters for Preampifier-Gain, VerticalShiftSpeed and the ADC/HorizontalSpeed.

These parameters are optimized for the faster mode, which means the maximum gain, the "fasten recommended" VSSpeed (i.e as returned by GetFastestRecommendedVSSpeed() SDK function call) and the ADC with the faster Horizontal speed.

All the parameters can be set and get using the corresponding methods, the default values (max speeds and gain) can be applied with -1 as passed value:

set/getPGain()

set/getVsSpeed()

set/getADCSpeed()

Some other methods are available but they can not be supported depending on which camera model you are using:

set/getHighCapacity()

set/getFanMode()

set/getBaselineClamp()

The above parameters, only support enumerate type for values.

Std capabilites

This plugin has been implemented in respect of the mandatory capabilites but with some limitations which are due to the camera and SDK features. We only provide here extra information for a better understanding of the capabilities for Andor cameras.

- HwDetInfo

  getCurrImageType/getDefImageType(): the methods call the SDK GetBitDepth() function to resolve the image data type. The bit-depth correspond to the AD channel dynamic range which depends on the selected ADC channel. By experience and with IKon detectors we only have Bpp16 of dynamic range, but the methods can return Bpp8 and Bpp32 as well.

  setCurrImageType(): this method do not change the image type which is fixed to 16bpp.

- HwSync

  get/setTrigMode(): the only supported mode are IntTrig, ExtTrigSingle, ExtGate and IntTrigMult

Optional capabilites

In addition to the standard capabilities, we make the choice to implement some optional capabilities which are supported by the SDK and the I-Kon cameras. A Shutter control, a hardware ROI and a hardware Binning are available.

- HwShutter

  setMode(): only ShutterAuto and ShutterManual modes are supported

- HwRoi

  There is no restriction for the ROI setting

- HwBin

  There is no restriction for the Binning but the maximum binning is given by the SDK function GetMaximumBinning() which depends on the camera model

Configuration

Plug your USB camera on any USB port of the computer, that's it !

How to use

This is a python code example for a simple test:

```
from Lima import FLI
from lima import Core

cam = Andor.Camera('/dev/fliusb0')
hwint = Andor.Interface(cam)
ct = Core.CtControl(hwint)

acq = ct.acquisition()

# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# set accumulation mode

acq_pars= acq.getPars()

#0-normal,1-concatenation,2-accumu
acq_pars.acqMode = 2
acq_pars.accMaxExpoTime = 0.05
acq_pars.acqExpoTime =1
acq_pars.acqNbFrames = 1

acq.setPars(acq_pars)
```

(continues on next page)

(continued from previous page)

```
# here we should have 21 accumalated images per frame
print acq.getAccNbFrames()

# now ask for 2 sec. exposure and 10 frames
acq.setAcqExpoTime(2)
acq.setNbImages(10)

ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)
```

7.3.12 imXPAD

Introduction

The imXpad detectors benefit of hybrid pixel technology, which leads to major advantages compared to the other technologies. These advantages are mainly provided by direct photon conversion and real time electronic analysis of X-ray photons. This allows for direct photon counting and energy selection.

XPAD detectors key features compared to CCDs and CMOS pixels detectors are:

- Noise suppression
- Energy selection
- Almost infinite dynamic range

- High Quantum Efficiency (DQE(0) ~100%, dose reduction)
- Ultra fast electronic shutter (10 ns)
- Frame rate > 500 Hz

Prerequisite

In order to operate the imXpad detector, the USB-server or the PCI-server must be running in the computer attached to the detector.

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_IMXPAD=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

imXpad camera must be initialisated using 2 parameters:

1) The IP adress where the USB or PCI server is running
2) The port number use by the server to communicate.

Std capabilities

- HwDetInfo

  getCurrImageType/getDefImageType():

- HwSync:

  get/setTrigMode(): the only supported mode are IntTrig, ExtGate, ExtTrigMult, ExtTrigSingle.

Refer to: http://imxpad.com/templates/SoftwareDocumentation/softwareDocumentation.html for a whole description of detector capabilities.

Optional capabilities

This plugin does not offer optional hardware capabilities.

How to use

This is a python code example for a simple test:

```
from Lima import imXpad
from Lima import Core
import time

# Setting XPAD camera (IP, port)
cam = imXpad.Camera('localhost', 3456)

HWI = imXpad.Interface(cam)
CT = Core.CtControl(HWI)
CTa = CT.acquisition()
CTs = CT.saving()

#To specify where images will be stored using EDF format
CTs.setDirectory("./Images")
CTs.setPrefix("id24_")
CTs.setFormat(CTs.RAW)
CTs.setSuffix(".bin")
CTs.setSavingMode(CTs.AutoFrame)
CTs.setOverwritePolicy(CTs.Overwrite)

#To set acquisition parameters
CTa.setAcqExpoTime(0.001) #1 ms exposure time.
CTa.setAcqNbFrames(10) # 10 images.
CTa.setLatencyTime(0.005) # 5 ms latency time between images.

#To change acquisition mode
cam.setAcquisitionMode(cam.XpadAcquisitionMode.Standard)

#To set Triggers. Possibilities: Core.IntTrig, Core.ExtGate, Core.ExtTrigMult, Core.
↪ExtTrigSingle.
CTa.setTriggerMode(Core.IntTrig)

#To set Outputs.
cam.setOutputSignalMode(cam.XpadOutputSignal.ExposureBusy)

#ASYNCHRONOS acquisition
CT.prepareAcq()
CT.startAcq()

#SYNCHRONOUS acquisition
CT.prepareAcq()
CT.startAcq()
cam.waitAcqEnd()

#To abort current process
#CT.stopAcq()

#Load Calibration from file
```

(continues on next page)

(continued from previous page)

```
#cam.loadCalibrationFromFile("./S70.cfg")

#Perform Calibrations 0-SLOW, 1-MEDIUM, 2-FAST
#cam.calibrationOTN(0)
#cam.calibrationOTNPulse(0)
#cam.calibrationBEAM(1000000,60,0) # 1s->exposure time, 60->ITHL_MAX, 0->SLOW
```

7.3.13 Merlin camera

Introduction

The Merlin Medipix3Rx Quad Readout detector system from Diamond Light Source Ltd is a photon counting soild state pixel detector with a silicon sensor.

The Lima module has only been tested in a 2 x 2 configuration, but is available in a 4 x 1 configuration

There is extensive documentation :ref: *Merlin_and_Medipix3_Documentation_v0.7.pdf*

Prerequisite

The Merlin detector system is based on a National Instruments FlexRIO PXI FPGA system. It incorporates an embedded PC running Windows with a LabView graphical user interface, incorporating a socket server, which this plugin communicates with. This program must be running prior to starting Lima.

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_MERLIN=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you good knowledge regarding camera features within the LIMA framework.

Camera initialisation

The camera has to be initialized using the `MerlinCamera` class. The constructor requires the hostname of the detector system.

Std capabilities

This plugin has been implemented with the mandatory capabilites, with some limitations due to the camera server implementation.

- HwDetInfo

  The detector is set to full image size at startup which means a binning of 1x1. There is no hardware binning

- HwSync

  The supported trigger modes are:

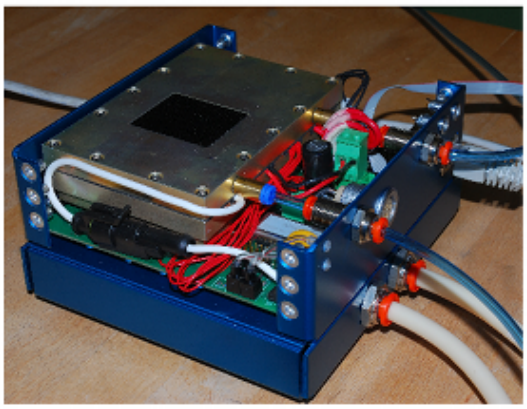
  - IntTrig
  - IntTrigMult
  - ExtTrigSingle
  - ExtTrigMult

Testing

This is a simple python test program:

```
from Lima import Merlin
from Lima import Core
import time

camera = Merlin.Camera('<hostname>')
interface = Merlin.Interface(camera)
control = Core.CtControl(interface)

acq = control.acquisition()

# check its OK
print camera.getDetectorType()
print camera.getDetectorModel()
print camera.getSoftwareVersion()

nframes=5
acqtime=3.0
# setting new file parameters and autosaving mode
saving=control.saving()

saving.setDirectory("/home/grm84/data")
saving.setFramesPerFile(nframes)
saving.setFormat(Core.CtSaving.HDF5)
saving.setPrefix("merlin_")
saving.setSuffix(".hdf")
saving.setSavingMode(Core.CtSaving.AutoFrame)
saving.setOverwritePolicy(Core.CtSaving.Append)

# do acquisition
acq=control.acquisition()
acq.setAcqExpoTime(acqtime)
acq.setAcqNbFrames(nframes)

control.prepareAcq()
control.startAcq()

# wait for last image (#4) ready
lastimg = control.getStatus().ImageCounters.LastImageReady
while lastimg !=nframes-1:
  time.sleep(0.01)
  lastimg = control.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = control.ReadImage(0)
```

7.3.14 PIXIRAD (PX1 and PX8) camera plugin



PIXIRAD-1

Top:
The first commercial PIXIRAD-1 module.
Side:
Inside of PIXIRAD-1 during its commissioning phase. The unit contains:
• the Gigabit Ethernet DAQ,
• the generation of High and Low Voltages supply,
• the distribution and the cooling control.
Externally only a 12 V power supply (laptop type) is needed.

Introduction

PIXIRAD Imaging Counters s.r.l. is an INFN Spin-off company introducing an innovative, high quality X-ray imaging sensor with intrinsic digital characteristics. It is based on Chromatic Photon Counting technology and represents a radical leap forward compared to the standard methods currently on the market.

The PIXIRAD imaging sensors are able to count individually the incident X-ray photons and to separate them in real time according to their energy (two color images per exposure).

- Global count rate > 200 GHz
- Energy range 1-100 keV
- Energy resolution better than 2 keV (FWHM) @20 keV

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_PIXIRAD=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

The camera has to be initialized using the `Pixirad::Camera` class. The default constructor does accept parameters:

Std capabilities

This plugin has been implement in respect of the mandatory capabilites but with some limitations which are due to the camera and SDK features. We only provide here extra information for a better understanding of the capabilities.

- HwDetInfo

  TODO

- HwSync
- The minimum latency time is 1 ms.
- The supported trigger modes are depending of the chosen frame mode:
  - IntTrig
  - ExtTrigMult

Optional capabilities

- HwReconstruction

  TODO

Specific control parameters

Some specific parameters are available within the camera hardware interface. Those parameters should be used carefully, please refer to the camera SDK (or user's guide) documentation for further information.

```
void autocalibration();

void setHighThreshold0(float t);
void getHighThreshold0(float& t) ;

void setLowThreshold0(float t);
void getLowThreshold0(float& t) ;

void setHighThreshold1(float t);
void getHighThreshold1(float& t) ;

void setLowThreshold1(float t);
void getLowThreshold1(float& t) ;

void setDeadTimeFreeMode(Camera::DeadTimeFreeMode dtf) ;
void getDeadTimeFreeMode(Camera::DeadTimeFreeMode &dtf) ;

void setNbiMode(Camera::SensorConfigNBI nbi) ;
void getNbiMode(Camera::SensorConfigNBI &nbi) ;

void setAsicMode(Camera::SensorConfigASIC asic);
void getAsicMode(Camera::SensorConfigASIC &asic);

void setHybridMode(Camera::SensorConfigHybrid hybrid);
void getHybridMode(Camera::SensorConfigHybrid &hybrid);

void setSensorConfigBuild(Camera::SensorConfigBuild build);
void getSensorConfigBuild(Camera::SensorConfigBuild &build);

void setRunConfigMode(Camera::RunConfigMode mode);
void getRunConfigMode(Camera::RunConfigMode &mode);

void setCoolingTemperatureSetpoint(float t);
void getCoolingTemperatureSetpoint(float& t) ;

void setCoolingMode(Camera::CoolingMode mode);
void getCoolingMode(Camera::CoolingMode &mode);

void setHighVoltageBiais(float hv);
void getHighVoltageBiais(float& hv) ;

void setHVBiasModePower(Camera::HVBiaisPower mode);
void getHVBiasModePower(Camera::HVBiaisPower &mode);

void setHVBiasMode(Camera::HVMode mode);
```

(continues on next page)

(continued from previous page)

```
void getHVBiasMode(Camera::HVMode &mode);

void setHighVoltageDelayBeforeOn(float sec);
void getHighVoltageDelayBeforeOn(float& sec);

void setHVRefreshPeriod(int nbOfImages);
void getHVRefreshPeriod(int& nbOfImages);

void setDelayBetweenFrames(int delayms);
void getDelayBetweenFrames(int& delayms);

void setColorMode(Camera::ColorMode color);
void getColorMode(Camera::ColorMode &color);

void setTrsfMode(Camera::TrsfMode mode);
void getTrsfMode(Camera::TrsfMode &mode);

// UDP
void setNCyclesUdpDelay(int nbcycles);
void getNCyclesUdpDelay(int& nbcycles);

void setSyncOutFunction(Camera::SyncOutFunction mode);
void getSyncOutFunction(Camera::SyncOutFunction &mode);

void setSyncOutPol(Camera::Polarity mode);
void getSyncOutPol(Camera::Polarity &mode);

void setSyncInPol(Camera::Polarity mode);
void getSyncInPol(Camera::Polarity &mode);

// Weather variable extracted from UDP stream, needs get/set
void getTemperaturePeltierCold(float& information);
void getTemperaturePeltierHot(float& information);
void getHighVoltageTension(float& information);
void getBoxHumidity(float& information);
void getBoxTemperature(float& information);
void getPeltierPower(float& information);

void getAlarmTempTooHot(bool& information);
void getAlarmTempTooHotEnabled(bool& information);
void getAlarmTempTooCold(bool& information);
void getAlarmTempTooColdEnabled(bool& information);
void getAlarmHumidity(bool& information);
void getAlarmHumidityEnabled(bool& information);
```

Basic network configuration

The camera has 192.168.0.1/24 adress. The detector pc has to be configured likewise. The recommended option is to have one good quality network interface dedicated to the pixirad, and one for the rest of the world.

- Case one (Recommended), dedicated interface:

```
auto eth1
iface eth1 inet static
address 192.168.0.100
```

(continues on next page)

(continued from previous page)

```
netmask 255.255.255.0
mtu 1500
```

- Case two, one interface, with a router handling two subnetworks:

  Configuration with an alias on interface eth0:

```
auto eth0:1
iface eth0:1 inet static
address 192.168.0.100
netmask 255.255.255.0
mtu 1500
```

Test examples

With python

- Test directly the camera within python:

```
from Lima import Core
from Lima import Pixirad as PixiradAcq
```

- Set the number of image treatment threads according to the number of CPU available on your mighty machine :

```
Core.Processlib.PoolThreadMgr.get().setNumberOfThread(20)
```

- Create your camera with its network settings and model (PX8 or PX1)

```
print "\n\n\n\n ======= INIT ======== \n"
camera = PixiradAcq.Camera("192.168.0.1", 2222, "PX8")
camera.init()
```

```
print "\n\n\n\n ======= INTERFACE ======== \n"
camera_interface = PixiradAcq.Interface(camera)
# Set some feature (check manual)
# color mode (only 1 col mode supported)
camera_interface.setColorMode(camera.COLMODE_1COL0)
# Set point (more than acheavable by the peliter to have full powa):
camera.setCoolingTemperatureSetpoint(-50)
# Set some energy thresholds (check manual, as they will fall in gain level␣
↪(ranges of energy).
camera.setLowThreshold0(10)
camera.setHighThreshold0(60)
camera.setLowThreshold1(10)
camera.setHighThreshold1(60)
# Some high tension management
camera.setHighVoltageBiais(2100)
camera.setHVBiasModePower(1)
camera.setHighVoltageDelayBeforeOn(3)
camera.setHVRefreshPeriod(1000);
# some ethernet interface
camera_interface.setTrsfMode(camera.UNMOD)
```

```
# Get control over things:
print "\n\n\n\n ======= CONTROL ======== \n"
control = Core.CtControl(camera_interface)
# set how much you want lima to buffer memory for treatment.
control.buffer().setMaxMemory(70)
```

```
# Get the object with whom you will play :
print "\n\n\n\n ======= ACQUISITION OBJECT ======== \n"
acq = control.acquisition()
# Define trigger:
acq.setTriggerMode(Core.IntTrig)
#acq.setTriggerMode(Core.ExtTrigMult)
```

```
# save somewhere
saving = control.saving()
pars=newsaving.getParameters()
pars.directory='/tmp/test'
pars.prefix=basename
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)
```

```
# Take images !
# expo time for one frame :
acq.setAcqExpoTime(0.01)
# number of frames:
acq.setAcqNbFrames(10)
# get it !
control.prepareAcq();
control.startAcq()
```

```
# pretty ones now !
# Take many (100) images and accumulate them to have better stats and one␣
↪image written:
acq.setAcqMode(Core.Accumulation)
# Max expo time per frame:
acq.setAccMaxExpoTime(0.01)
# Total time for the accumulation:
acq.setAcqExpoTime(1);
# how many accumulated images:
acq.setAcqNbFrames(1)
# get them all and keep one:
control.prepareAcq();
control.startAcq()
```

With Tango

- Properties

```
initial_model = PX8   // or PX1
ip_address    = 192.168.0.1
port_number   = 2222
```

- PyTango client connection examples:

```
import PyTango
pixi = PyTango.DeviceProxy("d05/pixirad/pixirad")
limaccd = PyTango.DeviceProxy("d05/pixirad/pixirad8")
pixi.cooling_temperature_setpoint = -50
pixi.high_voltage_biais = 2100
pixi.dead_time_free_mode = 'DEAD_TIME_FREE_MODE_OFF'
pixi.color_mode = 'COLMODE_1COL0'
pixi.low_threshold0 =  1
pixi.high_threshold0 = 99
pixi.low_threshold1 =  1
pixi.high_threshold1 = 99
#pixi.sensor_config_build = 'PX8'
pixi.h_v_bias_mode_power = 1
pixi.trsf_mode = "UNMOD"
limaccd.buffer_max_memory = 80
limaccd.acq_nb_frames = 0
limaccd.acq_expo_time = 0.01
limaccd.prepareAcq()
limaccd.startAcq()
```

Advanced configuration and optimization (optional)

The camera will send the images as small (1490) udp datagrams, as fast as it can, nearly saturating the bandwidth of the 1Gb ethernet link. Bad network cards, or high latency systems will result in a loss of part of the image. If this happens, several points needs checking. The ethernet card driver might drop packets (and as they are UDP, there won't be any chace to see them). The linux kernel UDP buffer might saturate and willingly drop packets (but you knows it at least). In this case, it means that your reading loop (reading from the linux udp buffer) is too slow.

Here are a couple of options:

- Using FIFO realtime mode can help.
- Tuning network buffers can help.
- Changing ethernet card can save your skin, and avoid you loosing weeks fine tuning muddy cards.

Realtime mode

In : /etc/security/limits.conf add :

```
username    -       rtprio  5
```

In soft :

```
pthread_t this_thread = pthread_self();
struct sched_param params;
params.sched_priority = 5;
ret = pthread_setschedparam(this_thread, SCHED_FIFO, &params);
if (ret != 0) { std::cout << "Check /etc/security/limits.conf " << std::endl; }
```

Kernel tuning

```
man udp
```

Change in `/etc/sysctl.conf` and validate with `sysctl -p`

```
net.core.rmem_max = 256217728
net.core.wmem_max = 256217728
net.ipv4.udp_mem = 131072 262144 524288
net.ipv4.udp_rmem_min = 65536
net.core.netdev_max_backlog = 65536
net.core.somaxconn = 1024
```

Network card driver tuning

```
ethtool -g eth1
Ring parameters for eth1:
Pre-set maximums:
RX:           4096
RX Mini:      0
RX Jumbo:     0
TX:           4096
Current hardware settings:
RX:           512         <<<<<< =====
RX Mini:      0
RX Jumbo:     0
TX:           512
```

Increased with :

```
ethtool -G eth1 rx 4096
```

Troubleshootings

UDP debug tips

If you suspect drop of UDP datagram due to a too small kernel buffer (the plugin is too slow to treat the buffer, it filled and drop frames)

```
cat /proc/net/udp

And check the drop column.
```

```
cat /proc/sys/net/core/rmem_max

tells you the buffer size
by default : 131071

Enough for 100 images:
```

```
net.core.rmem_max = 507217408
```

Possible problems with network adapters

List of known to work adapters

Embedded motherboard card on optiplex 980:

- Intel Corporation 82578DM Gigabit Network Connection (rev 05)

List of non working adapters

Intel pro 1000 on PCI card (82541GI) (debian 7 & 9):

- Intel Corporation 82541GI Gigabit Ethernet Controller
- Intel Corporation 82541PI Gigabit Ethernet Controller (rev 05)

Possible problems with Chillers

Symptoms : strippy images

The goal is to setup your temperature settings as to have the peltier full time @ max power. If the peltier is regulating the temperature, stripes appears in the images. A easy way is to setup a -50C unreachable goal for the detector and let it stabilise to wathever temperature it can reach based on chiller setting. Chiller is supposed to be set at 16degC. Going bellow needs a hutch humidity well controlled.

7.3.15 PointGrey

Introduction

> "Point Grey is a world-leading designer and manufacturer of innovative, high-performance digital cameras for industrial, life science, and traffic applications. We offer a unique and comprehensive portfolio of USB 3.0, GigE, FireWire, USB 2.0 and Camera Link products known for their outstanding quality, ease of use, and unbeatable price-performance."

The Lima module has been tested only with this GigE cameras models:

- Blackfly 1024x768 (model BFLY-PGE-05S2M)

Prerequisite

First, you have to install the PointGrey *FlyCapture* SDK. We only tested it on debian6 and using the SDK version 2.3.19 (the latest one compatible with debian6 libc).

PointGrey python module need at least the lima core module.

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_POINTGREY=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you good knowledge regarding camera features within the LIMA framework.

Camera initialisation

The camera has to be initialized using the `PointGreyCamera` class. The default constructor needs at least the serial number of your camera in order to get the network connection setting up. In Addition one can provide both `packate_size` and `packet_delay` parameters. By default no value is passed.

Std capabilities

This plugin has been implemented in respect of the mandatory capabilites but with some limitations which are due to the camera and SDK features. We only provide here extra information for a better understanding of the capabilities for Andor cameras.

- HwDetInfo

  getPixelSize(): the method just returns -1, it has to be implemented in further version. get/setImageType(): the plugin only supports Bpp8 and Bpp16

- HwSync

  get/setTriggerMode(): Depending of the camera model, but some can not support any trigger mode. Otherwise the only implemented modes are IntTrig and ExtTrigSingle. IntTrigMult is normally a mandatory mode (for any camera) and will be implemented in next version.

Optional capabilities

None has been implemented for this camera plugin.

Specific control parameters

Some specific paramaters are available within the camera hardware interface. Those parameters should be used carefully and one should refer to the camera SDK (or user's guide) documentation for a better understanding.

- get/setPacketSize()
- get/setPacketDelay()
- get/setGain()
- get/setAutoGain()
- getGainRange()

The following parameters can break the synchronisation with the LIMA HwSync layer by changing the camera internal exposure time.

- get/setAutoExpTime()
- get/setFrameRate()
- get/setAutoFrameRate()

Network Configuration

- Depending on your network infrastructure you will need to configure a fix IP address for the camera or use a DHCP setup instead.

  The linux SDK provides a configuation tool called `GiGEConfigCmd`. The Windows SDK version provides a graphical tool, `GigEConfigurator.exe`.

- Then in the PointGrey Tango device set the property `camera_serial` using the camera serial number (sticked on the camera).
- If you are running the server with linux kernel >= 2.6.13, you should add this line into `etc/security/limits.conf`. With the following line, the acquisition thread will be in real time mode:

```
USER_RUNNING_DEVICE_SERVER  -       rtprio  99
```

How to use

This is a python code example for a simple test:

```
from Lima import PointGrey
from lima import Core

cam = PointGrey.Camera(13125072)
hwint = PointGrey.Interface(cam)
control = Core.control(hwint)

acq = control.acquisition()
```

(continues on next page)

(continued from previous page)

```
# configure some hw parameters
hwint.setAutoGain(True)

# setting new file parameters and autosaving mode
saving=control.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# now ask for 10ms sec. exposure and 100 frames
acq.setAcqExpoTime(0.01)
acq.setNbImages(100)

control.prepareAcq()
control.startAcq()

# wait for last image (#99) ready
lastimg = control.getStatus().ImageCounters.LastImageReady
while lastimg !=99:
  time.sleep(.01)
  lastimg = control.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = control.ReadImage(0)
```

7.3.16 Prosilica

Introduction

AVT offers a large choice of FireWire and GigE cameras for machine vision, computer vision and other industrial or medical applications. Cameras by AVT and Prosilica include sensitive machine vision sensors (CCD and CMOS, VGA to 16 Megapixels) and fit a wide range of applications.

The Lima module as been tested with color and B/W GigE camera.

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_PROSILICA=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you good knowledge regarding camera features within the LIMA framework.

Camera initialisation

The camera will be initialized by creating a :cpp:`Prosilica::Camera` object. The contructor sets the camera with default parameters, only the ip address or hostname of the camera is mandatory.

Std capabilities

This plugin has been implemented in respect of the mandatory capabilites but with some limitations which are due to the camera and SDK features. Only restriction on capabilites are documented here.

- HwDetInfo

  getCurrImageType/getDefImageType(): it can change if the video mode change (see HwVideo capability).

  setCurrImageType(): It only supports Bpp8 and Bpp16.

- HwSync

  get/setTrigMode(): the only supported mode are IntTrig, IntTrigMult and ExtTrigMult.

Optional capabilities

In addition to the standard capabilities, we make the choice to implement some optional capabilities which are supported by the SDK. Video and Binning are available.

- HwVideo

  The prosilica cameras are pure video devices, so only video format for image are supported:

  Color cameras ONLY

  - BAYER_RG8

  - BAYER_RG16
  - RGB24
  - BGR24

Color and Monochrome cameras

  - Y8

Use get/setMode() methods of the cpp::class::*Video* object (i.e. CtControl::video()) to read or set the format.

- HwBin

  There is no restriction for the binning up to the maximum size.

Configuration

- First you have to setup ip address of the Prosilica Camera with `CLIpConfig` located in `camera/prosilica/sdk/CLIpConfig`
- list of all cameras available : `CLIpConfig -l` (If you do not see any camera, that's bad news!)
- finally set ip add : `CLIpConfig -u UNIQUE_NUMBER -s -i 169.254.X.X -n 255.255.255.0 -m FIXED` (It's an example!)
- Then in the Prosilica Tango device set the property `cam_ip_address` to the address previously set.

That's all....

How to use

This is a python code example for a simple test:

```
from Lima import Prosilica
from lima import Core

cam = Prosilica.Camera("192.169.1.1")

hwint = Prosilica.Interface(cam)
ct = Core.CtControl(hwint)

acq = ct.acquisition()

# set video  and test video

video=ct.video()
video.setMode(Core.RGB24)
video.startLive()
video.stopLive()
video_img = video.getLastImage()

# set and test acquisition

# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
```

(continues on next page)

(continued from previous page)

```
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.TIFF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

acq.setAcqExpoTime(0.1)
acq.setNbImages(10)
ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(0.01)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)
```

7.3.17 MarCCD

Introduction

The SX165 features a round, 165 mm diameter active area, and a versatile, high resolution CCD chip. It is the ideal X-ray detector for research applications with both synchrotrons and rotating anode X-ray sources.

Prerequisite

The MarCCD software server should be started on the MarCCD host computer, by running the command:

```
$ marccd -r
```

Then you can launch your lima/marccd client on another host, as the MarCCD server can be reached by network

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_MARCCD=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you good knowledge regarding camera features within the LIMA framework.

Camera initialisation

There are 4 parameters to be filled by your Lima client:

- The IPAddress of the host where the marccd server is running
- The port of the marccd server process
- The detector target path: the path where will be saved the marccd image files
- Reader timeout: in ms, the timeout after which the plugin will be in fault if no marccd image file is present

Std capabilities

This plugin has been implemented in respect of the mandatory capabilites but with some limitations according to some programmer's choices. We only provide here extra information for a better understanding of the capabilities for the MarCCD camera.

- HwDetInfo
  - Max image size is : 4096 * 4096
  - 16 bit unsigned type is supported
- HwSync
  - trigger type supported are:
    - IntTrig

Optional capabilities

- HwBin
  - 2 * 2
  - 4 * 4
  - 8 * 8
- HwRoi

  TODO

Configuration

No Specific hardware configuration is needed.

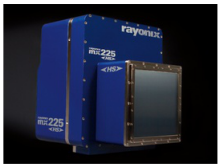
How to use

Here is the list of accessible fonctions to configure and use the MarCCD detector:

```
void getDetectorImageSize(Size& size);
void setImagePath(const std::string& path);
const std::string& getImagePath(void);
void setImageFileName(const std::string& imgName);
const std::string& getImageFileName();
void setImageIndex(int newImgIdx);
int getImageIndex();
int getFirstImage();

bool isStopSequenceFinished();
void saveBGFrame(bool);

void setBeamX(float);
float getBeamX();
void setBeamY(float);
float getBeamY();
void setDistance(float);
float getDistance();

void setWavelength(float);
float getWavelength();
```

7.3.18 Rayonix HS camera

rayonix High-performance X-ray technology



Introduction

The MX-HS series from Rayonix incorporates the new, exclusive HS frame-transfer technology for high speed X-ray data collection without compromising resolution or data quality. The result is a new type of high speed and ultra-low noise area detector that delivers the highest performance available for X-ray diffraction applications.

The Rayonix MX-HS detectors are ideal for taking advantage of high brilliance synchrotron sources, or for any other high frame rate application. Examples include: high throughput protein crystallography, Laue diffraction, time-resolved or static small-angle X-ray scattering (SAXS), wide-angle X-ray scattering (WAXS), powder diffraction, X-ray computed tomography (CT), X-ray imaging, and coherent diffraction imaging (CDI). With no count rate limitation, these detectors are also ideal for XFEL applications.

The Lima module as been tested only with the following models :

- MX170-HS (2x2 mdules)

Prerequisite

The Rayonix HS detector is been delivered today with its own control computer, a powerful computer embedded at leat 8GB of RAM, dual 4-Core CPU (8 cores) and a GPU card for the online image correction (background, flatfield ...). The computer is running redhat entprise Linux 6 (64bits).

The rayonix SDK is preinstalled on the detector node under the directory `/opt/rayonix`.

There is no special prerequisite, you can test that the device works properly by running the rayonix GUI, `caxpure`.

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_RAYONIXHS=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

The camera has to be initialized using the RayonixHsCamera class. The default constructor does not need any input parameter.

Std capabilities

This plugin has been implemented in respect of the mandatory capabilites but with some limitations which are due to the camera and SDK features. We only provide here extra information for a better understanding of the capabilities.

- HwDetInfo

  The detector is set to full image size at startup which means a binning of 1x1.

  Note: The recommended binning for most of the experiment is 2x2.

- HwSync
- The minimum latency time is 1 ms.
- The supported trigger modes are depending of the chosen frame mode. There are:
  - IntTrig
  - IntTrigMult
  - ExtTrigSingle
  - ExtTrigMult (only for SINGLE frame mode)
  - ExtGate (only for SINGLE frame mode)
  - ExtTrigReadout (only for FAST_TRANSFER frame mode).

Optional capabilities

- HwBin

  The supported hardware binning are 2x2, 3x3, 4x4, 5x5, 6x6, 7x7, 8x8, 9x9 and 10x10. By increasing the binning factor you can increase the readout speed from 2.6 fps to 140 fps which corresponds respectively to a pixel size of 44um and 440 um.

- HwShutter

  The Rayonix HS detectors provides 2 output channels one can choose a different source for each (see specific control parameters for more details about the output source control). For the SHUTTER source both opening and closing delay can be set.

  The Rayonix HS shutter capability only supports two modes:

  - ShutterAutoFrame
  - ShutterManual

Specific control parameters

Some specific paramaters are available within the camera hardware interface. Those parameters should be used carefully and one should refer to the camera SDK (or user's guide) documentation for a better understanding.

- get/setFrameTriggerType(type): signal type for the frame trigger input (channel #1)
- get/setSequenceGateSignalType(type): signal type for the gate input (channel #2), The supported signal types:
- OPTO
- OPTO_INVERTED

- CMOS
- CMOS_PULLDOWN
- CMOS_PULLUP
- CMOS_PULLDOWN_INVERTED
- CMOS_PULLUP_INVERTED
- SOFTWARE
- get/setOutputSignalType(channel, type): the signal type for the output channel (CHANNEL_1 or CHANNEL_2)
- get/setOutputSignalID(channel, id): the source id for the output channel, possible sources are:
  - ID_SHUTTER
  - ID_INTEGRATE
  - ID_FRAME
  - ID_LINE
  - ID_SHUTTER_OPENING
  - ID_SHUTTER_CLOSING
  - ID_SHUTTER_ACTIVE
  - ID_TRIGGER_RISE_WAIT
  - ID_TRIGGER_RISE_ACK
  - ID_TRIGGER_FALL_WAIT
  - ID_TRIGGER_FALL_ACK
  - ID_TRIGGER_2_RISE_WAIT
  - ID_TRIGGER_2_RISE_ACK
  - ID_INPUT_FRAME
  - ID_INPUT_GATE
- get/setElectronicShutterEnabled(): active or unactive the electronic shutter
- get/setCoolerTemperatureSetpoint(): the cooler temperature set-point
- get/setSensorTemperatureSetpoint(): the sensor temperature set-point
- get/setSensorTemperature(): the detector measured temperature
- get/setCooler(): stop or start the cooler controller
- get/setVacuumValve(): close or open the vacuum valve
- get/setFrameMode(): modes are SINGLE or FAST_TRANSFER.

> **Warning:** in FAST_TRANSFER mode the latency time is disabled and it has a fixed value of 1 ms which corresponds to the readout time. In addition to this the supported trigger mode will depend on the frame mode. The list of supported trigger modes is available in this document below.

Configuration

Cabling

The detector head should be connected to the detector computer on the cameralink and USB links. You must connect the USB on the PCI board (not the motherboard ones) and the cameralink on the first channel, the top connector.

Cooling

For an optimized condition wit dark current the detector has to be cooled down, the sensor temperature set-point should be at -120 deg and the cooler temperature set-point at -90 deg Celsuis. And of course the cooler controller should be started.

How to use

This is a simple python test program:

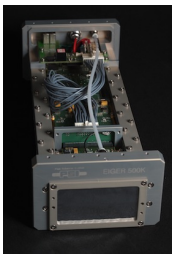
```
from Lima import RayonixHs
from lima import Core

cam = RayonixHs.Camera()
hwint = RayonixHs.Interface(cam)
control = Core.CtControl(hwint)

acq = control.acquisition()

# configure some hw parameters
sens_temp = hwint.getSensorTemperature()
cool_temp = hwint.getCoolerTemperatureSetpoint()
if sens_temp > -50:
  print " Hoops, detector is not cooled down, temp = ", sens_temp

# setting new file parameters and autosaving mode
saving=control.saving()

pars=saving.getParameters()
pars.directory='/somewhere/'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# set a new binning to increase the frame rate
image = control.image()
image.setBin(Core.Bin(2,2))

# now ask for 10ms sec. exposure and 100 frames
acq.setAcqExpoTime(0.01)
acq.setNbImages(100)

control.prepareAcq()
control.startAcq()
```

(continues on next page)

(continued from previous page)

```
# wait for last image (#xi99) ready
lastimg = control.getStatus().ImageCounters.LastImageReady
while lastimg !=99:
  time.sleep(1)
  lastimg = control.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = control.ReadImage(0)
```

7.3.19 SlsDetector camera



Introduction

The PSI/SLS Detector Group has developed a family of X-ray detectors: Mythen, Pilatus, Gotthard, Eiger, Moench, Jungrau, among others. Most of them are controlled through Ethernet interfaces, with optional dedicated data link(s). A common protocol has been developed to control these detectors, based on the *slsDetector* class. A separate software entity receives and dispatch the data: *slsReceiver*. The SlsDetector LIMA plugin instantiates the necessary software objects to perform data aquisitions with the detectors supported by the slsDetectorsPackage.

The current implementation only works with the PSI/Eiger detectors.

Prerequisite

The *slsDetectorPackage-v2.3.x* is needed by the SlsDetector LIMA plugin. As explained in installation, the *slsDetectorPackage* is included as a submodule in the SlsDetector camera plugin. It will be automatically compiled and installed during the LIMA build procedure.

In addition to that, a *configuration file*, containing the commands necessary to initialise both the *slsDetector" and *slsReceiver* instances, is required.

The library protocol uses Unix System-V IPC shared memory blocks to exchange information between processes. The segments, referred to by keys matching hex *000016xx*, must be owned by the user running the plugin, if it is not *root*. The following command, which removes the existing segments, must be run by the segments' owner (or *root*) so they can be deleted/created by another user:

```
ipcs -m | \
  grep -E '^0x000016[0-9a-z]{2}' | \
  awk '{print $2}' | while read m; do \
    ipcrm -m $m; \
done
```

High-performance Acquisitions

High-performance acquisitions require a specific backend computer setup. Please refer to the installation.

Installation & Module configuration

- Follow the steps indicated in installation

As a reference, see:

- linux_installation
- linux_compilation
- *PyTango Device Server*

Initialisation and Capabilities

In order to help people to understand how the camera plugin has been implemented in LImA this section provides some important information about the developer's choices.

Camera initialisation

The SlsDetector plugin exports two kind classes: one generic *SlsDetector::Camera* class, with the common interface to *slsDetector* and *slsReceiver* classes, and detector-specific classes, like *SlsDetector::Eiger* which manage the particularities of each model.

First, the *SlsDetector::Camera* must be instantiated with the configuration file, and once the connection to the detector is established, a specific class is created depending on the detected type:

```
cam = SlsDetector.Camera(config_fname)
if cam.getType() == SlsDetector.Camera.EigerDet:
    eiger = SlsDetector.Eiger(cam)
else:
    raise RuntimeError("Non-supported type: %s" % cam.getType())

hw_inter = SlsDetector.Interface(cam)
ct = Core.CtControl(hw_inter)
```

The raw images returned by the *slsReceiver* class might need to be reconstructed, like in the case of the PSI/Eiger detector. A LImA software reconstruction task must be then created from the LImA plugin and registered to the *Core::CtControl* layer:

if cam.getType() == SlsDetector.Camera.EigerDet: corr = eiger.createCorrectionTask()
ct.setReconstructionTask(corr)

Std capabilites

This plugin has been implemented in respect of the mandatory capabilites but with limitations according due to the detector specific features and with some programmer's choices. We do not explain here the standard Lima capabilites but you can find in this section the useful information on the SlsDetector specfic features.

- HwDetInfo

TODO

- HwSync

The following trigger modes are currently implemented:

- IntTrig
- ExtTrigSingle
- ExtTrigMult
- ExtGate

The minimum *latency_time* and the *max_frame_rate* are automatically updated depending on the *PixelDepth* (4, 8, 16, 32), the *ClockDiv* (Full-, Half-, Quarter-, SuperSlow-Speed), and the *ReadoutFlags* (Parallel, Non-Parallel).

Optional capabilites

In addition to the standard capabilities, we make the choice to implement some optional capabilities in order to have an improved simulation.

- HwShutter

Not implemented

- HwRoi

Not implemented

- HwBin

Not implemented

Configuration

The main configuration will consist in providing the correct *config file* file to the *slsDetector API*. As mentioned before, the file is a list of commands accepted by *sls_detector_put*, and it should also work with the *slsDetectorGui* application.

Two important parameters define the image frame dimension:

- PixelDepth:
  - 4 bit (not implemented yet)
  - 8 bit
  - 16 bit
  - 32 bit
- RawMode:

If set to *True*, the image is exported to LiMA as given from the Receiver(s), without any software reconstruction.

How to use

The LimaCCDs Tango server provides a complete interface to the SlsDetector plugin so feel free to test.

For a quick test one can use Python, this a short code example to work with the PSI/Eiger detector:

```
from Lima import SlsDetector
from Lima import Core
import time
import sys

config_fname = sys.argv[1]

cam = SlsDetector.Camera(config_fname)
if cam.getType() != SlsDetector.Camera.EigerDet:
  raise RuntimeError("Non-supported type: %s" % cam.getType())

eiger = SlsDetector.Eiger(cam)
hw_inter = SlsDetector.Interface(cam)
ct = Core.CtControl(hw_inter)
corr = eiger.createCorrectionTask()
ct.setReconstructionTask(corr)

acq = ct.acquisition()

# setting new file parameters and autosaving mode
saving = ct.saving()

pars = saving.getParameters()
pars.directory = '/tmp'
pars.prefix = 'test_slsdetector_'
pars.suffix = '.edf'
pars.fileFormat = Core.CtSaving.EDF
pars.savingMode = Core.CtSaving.AutoFrame
saving.setParameters(pars)

# now ask for 0.2 sec. exposure and 10 frames
acq.setAcqExpoTime(0.2)
acq.setAcqNbFrames(10)

ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg != 9:
  time.sleep(0.1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)

# cleanup in good order
import gc
del acq; gc.collect()
del ct; gc.collect()
del corr; gc.collect()
del eiger; gc.collect()
```

(continues on next page)

(continued from previous page)

```
del hw_inter; gc.collect()
del cam; gc.collect()
```

A more complete **test_slsdetector_control.py** Python script can be found under the *camera/slsdetector/test* directory.

7.3.20 Ueye

Introduction

Industrial Cameras for digital imaging and visualization (USB,GigE).

home site: http://www.ids-imaging.com/

Installation & Module configuration

First, you have to install the Ueye SDK. See the sdk README provide in the ueye module

Then, follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_UEYE=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

The camera will be initialized by creating a `Ueye::Camera` object. The contructor sets the camera with default parameters, only the video address (e.g. 0) of the camera is mandatory.

Std capabilites

This plugin has been implement in respect of the mandatory capabilites but with some limitations which are due to the camera and SDK features. Only restriction on capabilites are documented here.

- HwDetInfo

  getCurrImageType/getDefImageType(): it can change if the video mode change (see HwVideo capability).

  setCurrImageType(): It only supports Bpp8 and Bpp16.

- HwSync

  get/setTrigMode(): the only supported mode are IntTrig, IntTrigMult ExtTrigSingle and ExtTrigMult.

Optional capabilites

In addition to the standard capabilities, we make the choice to implement some optional capabilities which are supported by the SDK. **Video** and Binning are available.

- HwVideo

  The prosilica cameras are pure video device, so video format for image are supported:

  For color cameras ONLY

  - BAYER_RG8
  - BAYER_RG16
  - BAYER_BG8
  - BAYER_BG16
  - RGB24
  - YUV422

  Color and Monochrome cameras

  - Y8
  - Y16

  Use get/setMode() methods of the *video* object (i.e. CtControl::video()) to read or set the format.

- HwBin

  There is no restriction for the binning up to the maximum size.

Configuration

See the SDK `README` in `camera/ueye/sdk/` directory.

How to use

A python code example for testing your camera:

```
from Lima import Ueye
from lima import Core

#-----------------+
#                 |
#                 v the video address
cam = Ueye.Camera(0)

hwint = Ueye.Interface(cam)
ct = Core.CtControl(hwint)

acq = ct.acquisition()


# set video  and test video, supposing we have a color camera !!
#

video=ct.video()
video.setMode(Core.YUV422)
video.setExposure(0.1)
video.startLive()
video.stopLive()
video_img = video.getLastImage()


# set and test acquisition
#

# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.TIFF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

acq.setAcqExpoTime(0.1)
acq.setNbImages(10)
ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(0.1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady
```

(continues on next page)

(continued from previous page)

```
# read the first image
im0 = ct.ReadImage(0)
```

7.3.21 Ultra

Introduction

"The ULTRA Detector System enables capture of one dimensional spectra at extremely high rates. Where CCDs were used to capture a line of data at a time, the ULTRA Detector System offers many orders of magnitude faster time framing. ULTRA is a compact turnkey system. The data acquisition system is attached in a compact form factor unit with gigabit Ethernet out and multiple I/O options onboard."

Table 1: Ultra Specification

Sustained Spectral Rate	20 KHz (spectra per second) Maximum
Frame Period	<500 ns Minimum
Spectral Sensitivity	5 – 17KeV 300µm thickness. 500µm also available.
Output	Gigabit Ethernet
Pixel configuration	Si 512 linear strips @ 50µm pitch
ADC Dynamic Range	16 Bit
Synchronisation Inputs	TTL or Fibre Optic
Integration Time	<1us - 650us frames
TriggeringExternal	(TTL or Fibre) or Internal (10KHz fixed)

Prerequisite

The default network setup is (excluding the site network connection):

1GBit Copper network for control communinication between the PC and the Ultra box.

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_ULTRA=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

The camera will be initialized within the :cpp::class::*Ultra::Camera* object. A TCP and UDP socket connections on the 1GBit port are established

The Ultra requires the following parameters with the recommended settings:

```
headname          = 192.168.1.100
hostname          = 192.168.1.103
tcpPort           = 7
udpPort           = 5005
npixels           = 512
```

Std capabilites

This plugin has been implemented with respect of the mandatory capabilites but with some limitations which are due to the camera. We only provide here extra information for a better understanding of the capabilities for Ultra cameras.

- HwDetInfo

  getCurrImageType/getDefImageType(): is set to Bpp16

- HwSync

  get/setTrigMode(): the only supported modes are IntTrig, ExtTrigMult and IntTrigMult

Optional capabilities

TODO

7.3.22 V4l2 camera

Introduction

V4L2 stands for Video for Linux 2. This new plugin aims to interface any v4l2 camera devices to LIMA framework. Some USB Webcams have been tested successfully. Video for Linux 2 supports most of the market products, however you may encountered some limitations using Lima, please report your problem and or your patch to lima@esrf.fr, we will be happy to improve this code for you.

Useful links:

- http://linuxtv.org
- http://en.wikipedia.org/wiki/Video4Linux

Installation & Module configuration

Depending or your linux flavor you may need to intall/update the v4l2 packages.

The package libv4l-dev is mandatory to compile the lima v4l2 plugin.

We recommend to install a useful tool `qv4l2`, a Qt GUI. You can test your device and check supported video formats and if the camera is supporting fixed exposure for instance.

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_V4L2=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

The camera will be initialized by creating a `V4l2::Camera` object. The contructor sets the camera with default parameters, and a device path is required, e.g. `/dev/video0`.

Std capabilities

This plugin has been implemented in respect of the mandatory capabilites but with some limitations.

It is mainly a video controller, see `HwVideoCtrlObj`, with a minimum set of feature for standard acquisition. For instance the exposure control can not be available if the camera only support the auto-exposure mode.

- HwDetInfo

  getCurrImageType/getDefImageType(): it can change if the video mode change (see HwVideo capability).

  setCurrImageType(): It only supports Bpp8 and Bpp16.

- HwSync

  get/setTrigMode(): Only IntTrig mode is supported.

Optional capabilites

The V4L2 camera plugin is a mostly a **Video** device which provides a limited interface for the acquisition (i.e, exposure, latency ..).

- HwVideo

  The v4l2 cameras are pure video device we are supporting the commonly used formats:

  Bayer formats

  - BAYER_BG8
  - BAYER_BG16

  Luminence+chrominance formats

  - YUV422
  - UYV411
  - YUV444
  - I420

  RGB formats

  - RGB555
  - RGB565
  - BGR24
  - RGB24
  - BGR32
  - RGB32

  Monochrome formats

  - Y8
  - Y16
  - Y32
  - Y64

Use get/setMode() methods of the *video* object (i.e CtControl::video()) for accessing the video format. The lima plugin will initialise the camera to a *preferred* video format by choosing one of the format the camera supports but through ordered list above.

Configuration

Simply plug your camera (USB device or other interface) on your computer, it should be automatically detected and a new device file is created like `/dev/video0`. The new device is maybe owned by `root:video`, so an other user cannot access the device. In that case you should update `/etc/group` to add that user to the video group.

How to use

This is a python code example for a simple test:

```
from Lima import v4l2
from lima import Core


#------------------+
# V4l2 device path |
#                  v
cam = v4l2.Camera('/dev/video0')


hwint = v4l2.Interface(cam)
ct = Core.CtControl(hwint)

acq = ct.acquisition()


# set and test video
#

video=ct.video()
# to know which preferred format lima has selected
print (video.getMode())
video.startLive()
video.stopLive()
video_img = video.getLastImage()

# set and test an acquisition
#

# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.TIFF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# now ask for and 10 frames
```

(continues on next page)

(continued from previous page)

```
acq.setNbImages(10)

ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)
```

7.3.23 Xpad

Introduction

The XPAD detector is based on the photon counting technology providing a quasi noiseless imaging as well as a very high dynamic range and a fast frame rate (500 images/s). This is a detector stemming from the collaboration of Soleil, CPPM and ESRF(D2AM). It is now supported by the ImXPAD company.

This plugin support the following models:

- S70,
- S140,
- S340,
- S540

The XPAD runs under Linux, with the help of a PCI express board from PLDA.

Prerequisite

The host where the PCI express board is installed, should have the PLDA driver installed.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

The camera will be initialized within the `Xpad::Camera` object. One should pass to the constructor, the Xpad type as a string. Possible values are:

- "IMXPAD_S70",
- "IMXPAD_S140",
- "IMXPAD_S340",
- "IMXPAD_S540"

Synchrone or Asynchrone acquisition should be selected with a call `setAcquisitionType()`.

Std capabilities

This plugin has been implemented in respect of the mandatory capabilites but with some limitations according to some programmer's choices. We only provide here extra information for a better understanding of the capabilities for the xpad camera.

HwDetInfo

- 16 or 32 bit unsigned type are supported
- the size of the image will depend of the type of Xpad

HwSync

Trigger type supported are:

- IntTrig
- ExtTrigSingle
- ExtGate : 1 external trigger start N internal gates (gates being configured by software)
- ExtTrigMult : N external trigger start N internal gates (gates being configured by software)

Optional capabilities

There are no optional capabilities.

Configuration

No Specific hardware configuration is needed.

How to use

Here is a list of accessible fonctions to configure and use the Xpad detector:

```
//! Set all the config G
void setAllConfigG(const std::vector<long>& allConfigG);
//!     Set the Acquisition type between synchrone and asynchrone
void setAcquisitionType(short acq_type);
//!     Load of flat config of value: flat_value (on each pixel)
void loadFlatConfig(unsigned flat_value);
//! Load all the config G
void loadAllConfigG(unsigned long modNum, unsigned long chipId , unsigned long*␣
↪config_values);
//! Load a wanted config G with a wanted value
void loadConfigG(const std::vector<unsigned long>& reg_and_value);
//! Load a known value to the pixel counters
void loadAutoTest(unsigned known_value);
//! Save the config L (DACL) to XPAD RAM
void saveConfigL(unsigned long modMask, unsigned long calibId, unsigned long chipId,␣
↪unsigned long curRow,unsigned long* values);
//! Save the config G to XPAD RAM
void saveConfigG(unsigned long modMask, unsigned long calibId, unsigned long reg,
↪unsigned long* values);
//! Load the config to detector chips
void loadConfig(unsigned long modMask, unsigned long calibId);
//! Get the modules config (Local aka DACL)
unsigned short*& getModConfig();
//! Reset the detector
void reset();
//! Set the exposure parameters
void setExposureParameters( unsigned Texp,unsigned Twait,unsigned Tinit,
                                                    unsigned Tshutter,unsigned␣
↪Tovf,unsigned mode, unsigned n,unsigned p,
                                                    unsigned nbImages,unsigned␣
↪BusyOutSel,unsigned formatIMG,unsigned postProc,
                                                    unsigned GP1,unsigned GP2,
↪unsigned GP3,unsigned GP4);
//! Calibrate over the noise Slow and save dacl and configg files in path
void calibrateOTNSlow (const std::string& path);
//! Calibrate over the noise Medium and save dacl and configg files in path
void calibrateOTNMedium (const std::string& path);
//! Calibrate over the noise High and save dacl and configg files in path
void calibrateOTNHigh (const std::string& path);
//! upload the calibration (dacl + config) that is stored in path
void uploadCalibration(const std::string& path);
//! upload the wait times between each images in case of a sequence of images (Twait␣
↪from setExposureParameters should be 0)
```

(continues on next page)

(continued from previous page)

```
void uploadExpWaitTimes(unsigned long *pWaitTime, unsigned size);
//! increment the ITHL
void incrementITHL();
//! decrement the ITHL
void decrementITHL();
//! set the specific parameters (deadTime,init time, shutter ...
void setSpecificParameters( unsigned deadtime, unsigned init,
                                                    unsigned shutter, unsigned␣
↪ovf,
                                                    unsigned n,       unsigned p,
                                                    unsigned busy_out_sel,
                                                    bool geom_corr,
                                                    unsigned GP1,     unsigned␣
↪GP2,    unsigned GP3,      unsigned GP4);

//! Set the Calibration Adjusting number of iteration
void setCalibrationAdjustingNumber(unsigned calibration_adjusting_number);
```

7.3.24 Xspress3

Introduction

Many solid state detectors are not limited by their intrinsic rate capability, but by the readout system connected to them. The Quantum Detectors Xspress 3 was developed to maximise the throughput and resolution of such detectors and remove the bottleneck at the readout stage. With output count rates of over 3 Mcps, this detector is easily 10X faster than the systems many users have on their beamlines. Xspress 3 can open up the beamline to much faster data collection, its dynamic range can reduce the number of scans required and save large amounts of time with attenuation selection.

The XSPRESS3 system contains a Xilinx Virtex-5 FPGA with two embedded PowerPC processors. PPC1 manages the DMA engines. PPC2 runs the Xilinx micro kernel and communicates to the Intel 64 bit Linux server PC by 1 GBit Ethernet,TCP sockets. Bulk data and event lists to be histogrammed are sent from the firmware to the Server PC by 10G Ethernet, UDP.

The Software Development Toolkit (SDK) is provided for Linux only.

Prerequisite

Unpack the SDK distribution into either the `camera/xspress3/sdk` directory or `/usr/local/lib`. Then ensure the libraries are in the `LD_LIBRARY_PATH`.

The SDK has shared libraries which has been compiled on recent linux kernel. g++ (GCC) 4.1.2 20080704 (Red Hat 4.1.2-50), check first you have the right kernel and libc available by compiling the test program.

The default network setup is (excluding the site network connection):

1GBit Copper network for control communinication between the PC and the XSPRESS3 box. With more than 1 XSPRESS3 box connected this network uses a ethernet switch A private network with 64 addresses allocated:

```
$ ifconfig eth1

eth1      Link encap:Ethernet  HWaddr d4:ae:52:7d:5f:84
          inet addr:192.168.0.1  Bcast:192.168.0.63  Mask:255.255.255.192
          inet6 addr: fe80::d6ae:52ff:fe7d:5f84/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:9000  Metric:1
          RX packets:1567 errors:0 dropped:5766 overruns:0 frame:0
          TX packets:158 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:173937 (169.8 KiB)  TX bytes:37252 (36.3 KiB)
          Interrupt:48 Memory:da000000-da012800
```

A 10GBit Fibre network for data transfer, point to point with 4 addresses allocated. With more that 1 XSPRESS3 box there would be multiple 10G Ports on the PC with multiple 4 address range subnets:

```
$ ifconfig eth2

eth2      Link encap:Ethernet  HWaddr 00:07:43:05:7c:65
          inet addr:192.168.0.65  Bcast:192.168.0.67  Mask:255.255.255.252
          inet6 addr: fe80::207:43ff:fe05:7c65/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:9000  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:702 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:154963 (151.3 KiB)
          Interrupt:41 Memory:dd7fe000-dd7fefff
```

Note the carefully picked subnet masks etc and the MTU 9000 We then have a script that should be executed automatically at boot.

```
$ cat /etc/init.d/xspress3.sh

#!/bin/bash
#
# static-arp        This is to register a static ARP address in the arp table at boot
#
# Kept as simple as possible hopefully this will auto register the associated
# MAC with the private network address to allow the machine to communicate with the
# test boards for xspress3
# Derived from work by Duncan Russell, by William Helsby
PATH=/sbin:/bin:/usr/bin:/usr/sbin
arp -i eth2 -s 192.168.0.66 02:00:00:00:00:00
#route -v add -host 192.168.0.66 eth2
# Setting default and max buffer sizes for networking.
sysctl -w net.core.rmem_max=1073741824
sysctl -w net.core.rmem_default=1073741824
```

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_XSPRESS3=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

In order to help people to understand how the camera plugin has been implemented in LImA this section provide some important information about the developer's choices.

Camera initialisation

The camera will be initialized within the `Xspress3::Camera` object. A TCP socket connection on the 1GBit port is established and optionally a UDP connection on the 10Gbit port (depends on boolean constructor flag noUDP). The ROI's are reset, the first card in a multicard system or the single card, is set to be the master and the run flags are set to initiate Scaler and Histogram modes. The register and configuration settings (as optimised by QD on delivery) are uploaded to the Xspress3.

The Xspress3 requires the following parameters with the recommended settings:

```
nbCards           = 1 (number of Xspress3 boxes)
maxFrames         = 16384
baseIPaddress     = "192.168.0.1"
basePort          = 30123
baseMACaddress    = "02.00.00.00.00.00"
nbChans           = 4/6/8 (depends on the firmware)
createScopeModule = true/false
scopeModuleName   = "a-name-of-your-choice"
debug             = 0 is off, 1 is on, 2 is verbose
cardIndex         = 0 (for a 1 xspress system)
noUDP             = true/false
directoryName     = "directory containing xspress3 configuration settings"
```

The `Xspress3::Camera` contructor sets the camera with default parameters for Number of Pixels (4096), the imageType (Bpp32), Number of Frames (1) and the trigger mode (IntTrig)

Std capabilities

This plugin has been implemented with respect of the mandatory capabilites but with some limitations which are due to the camera and SDK features. We only provide here extra information for a better understanding of the capabilities for Xspress3 cameras.

- HwDetInfo
  - getCurrImageType/getDefImageType(): is set to Bpp32
  - setCurrImageType(): will not change the image type.
  - getMaxImageSize/getDetectorImageSize(): is defined as number of pixels + number of scalers x number of channels, i.e. (4096+8) x 4 for a 4 channel xspress3 system
  - getPixelSize(): is hardcoded to be 1x1
  - getDetectorModel(): reads and reports the xspress3 firmware version.
- HwSync

  get/setTrigMode(): the only supported modes are IntTrig, ExtGate and IntTrigMult

Optional capabilities

None

Data Format

The raw data is saved in .edf file format. Each frame is saved as it completes. To allow Lima to save both histogram and scaler data, the latter is appended to the histogram data.

```
histogram                                          scaler
[0] [0 ... 4095, 4096 ... 5003]                    channel 0
[1] [0 ... 4095, 4096 ... 5003]                    channel 1
[2] [0 ... 4095, 4096 ... 5003]                    channel 2
[3] [0 ... 4095, 4096 ... 5003]                    channel 3
```

- `Camera::readScalers()`: returns the raw scaler data from the Lima buffers from the specified frame and channel
- `Camera::readHistogram()`: returns the raw histogram data from the Lima buffers from the specified frame and channel
- `Camera::setUseDtc()` and `Camera::getUseDtc()`: set to true will dead time correct the data returned from the Lima buffers (default is false)
- `Camera::setUseHW()` and `Camera::getUseHw()`: set to true will return raw histogram data from the H/W data buffers, including the current frame.

How to use

See example in the test directory. Playback data should be extracted from the tarball.

7.3.25 XH camera

Introduction

> "XH is the worlds first 50m pitch Ge Strip detector which has been designed specifically for Energy Dispersive EXAFS (EDE). Carrying on from the CLRC development of XSTRIP1, a Si based detector system, XH makes use of amorphous germanium (a-Ge) contact technology produced by LBNL2 and readout ASICs developed by CLRC. XH is designed to address the issues of detection efficiency and radiation damage that limit the effectiveness of the original XSTRIP system."

The system is controlled from its own PC or via a TCP/IP connection from a beamline computer system.

The Lima plugin has been tested only at ESRF for a unique XH detector on BM23 and ID24 beamlines.

Prerequisite Linux OS

The plugin is only working for Linux distribution and been tested on Redhat E4 i386 and debian 6 x86_64.

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_XH=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

TODO

Std capabilities

This plugin has been implemented in respect of the mandatory capabilites but with some limitations which are due to the camera and SDK features. We only provide here extra information for a better understanding of the capabilities for Andor cameras.

- HwDetInfo

  TODO

- HwSync

TODO

Optional capabilities

In addition to the standard capabilities, we make the choice to implement some optional capabilities which are supported by the SDK and the I-Kon cameras. A Shutter control, a hardware ROI and a hardware Binning are available.

- HwShutter

  TODO

- HwRoi

  TODO

- HwBin

  TODO

Configuration

TODO

How to use

This is a python code example for a simple test:

```
from Lima import Xh
from lima import Core

#                 hostname     port  config name
cam = Xh.Camera('xh-detector', 1972, 'config_xhx3')
hwint = Xh.Interface(cam)
ct = Core.CtControl(hwint)

acq = ct.acquisition()

# configure some hw parameters

# set some low level configuration

# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# now ask for 2 sec. exposure and 10 frames
acq.setAcqExpoTime(2)
acq.setNbImages(10)
```

(continues on next page)

(continued from previous page)

```
ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(0.1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)
```

7.3.26 Zwo (Zhen Wang Optical)

Introduction

ZWO offers a large choice of cameras for astronomical applications. The cameras are connected via USB. The delivered driver library is available for Linux, Mac, and Windows.

The LImA module has been tested with the ASI 178MM-Cool model on Linux.

Prerequisite

Installation & Module configuration

- follow first the steps for the linux installation linux_installation
- follow first the steps for the windows installation windows_installation

The minimum configuration file is *config.inc* :

```
COMPILE_CORE=1
COMPILE_SIMULATOR=0
COMPILE_SPS_IMAGE=1
COMPILE_ESPIA=0
COMPILE_FRELON=0
COMPILE_MAXIPIX=0
COMPILE_PILATUS=0
COMPILE_BASLER=0
COMPILE_PROSILICA=0
COMPILE_ROPERSCIENTIFIC=0
COMPILE_MYTHEN=0
COMPILE_ADSC=0
COMPILE_UEYE=0
COMPILE_XH=0
COMPILE_XSPRESS3=0
COMPILE_XPAD=0
COMPILE_PERKINELMER=0
COMPILE_ANDOR=0
COMPILE_PHOTONICSCIENCE=0
COMPILE_PCO=0
COMPILE_MARCCD=0
COMPILE_POINTGREY=0
COMPILE_IMXPAD=0
COMPILE_DEXELA=0
COMPILE_ZWO=1
COMPILE_RAYONIXHS=0
COMPILE_CBF_SAVING=0
COMPILE_NXS_SAVING=0
COMPILE_FITS_SAVING=0
COMPILE_EDFGZ_SAVING=0
COMPILE_TIFF_SAVING=0
COMPILE_CONFIG=1
LINK_STRICT_VERSION=0
export COMPILE_CORE COMPILE_SPS_IMAGE COMPILE_SIMULATOR \
      COMPILE_ESPIA COMPILE_FRELON COMPILE_MAXIPIX COMPILE_PILATUS \
      COMPILE_BASLER COMPILE_PROSILICA COMPILE_ROPERSCIENTIFIC COMPILE_ADSC \
      COMPILE_MYTHEN COMPILE_UEYE COMPILE_XH COMPILE_XSPRESS3 COMPILE_XPAD COMPILE_
↪PERKINELMER \
      COMPILE_ANDOR COMPILE_PHOTONICSCIENCE COMPILE_PCO COMPILE_MARCCD COMPILE_DEXELA␣
↪COMPILE_ZWO\
```

(continues on next page)

(continued from previous page)

```
      COMPILE_POINTGREY COMPILE_IMXPAD COMPILE_RAYONIXHS COMPILE_CBF_SAVING COMPILE_
↪NXS_SAVING \
      COMPILE_FITS_SAVING COMPILE_EDFGZ_SAVING COMPILE_TIFF_SAVING COMPILE_CONFIG\
      LINK_STRICT_VERSION
```

- start the compilation linux_compilation
- finally for the Tango server installation *PyTango Device Server*

Initialisation and Capabilities

In order to help people to understand how the camera plugin has been implemented in LImA this section provide some important information about the developer's choices.

Camera initialisation

TODO

Std capabilites

This plugin has been implement in respect of the mandatory capabilites but with some limitations according to some programmer's choices. We only provide here extra information for a better understanding of the capabilities for the Zwo camera.

- HwDetInfo

TODO

- HwSync

TODO

Optional capabilites

In addition to the standard capabilities, we make the choice to implement some optional capabilities in order to have an improved simulation.

TODO

- BinCtrl

  TODO

- BufferCtrl

  TODO

- FlipCtrl

  TODO

- RoiCtrl

  TODO

- ShutterCtrl

  TODO

- SavingCtrl

  TODO

- VideoCtrl

  TODO

Configuration

TODO

How to use

The LimaCCDs tango server provides a complete interface to the zwo plugin so feel free to test.

For a quick test one can use python, is this a short code example:

```
from Lima import Zwo
from lima import Core
import time

cam = Zwo.Camera(0)
hwint = Zwo.Interface(cam)

control = Core.CtControl(hwint)

acq = control.acquisition()

# setting new file parameters and autosaving mode
saving = control.saving()

pars = saving.getParameters()
pars.directory = '/tmp/'
pars.prefix = 'testsimul_'
pars.suffix = '.edf'
pars.fileFormat = Core.CtSaving.EDF
pars.savingMode = Core.CtSaving.AutoFrame
saving.setParameters(pars)

# now ask for 2 sec. exposure and 10 frames
acq.setAcqExpoTime(2)
acq.setNbImages(10)

control.prepareAcq()
control.startAcq()

# wait for last image (#9) ready
lastimg = control.getStatus().ImageCounters.LastImageReady
while lastimg != 9:
    time.sleep(0.1)
    lastimg = control.getStatus().ImageCounters.LastImageReady
```

(continues on next page)

(continued from previous page)

```
# read the first image
im0 = control.ReadImage(0)
```

7.4 Windows and Linux

7.4.1 Andor SDK2 camera plugin

Introduction

Andor Technology manufactuer offers a large catalogue of scientific cameras. Covered scientific applications are low ligth imaging, spectroscopy, microscopy, time-resolved and high energy detection. Andor is providing a unique Software Development Tool (SDK) for both Windows and Linux, supporting different interface buses such as USB, CameraLink and also some specific acquisition PCI board.

The Lima module as been tested only with these camera models:

- IKon-M and IKon-L (USB interface, Linux OS debian 6)
- IKon-L (USB interface, Windows XP - 32bits)

Prerequisites

Linux

First, you have to install the Andor Software developpement Kit (SDK) in the default path (/usr/local). For our tests, we used the SDK for Linux version **V2.91.30001.0** and ran the install script `install_andor` for which option 5 (All USB Cameras) was selected, the default installation is made under `/usr/local/` with:

- `/usr/local/include`, header files
- `/usr/local/lib`, library files
- `/usr/local/etc/andor`, configuration files

The Linux SDK 2.91 has shared libraries which has been compiled on recent linux kernel, check first you have the right kernel and libc available by compiling one of the example program available under examples/console. Andor python module needs at least the lima core module.

For the USB camera the SDK is using the libusb under linux, check first your system is equiped with the libusb package otherwise you will not compile the Andor Lima plugin.

Windows XP - 32 bits

First, you have to install the Andor Software developpement Kit (SDK) in default path (`C:\\Program Files (x86)\\Andor iKon\\Drivers`).

Add the location of the file `\\Lima\\camera\\andor\\sdk\\msvc\\bin\\ATMCD32D.DLL` to your `PATH` environment variable.

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_ANDOR=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

The camera will be initialized within the `AndorCamera` object. The `AndorCamera()` contructor sets the camera with default parameters for Preampifier-Gain, VerticalShiftSpeed and the ADC/HorizontalSpeed.

These parameters are optimized for the faster mode, which means the maximum gain, the "fasten recommended" VSSpeed (i.e as returned by GetFastestRecommendedVSSpeed() SDK function call) and the ADC with the faster Horizontal speed.

All the parameters can be set and get using the corresponding methods, the default values (max speeds and gain) can be applied with -1 as passed value:

set/getPGain()

set/getVsSpeed()

set/getADCSpeed()

Some other methods are available but they can not be supported depending on which camera model you are using:

set/getHighCapacity()

set/getFanMode()

set/getBaselineClamp()

The above parameters, only support enumerate type for values.

Std capabilities

This plugin has been implemented in respect of the mandatory capabilites but with some limitations which are due to the camera and SDK features. We only provide here extra information for a better understanding of the capabilities for Andor cameras.

- HwDetInfo

  getCurrImageType/getDefImageType(): the methods call the SDK GetBitDepth() function to resolve the image data type. The bit-depth correspond to the AD channel dynamic range which depends on the selected ADC channel. By experience and with IKon detectors we only have Bpp16 of dynamic range, but the methods can return Bpp8 and Bpp32 as well.

  setCurrImageType(): this method do not change the image type which is fixed to 16bpp.

- HwSync

  get/setTrigMode(): the only supported mode are IntTrig, ExtTrigSingle, ExtGate and IntTrigMult

Optional capabilities

In addition to the standard capabilities, we make the choice to implement some optional capabilities which are supported by the SDK and the I-Kon cameras. A Shutter control, a hardware ROI and a hardware Binning are available.

- HwShutter

  setMode(): only ShutterAuto and ShutterManual modes are supported

- HwRoi

  There is no restriction for the ROI setting

- HwBin

  There is no restriction for the Binning but the maximum binning is given by the SDK function GetMaximumBinning() which depends on the camera model

Configuration

Plug your USB camera on any USB port of the computer, that's all !

How to use

This is a python code example for a simple test:

```
from Lima import Andor
from lima import Core

cam = Andor.Camera("/usr/local/etc/andor", 0)
hwint = Andor.Interface(cam)
ct = Core.CtControl(hwint)

acq = ct.acquisition()

# configure some hw parameters
hwint.setTemperatureSP(-30)
hwint.setCooler(True)
.... wait here for cooling

# set some low level configuration
hwint.setPGain(2)
hwint.setCooler(True)
hwint.setFanMode(cam.FAN_ON_FULL)
hwint.setHighCapacity(cam.HIGH_SENSITIVITY)
hwint.setBaselineClamp(cam.BLCLAMP_ENABLED)
hwint.setFastExtTrigger(False)
hwint.setShutterLevel(1)


# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
```

(continues on next page)

(continued from previous page)

```
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# set accumulation mode

acq_pars= acq.getPars()

#0-normal,1-concatenation,2-accumu
acq_pars.acqMode = 2
acq_pars.accMaxExpoTime = 0.05
acq_pars.acqExpoTime =1
acq_pars.acqNbFrames = 1

acq.setPars(acq_pars)
# here we should have 21 accumalated images per frame
print acq.getAccNbFrames()

# now ask for 2 sec. exposure and 10 frames
acq.setAcqExpoTime(2)
acq.setNbImages(10)

ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)
```

7.4.2 Basler camera

Introduction

Basler's area scan cameras are designed for industrial users who demand superior image quality and an excellent price/performance ratio. You can choose from an area scan portfolio that includes monochrome or color models with various resolutions, frame rates, and sensor technologies.

The Lima module has been tested only with this GigE cameras models:

- Scout
- Pilot
- Ace

The Lima module has been tested with Pylon SDK versions **3.2.2** and **5.0.1**.

Monochrome and color cameras are supported with these SDK versions.

Installation & Module configuration

First, you have to install the Basler SDK *Pylon* to the default path `/opt/pylon`.

Then, follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_BASLER=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

The camera will be initialized by creating Basler ::Camera object. The Basler camera can be idenfified either by:

- IP/hostname (examples: `ip://192.168.5.2`, `ip://white_beam_viewer1.esrf.fr`) or
- Basler serial number (example: `sn://12345678`) or
- Basler user name (example: `uname://white_beam_viewer1`)

In case an IP is given, the `ip://` scheme prefix is optional.

Only the camera ID is mandatory.

Small example showing possible ways to initialize:

```
from Lima import Basler
from lima import Core

# From an IP (notice ip:// prefix is optional)
cam = Basler.Camera('192.168.5.2')

# From a basler serial number
cam = Basler.Camera('sn://12345678')
```

(continues on next page)

(continued from previous page)

```
# From a basler user name
cam = Basler.Camera('uname://white_beam_viewer1')
```

Std capabilites

This plugin has been implemented in respect of the mandatory capabilites but with some limitations which are due to the camera and SDK features. Only restriction on capabilites are documented here.

- HwDetInfo

  getCurrImageType/getDefImageType(): it can change if the video mode change (see HwVideo capability).

  setCurrImageType(): It only supports Bpp8 and Bpp16.

- HwSync

  get/setTrigMode(): the supported mode are IntTrig, IntTrigMult, ExtTrigMult and ExtGate.

Optional capabilites

In addition to the standard capabilities, we make the choice to implement some optional capabilities which are supported by the SDK. **Video**, Roi and Binning are available.

- HwVideo

  The basler cameras are pure video device, so video format for image are supported:

  Color cameras ONLY

  - BAYER_RG8
  - BAYER_BG8
  - BAYER_RG16
  - BAYER_BG16
  - RGB24
  - BGR24
  - RGB32
  - BGR32
  - YUV411
  - YUV422
  - YUV444

  Color and Monochrome cameras

  - Y8
  - Y16

  Use get/setMode() methods of the *video* object (i.e. CtControl::video()) to read or set the format.

- HwBin

  There is no restriction for the binning up to the maximum size.

- HwRoi

  There is no restriction for the Roi up to the maximum size.

Configuration

- First you need to decide how you want to reference your camera (by IP/hostname, serial number or user name)
- Second, you have to setup the IP address of the Basler Camera by using *IpConfigurator* (`/opt/pylon/bin/IpConfigurator`) or by matching the MAC address with a choosen IP into the DHCP. If you plan to reference the camera by user name you should also set it in *IpConfigurator*. If you plan to reference the camera by serial number you should note down the serial number that appears in the label of your camera.
- Then in the Basler Tango device, set the property *camera_id* according to the type of ID you choose (see *Basler Tango device* for more details)
- If you are running the server with linux kernel >= 2.6.13, you should add this line into */etc/security/limits.conf*. With this line, the acquisition thread will be in real time mode.

```
USER_RUNNING_DEVICE_SERVER    -       rtprio  99
```

How to use

This is a python code example for a simple test:

```
from Lima import Basler
from lima import Core

#---------------------------------------+
#                        packet-size    |
#                                       |
#-------------------------------------+  |
#              inter-packet delay     |  |
#                                     |  |
#-----------------------------------+  |  |
#      frame-transmission delay     |  |  |
#                                   |  |  |
#--------------------+              |  |  |
# cam ip or hostname |              |  |  |
#                    v              v  v  v
cam = Basler.Camera('192.168.1.1', 0, 0, 8000)

hwint = Basler.Interface(cam)
ct = Core.CtControl(hwint)

acq = ct.acquisition()


# set and test video
#

video=ct.video()
video.setMode(Core.RGB24)
video.startLive()
video.stopLive()
video_img = video.getLastImage()
```

(continues on next page)

(continued from previous page)

```
# set and test an acquisition
#

# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.TIFF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# now ask for 2 sec. exposure and 10 frames
acq.setAcqExpoTime(2)
acq.setNbImages(10)

ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)
```

7.4.3 RoperScientific / Princeton

Introduction

This plugin control a RoperScientific/Princeton camera under Windows and Linux, using the PVCAM (Photometrics Virtual Camera Access Method) libraries.

It is in production at SOLEIL under windows and it has been tested at Desy under Linux. Model used at SOLEIL: PI-MTE:2048B

Prerequisite

The RoperScientific is connected to a specific computer with a PCI board. The Lima/RoperScientific client must run on this PC.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

The camera will be initialized within the :cpp:`RoperScientific::Camera` object. The camera number (as an integer) should be given to the constructor. For example: 0.

Std capabilites

This plugin has been implemented in respect of the mandatory capabilites but with some limitations according to some programmer's choices. We only provide here extra information for a better understanding of the capabilities for the RoperScientific camera.

- HwDetInfo
- Max image size is : 2048 * 2048
- 16 bit unsigned type is supported
- HwSync

  Trigger type supported are:

  - IntTrig
  - ExtTrigSingle
  - ExtTrigMult
  - ExtGate

Optional capabilites

- HwBin:
  - all values are accepted
- HwRoi

Specific control parameters

Some specific paramaters are available within the camera hardware interface. Those parameters should be used carefully and one should refer to the camera SDK (or user's guide) documentation for a better understanding.

- getTemperature()
- set/getTemperatureSetPoint()
- set/getGain()
- set/getInternalAcqMode()
- "FOCUS"
- "STANDARD"
- set/getSpeedTableIndex()

Configuration

No Specific hardware configuration are needed

How to use

Here is the list of accessible fonctions to configure and use the RoperScientific detector:

```
void setGain(long);
long getGain();

void setFullFrame(rgn_type* roi);
void setBinRoiParameters(rgn_type* roi);

void setSpeedTableIndex(unsigned);
unsigned getSpeedTableIndex(void);
const std::string& getADCRate(void);

double  getTemperature();
double  getTemperatureSetPoint();
void    setTemperatureSetPoint(double temperature);
```

Code example in python:

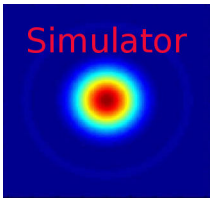
```
from Lima import RoperScientific
from lima import Core

cam = RoperScientific.Camera(0)

hwint = RoperScientific.Interface(cam)
```

(continues on next page)

(continued from previous page)

```
ct = Core.CtControl(hwint)

acq = ct.acquisition()

# set some configuration
cam.setTemperatureSetPoint(0)
cam.setAdcRate(0) # 0-1MHz, 1-100KHz


# setting new file parameters and autosaving mode
saving=ct.saving()

pars=saving.getParameters()
pars.directory='/buffer/lcb18012/opisg/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# now ask for 2 sec. exposure and 10 frames
acq.setAcqExpoTime(2)
acq.setNbImages(10)

ct.prepareAcq()
ct.startAcq()

# wait for last image (#9) ready
lastimg = ct.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(0.1)
  lastimg = ct.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = ct.ReadImage(0)
```

7.4.4 Simulator



Introduction

This is the official Lima camera simulator. It has been made to help you getting started with Lima and to test/play Lima without any hardware.

The simulator provides two modes of operations:

- **Frame Builder** generates frames with diffraction patterns and a set of parameters can be tuned to change those patterns like for instance the number and position of gaussian peaks;
- **Frame Loader** loads frames from files.

Both modes have a preteched variant, where the frames are preteched in memory before the acquisition is started. This feature allows to simulate high frame rates detectors.

Prerequisite

There is no special prerequisite, the simulator can be compiled and tested on both Linux and Windows platforms.

Installation & Module configuration

Follow the generic instructions in *Build and Install*. If using CMake directly, add the following flag:

```
-DLIMACAMERA_SIMULATOR=true
```

For the Tango server installation, refers to *PyTango Device Server*.

Initialisation and Capabilities

Implementing a new plugin for new detector is driven by the LIMA framework but the developer has some freedoms to choose which standard and specific features will be made available. This section is supposed to give you the correct information regarding how the camera is exported within the LIMA framework.

Camera initialisation

The camera will be initialized within the `Camera` object. The `Camera()` constructor takes an optional mode parameter.

This simulator plugin architecture is based on the `FrameGetter` interface that have multiple implementations.

The `SimulatorCamera` class provides a specific member function `SimulatorCamera::getFrameGetter()` that returns the `FrameGetter` instance.

Depending on the current mode, `FrameGetter` can be dynamically casted to either:

- `FrameBuilder`
- `FrameLoader`
- `FramePrefetcher`
- `FramePrefetcher`

The class `FrameBuilder` can be parametrized with:

- `setFrameDim()`: set a new frame dimension (max. is 1024x1024)
- `setPeaks()`: set a list of GaussPeak positions (GaussPeak struct -> x, y, fwhm, max)

- `setPeakAngles()`: set a list of GaussPeak angles
- `setFillType()`: set the image fill type Gauss or Diffraction (default is Gauss)
- `setRotationAxis()`: set the rotation axis policy Static, RotationX or RotationY (default is RotationY)
- `setRotationAngle()`: set a peak rotation angle in deg (default is 0)
- `setRotationSpeed()`: set a peak rotation speed ixin deg/frame (default is 0)
- `setGrowFactor()`: set a growing factor (default is 1.0)
- `setDiffractionPos()`: set the source diplacement position x and y (default is center)
- `setDiffractionSpeed()`: set the source diplacement speed sx and sy (default is 0,0)

The class `FrameLoader` can be parametrized with:

- `setFilePattern()`: set the file pattern used to load the frames than may include globing pattern, i.e. `input/test_*.edf`

The `template <typename FrameGetterImpl> FramePrefetcher` variants have an addition parameter:

- `setNbPrefetchedFrames()`: set the number of frames to prefetch in memory

Standard capabilities

This plugin has been implemented in respect of the standard capabilites of a camera plugin but with some limitations according to some programmer's choices. We only provide here extra information for a better understanding of the capabilities for the simulator camera.

- `HwDetInfo`: The default (and max.) frame size if about 1024x1024-Bpp32, but one can only change the image type by calling `DetInfoCtrlObj::setCurrImageType()`.
- `HwSync`: Only IntTrig trigger mode is supported. For both exposure time and latency time min. is 10e-9 and max. is 10e6.

Optional capabilities

In addition to the standard capabilities, some optional capabilities are implemented:

- `HwShutter`: The simulator only support ShutterAutoFrame and ShutterManual modes.
- `HwRoi`: There is no restriction for the ROI.
- `HwBin`: Bin 1x1 or 2x2 only.

Configuration

No hardware configuration of course!

How to use

The LimaCCDs tango server provides a complete interface to the simulator plugin so feel free to test.

For a quick test one can use the python binding, here is a short code example:

```
from Lima import Simulator
from Lima import Core
import time

def test_mode_generator(cam, nb_frames_prefetched = 0):
    if nb_frames_prefetched:
        cam.setMode(Simulator.Camera.MODE_GENERATOR_PREFETCH)
        fb = cam.getFrameGetter()
        fb.setNbPrefetchedFrames(nb_frames_prefetched);
    else:
        cam.setMode(Simulator.Camera.MODE_GENERATOR)
        fb = cam.getFrameGetter()

    # Add a peak
    p1 = Simulator.GaussPeak(10, 10, 23, 1000) # peak at 10,10 fwhm=23 and max=1000
    fb.setPeaks([p1])


def test_mode_loader(cam, nb_frames_prefetched = 0):
    if nb_frames_prefetched:
        cam.setMode(Simulator.Camera.MODE_LOADER_PREFETCH)
        fb = cam.getFrameGetter()
        test = fb.getNbPrefetchedFrames();
    else:
        cam.setMode(Simulator.Camera.MODE_LOADER)
        fb = cam.getFrameGetter()

    # Set file pattern
    fb.setFilePattern(b'input\\test_*.edf')

cam = Simulator.Camera()

# Select one of the mode to test
#test_mode_generator(cam)
#test_mode_generator(cam, 10)
#test_mode_loader(cam)
test_mode_loader(cam, 100)

# Get the hardware interface
hwint = Simulator.Interface(cam)

# Get the control interface
control = Core.CtControl(hwint)

# Get the acquisition control
acq = control.acquisition()

# Set new file parameters and autosaving mode
saving=control.saving()

pars=saving.getParameters()
pars.directory='/tmp/'
```

(continues on next page)

(continued from previous page)

```
pars.prefix='testsimul_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# Now ask for 2 sec. exposure and 10 frames
acq.setAcqExpoTime(2)
acq.setAcqNbFrames(10)

control.prepareAcq()
control.startAcq()

# Wait for last image (#9) ready
lastimg = control.getStatus().ImageCounters.LastImageReady
while lastimg !=9:
  time.sleep(0.1)
  lastimg = control.getStatus().ImageCounters.LastImageReady

# read the first image
im0 = control.ReadImage(0)
```

CHAPTER

EIGHT

FUTURE CAMERAS

8.1 Acknowledgement

Many contributors contributed to new camera plugins, including:

- ESRF,
- SOLEIL,
- DESY,
- ALBA,
- FRMII,
- ANKA.

thank you for your support.

8.2 Under development

During the coming year, several new detector plugins should be released:

- Arinax Bi-zoom (Arinax ltd.)
- QHYCCD model Q178-Cool (FRMII)

8.3 Foreseen

- Ximea for high resolution, 6kx6k pixel (ESRF)

CHAPTER

NINE

PYTHON TANGO SERVER

This is the python Tango devices server by the ESRF team.

This server provides a main device for the standard camera control, a camera specific device for the camera configuration and a set of "plugin" devices for extra operations or just to provide some specific API for clients.

Thanks to the Lima framework, the control can be achieved through a common server and a set of software operations (Mask,Flatfield,Background,RoiCounter,PeakFinder...) on image as well. The configuration of the detector is done by the specific detector device. At ESRF we decided to develop the Tango devices only in python language which implies that all the detector C++ interfaces have been wrapped in python.

9.1 Main device: LimaCCDs

LimaCCDs is the generic device and it provides a unique interface to control any supported cameras. One can find below the commands, the attributes and the properties.

To run a LimaCCDs server you will need at least to configure the **LimaCameraType** property. This property is used by the LimaCCDs server to create the proper camera device. Pleas refer a specific camera (e.g Basler) device chapter for further information.

9.1.1 Property

Property name	Mandatory	Default value	Description
AccThresholdCallbackModule	No	""	Plugin file name which manages threshold, see acc_saturated_* attributes and the *AccSaturated* commands to activate and use this feature
BufferMaxMemory	No	70	The maximum among of memory in percent of the available RAM that Lima is using to allocate frame buffer.
ConfigurationFilePath	No	~/lima_<servname>.cfg	The default configuration file path
ConfigurationDefaultName	No	"default"	Your default configuration name
IntrumentName	No	""	The instrument name, e.g ESRF-ID02 (*)
LimaCameraType	Yes	N/A	The camera type: e.g. Maxipix
MaxVideoFPS	No	30	Maximum value for frame-per-second
NbProcessingThread	No	1	The max number of thread for processing. Can be used to improve the performance when more than 1 task (plugin device) is activated
TangoEvent	No	False	Activate Tango Event for counters and new images
UserDetectorName	No	""	A user detector identifier, e.g frelon-saxs, (*)

(*) Properties only used to set meta-data in HDF5 saving format.

9.1.2 Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrStringValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name
prepareAcq	DevVoid	DevVoid	Prepare the camera for a new acquisition, has to be called each time a parameter is set.
startAcq	DevVoid	DevVoid	Start the acquisition
stopAcq	DevVoid	DevVoid	Stop the acquisition after current frame is acquired, and wait for all tasks to finish
abortAcq	DevVoid	DevVoid	Abort the acquisition, the current frame is lost
setImageHeader	DevVarStringArray: Array of string header	DevVoid	**Set the image header:** • [0]=”ImageId0 delimiter imageHeader0, • [1] = ImageId1 delimiter imageHeader1..
resetCommonHeader	DevVoid	DevVoid	Reset the common header
resetFrameHeaders	DevVoid	DevVoid	Reset the frame headers
getImage	DevLong: Image number(0-N)	DevVarCharArray: Image data	Return the image data in raw format (char array)
getBaseImage	DevLong: Image number(0-N)	DevVarCharArray: Image data	Return the base image data in raw format (char array). Base image is the raw image before processing
readImage	DevLong: Image number(0-N)	DevEncoded: Encoded image	Return the image in encoded format of type “**DATA_ARRAY**” (see *DevEncoded DATA_ARRAY*)
readImageSeq	DevLongArray: Image number(0-N) list	DevEncoded: Encoded image(S)	Return a stack of images in encoded format of type “**DATA_ARRAY**” (see *DevEncoded DATA_ARRAY*)
writeImage	DevLong: Image number(0-N)	DevVoid	Save manually an image
readAccSaturatedImageCounter	DevLong: Image number	DevVarUShortArray: Image counter	The image counter
readAccSaturatedSumCounter	DevLong: from image id	DevVarLongArray: number of readed images,sum counter of raw image #0 of image #0,sum counter of raw	

9.1.3 Attributes

You will here a long list of attributes, this reflects the richness of the LIMA library. We organized them in modules which correspond to specific functions. A function module is identified by an attribute name prefix (excepted for informationnal attributes), for instance the **Acquisition** module attributes are always named **acq_<attr-name>**. The available modules are :

- General Information
- Status (prefix *last_* and *ready_*)
- Acquisition (prefix *acq_* for most of them sorry)
- Accumulation (prefix *acc_*)
- Saving (prefix *saving_*)
- Image (prefix *image_*)
- Shutter (prefix *shutter_*)
- Debug (prefix *debug_*)
- Video (prefix *video_*)
- Shared Memory (prefix *shared_memory_*)
- Configuration (prefix *config_*)
- Buffer (prefix *buffer_*)
- Plugin (prefix *plugin_*)

Many attributes are of type DevString and they have a fixed list of possible values. you can get the list by calling the special command **getAttrStringValueList**. Because a camera cannot support some attribute values , the command getAttrStringValueList will give you the the value list for the camera. For instance the attribute *video_mode* supports up to 14 different video formats, but a camera can only supports few of them.

Attribute name	RW	Type	Description
		GENERAL INFORMATION	
lima_version	ro	DevString	The lima core library version number
lima_type	ro	DevString	LImA camera type: Maxipix,Pilatus,Frelon,Pco, Basler ...
camera_type	ro	DevString	Like lima_type but in upper-case !!
camera_pixelsize	ro	DevDouble[x,y]	The camera pixel size in x and y dimension
camera_model	ro	DevString	Camera model return by the detector layer:.e.g. 5x1- TPX1
		STATUS	
last_base_image_ready	ro	DevLong	The last base (before treatment) ready
last_image_ready	ro	DevLong	The last acquired image number, ready for reading

continues on next page

Table 1 – continued from previous page

Attribute name	RW	Type	Description
last_image_saved	ro	DevLong	The last saved image number
last_image_acquired	ro	DevLong	The last acquired image number
last_counter_ready	ro	DevLong	Tell which image counter is last ready
ready_for_next_image	ro	DevBoolean	True after a camera readout, otherwise false. Can be used for fast synchronisation with trigger mode (internal or external).
ready_for_next_acq	ro	DevBoolean	True after end of acquisition, otherwise false.
user_detector_name	rw	DevString	User detector name
instrument_name	rw	DevString	Intrument/beamline name
		ACQUISITION	
acq_status	ro	DevString	Acquisition status: Ready, Running, Fault or Configuration
acq_status_fault_error	ro	DevString	In case of Fault state, return the error message
acq_mode	rw	DevString	**Acquisition mode:** • **Single**, default mode one frame per image • **Concatenation**, frames are concatenated in image • **Accumulation**, powerful mode to avoid saturation of the pixel, the exposure is shared by multiple frames, see acc_ attributes for more
acq_nb_frames	rw	DevLong	Number of frames to be acquired, Default is 1 frame

continues on next page

Table 1 – continued from previous page

Attribute name	RW	Type	Description
acq_trigger_mode	rw	DevString	**Trigger mode:** • **Internal_trigger**, the software trigger, start the acquisition immediately after an acqStart() call, all the acq_nb_frames are acquired in an sequence. • **External_trigger**, wait for an external trigger signal to start the an acquisition for the acq_nb_frames number of frames. • **External_trigger_multi**, as the previous mode except that each frames need a new trigger input (e.g. for 4 frames 4 pulses are waiting for) • **Internal_trigger_multi**, as for internal_trigger except that for each frame the startAcq() has to called once. • **External_gate**, wait for a gate signal for each frame, the gate period is the exposure time. • **External_start_stop**

Table 1 – continued from previous page

Attribute name	RW	Type	Description
latency_time	rw	DevDouble	Latency time in second between two frame acquisitions, can not be zero, the minimum time corresponds to the readout time of the detector.
valid_ranges	ro	DevDouble[4]	min exposure, max exposure, min latency, max latency
concat_nb_frames	rw	DevLong	The nb of frames to concatenate in one image
acq_expo_time	rw	DevDouble	The exposure time of the image, Default is 1 second
		ACCUMULATION	
acc_expotime	ro	DevDouble	The effective accumulation total exposure time.
acc_nb_frames	ro	DevLong	The calculated accumulation number of frames per image.
acc_max_expotime	rw	DevDouble	The maximum exposure time per frame for accumulation
acc_time_mode	rw	DevString	**Accumulation time mode:** • **Live**,acq_expo_time = acc_live_time • **Real**,acq_expo_time = acc_dead_time + acc_live_time
acc_dead_time	ro	DevDouble	Total accumulation dead time
acc_live_time	ro	DevDouble	Total accumulation live time which corresponds to the detector total counting time.
acc_offset_before	rw	DevLong	Set a offset value to be added to each pixel value
acc_saturated_active	rw	DevBoolean	To activate the saturation counters (i.e. readAccSaturated commands)

continues on next page

Table 1 – continued from previous page

Attribute name	RW	Type	Description
acc_saturated_cblevel	rw	DevLong	Set at which level of total saturated pixels the callback plugin (if set with the AccThresholdCallbackModule property) will be called
acc_saturated_threshold	rw	DevLong	The threshold for counting saturated pixels
acc_threshold_before	rw	DevLong	Set a threshold value to be substract to each pixel value
		SAVING	
saving_mode	rw	DevString	**Saving mode:** • **Manual**, no automatic saving, a command will be implemented in a next release to be able to save an acquired image. • **Auto_Frame**, Frames are automatically saved according the saving parameters (see below). • **Auto_header**, Frames are only saved when the setImageHeader() is called in order to set header information with image data.
saving_directory	rw	DevString	The directory where to save the image files
saving_prefix	rw	DevString	The image file prefix
saving_suffix	rw	DevString	The image file suffix

continues on next page

Table 1 – continued from previous page

Attribute name	RW	Type	Description
saving_next_number	rw	DevLong	The image next number The full image file name is: /saving_directory/saving_prefix+sprintf("%04d",s
saving_format	rw	DevString	**The data format for saving:** • **Raw**, save in binary format • **Edf**, save in ESRF Data Format • **edfgz** (or edf.gz), EDF with gz compression • **Tiff**, The famous TIFF format • **Cbf**, save in CBF format (a compressed format for crystallography)
saving_overwrite_policy	rw	DevString	**In case of existing files an overwite policy is** • **Abort**, if the file exists the saving is aborted • **Overwrite**, if the file exists it is overwritten • **Append**, if the file exists the image is append to the file
saving_frame_per_file	rw	DevLong	Number of frames saved in each file
saving_common_header	rw	DevString[]	Common header with multiple entries

continues on next page

Table 1 – continued from previous page

Attribute name	RW	Type	Description
saving_header_delimiter	rw	DevString[]	The header delimiters, [0] = key header delimiter, [1] = entry header delimiter, [2] = image number header delimiter. Default : [0] = "=", [1] = "n", [2] = ";"
saving_max_writing_task	rw	DevShort	Set the max. tasks for saving file, default is 1
saving_statistics	ro	DevDouble[]	Return stats: saving speed, compression ratio, compression speed and incoming speed (speed in byte/s)
saving_statistics_history	rw	DevLong	Set size of history for stats calculation, default is 16 frames
		IMAGE	
image_type	ro	DevString	**Return the current image data type, bit per** • Bpp8, Bpp8S, Bpp10, Bpp10S, Bpp12, Bpp12S, Bpp14, • Bpp14S, Bpp16, Bpp16S, Bpp32, Bpp32S , Bpp32F.
image_width	ro	DevLong	Width size of the detector in pixel
image_height	ro	DevLong	Height size of the detector in pixel
image_sizes	ro	DevULong[4]	Signed(0-unsigned,1-signed), depth(nb bytes), width and height
image_max_dim	ro	DevULong[2]	Maximum image dimension, width and height in pixel
image_roi	rw	DevLong[4]	Region Of Interest on image, [0] = Begin X, [1] = End X, [2] Begin Y, [3] = End Y, default ROI is [0,0,0,0] (no ROI)

continues on next page

Table 1 – continued from previous page

Attribute name	RW	Type	Description
image_bin	rw	DevLong[2]	Binning on image, [0] = Binning factor on X, [1] = Binning factor on Y. Default binning is 1 x 1
image_flip	rw	DevBoolean[2]	Flip on the image, [0] = flip over X axis, [1] flip over Y axis. Default flip is False x False
image_rotation	rw	DevString	Rotate the image: “0”, “90”, “180” or “270”
		SHUTTER	
shutter_ctrl_is_available	ro	DevBoolean	Return true if the camera has a shutter control
shutter_mode	rw	DevString	**Synchronization for shutter, modes are avai** • **Manual** • **Auto_frame**, the output signal is activated for each individual frame of a sequence • **Auto_sequence**, the output signal is activated during the whole sequence
shutter_open_time	rw	DevDouble	Delay (sec.) between the output shutter trigger and the beginning of the acquisition, if not null the shutter signal is set on before the acquisition is started.
shutter_close_time	rw	DevDouble	Delay (sec.) between the shutter trigger and the end of the acquisition, if not null the shutter signal is set on before the end of the acquisition.
shutter_manual_state	rw	DevString	To open/close manually the shutter (if Manual mode is supported, see shutter_mode)

continues on next page

Table 1 – continued from previous page

Attribute name	RW	Type	Description
		DEBUG	
debug_module_possible	ro	DevString[]	Return the list of possible debug modules
debug_modules	rw	DevString[]	**Set the debug module level of LImA:** • "None" • "Common" • "Hardware" • "HardwareSerial" • "Control" • "Espia" • "EspiaSerial" • "Focla" • "Camera" • "CameraCom" • "Test" • "Application"
debug_types_possible	ro	DevString[]	Return the list of the possible debug types
debug_types	rw	DevString[]	**Set the debug type level of LImA:** • "Fatal" • "Error" • "Warning" • "Trace" • "Funct" • "Param" • "Return" • "Always"
		VIDEO	
video_active	rw	DevBoolean	Start the video mode (or not)
video_live	rw	DevBoolean	Start the video streaming (or not)
video_exposure	rw	DevDouble	The video exposure time (can be different to the acq_expo_time)
video_gain	rw	DevDouble	The video gain (if supported by the hardware)

continues on next page

Table 1 – continued from previous page

Attribute name	RW	Type	Description
video_mode	rw	DevString	**The video mode is the video format support** • Y8, grey image 8bits • Y16, grey image 16bits • Y32, grey image 32bits • RGB555, color image RGB 555 encoding • RGB564, color image RGB 555 encoding • RGB24, color image RGB 24bits encoding • RGB32, color image RGB 32bits encoding • BGR24, color image BGR 24bits encoding • BGR32, color image BGR 32bits encoding • BAYER_RG8, color image BAYER RG 8bits encoding • BAYER_RG16, color image BAYER RG 16bits encoding • I420, color image I420 (or YUV420) planar encoding • YUV411, color image YUV411 planar encoding • YUV422PACKED, color image YUV422 pla-

Table 1 – continued from previous page

Attribute name	RW	Type	Description
video_roi	rw	DevLong[4]	A ROI on the video image (independent of the image_roi attribute)
video_bin	rw	DevULong[2]	A Binning on the video image (independt of the image_bin attribute)
video_last_image	rw	DevEncoded	The last video image, in DevEncoded "**VIDEO_IMAGE**" format, and using the video_mode set, see the DevEncoded definition *DevEncoded VIDEO_IMAGE*
video_source	rw	DevString	The source for video image, BASE_IMAGE (raw image) or LAST_IMAGE (after soft operation) Only valid with monochrome or scientific cameras
video_last_image_counter	rw	DevLong64	The image counter
		SHARED MEMORY	
shared_memory_names	rw	DevString[2]	Firstname and surname of the SPS typed shared memory (default is LimaCCDs,<camera_type>)
shared_memory_active	rw		Activate or not the shared memory. The shared memory is for image display
		CONFIG	
config_available_module	ro	DevString[]	List of possible config modules,
config_available_name	ro	DevString[]	List of existing config names
		BUFFER	
buffer_max_memory	rw	DevShort	The maximum among of memory in percent of the available RAM that Lima is using to allocate frame buffer.
		PLUGIN	
plugin_type_list	ro	DevString[]	List of the available plugin type, to get one device name use instead the **getPluginDeviceNameFromType** command

continues on next page

Table 1 – continued from previous page

Attribute name	RW	Type	Description
plugin_list	ro	DevString[]	List of the available plugin as couple of type, device name

DevEncoded DATA_ARRAY

The DATA_ARRAY DevEncoded has been invented for special Tango client like SPEC. It is used by the **readImage** command. It can only embed raw data (no video data). The supported image format can be retrieve with the **image_type** attribute (Bpp8,Bpp8S, ..., Bpp16,..) This encoded format is very generic and it supports many different type of data from scalar to image stack (see DataArrayCategory enumerate C-type). The readImage command only supports *Image* data array category.

The DATA_ARRAY format is composed of a fixed header followed by the raw data. The header is a C-like structure, with **little-endian** byte order and no alignment:

```
# The DATA_ARRAY definition
struct {
    unsigned int       magic= 0x44544159; // magic key
    unsigned short     version;           // version, only 1 supported
    unsigned  short    header_size;       // size of the header
    DataArrayCategory  category;          // data array category, see␣
↪DataArrayCategory enumerate
    DataArrayType      data_type;         // data type, see DataArrayType enumerate
    unsigned short     endianness;        // 0-little-endian, 1-big-endian
    unsigned short     nb_dim;            // number of dimension (0 to 7 max)e.g 2␣
↪for image
    unsigned short     dim[8];            // size for each dimension, e.g [width,
↪height]
    unsigned int       dim_step[8];       // step size in pixel for each dimension, e.
↪g [1,height]
} DATA_ARRAY_STRUCT;

enum DataArrayCategory {
    ScalarStack = 0;
    Spectrum;
    Image;
    SpectrumStack;
    ImageStack;
};

enum DataArrayType{
    DARRAY_UINT8 = 0;
    DARRAY_UINT16;
    DARRAY_UINT32;
    DARRAY_UINT64;
    DARRAY_INT8;
    DARRAY_INT16;
    DARRAY_INT32;
    DARRAY_INT64;
    DARRAY_FLOAT32;
    DARRAY_FLOAT64;
};
```

DevEncoded VIDEO_IMAGE

The VIDEO_IMAGE DevEncoded has been implemented for the **video_last_image** attribute to return the last image. It can embed any of the supported video format depending of the **video_mode** attribute value.

The VIDEO_IMAGE format is composed of a fixed header followed by the data. The header is a C-like structure, with **big-endian** byte order and no alignment:

```
struct {
    unsigned int     magic_number = 0x5644454f;
    unsigned short   version;        // only version 1 is supported
    unsigned short   image_mode;     // Y8,Y16,....
    long     long    frame_number;   // the frame number (counter)
    int              width;          // the frame width in pixel (horizontal size)
    int              height          // the frame height in pixel (vertical size)
    unsigned short   endianness;     // 0-little-endian, 1-big-endian
    unsigned short   header_size;    // this header size in byte
    unsigned short   padding[2];     // 4 bytes of padding (for alignment)
} VIDEO_IMAGE_STRUCT;
```

9.2 Camera devices

Each camera has a configuration device with its own property/attribute/command lists. The camera configuration device is supposed to give you access to the "private" parameters of the detector that LIMA does not need but you may want to set. For instance some detectors provides a temperature control with set-points and/or start/stop commands for a auxillary cooling system.

For more details about the camera device interface, please have a look on the following sections:

9.2.1 Andor Tango device

This is the reference documentation of the Andor Tango device.

you can also find some useful information about prerequisite/installation/configuration/compilation in the *Andor camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
adc_speed	No	max.	The adc/Horiz. speed pair
baseline_clamp	No	Off	Clamping for baseline threshold, ON or OFF
camera_number	No	N/A	The camera number, default is 0
cooler	No	Off	Start/stop the cooling system of the camera mode
config_path	No	N/A	The configuration path, for linux default is /usr/local/etc/andor
fast_ext_trigger	No	Off	Fast external trigger mode, see Andor documentation for usage
fan_mode	No	N/A	FAN mode, FAN_ON_FULL/FAN_ON_LOW/FAN_OFF
high_capacity	No	High_capacity	Camera can run in two modes, HIGH_CAPACITY or HIGH_SENSITIVITY
p_gain	No	max.	The preamplifier gain [X1-Xn] (see detector spec.)
shutter_level	No	High	The shutter output level mode
temperature_sp	No	N/A	The temperature setpoint in Celsius
vs_speed	No	fasten	The vertical shift speed (see detector spec.)

Attributes

Attribute name	RW	Type	Description
adc_speed	rw	DevString	The ADC and Horizontal shift speed, in ADC-channel/Freq.Mhz, check the documentatio for more help **(*)**
baseline_clamp	rw	DevString	**The baseline clamping for threshold: (**)** • **ON** • **OFF**
cooler	rw	DevString	**Start/stop the cooling system of the camera** • **ON**, the cooler is started • **OFF**, the cooler is stopped
cooling_status	ro	DevString	The status of the cooling system, tell if the setpoint temperature is reached
fan_mode	rw	DevString	**The FAN mode for extra-cooling: (**)** • **FAN_OFF** • **FAN_ON_FULL** • **FAN_ON_LOW**
fast_ext_trigger	rw	DevString	**Fast external trigger mode, see Andor docu** • **ON**, fast mode, the camera will not wait until the a keep clean cycle has been completed before accepting the next trigger • **OFF**, slow mode
high_capacity	rw	DevString	**Off/On the High Capacity mode: (**)** • **HIGH_CAPACITY** • **HIGH_SENSITIVITY**
p_gain	rw	DevString	The preamplifier gain

(*) Use the command getAttrStringValueList to get the list of the supported value for these attributes.

(**) These attributes can not be supported by some camera models and the return value will be set to **UNSUPPORTED**.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrStringValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.2 Basler Tango device

This is the reference documentation of the Basler Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Basler camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
camera_id	No	uname://<*server instance name*>	The camera ID (see details below)
packet_size	No	8000	the packet size
inter_packet_delay	No	0	The inter packet delay
frame_transmission_delay	No	0	The frame transmission delay

camera_id property identifies the camera in the network. Several types of ID might be given:

- IP/hostname (examples: *ip://192.168.5.2*, *ip://white_beam_viewer1.esrf.fr*)
- Basler serial number (example: *sn://12345678*)
- Basler user name (example: *uname://white_beam_viewer1*)

If no *camera_id* is given, it uses the server instance name as the camera user name (example, if your server is called *LimaCCDs/white_beam_viewer1*, the default value for *camera_id* will be *uname://white_beam_viewer1*).

To maintain backward compatibility, the old *cam_ip_address* is still supported but is considered deprecated and might disappear in the future.

Both inter_packet_delay and frame_tranmission_delay properties can be used to tune the GiGE performance, for more information on how to configure a GiGE Basler camera please refer to the Basler documentation.

Attributes

This camera device has not attribute.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrString-ValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.3 Dexela Tango device

This is the reference documentation of the Dexela Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Dexela camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
database_path	Yes	DexelaConfig.cfg	The database path file, e.g C:DexelaConfig.cfg
sensor_format	Yes	sensor2923	The detector model

Attributes

Attribute name	RW	Type	Description
full_well_mode	ro	DevString	The well-mode, can be set to HIGH or LOW

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrString-ValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.4 Frelon Tango device

This is the reference documentation of the Frelon Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Frelon camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
espia_dev_nb	No	0	The acquisition Espia board number

Attributes

Attribute name	RW	Type	Description
espia_dev_nb	ro	DevString	The Espia board number.
image_mode	rw	DevString	**The acquisition image mode:** • **Frame transfert** • **Full frame**
input_channel	rw	DevString	**The Inputs ADC channels:** • **1** • **2** • **3** • **4** • **1-2** • **3-4** • **1-3** • **2-4** • **1-2-3-4**
e2v_correction	rw	DevString	**Active/Desactive the corrstion for e2v came** • **On** • **Off**
roi_mode	rw	DevString	**The roi mode:** • **None** • **Slow** • **Fast** • **Kinetic**
roi_bin_offset	rw	DevLong	The roi offset in line
spb2_config	rw	DevString	The internal config for pixel rate, **precision** or **speed**. Depending on your camera model, the pixel rates are factory defined
seq_status	ro	DevLong	

Please refer to the *Frelon User's Guide* for more information about the above specfic configuration parameters.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrString-ValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name
execSerial-Command	DevString command	DevString command result	Send a command through the serial line
resetLink	DevVoid	DevVoid	reset the espia link

9.2.5 ImXPAD Tango device

This is the reference documentation of the ImXPAD Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *ImXPAD camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
camera_ip_address	Yes	N/A	IP address
port	No	3456	socket port number
model	No	XPAD_S70	detector model
usb_device_id	No	N/A	reserved, do not use
config_path	Yes	N/A	The configuration directory path (see loadConfig command)

Attributes

This camera device has no attribute.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrString-ValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name
loadConfig	DevString	DevVoid	the config file prefix, the property config_path is mandatory

9.2.6 Basler Tango device

This is the reference documentation of the Basler Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Marccd camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
camera_ip	Yes	n/a	The camera hostname or ip address
port_number	Yes	n/a	Socket port number
image_path	Yes	n/a	The inter packet delay

Attributes

Attribute name	RW	Type	Description
source_beam_x	rw	DevFloat	.
source_beam_y	rw	DevFloat	.
source_distance	rw	DevFloat	.
source_wavelength	rw	DevFloat	.
header_beam_x	ro	DevFloat	.
header_beam_y	ro	DevFloat	.
header_distance	ro	DevFloat	.
header_pixelsize_x	ro	DevFloat	.
header_pixelsize_y	ro	DevFloat	.
heaer_integration_time	ro	DevFloat	.
header_exposure_time	ro	DevFloat	.
header_readout_time	ro	DevFloat	.
header_wavelength	ro	DevFloat	.
header_acquire_timestamp	ro	DevFloat	.
header_header_timestamp	ro	DevFloat	.
header_save_timestamp	ro	DevFloat	.
header_mean_bias	ro	DevFloat	.
header_mean	ro	DevFloat	.
header_rms	ro	DevFloat	.
header_temperature	ro	DevFloat[9]	.
header_pressure	ro	DevFloat[9]	.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string

9.2.7 Maxipix Tango device

This is the reference documentation of the Maxipix Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Maxipix camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
config_name	Yes	N/A	The configuration name
config_path	Yes	N/A	The configuration directory path where the files are available
espia_dev_nb	No	0	The acquisition Espia board number
reconstruction_active	No	True	Activate the reconstruction or not
fill_mode	No	Raw	**the chip-gap filling mode, Raw, Zero, Dispatch** or **Mean**.
gate_level	No	High_Rise	The Input gate level, **High_rise** or **Low_Fall**
gate_mode	No	Inactive	The gate mode, **Inactive** or **Active**
ready_level	No	High_Rise	The output ready level, **High_rise** or **Low_Fall**
ready_mode	No	Exposure	The output Ready mode, **Exposure** or **Exposure_Readout**
shutter_level	No	High_Rise	The output Shutter level, **High_rise** or **Low_Fall**
trigger_level	No	High_Rise	The output Trigger level, **High_rise** or **Low_Fall**

Attributes

Attribute name	RW	Type	Description
config_name	rw	DevString	the configuration name. If changed the detector is re-configured and reset.
config_path	rw	DevString	the configuration directory path where the files are available
energy_calibration	rw	Spectrum DevDouble	The energy calibration, [0] = threshold setpoint , [1] threshold step-size (keV)
energy_threshold	rw	DevDouble	The threshold in energy (keV)
threshold	rw	DevDouble	The detector threshold
threshold_noise	rw	Spectrum DevDouble	The threshold noise of each chip, [0] =chip0 thl, [0] = chip1 thl, ...
espia_dev_nb	rw	DevString	The Espia board number.
fill_mode	rw	DevString	**The chip-gap filling mode:** • **Raw**, the border pixel values are copied • **Zero**, border and gap pixel are set to zero • **Dispatch**, the border pixel values are interpolated over the full gap • **Mean**, the gap pixels are filled with the border pixels average value.
gate_level	rw	DevString	**The Input gate level:** • **High_rise** • **Low_Fall**
gate_mode	rw	DevString	**The gate mode:** • **Inactive** • **Active**
ready_mode	rw	DevString	**The output Ready mode:** • **Exposure** • **Exposure_Readout**
shutter_level	rw	DevString	**The output Shutter level**

Warning: we recommend to not change the DAC register values (dac_name and dac_value attributes) excepted if you well know what you are doing, if you have some troubles with the detector please contact the ESRF Detector Unit first.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrString-ValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.8 Merlin Tango device

This is the reference documentation of the Merlin Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Merlin camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
HostName	Yes	none	The detector IP address
CmdPort	No	6431	The tcp command port
DataPort	No	6432	The tcp data port
ImageWidth	No	512	The number of detector pixels
ImageHeight	No	512	The number of detector rasters
Chips	No	4	The number of detector medipix3 chips
Simulate	No	0	Command simulation mode

Attributes

Attribute name	RW	Type	Description
acqRunning	ro	DevBoolean	Is acquisition active
chargeSumming	rw	DevString	Charge Summming mode (**ON/OFF**)
colourMode	rw	DevString	Colour mode (**MONOCHROME/COLOUR**)
continuousRW	rw	DevString	Continuous Collection (**ON/OFF**)
counter	rw	DevString	Counter (**COUNTER0/COUNTER1/BOTH**)
depth	rw	DevString	Counter depth (**BPP1/BPP6/BPP12/BPP24**)
fileDirectory	rw	DevString	Directory name if saving on Merlin PC
fileEnable	rw	DevString	Enable file saving to Merlin PC (**ON/OFF**)
fileName	rw	DevString	Filename if saving on Merlin PC
gain	rw	DevString	Gain Settings (**SHGM/HGM/LGM/SLGM**)
operatingEnergy	rw	DevFloat	Energy keV (0 < e < 999.99)
softwareVersion	ro	DevFloat	Software version number

Table 2 – continued from previou

Attribute name	RW	Type	Description
temperature	ro	DevFloat	Temperature degrees C
threshold0	rw	DevFloat	Threshold 0 keV (0 < th < 999.99)
threshold1	rw	DevFloat	Threshold 1 keV (0 < th < 999.99)
threshold2	rw	DevFloat	Threshold 2 keV (0 < th < 999.99)
threshold3	rw	DevFloat	Threshold 3 keV (0 < th < 999.99)
threshold4	rw	DevFloat	Threshold 4 keV (0 < th < 999.99)
threshold5	rw	DevFloat	Threshold 5 keV (0 < th < 999.99)
threshold6	rw	DevFloat	Threshold 6 keV (0 < th < 999.99)
threshold7	rw	DevFloat	Threshold 7 keV (0 < th < 999.99)
triggerStartType	rw	DevString	Trigger start mode (**INTERNAL/RISING_EDGE_TTL/FALLING_EDGE_TTL/R**
triggerStopType	rw	DevString	Trigger stop mode (**INTERNAL/RISING_EDGE_TTL/FALLING_EDGE_TTL/R**
triggerOutTTL	rw	DevString	TTL Trigger stop mode (**TTL/LVDS/TTL_DELAYED/LVDS_DELAYED/FOLLO**
triggerOutLVDS	rw	DevString	LVDS Trigger stop mode (**TTL/LVDS/TTL_DELAYED/LVDS_DELAYED/FOLL**
triggerOutTTLInvert	rw	DevString	TTL Trigger invert mode (**NORMAL/INVERTED**)
triggerOutLVDSInvert	rw	DevString	LVDS Trigger invert mode (**NORMAL/INVERTED**)
triggerOutTTLDelay	rw	DevLong64	TTL Trigger delay ns (0 < del < 68719476720)
triggerOutLVDSDelay	rw	DevLong64	LVDS Trigger delay ns (0 < del < 68719476720)
triggerUseDelay	rw	DevString	Use Trigger delay (**ON/OFF**)
thScanNum	rw	DevLong	Threshold number to scan (0 < n < 7)
thStart	rw	DevFloat	Threshold scan start energy keV (0 < e < 999.99)
thStep	rw	DevFloat	Threshold scan step energy keV (0 < e < 999.99)
thStop	rw	DevFloat	Threshold scan stop energy keV (0 < e < 999.99)

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
SoftTrigger	DevVoid	DevVoid	Perform soft trigger
Abort	DevVoid	DevVoid	Abort
THScan	DevVoid	DevVoid	Perform threshold scan
ResetHW	DevVoid	DevVoid	Reset

9.2.9 Eiger Tango device

This is the reference documentation of the Dectris Eiger Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Dectris Eiger camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
detector_ip_address	Yes	N/A	The ip address or the hostname of the detector computer interface

Attributes

Attribute name	RW	Type	Description
auto_summation	rw	DevString	If enable image depth is bpp32 and, if not image depth is bpp16 **(*)**
countrate_correction	rw	DevString	Enable or disable the countrate correction **(*)**
efficency_correction	rw	DevString	Enable the efficienty correction
flatfield_correction	rw	DevString	Enable or disable the internal (vs. lima) flatfield correction **(*)**
humidity	ro	DevFloat	Return the humidity percentage
pixel_mask	rw	DevString	Enable or disable the pixel mask correction **(*)**
photon_energy	rw	DevFloat	The photon energy,it should be set to the incoming beam energy. Actually it's an helper which set the threshold
threshold_energy	rw	DevFloat	The threshold energy, it will set the camera detection threshold. This should be set between 50 to 60 % of the incoming beam energy.
temperature	ro	DevFloat	The sensor temperature
virtual_pixel_correction	rw	DevString	Enable or disable the virtual-pixel correction **(*)**

(*) These attributes can take as value **ON** or **OFF**. Please refer to the Dectris documention for more information regarding the online corrections.

Commands

Command name	Arg. in	Arg. out	Description
deleteMemory-Files	DevVoid	DevVoid	To remove the temporary mem. files
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrString-ValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.10 Mythen3 Tango device

This is the reference documentation of the Mythen3 Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Xspress3 camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
HostName	Yes		The Mythen detector socket server IP address
TcpPort	No	1031	The tcp communication port.
Simulate	No	0	Command simulation mode.

Attributes

Attribute name	RW	Type	Description
acqRunning	ro	DevBoolean	Is acquisition active
assemblyDate	ro	DevString	Assembly date of the Mythen system
badChannelInterpolation	rw	DevString	Enable/Disable Bad Channel Interpolation Mode (**ON/OFF**)
badChannels	ro	DevLong[1280*Nb]	Display state of each channel for each active module [Nb = nbModules]
commandID	ro	DevLong	Command identifier (increases by 1)
continuousTrigger	rw	DevString	Enable/Disable continuous trigger mode (**ON/OFF**)
cutoff	ro	DevLong	Count value before flatfield correction
delayBeforeFrame	rw	DevLong64	Time delay between trigger & start (100ns increments)
energy	rw	DevFloat[Nb]	X-ray Energy (4.09 < e keV < 40) [Nb = nbModules]
energyMax	ro	DevFloat	Maximum X-ray Energy keV
energyMin	ro	DevFloat	Minimum X-ray Energy keV
flatField	ro	DevLong[1280*Nb]	Flat field correction values
flatFieldCorrection	rw	DevString	Enable/Disable Flat Field Correction Mode (**ON/OFF**)
gateMode	rw	DevString	Enable/Disable gate mode (**ON/OFF**)
gates	rw	DevLong	Number of gates per frame
hwStatus	ro	DevString	The hardware status
inputSignalPolarity	rw	DevString	Input Signal Polarity (**RISING_EDGE/FALLING_EDGE**)
kthresh	ro	DevFloat[Nb]	Threshold Energy (4.0 < e keV < 20) [Nb = nbModules]

continues on next page

Table 3 – continued from previous page

Attribute name	RW	Type	Description
kthreshEnergy	w	DevFloat[2]	Threshold & Energy keV
kthreshMax	ro	DevFloat	Maximum Threshold Energy keV
kthreshMin	ro	DevFloat	Minimum Threshold Energy keV
maxNbModules	ro	DevLong	Maximum nos. of Mythen modules
module	rw	DevLong	Number of selected module (-1 = all)
nbits	rw	DevString	Number of bits to readout (**BPP24/BPP16/BPP8/BPP4**)
nbModules	rw	DevLong	Number of modules in the system
outputSignalPolarity	rw	DevString	Output Signal Polarity (**RISING_EDGE/FALLING_EDGE**)
predefinedSettings	w	DevString	Load predefined energy/kthresh settings (**Cu/Ag/Mo/Cr**)
rateCorrection	rw	DevString	Enable/Disable rate correction mode (**ON/OFF**)
sensorMaterial	ro	DevLong	The sensor material (0=silicon)
sensorThickness	ro	DevLong	The sensor thickness um
serialNumbers	ro	DevLong[Nb]	Serial nos. of Mythen modules [Nb = nbModules]
systemNum	ro	DevLong	The serial number of the Mythen
tau	rw	DevFloat[Nb]	Dead time constants for rate correction [Nb = nbModules]
testPattern	ro	DevLong[1280*Nb]	Read back a test pattern
triggered	rw	DevString	Enable/Disable triggered mode (**ON/OFF**)
useRawReadout	rw	DevString	Raw readout packed Mode (**ON/OFF**)
version	ro	DevString	The software version of the socket server

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
LogStart	DevVoid	DevVoid	Start logging server activity (use sparingly)
LogStop	DevVoid	DevVoid	Stop logging server activity
LogRead	DevVoid	DevVoid	Print logging file to terminal
ReadFrame	DevLong	DevVarULongArray	[in] frame number [out] a frame of mythen data
ReadData	DevVoid	DevVarULongArray	[out] all frames of mythen data
ResetMythen	DevVoid	DevVoid	Reset

9.2.11 Pilatus Tango device

This is the reference documentation of the Pilatus Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Pilatus camera plugin* section.

Properties

This camera device has no property.

Property name	Mandatory	Default value	Description
TmpfsSize	No	0	OBSOLETE

Attributes

Attribute name	RW	Type	Description
threshold_gain	rw	DevString	The detector threshold gain (**LOW,MID,HIGH,ULTRA HIGH**)
fill_mode	rw	DevString	The gap fill mode (**ON,OFF**)
threshold	rw	DevLong	The threshold level of detector in eV
energy_threshold	rw	DevFloat	The energy threshold in keV (set the gain and the threshold)
trigger_delay	rw	DevDouble	The start exposure delay after the hard trigger
nb_exposure_per_frame	rw	DevLong	The number of exposure/frame to set an accumulation of frames Very useful to not saturate the pixel counters.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrStringValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.12 PCO Tango device

This is the reference documentation of the PCO Tango device.

You can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *PCO camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
debug_control	No	0	Enable/Disble the debug (0/1)
debug_module	No	0	**To set the debug module list (in hex format** • None = 0x001 • Common = 0x002 • Hardware = 0x004 • HardwareSerial = 0x008 • Control = 0x010 • Espia = 0x020 • EspiaSerial = 0x040 • Focla = 0x080 • Camera = 0x100 • CameraCom = 0x200 • Test = 0x400 • Application = 0x800
debug_format	No	0	**To set the debug format (in hex format 0x..** • DateTime = 0x001 • Thread = 0x002 • Module = 0x004 • Obj = 0x008 • Funct = 0x010 • FileLine = 0x020 • Type = 0x040 • Indent = 0x080 • Color = 0x100
debug_type	No	0	**To set the debug type (in hex format 0x....)** • Fatal = 0x001 • Error = 0x002 • Warning = 0x004 • Trace = 0x008 • Funct = 0x010 • Param = 0x020 • Return = 0x040

Attributes

Attribute name	RW	Type	Description
acqTimeoutRetry	rw	DevLong	Maximum Timeout retries during acq (0 - infinite)
adc	rw	DevLong	Number of working ADC's
adcMax	ro	DevLong	Maximum number of ADC's
binInfo	ro	DevLong	PCO hw binning info
bitAlignment	rw	DevString	**Bit alignment** • MSB (0) • LSB (1)
bytesPerPixel	ro	DevLong	Bytes per Pixel
camerasFound	ro	DevString	List of cameras found during the Open search
camInfo	ro	DevString	General camera parameters information
camName	ro	DevString	Camera Name
camNameBase	ro	DevString	Camera Name (Pco)
camNameEx	ro	DevString	Camera Name, Interface, Sensor
camType	ro	DevString	Camera Type
cdiMode	rw	DevLong	**Correlated Double Imaging Mode** • enabled/disabled = 1/0 (rw) • not allowed = -1 (ro)
clXferPar	ro	DevString	General CameraLink parameters
cocRunTime	ro	DevDouble	cocRunTime (s) - only valid after the camera is armed
coolingTemperature	ro	DevDouble	Cooling Temperature
debugInt	rw	DevString	PCO plugin internal debug level (hex format: 0x....)
debugIntTypes	ro	DevString	PCO plugin internal debug types

continues on next page

Table 4 – continued from previous page

Attribute name	RW	Type	Description
doubleImageMode	rw	DevLong	**Double Image Mode** • enabled/disabled = 1/0 (rw) • not allowed = -1 (ro)
firmwareInfo	ro	DevString	Firmware info
frameRate	ro	DevDouble	Framerate, calculated as: 1/cocRunTime (1/s)
generalCAPS1	ro	DevString	General PCO CAPS1 value (hex and bin)
info	ro	DevString	General camera parameters information
lastError	ro	DevString	The last PCO error message
lastImgAcquired	ro	DevLong	Last image acquired (during recording)
lastImgRecorded	ro	DevLong	Last image recorded (during recording)
logMsg	ro	DevString	Last Log msgs
logPcoEnabled	ro	DevLong	PCO logs are enabled
maxNbImages	ro	DevLong	The maximum number of images which can be acquired by the camera (recording mode)
paramsInfo	ro	DevString	Values of the PCO properties **params**
pixelRate	ro	DevLong	Actual Pixel Rate (Hz)
pixelRateInfo	ro	DevString	Pixel Rate information
pixelRateValidValues	ro	DevString	Allowed Pixel Rates
recorderForcedFifo	rw	DevLong	Forced Fifo Mode (**only for recording cams**)
roiInfo	ro	DevString	PCO ROI info
roiLastFixed	ro	DevString	Last fixed ROI info
rollingShutter	rw	DevLong	**Rolling Shutter Mode as int (only for some** • 1 = ROLLING • 2 = GLOBAL • 4 = GLOBAL RESET
rollingShutterInfo	ro	DevString	Rolling Shutter info
rollingShutterStr	rw	DevLong	Rolling Shutter Mode as str (**only for some types of EDGE**)
temperatureInfo	ro	DevString	Temperature info

continues on next page

Table 4 – continued from previous page

Attribute name	RW	Type	Description
test	rw	DevString	Debug test function (**do not use it**)
timestampMode	rw	DevLong	**Timestamp mode** • 0 = none • 1 = BCD coded stamp in the first 14 pixel • 2 = BCD coded stamp in the first 14 pixel + ASCII text • 3 = ASCII text (**only for some cameras**)
traceAcq	ro	DevString	Debug information for some types of acq
version	ro	DevString	Version information of the plugin
versionAtt	ro	DevString	Version of att file
versionSdk	ro	DevString	PCO SDK Release

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do NOT use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrStringValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name
talk	DevString	DevString	**WARNING**: use this command for test only, This is a backdoor cmd and it can distrub Lima

9.2.13 PerkinElmer Tango device

This is the reference documentation of the PerkinElmer Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *PerkinElmer camera plugin* section.

Properties

This device has no property.

Attributes

Attribute name	RW	Type	Description
correction_mode	rw	DevString	'NO', 'OFFSET ONLY' or 'OFFSET AND GAIN'
gain	rw	DevLong	The gain value, from 0 to 63
keep_first_image	rw	DevString	'YES' or 'NO', you can decide to trash the 1st image

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrStringValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name
startAcqOffsetImage	DevVarDoubleArray: nb_frames, exposure_time	DevVoid	Start acquisition for an offset calibration
startAcqGainImage	DevVarDoubleArray: nb_frames, exposure_time	DevVoid	Start an acquisition for an gain calibration

9.2.14 Pixirad Tango device

This is the reference documentation of the Pixirad Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Pixirad camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
ip_address	Yes	N/A	The ip address or the hostname of the detector computer interface
port_number	No	6666	The port number for detector (DAQ commmands)
initial_model	No	PX8	Model type PX1, PX2 or PX8

Attributes

Attribute name	RW	Type	Description
high_threshold0	rw	DevDouble	High Energy threshold 0 (KeV)
low_threshold0	rw	DevDouble	Low Energy threshold 0 (KeV)
high_threshold1	rw	DevDouble	High Energy threshold 1 (KeV)
low_threshold1	rw	DevDouble	Low Energy threshold 1 (KeV)
dead_time_free_mode	rw	DevString	**Enable or disable the free mode dead-time:** • **DEAD_TIME_FREE_MODE_C** • **DEAD_TIME_FREE_MODE_C**
cooling_temperature_setpoint	rw	DevDouble	Cooling temperature set-point for the peltier module of the detector
high_voltage_biais	rw	DevDouble	Bias tension for the high voltage in manual mode
high_voltage_delay_before_ac	rw	DevDouble	Delay for the hv before acquisition
h_v_refresh_period	rw	DevShort	How many image before hv is reset
delay_between_frames	rw	DevShort	Delay between frame in loop acquisition (millisecond)
color_mode	rw	DevString	**Color mode:** • **COLMODE_1COL0** • **COLMODE_2COL** • **COLMODE_1COL1** • **COLMODE_DTF** • **COLMODE_4COL**
sensor_config_build	rw	DevString	**The configuration build:** • **PX1** • **PX2** • **PX8**
trsf_mode	rw	DevString	**Moderated or unmoderated udp transport,** • **UMOD** • **UNMODH** • **MOD**
h_v_bias_mode_power	rw	DevBoolean	Enable (True) or disable (False) the high voltage
hybrid_mode	rw	DevString	**CDTF or GAAS**

Please refer to the Pixirad documention for more information on parameter meanings.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrString-ValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.15 PhotonicScience Tango device

This is the reference documentation of the PhotonicScience Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *PhotonicScience camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
camera_library_path	Yes	N/A	the path to the camera DLL library file e.g.: ImageStar4022_v2.5imagestar4022control.dll

Attributes

This camera device has no attribute.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrString-ValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.16 PointGrey Tango device

This is the reference documentation of the PointGrey Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *PointGrey camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
camera_serial	Yes	N/A	The serial number of the camera, used to get the connection
packet_size	No	-1	The packet size, in byte
packet_delay	No	-1	The packet inter delay , in us last both parameters can be used to tune the camera GigE bandwidth, please refer to the camera documentation for more information

Attributes

Attribute name	RW	Type	Description
gain	rw	DevDouble	The camera gain factor, in dB
auto_gain	rw	DevBoolean	Auto gain mode can be switched on or off
auto_exp_time	rw	DevBoolean	The camera can be set to auto-exposure mode
auto_frame_mode	rw	DevBoolean	The camera can be set to auto frame rate mode
frame_rate	rw	DevDouble	The frame rate, in fps
packet_size	rw	DevLong	See the corresponding property
packet_delay	rw	DevLong	See the corresponding property
exp_time_range	ro	DevDouble[]	Return the exposure time range (min,max) in ms
gain_range	ro	DevDouble[]	Return the gain range (min,max) in dB
frame_rate_range	ro	DevDouble[]	Return the frame rate range (min,max) in fps

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrStringValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.17 Prosilica Tango device

This is the reference documentation of the Prosilica Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the Prosilica camera plugin section.

Properties

Property name	Mandatory	Default value	Description
cam_ip_address	Yes	N/A	The camera's ip or hostname

Attributes

This device has no attribute.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrStringValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.18 RayonixHs Tango device

This is the reference documentation of the RayonixHs Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *RayonixHs camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
frame_mode	No	single	The frame mode, **single** or **fast_transfer**
frame_trigger_signal_type	No	opto	The frame trigger signal type (input #1)
sequence_gate_signal_type	No	opto	The gate signal type (input #2)
electronic_shutter_enabled	No	false	The electronic shutter **true** or **false** to activate or not
cooler_temperature_setpoint	No	-120	The cooling system temperature setpoint in Celsuis
sensor_temperature_setpoint	No	-80	The detector (sensor) temperature setpoint in Celsuis
output1_signal_type	No	cmos	The output #1 signal type
output2_signal_type	No	cmos	The output #2 signal type
output1_id	No	shutter	The output #1 signal source
output2_id	No	frame	the output #2 signal source

The Rayonix HS input/output system supports different type of signals:

- OPTO/OPTO_INVERTED/CMOS/CMOS_PULLDOWN/CMOS_PULLUP/CMOS_PULLDOWN_INVERTED/CMOS_P

And it provides a output multiplexer for both outputs within the following list of sources:

- SHUTTER/INTEGRATE/FRAME/LINE/SHUTTER_OPENING/SHUTTER_CLOSING/SHUTTER_ACTIVE/TRIGGER_

Attributes

Attribute name	RW	Type	Description
frame_mode	rw	DevString	The frame mode, **single** or **fast_transfer**
frame_trigger_signal_type	rw	DevString	The frame trigger signal type (input #1)
sequence_gate_signal_type	rw	DevString	The gate signal type (input #2)
electronic_shutter_enabled	rw	DevString	The electronic shutter **true** or **false** to activate or not
cooler_temperature_setpoint	rw	DevDouble	The cooling system temperature setpoint in Celsuis
sensor_temperature_setpoint	rw	DevDouble	The detector (sensor) temperature setpoint in Celsuis
output1_signal_type	rw	DevString	The output #1 signal type
output2_signal_type	rw	DevString	The output #2 signal type
output1_id	rw	DevString	The output #1 signal source
output2_id	rw	DevString	The output #2 signal source
vacuum_valve	rw	DevString	The vacuum valve command **true** or **false** to open or close

Warning: be careful with the temperature setting (and vacuum valve), the operating temperature is factory-determined and should never be changed. There is no reason to run the detector at a warner temperature.

For the signal type and source the possible values are listed above in the *Properties* section.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrString-ValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.19 Simulator Tango device

This is the reference documentation of the Simulator Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Simulator camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
peaks	No	N/A	A gauss peak list [x0,y0,w0,A0,x1,y1,w1,A1...]
peak_angles	No	N/A	The base rotation angle for each peak
fill_type	No	Gauss	The image fill type: Gauss or Diffraction
rotation_axis	No	rotationy	Peak move policy: STATIC, ROTATIONX, ROTATIONY

Attributes

Attribute name	RW	Type	Description
peaks	rw	Spectrum,DevDouble	The gauss peak list [x0,y0,w0,A0,x1,y1,w1,A1...]
peak_angles	rw	Spectrum,DevDouble	The base rotation angle for each peak
grow_factor	rw	DevDouble	The Grow factor for gauss peaks
fill_type	rw	DevString	The image fill type: Gauss or Diffraction
rotation_axis	rw	DevString	The rotation axis policy: Static, RotationX or RotationY
diffraction_pos	rw	Spectrum,DevDouble	The source diplacement position: x and y
diffraction_speed	rw	Spectrum,DevDouble	The source diplacement speed: sx and sy
rotation_angle	rw	DevDouble	The peak rotation angle in deg
rotation_speed	rw	DevDouble	The peak rotation speed in deg/frame

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrString-ValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.20 SlsDetector Tango device

This is the reference documentation of the PSI SlsDetector Tango device.

You can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *SlsDetector camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
config_fname	Yes	•	Path to the SlsDetector config file
apply_corrections	No	True	Perform corrections on each frame
high_voltage	No	0	Initial detector high voltage (V) (set to 150 if already tested)
fixed_clock_div	No	0	Initial detector fixed-clock-div
threshold_energy	No	0	Initial detector threshold energy (eV)
tolerate_lost_packets	No	True	Initial tolerance to lost packets
pixel_depth_cpu_affinity_map	No	[]	Default PixelDepthCPUAffinityMap as Python string(s) defining a dict: {<pixel_depth>: <global_affinity>}, being global_affinity a tuple: (<recv_list>, <lima>, <other>, <netdev_grp_list>), where recv_list is a list of tupples in the form: (<listeners>, <port_threads>), where listeners and port_threads are tuples of affinities, lima and and other are affinities, and netdev_grp_list is a list of tuples in the form: (<comma_separated_netdev_name_list>, <rx_queue_affinity_map>), the latter in the form of: {<queue>: (<irq>, <processing>)}. Each affinity can be expressed by one of the functions: Mask(mask) or CPU(<cpu1>[, ..., <cpuN>]) for independent CPU enumeration

Attributes

Attribute name	RW	Type	Description
config_fname	ro	DevString	Path to the SlsDetector config file
hostname_list	ro	DevVarStringArray	The list of the Eiger half-modules' hostnames
apply_corrections	ro	DevBoolean	Pixel software corrections are applied on each frame
dac_name_list	ro	DevVarStringArray	The list of the DAC signals' names
dac_<signal_name>	rw	DevVarLongArray	Array with the DAC <signal_name> value for each half-module, in A/D units
dac_name_list_mv	ro	DevVarStringArray	The list of the DAC signals' names supporting milli-volt units
dac_<signal_name>_mv	rw	DevVarLongArray	Array with the DAC <signal_name> value for each half-module, in milli-volt units
adc_name_list	ro	DevVarStringArray	The list of the ADC signals' names
adc_<signal_name>	rw	DevVarDoubleArray	Array with the ADC <signal_name> value for each half-module, in user units (deg C, etc.)
pixel_depth	rw	DevString	**The image pixel bit-depth:** • **4** (not implemented in LImA yet) • **8** • **16** • **32**
raw_mode	rw	DevBoolean	Publish image as given by the Receivers (no SW reconstruction)
threshold_energy	rw	DevLong	The energy (in eV) the pixel discriminator thresholds (Vcmp & Trim bits) is set at
high_voltage	rw	DevShort	The detector high voltage (in V)
tx_frame_delay	rw	DevLong	Frame Tx delay (6.2 ns units)
all_trim_bits	rw	DevVarLongArray	Array with the pixel trimming value [0-63] for each half-module, if all the pixels in the half-module have the same trimming value, -1 otherwise
clock_div	rw	DevString	**The readout clock divider:** • **FULL_SPEED** •

Please refer to the *PSI/SLS Eiger User's Manual* for more information about the above specfic configuration parameters.

Note: CPU-affinity control now acts, in a per-pixel_depth basis, on the following execution elements:

- Receiver listener threads
- Receiver writer threads
- Lima control & processing threads
- Other processes in the OS
- Network devices' processing tasks (kernel space)

Network devices can be grouped, each group will have the same CPU-affinity for the processing tasks.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrStringValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name
putCmd	DevString	DevVoid	Command setting a SlsDetector parameter (no response)
getCmd	DevString: get command	DevString: command result	Command getting a SlsDetector parameter (with response)
getNbBadFrames	DevLong: port_idx	DevLong: nb_bad_frames	Get the number of bad frames in the current (or last) acquisition for the given receiver port (-1=all)
getBadFrameList	DevLong: port_idx	DevVarLongArray: bad_frame_list	Get the list of bad frames in the current (or last) acquisition for the given receiver port (-1=all)

9.2.21 Ueye Tango device

This is the reference documentation of the Ueye Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Ueye camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
address	No	0	The video address

Attributes

This device has no attribute.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrString-ValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.22 Ultra Tango device

This is the reference documentation of the Ultra Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Ultra camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
headIpaddress	No	192.168.1.100	The detector head IP address
hostIpaddress	No	192.168.1.103	The host IP address
tcpPort	No	7	The tcp echo port
udpPort	No	5005	The upd port
nPixels	No	512	The number of detector pixels

Attributes

Attribute name	RW	Type	Description
headColdTemp	ro	DevFloat	The head cold temperature in K
heatHotTemp	ro	DevFloat	The head hot temperature in K
tecColdTemp	ro	DevFloat	
tecSupplyVolts	ro	DevFloat	
adcPosSupplyVolts	ro	DevFloat	
adcNegSupplyVolts	ro	DevFloat	
vinPosSupplyVolts	ro	DevFloat	
vinNegSupplyVlots	ro	DevFloat	
headADCVdd	ro	DevFloat	
headVdd	rw	DevFloat	
headVref	rw	DevFloat	
headVrefc	rw	DevFloat	
headVpupref	rw	DevFloat	
headVclamp	rw	DevFloat	

continues on next page

Table 5 – continued from previous page

Attribute name	RW	Type	Description
headVres1	rw	DevFloat	
headVres2	rw	DevFloat	
headVTrip	rw	DevFloat	
fpgaXchipReg	rw	DevULong	
fpgaPwrReg	rw	DevULong	
fpgaSyncReg	rw	DevULong	
fpgaAdcReg	rw	DevULong	
frameCount	ro	DevULong	
frameError	ro	DevULong	
headPowerEnabled	rw	DevBoolean	
tecPowerEnabled	rw	Devboolean	
biasEnabled	rw	Devboolean	
syncEnabled	rw	Devboolean	
calibEnabled	rw	Devboolean	
8pCEnabled	ro	DevBoolean	
tecOverTemp	ro	DevBoolean	
adcOffset	rw	DevFloat[16]	
adcGain	rw	DevFloat[16]	
aux1	rw	DevULong[2]	
aux2	rw	DevULong[2]	
xchipTiming	rw	DevULong[9]	

Please refer to the manufacturer's documentation for more information about the above listed parameters and how to use them.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrStringValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name
SaveConfiguration	DevVoid	DevVoid	Save the current configuration
RestoreConfiguration	DevVoid	DevVoid	Restore the latest configuration

9.2.23 V4l2 Tango device

This is the reference documentation of the V4l2 Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *V4l2 camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
video_device	No	/dev/video0	The video device path

Attributes

This device has no attribute.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrString-ValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.24 Xh Tango device

This is the reference documentation of the Xh Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Xh camera plugin* section.

Properties

Property name	Mandatory	Default value	Description
cam_ip_address	Yes	N/A	The detector IP address
port	No	1972	The port number
config_name	No	“config”	The default configuration filename

Attributes

Attribute name	RW	Type	Description
clockmode	wo	DevString	The clockmode, **XhInternalClock**, **XhESRF5468Mhz** or **XhESRF1136Mhz**
nbscans	wr	DevLong	the number of scans for accumulation

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrStringValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name
reset	DevVoid	DevVoid	Perform a hardware reset of the detector
setHeadCaps	DevVarULongArray	DevVoid	Caps for AB, Caps for CD
sendCommand	DevString	DevVoid	Backdoor command to send direct command to the *da.server* server

9.2.25 Xpad Tango device

This is the reference documentation of the Xpad Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Xpad camera plugin* section.

Properties

None.

Attributes

None.

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrStringValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name

9.2.26 Xspress3 Tango device

This is the reference documentation of the Xspress3 Tango device.

you can also find some useful information about the camera models/prerequisite/installation/configuration/compilation in the *Xspress3 camera plugin* section.

test reference to camera plugin section: *ADSC camera*

Properties

Property name	Mandatory	Default value	Description
basIpaddress	No	none	Override the base IP address (e.g. 192.168.0.1) from which all other addresses are calculated or NULL to use the default
basMacAddress	No	none	Override the base MAC address (e.g. 02.00.00.00.00) from which all other card MAC address`s are calculated or NULL to use the default
basePort	No	none	Override the base IP port number or 0 to use the default
createScopeModule	No	False	true = do not create a scope data module
nbFrames	No	1	Number of 4096 energy bin spectra timeframes
scopeModName	No	NULL	The scope data module filename or NULL to use the default
nbCards	No	1	The number of xspress3 cards that constitute the xspress3 system, between 1 and XSP3_MAX_CARDS
nbChans	No	-1	Limit the number of channels
debug	No	0	debug message (0 = off, 1=normal, 2=verbose)
noUDP	No	False	True = do not do UDP connection
cardIndex	No	none	Starting card index
directoryName	No	non	The directory name to save and restore configurations

Attributes

Attribute name	RW	Type	Description
card	rw	DevLong	
numChan	ro	DevLong	
numCards	ro	DevLong	
chansPerCard	ro	DevLong	
maxNumChan	ro	DevLong	
binsPerMca	ro	DevLong	
windows	rw	DevLong[32]	
runMode	rw	DevBoolean[4]	
clocks	rw	Devbooleanp[3]	
goodsThreshold	rw	DevLong[16]	
dtcEnergy	rw	DevDouble	
dtcParameters	rw	DevDouble[48]	
scaling	rw	DevDouble[8]	
fanTemperatures	rw	DevDouble[50]	
fanController	rw	DevDouble[2]	
setPoint	wo	DevDouble	
roi	wo	DevLong[25]	
useDtc	rw	DevBoolean	
setTiming	wo	DevLong	
adcTempLimit	wo	DevLong	
setPlayback	wo	DevBoolean	
playbackfilename	wo	DevString	
dataSource	rw	DevLong[8]	

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
getAttrStringValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name
Reset	DevVoid	DevVoid	
InitBrams	DevLong: channel	DevVoid	
Pause	DevVoid	DevVoid	
Restart	DevVoid	DevVoid	
Arm	DevVoid	DevVoid	
Clear	DevVoid	DevVoid	
SaveSettings	DevVoid	DevVoid	
RestoreSettings	DevBoolean	DevVoid	Force restore if major revision of saved file does not match the firmware revision
InitRois	DevLong: channel	DevVoid	
ReadHistogram	DevVarLongArray: frame, channel	DevVarULongArray:	Return the histogram data
ReadScalers	DevVarLongArray: frame, channel	DevVarULongArray:	Return the scaler data
StartScope	DevVoid	DevVoid	
LoadPlayback	DevVarLongArray: src0,src1, [num_streams, digital]	DevVoid	
FormatRun	DevVarLongArray: chan,[nbits_eng, aux1_mode, adc_bits, min_samples, aux2_mode, pileup_reject	DevVoid	

9.3 Plugin devices: software operation and extra interfaces

User-defined software plugins can be used to execute arbitrary image-based operations. An entry point in the control layer completely exports the ProcessLib functionality, allowing an external code to be called on every frame. The software operation can be implemented in C++ or Python.

The software operations on image are embedded into individual Tango devices and are available in the **plugins/** directory. They are automatically exported by the LimaCCDs server.

The software operations are of two types, *Sink* or *Link* :

- **Link** operation is supposed to modify the frame data, so it gets the frame data as input parameter and it will return a "corrected" image (e.g. Mask/Flatfield/BackgroundSubstraction).
- **Sink** operation is taken the frame data as input parameter to apply some software operation in order to return new data like statistics, peak positions, alarm on saturation . . . etc.

In addition to sink/link plugin device, a plugin can just be implemented to provide/export a subset of the Lima interface or a legacy interface for some specific client applications (e.g SPEC, LimaTacoCCD plugin).

Today there are about 8 standard plugin devices:

- BackgroundSubstraction : link operation, to correct the frames with a background image (substraction)
- FlatField: link operation to correct the frames with a flatfield image (divide + option normalisation)
- Mask: link operation to mask pixels. Very useful if some pixel are not working properly and if you want to set then to a fix value or to zero.
- PeakFinder: thanks to Teresa Numez from DESY, a sink operation which can detect diffraction peaks.
- Roi2Spectrum: sink operation to apply ROI spectrum on the frames. You can define more than one spectra with ROI coordinates and by specifying in which direction you need to bin the values, vertical or horizontal.
- RoiCounter: sink operation to get calculating statistics on image regions.
- LimaTacoCCD: extra interface for TACO clients, it only provides commands (TACO does not have attribute !), it is still used at ESRF for SPEC.
- LiveViewer: extra interface to provide a live view of the last acquired image, can be used from atkpanel.

If you need to implement your own plugin device we can provide you some example codes, use the mailing-list lima@esrf.fr to get help.

9.3.1 Background Substraction

The Background substraction correction is a simple operation you can active when a detector has some dark-current noise independent of the dose of photons it will receive. To set the correction you must provide to the device a background image file (**setBackgroundImage** command) and then start the correction (**start** command). Instead of providing an external image file you can simply ask the device to use an image taken. Call the command **takeNextAcquistionAsBackground** to set the internal background image from an acquisition image. One can apply an extra offset correction using the **offset** attribute value.

Properties

This device has no property.

Attributes

Attribute name	RW	Type	Description
delete_dark_after_read	rw	DevBoolean	If true the device will delete the file after reading Can be useful to not keep obsolete dark image file after use
offset	rw	DevLong	Set a offset level to be applied in addition to the background correction
RunLevel	rw	DevLong	Run level in the processing chain, from 0 to N
State	ro	State	OFF or ON (stopped or started)
Status	ro	DevString	“OFF” “ON” (stopped or started)

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
setBackgroundImage	DevString	DevVoid	Full path of background image file
Start	DevVoid	DevVoid	Start the correction for next image
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
Stop	DevVoid	DevVoid	Stop the correction after the next image
takeNextAcquisitionAsBackground	DevVoid	DevVoid	next taken image will replace the background

9.3.2 Bpm

This is the BPM (Beam Position Monitoring) device. It aims to detect an X-ray beam spot and returns statistics (x,y positions, FWHM, ...). It takes images and calculates the beam position using the builtin task BPM of the processlib library. It can also push Tango event containing jpeg view of the image and several statistics and information (listed bellow) in a DevEncoded attribute name bvdata.

Properties

Propertie name	RW	Type	Description
enable_tango_event	RW	DevBoolean	if set to false, Bpm won't push bvdata or other attributes through Tango.
calibration	RW	DevVarDoubleArray	Contains the calibration in X and Y ([X,Y]), value in unit/pixel. |
beammark	RW	DevVarLongArray	Contains coordinates (X,Y) in pixels of a beam mark set by the user.

Attributes

Attribute name	RW	Type	Description
buffer_size	RW	Dev_Long	Size of the buffer where a certain amount of images will be store before re-writing on the first one.
x	RO	Dev_Double	coordinate on the x axis of the beam return by the BPM task. If the algorithm couldn't find a X value then it is set at -1.
y	RO	Dev_Double	Same as x but for Y axis.
txy	RO	Dev_Double	Return an array [timestamp,x,y] of the last acquisition.
automatic_aoi	RW	Dev_Boolean	true or false for the AOI mode.
intensity	RO	Dev_Double	Intensity of the area around beam.
max_intensity	RO	Dev_Double	Maximum intensity on the image.
proj_x	RO	Dev_Long	Array containing sum of all pixel´s intensity on axis x
proj_y	RO	Dev_Long	Same as proj_x but on y axis.
fwhm_x	RO	Dev_Double	Full width at half of maximum on the profil X.
fwhm_y	RO	Dev_Double	same as fwhm_x but on y axis profil.
autoscale	RW	Dev_Boolean	Activate autoscale transformation on the image. (use min and max intensity on it in order to scale).
lut_method	RW	DevString	Method used in the transformation of image. can be "LOG" or "LINEAR".
color_map	RW	Dev_Boolean	Image in black and white(color_map=false), or use a color map to display colors based on intensity.
bv_data	RO	DevEncoded	Attribute regrouping the image (jpeg format) and numerous information on it, such as timestamp, number of the frame, x, y, txy, … Everything is pack throught struck module and is either send in a Tango event or directly read. WARNING : You need to have the decode function in order to read (can be found in the webserver Bpm, currently here : https://gitlab.esrf.fr/limagroup/bpm-web)
cal_	RW	De_	Attribute version of the calibration property.

Commands

Commands name	Arg.IN	Arg.OUT	Description
Start	DevVoid	DevVoid	Start Bpm device.
Stop	DevVoid	DevVoid	Stop Bpm device.
getResults	DevLong	DevVarDoubleArray	Take a number as parameter and return an array containing (framenb,x,y) values, starting to the frame number ask until there is no more image.
GetPixelIntensity	DevVarLongArray	DevLong	Return the intensity of pixel (x,y) passed as parameters
HasBackground	DevVoid	DevBoolean	Is there a background already in place ?
TakeBackground	DevVoid	DevVoid	Take the current image and set it as Background, using the Core.BACKGROUNDSUBSTRACTION module.
ResetBackground	DevVoid	DevVoid	Reset the Background.

NOTE

This plugin is supposed to replace the old BeamViewer plugin but with limited functionalities for the moment. Some other plugins will be created in the future. This plugin is mainly used in conjunction with the bpm webserver application

9.3.3 FlatField

The flat fied correction can be used to remove artifacts from the images that are caused by variations in the pixel-to-pixel sensitivity of the detector and/or by the distortions in the optical path. Here the correction consists in providing a reference image taken using a uniform photon exposure. Then each raw image will be corrected by dividing the pixel values by their corresponding reference values (flatfield image pixels).

To set the correction you must provide to the device a flatfield image file (**setFlatFieldImage** command) and then start the correction (**start** command).

Properties

This device has no property.

Attributes

Attribute name	RW	Type	Description
RunLevel	rw	DevShort	Run level in the processing chain, from 0 to N
normalize	rw	DevBoolean	If true the flatfield image will be normalized first (using avg signal)
State	ro	State	OFF or ON (stopped or started)
Status	ro	DevString	"OFF" "ON" (stopped or started)

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
setFlatFieldImage	DevString	DevVoid	Full path to flatfield image file
Start	DevVoid	DevVoid	Start the correction for next image
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
Stop	DevVoid	DevVoid	Stop the correction after the next image

9.3.4 Mask

The mask correction is very useful when you have some defective pixels on your detector sensor. Then you can provide a mask image file which can either applies a fixed value for those defective pixel (mask type == **DUMMY**) or sets those pixels to zero count (mask type = **STANDARD**).

To set the correction you must provide to the device a flatfield image file (**setFlatMaskImage** command) and then start the correction (**start** command).

Properties

This device has no property.

Attributes

Attribute name	RW	Type	Description
RunLevel	rw	DevShort	Run level in the processing chain, from 0 to N
type	rw	DevString	**Set the type of mask correction:** • **DUMMY**, replace the pixel value with the mask image pixel value • **STANDARD**, if the mask pixel value is equal to zero set the image pixel value to zero otherwise keep the image pixel value unchanged
State	ro	State	OFF or ON (stopped or started)
Status	ro	DevString	"OFF" "ON" (stopped or started)

Commands

Command name	Arg. in	Arg. out	Description
getAttrStringValueList	DevString: Attribute name	DevVarStringArray: String value list	Return the authorized string value list for a given attribute name
Init	DevVoid	DevVoid	Do not use
setMaskImage	DevString	DevVoid	full path for the mask image file
Start	DevVoid	DevVoid	set the correction active
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
Stop	DevVoid	DevVoid	set the correction inactive

9.3.5 PeakFinder

This is a nice plugin developed at DESY which can find peaks on an image and returns the positions of the peaks.

Once the configuration is ok you can start the task using **Start** command and stop the task calling the **Stop** command.

Properties

This device has no property.

Attributes

Attribute name	RW	Type	Description
BufferSize	rw	DevLong	Circular buffer size in image, default is 128
ComputingMode	rw	DevString	**The computing algorithm :** • **MAXIMUM**, find peak at maximum • **CM**, find peak at center of mass
CounterStatus	ro	DevLong	Counter related to the current number of proceeded images
RunLevel	rw	DevLong	Run level in the processing chain, from 0 to N
State	ro	State	OFF or ON (stopped or started)
Status	ro	DevString	"OFF" "ON" (stopped or started)

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
readPeaks	DevVoid	DevVarDoubleArray frame0,x,y,frame1,..	Return the peaks positions
setMaskFile	DevVarStringArray	DevVoid	Full path of mask file
Start	DevVoid	DevVoid	Start the operation on image
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
Stop	DevVoid	DevVoid	Stop the operation on image

9.3.6 Roi2Spectrum

The Region-of-Interest to Spectrum operation is very useful to provide online integration of some areas of your detector. The integration of the pixel values can set along the Y direction or the X direction. You must create first the Rois by providing unique names (**addNames** command) and then set the Roi position using the index and the x,y, width, height (**setRois** command). The direction for integration (so-called mode) can be set using te **setRoiModes** command. Once the configuration is ok you can start the task using **Start** command and stop the task calling the **Stop** command. The spectrum data can be retrieved by calling the **readImage** command, the command returns the spectrums as a stack stored into an image.

Properties

This device has no property.

Attributes

Attribute name	RW	Type	Description
BufferSize	rw	DevLong	Circular buffer size in image, default is 128
CounterStatus	ro	DevLong	Counter related to the current number of proceeded images
RunLevel	rw	DevLong	Run level in the processing chain, from 0 to N
State	ro	State	OFF or ON (stopped or started)
Status	ro	DevString	“OFF” “ON” (stopped or started)

Commands

Command name	Arg. in	Arg. out	Description
addNames	DevVarStringArray list of Roi names	DevVarStringArray list of Roi indexes	Set the names and return the corresponding indexes
clearAllRois	DevVoid	DevVoid	Remove the Rois
getNames	DevVoid	DevVarStringArray	Return the list of Roi names
getRoiModes	DevVarStringArray	DevVarStringArray	Return the Roi modes
getRois	DevVarStringArray list of Roi names	DevVarStringArray list of Roi position (roi_id,x,y,width,heigth,...)	Return the Roi positions
Init	DevVoid	DevVoid	Do not use
readImage	DevVarLongArray	DevVarLongArray	
removeRois	roi_id,first image	spectrum stack	Return the stack of spectrum from the specified image index until the last image acquired
setRois	DevArLongArray (roi_id,x,y,w,h,...)	DevVoid	Set roi positions
Start	DevVoid	DevVoid	Start the operation on image
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
Stop	DevVoid	DevVoid	Stop the operation on image

9.3.7 RoiCounter

The Region-of-Interest to Counter operation is very useful to provide online statistics on some detector areas. The operation will calculate for each image acquired the **average**, the **standard deviation**, the **sum**, the **minimum** and the **maximum pixel** values.

The Roi can be defined either with rectangle coordinates (x begin,y begin, width, height) or with arc coordinates (center x, center y, radius1, radius2, angle start, angle end). Different commands are provided for that purpose: **setRois** and **setArcRois**.

You must create first the Rois by providing unique names (**addNames** command) and then set the Roi position using the Roi index and the position (rectangle or arc position).

The statistics can be retrieved by calling the **readCounters** command, the command returns a list of statistics per Roi and frame.

In addition to the statistics calculation one can set a mask file (**setMask** command) where null pixel will not be taken into account.

Properties

This device has no property.

Attributes

Attribute name	RW	Type	Description
BufferSize	rw	DevLong	Circular buffer size in image, default is 128
CounterStatus	ro	DevLong	Counter related to the current number of proceeded images
RunLevel	rw	DevLong	Run level in the processing chain, from 0 to N
State	ro	State	OFF or ON (stopped or started)
Status	ro	DevString	"OFF" "ON" (stopped or started)

Commands

Command name	Arg. in	Arg. out	Description
addNames	DevVarStringArray list of Roi names	DevVarStringArray list of Roi indexes	Set the names and return the corresponding indexes
clearAllRois	DevVoid	DevVoid	Remove the Rois
getNames	DevVoid	DevVarStringArray	Return the list of Roi names
getRoiModes	DevVarStringArray	DevVarStringArray	Return the Roi modes
getRois	DevVarStringArray list of Roi names	DevVarStringArray list of Roi position (roi_id,x,y,width,heigth,...)	Return the Roi positions
getArcRois	DevVarStringArray list of ArcRoi names	**DevVarStringArray** list of ArcRoi position (roi_id,x,y,width,heigth,...)	Return the ArcRoi positions
Init	DevVoid	DevVoid	Do not use
readCounters	DevVarLongArray	DevVarLongArray	
removeRois	roi_id,first image	spectrum stack	Return the stack of spectrum from the specified image index until the last image acquired
setArcRois	DevVarDoublArray (roi_id0,centerx,centery, radius1,raduis2,start_angle, end_angle,roi_id1,...)	DevVoid	Set the Arc Rois
setMaskFile	DevVarStringArray full path file	DevVoid	Set the mask file
setRois	DevArLongArray (roi_id0,x,y,w,h,roi_id1..)	DevVoid	Set roi positions
Start	DevVoid	DevVoid	Start the operation on image
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
Stop	DevVoid	DevVoid	Stop the operation on image

9.3.8 LimaTacoCCD

This device has been created by legacy and it provides the only interface that SPEC software is supporting for "ESRF General CCD Dev" CCD-like controller.

Properties

Property name	Mandatory	Default value	Description
ManualAsynchronousWrite	No	False	Flag for manual writting, can improve the performance of data saving

Attributes

This device has no attributes.

Commands

Command name	Arg. in	Arg. out	Description
TacoState	DevVoid	DevLong	Return the device taco-like state
DevCcdStart	DevVoid	DevVoid	Start the acquisition
DevCcdStop	DevVoid	DevVoid	Stop the acquisition
DevCcdRead	DevVarLongArray[2]: frame_nb,frame_size	DevVarCharArray: the raw image	Return the image as a string
DevCcdReadAll	DevLong: frame_size	DevEncoded	Return the concatenated frames in a DevEncoded format DATA_ARRAY (see *DevEncoded DATA_ARRAY*)
DevCcdReadJPeg	DevShort: jpeg compression	DevVarCharArray: Jpeg image	Return a jpeg image
DevCcdWrite	DevVoid	DevVoid	Save the last image
DevCcdSetExposure	DevFloat	DevVoid	Set the exposure time in second
DevCcdGetExposure	DevVoid	DevFloat	Return the exposure time in second
DevCcdSetRoI	DevVarLongArray[4]: startx,endx,starty, endy	DevVoid	Set the new Region-of-Interest
DevCcdGetRoi	DevVoid	DevVarLongArray[4]: startx,endx,starty, endy	Return the last Region-of-Interest
DevCcdSetFilePar	DevStringArray[5]		
DevCcdHeader			
DevCcdImageHeader			
DevCcdHeaderDelimiter			
DevCcdGetFilePar			
DevCcdDepth			
DevCcdYSize			

continues on next page

Table 6 – continued from previous page

Command name	Arg. in	Arg. out	Description
DevCcdXSize			
DevCcdReset			
DevCcdSetMode			
DevCcdGetMode			
DevCcdWriteFile			
DevCcdGetBin			
DevCcdSetBin			
DevCcdSetFrames			
DevCcdGetFrames			
DevCcdSetTrigger			
DevCcdGetTrigger			
DevCcdReadValues			
DevCcdSigValues			
DevCcdGetLstErrMsg			
DevCcdGetCurrent			
DevGetDebugFlags			
DevSetDebugFlags			

9.3.9 LiveViewer

This device was create for backward compatibility with former graphical applications used at ESRF by the diagnostic group for the monitoring of the electron beam. It is no longer maintain. Instead we recommend to use the video API provided via the main device LimaCCDs.

Nevertheless you will find here the of the available properties, attributes and commands.

Properties

Property name	Mandatory	Default value	Description
AcquisitionAutoStart	No	False	If true start the acquistion at device startup

Attributes

Attribute name	rw	Type	Description
Depth	ro	DevShort	Image depth in byte
Exposure	rw	DevDouble	Exposure time in second
ExternalTrigger	rw	DevBoolean	External trigger active if true
FrameRate	rw	DevDouble	Frame rate in fps
Frames	rw	DevLong	Number of frames to acquire
Gain	rw	DevDouble	Gain, support depends on the camera model
Image	ro	Image, DevUShort	The last image taken
ImageCounter	ro	DevLong	The image counter
JpegImage	ro	DevEncoded	The last image in JPEG format, only supported for B/W cameras.
JpegQuality	rw	DevLong	JPEG quality factor from 0 to 10
Roi	rw	DevLong,Spectrum	The Roi position, start x, start y, width, height
State	ro	State	OFF or ON (stopped or started)
Status	ro	DevString	“OFF” “ON” (stopped or started)

Commands

Command name	Arg. in	Arg. out	Description
Init	DevVoid	DevVoid	Do not use
Reset	DevVoid	DevVoid	Reset the camera, factory setting is apply
ResetRoi	DevVoid	DevVoid	Remove the Roi, camera set to full size
Start	DevVoid	DevVoid	Start the camera for live acquisition
State	DevVoid	DevLong	Return the device state
Status	DevVoid	DevString	Return the device state as a string
Stop	DevVoid	DevVoid	Stop the camera live

CHAPTER

TEN

UNDERSTAND THE PLUGIN ARCHITECTURE

10.1 Library structure

The library structure is divided into two main layers: the control, containing the common control and processing code, and the hardware which is implementing the detector-specific part. The control layer provides the library interface to the high level application. User requests to configure and control the acquisition are gathered by the control layer, so the hardware layer functionality is limited to the generation the image frames in a best-effort basis.

The control layer is responsible of:

- Adapting the received image geometry if it does not match the user requests,
- Executing the frame processing chain.

10.2 Generic Interface

The Hardware Layer defines the interface between the Control Layer and the controller library. It provides the minimal functionality needed for the Control Layer to satisfy the user requests. The main class in the Hardware Layer is the *lima::HwInterface*, providing the interface to the Control Layer. In order to provide a flexible and evolvable interface, the configuration of this layer is implemented as a set of features (capabilities) that may or may not be implemented by the hardware.

The capabilities can be grouped in three categories:

1. **Standard.** Includes the synchronization parameters (exposure time, ext. trigger, etc), the detector information (Detector model, Max size, etc..) is considered standard and must be implemented for all detectors.
2. **Extended.** Optional common features like image transformations (binning, RoI, flip), advanced acquisition modes (kinetics, frame transfer), and extended mechanisms (camera serial line)
3. **Specific.** These are detector-specific features that can not be treated in a generic interface

As a camera plugin developer, your mission, should you choose to accept it, will consist in writing the code for the *lima::HwInterface* class and its depending classes (.e.g the capabilities classes).

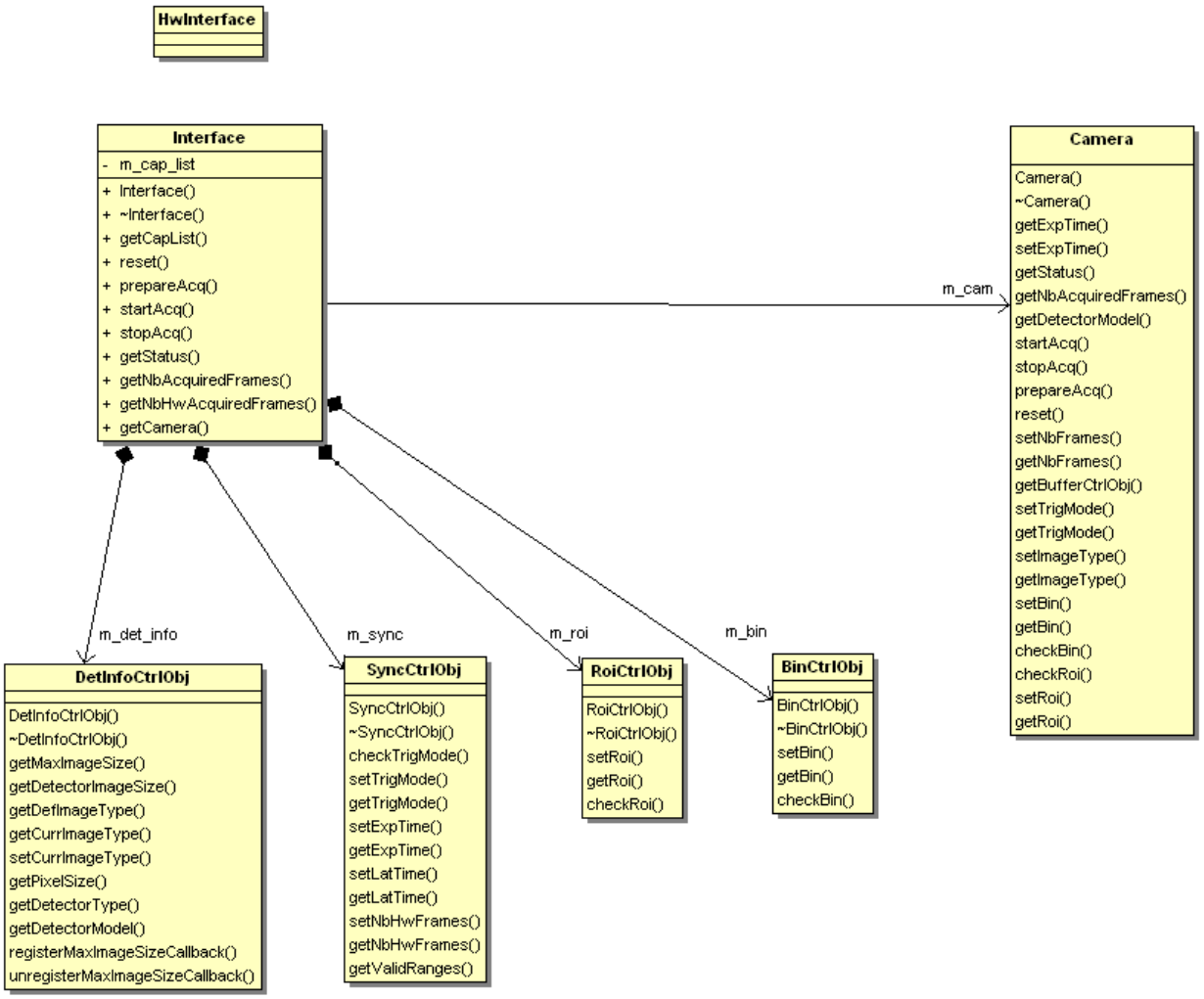


Fig. 1: Figure 1. Class diagram of a camera plugin.

10.3 Hardware Interface

lima::HwInterface is the glue layer between the Control Layer and the camera plugin implementation. It informs LImA about the capabilities provided by the hardware.

class lima::**HwInterface**

As an interface to the Control Layer, this class exports the capabilities provided by the hardware.

It is implemented by every camera plugins.

Public Functions

void **getCapList**(CapList&) **const** = 0

Returns a list of capabilities.

void **reset**(ResetLevel *reset_level*) = 0

Reset the hardware interface.

void **prepareAcq**() = 0

Prepare the acquisition and make sure the camera is properly configured.

This member function is always called before the acquisition is started.

void **startAcq**() = 0

Start the acquisition.

void **stopAcq**() = 0

Stop the acquisition.

void **getStatus**(*StatusType* &*status*) = 0

Returns the current state of the hardware.

int **getNbAcquiredFrames**()

Returns the number of acquired frames.

int **getNbHwAcquiredFrames**() = 0

Returns the number of acquired frames returned by the hardware (may differ from getNbAcquiredFrames if accumulation is on)

The *lima::HwInterface::getStatus()* member function should return the following information:

struct lima::*HwInterface*::**Status**

A tuple of status with acquisition and detector status / mask.

Public Types

enum Basic

Basic detector states (some detectors may have additional states)

Values:

enumerator Fault

Fault.

enumerator Ready

Ready for acquisition.

enumerator Exposure

Counting photons.

enumerator Readout
Reading data from the chip.

enumerator Latency
Latency between exposures.

enumerator Config
Fault.

Public Members

AcqStatus **acq**
Global acquisition status.

DetStatus **det**
Compound bit flags specifying the current detector status.

DetStatus **det_mask**
A mask specifying the detector status bits that are supported by the hardware.

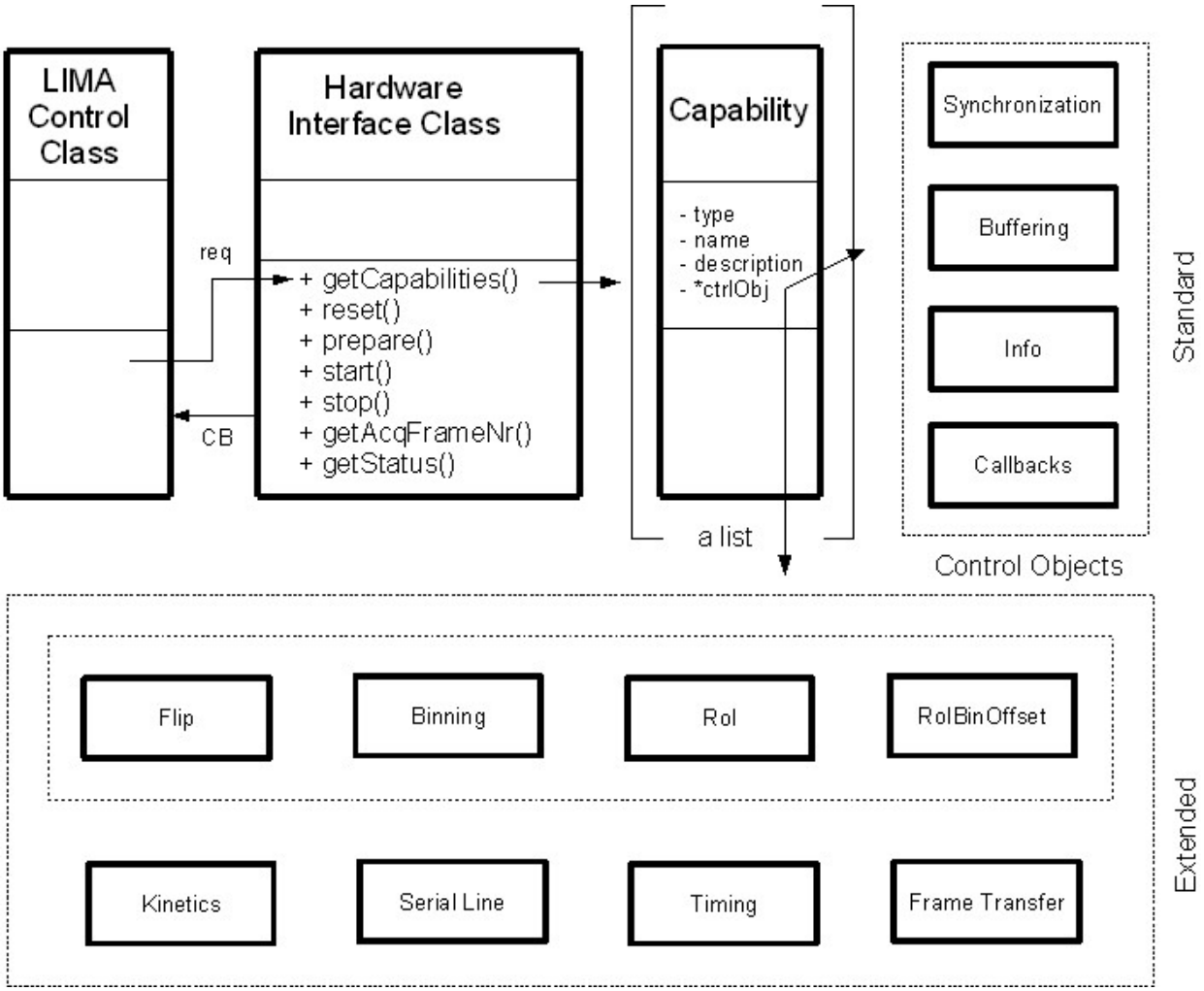


Fig. 2: Figure 2. Hardware capabilites block diagram

10.4 Standard Capabilities

These capabilities are mandatory for all the detectors. They define the minimum functionality necessary for image acquisition. Three capability classes (DetInfo, Sync and BuffCtrl) are listed below with their set/get methods which have to be provided within the new camera plugin code.

10.4.1 Detector Information

The interface *lima::HwDetInfoCtrlObj* returns static information about the detector and the current image dimension.

class lima::**HwDetInfoCtrlObj**

Provides static information about the detector and the current image dimension.

Public Functions

void **getMaxImageSize**(Size &*max_image_size*) = 0
Return the maximum size of the image.

void **getDetectorImageSize**(Size &*det_image_size*) = 0
Return the size of the detector image, it is always equal or greater than the MaxImageSize.

void **getDefImageType**(ImageType &*def_image_type*) = 0
Returns the default data type of image (ushort, ulong, ...)

void **getCurrImageType**(ImageType &*curr_image_type*) = 0
Returns the current data type of image (ushort, ulong, ...).

void **getPixelSize**(double &*x_size*, double &*y_size*) = 0
Physical size of pixels (in mm)

void **getDetectorType**(std::string &*det_type*) = 0
Returns the type of the detector (Frelon, Maxipix, ...)

void **getDetectorModel**(std::string &*det_model*) = 0
Returns the model of the detector.

void **registerMaxImageSizeCallback**(HwMaxImageSizeCallback &*cb*) = 0
Register a callback called when the detector is reconfigured with a different geometry.

void **unregisterMaxImageSizeCallback**(HwMaxImageSizeCallback &*cb*) = 0
Unregister a callback previsouly registered with registerMaxImageSizeCallback.

void **setUserDetectorName**(**const** std::string &*username*)
Set a detector user name.

void **getUserDetectorName**(std::string &*username*)
Get a detector user name.

Note: The `HwMaxImageSizeCallback` callback functions let the hardware inform the Lima library of a change of the detector maximum image size. This change can happen with some detectors which can be reconfigured with a different geometry. This camera capability is *NOT* a Roi *nor* a Bin capability. For instance, the maxipix detector is a mosaic of several individual sensor chips and it can be configured and reconfigured with different geometries according to user needs. A 2x2 maxipix detector can be configured in a 1x1 geometry.

10.4.2 Synchronization

The interface *lima::HwSyncCtrlObj* controls the acquisition parameters related to synchronization.

Parameters	Description
set/getExpTime	Frame exposure time
set/getLatTime	Latency time between frames
checkTrigMode	A check method which returns True/False for the supported trigger modes
set/getTrigMode	**Triggering mode:** • Internal: software triggering • ExtStart: one external signal to start the whole sequence acquisition (one or more frames per sequence) • MultExtStart: one external signal for each frame in the acquisition sequence • Gate: controls start and stop of each frame • ExtStartStop: one start signal to start acquisition of one frame and one signal to stop it

10.4.3 Buffer Management

The interface *lima::HwBufferCtrlObj* controls the image memory buffer allocation and management. They are used:

- As temporary frame storage before saving, allowing disk/network speed fluctuations.
- To permanently hold images that can be read by the user after the acquisition is finished.

These buffer functionalities may be implemented by the hardware layer (kernel driver in the case of the Espia). If not, an auxiliary buffer manager class will be provided to facilitate (and unify) its software implementation. The buffer management parameters are:

Parameters	Description
NbBuffers	Number of image buffers in memory.
NbConcatFrames	The number of concatenated frames per buffer.
NbAccFrames	The number of detector frames to accumulate into a single buffer.
MaxNbBuffers	This Read-Only parameter indicates the maximum number of buffers that can be allocated, given the size of the frame and the number of (concatenated) frames per buffer.
BufferMode	Buffer filling mode (linear or circular)

The buffer manager must also provide the following member functions:

- *lima::HwBufferCtrlObj::getBufferPtr()*
- *lima::HwBufferCtrlObj::getFramePtr()*
- *lima::HwBufferCtrlObj::getFrameInfo()*

In most of simple cases, one just need to create a *lima::SoftBufferCtrlObj* class instance within the Camera class instance to store the frames. A good example of a simple implementation is available in the Andor camera plugin code.

10.4.4 Frame callback

The hardware must provide callbacks after each acquired frame. The callback function should receive the following information:

Parameters	Description
AcqFrameNb	Index of the frame since the start of the acquisition
FramePtr	Pointer to the frame memory
FrameDim	Structure holding the width, height and type of the frame
TimeStamp	Time (in sec.) since the start of the acquisition

The frame callbacks are implemented by means of an auxiliary class *lima::HwFrameCallback*, which will be used by the Control Layer. From the Hardware Layer point of view, the standard capability control object must implement two functions:

- setFrameCallbackActive(bool cb_active)
- frameReady(<callback_frame_info>)

CHAPTER

ELEVEN

SETTING UP A DEVELOPMENT ENVIRONMENT

LImA build dependency were updated with the latest version of LImA and that may be an issue on older distro where the tools are not available, namely:

- CMake >= 3.1
- GCC with C++11 support >= 4.8.1

The first option is to build these packages from source but it is a PITA. One other option is to build with packages managed by Conda and the following instruction should get you started.

11.1 Install Conda

If you don't have Conda installed, get Miniconda and follow the install instruction.

11.2 Create a build environment

A good practice would be not to pollute the base environment and work in a dedicated `lima` environment:

```
conda create -n lima python=3
source activate lima
```

Then install the build tools:

For linux

```
conda install cmake gxx_linux-64
```

For windows, just be sure you have visual studio 2017 x64 installed

You might need to leave the *Conda* environment and enter it again so that the environment variables (*CXX*) needed by CMake are set:

```
source deactivate
source activate lima
```

Finally, install the `lima-core` package (and dependencies) with *Conda*:

```
conda install lima-core
```

And you are good to code! A good way to start is to use our seed project at:

```
git clone --bare https://github.com/esrf-bliss/Lima-camera-template.git
cd Lima-camera-template.git
git push --mirror https://github.com/esrf-bliss/Lima-camera-mycamera.git
```

Once you have your new repo ready, clone it and happy coding!

```
git clone https://github.com/esrf-bliss/Lima-camera-mycamera.git
cd Lima-camera-mycamera
git checkout develop
```

Once you are ready to build, here are the typical CMake commands for an out of source build (in the *build* folder) and for installing in the current Conda environment (`$CONDA_PREFIX`)

For linux:

```
cmake -Bbuild -H. -DLIMA_ENABLE_PYTHON=1 -DCAMERA_ENABLE_TESTS=1 -DCMAKE_FIND_ROOT_
↪PATH=$CONDA_PREFIX -DCMAKE_INSTALL_PREFIX=$CONDA_PREFIX -DPYTHON_SITE_PACKAGES_DIR=
↪$CONDA_PREFIX/<Python site package location>
cmake --build build --target install
```

For windows:

```
cmake -Bbuild -H. -DLIMA_ENABLE_PYTHON=1 -DCAMERA_ENABLE_TESTS=1 -DCMAKE_FIND_ROOT_
↪PATH=%CONDA_PREFIX% -DCMAKE_INSTALL_PREFIX=%CONDA_PREFIX% -DPYTHON_SITE_PACKAGES_
↪DIR=%CONDA_PREFIX%/<Python site package location>
cmake --build build --target install --config Release
```

CHAPTER

TWELVE

SOURCE CODE ORGANIZATION

This chapter provides general guidelines to follow, to share a plugin with the community.

12.1 Source code

12.1.1 Plug-ins submodules

The source files and documentation of each new plug-in must be located under Lima/Camera as shown figure below.

```
├──camera
    └──mycamera
        ├──cmake
        ├──conda
        │   ├──camera
        │   └──tango
        ├──doc
        ├──include
        ├──python
        ├──sip
        ├──src
        ├──tango
        └──test
```

To maintain homogeneity between the different plug-ins, each plug-in must have at minimum the following folders:

- `/src` : contains the source files. Plug-ins must be developed in C++. The "src" folder must contain the following files :
  - `DetectorNameInterface.cpp` : interface class between detector capabilities from the hardware interface and the control layer **(mandatory)**
  - `DetectorNameDetInfoCtrObj.cpp` : capabilities to get static informations about the detector **(mandatory)**
  - `DetectorNameBufferCtrlObj.cpp` : capabilities to control the image memory buffer allocation **(mandatory)**
  - `DetectorNameSyncCtrlObj.cpp` : capabilities to control the image memory buffer allocation **(mandatory)**
  - `DetectorNameRoiCtrlObj.cpp` : capabilities to get a ROI **(optional)**
  - `DetectorNameBinCtrlObj.cpp` : capabilities to make pixel binning **(optional)**

  - `DetectorNameVideoCtrlObj.cpp` : capabilities to make video mode only for non-scientific detectors **(optional)**
  - `DetectorNameShutterCtrlObj.cpp` : capabilities to control shutter **(optional)**
  - `DetectorNameFlipCtrlObj.cpp` : capabilities to flip image **(optional)**
  - `DetectorNameEventCtrlObj.cpp` : capabilities to generate event **(optional)**
  - `DetectorNameSavingCtrlObj.cpp` : capabilities to save images in different formats **(optional)**
- `/include` : contains the header files relative to the sources files described before.
- `/doc` : contains at least `index.rst` for plug-in documentation. Other files such as image can be added. The minimum content of the index file is detailed in the documentation section.
- Other folders can be added based on need. The contents of this file must be described in the documentation.

Note: If optional capabilities are not defined, they are emulated by the Lima Core.

12.1.2 Camera device

Once the plug-in was developed, you must create a camera device to execute all commands on the camera. This device can be developed in Python or C++. Python devices must be located on "Lima/applications/tango/camera", C++ devices on "Lima/applications/tango/LimaDetector"

In order to enhance the general software quality of Device Servers developed by the various institutes using Tango, a Design and Implementation Guidelines document has been written by SOLEIL. This document can be downloaded here.

It is recommended that the camera device comply with these design guidelines.

12.2 Class names

Again, to maintain homogeneity, it is recommended to follow this nomenclature for the class names:

- **DetectorName**::Camera
- **DetectorName**::Interface
- **DetectorName**::SyncCtrlObj
- **DetectorName**::DetInfoCtrlObj

As an example, one can look at the Prosilica plugin for a real implementation or at the simulator plugin for a mock implementation.

12.3 How to test the new plugin with python

In order to communicate with the underlying detector hardware, the lima client must instantiate the main object of the LImA framework *lima::CtControl*. To be instantiated, *lima::CtControl* requires an interface inherited from common *lima::HwInterface*. This interface requires the Camera object that encapsulates dependency with detector and its SDK.

For instance if you are using the python binding for the Prosilica camera, a client application initialization should do:

```
from Lima import Prosilica as ProsilicaAcq
from Lima import Core

my_prosilica_ip_address = 192.168.1.2
# we need the camera object first
camera = ProsilicaAcq.Camera(my_prosilica_ip_address)

# create the HwInterface which needs the camera as unique parameter
camera_interface =  ProsilicaAcq.Interface(camera)

# Now create the :cpp:class:`lima::CtControl` and passed to Lima the new HwInterface
control = Core.CtControl(camera_interface)
```

The camera is now under control and it can be used to acquire images ! First get the sub-objects for the parameter setting of the detector, acquisition, saving and more if necessary.

```
acq = control.acquisition()
saving = control.saving()

acq.setAcqExpoTime(0.1)
acq.setAcqNbFrames(10)

pars=saving.getParameters()
pars.directory='/buffer/test_lima'
pars.prefix='test1_'
pars.suffix='.edf'
pars.fileFormat=Core.CtSaving.EDF
pars.savingMode=Core.CtSaving.AutoFrame
saving.setParameters(pars)

# pass parameters to camera hw interface
control.prepareAcq()

# start the acquisition
control.startAcq()
```

Note: Camera object is only used to enhance the separation between the generic interface and the API driver of the detector. It is similar to a proxy.

The camera class is also supposed to provide an access to the specific configuration of the detector. For instance if your detector has a threshold setting or a built-in background correction available you should implement these features in the Camera class. The *lima::HwInterface* will not know about the specific configuration and a client application should explicitly implement the configuration. A good example is the Andor camera, where there are few extra features like the temperature set-point (set/getTemperatureST()) or the cooler control (set/getCooler(bool)).

With the Andor camera one can set the cooling as:

```
camera.setTemperatureSP(-50)
camera.setCooler(True)

current_temp = camera.getTemperature()
```

The Lima project code provides some client application based on TANGO protocol for the remote access. One can find a python implementation under applications/tango and a C++ version in applications/tango/LimaDetector. The python server has been developed at ESRF and being used on lot of beamlines and the C++ server is the SOLEIL version which is also used on beamlines.

The `LimaCCDs` python server has its own documentation here.

CHAPTER

THIRTEEN

IMPLEMENTATION RECOMMENDATIONS

Use the pImpl idiom to implement the Camera class, breaking compile-time dependency between the vendor SDK and the rest of LImA and downstream applications.

The C++ ABI is sadly [known to be not stable](https://isocpp.org/files/papers/n4028.pdf) between versions of compilers and even between build compiled with the same toolset but different switches. Most vendor SDKs are closed source and cannot be recompiled at will which is the reason why we recommend to use their C version if it exists. Wrapping the C++ API in a C API is a possible workaround.

CHAPTER

FOURTEEN

WRITE A DOCUMENTATION

Plugin documentation must be located in “Lima/camera/detector/name/doc”. It is composed of at least an “index.rst” file which contains information to install, configure and implement a camera plugin. The presence of this documentation is required to share a plugin with Lima community.

Plugins documentation is available in the section “Supported Cameras”.

The table below describes information that must be present in the index file :

Detector Name

Picture of the detector

Introduction

In this section you should describe the detector :
- Manufacter, model
- Interface buses (USB, GIGE, CameraLink, specific acquisition boards,...)
- Type of applications (scientific, industrial, medical, ...)
- OS Supported

Prerequisite

In this section you should specify librairies, driver or software packages required to compile the plugin :
- Version
- Installation path
- Specific procedure for installation (script to execute, environment variables,...)

Installation & Module configuration

In this section you should describe specific procedure for plugin installation :
- Configuration file "config.inc"
- Post installation actions
- Refer to the installation section to compile and install the plugin

Capabilities

Standard capabilities :

Although the plugin as been implement in respect of the mandatory capabilities, some limitations which are due to the camera and SDK features can exist. You should provide here extra information for a better understanding of the three mandatory capabilities below :
- HwDetInfo
- HwSync
- HwBuffer

Optional capabilities :

If optional capabilities are supported by the detector, they should be listed in this section. If some limitiations exist, they should be described here. Available optional capabilities are :
- HwRoi
- HwBin
- HwVideo
- HwShutter
- HwFlip
- HwEvent
- HwSaving

Configuration

This section must summarize different actions to configure the device server and the camera :
- Procedure to configure camera (external tools to set ip adress, ...)
- Properties of the device server to configure
- How to connect the camera
- Others

How to use

In this section you should give a code example to test the plugin. Code may be written in C++ or Python

CHAPTER

FIFTEEN

C++ API

Unfortunately very limited documentation is available from the source but that should improve over time.

15.1 User API

In this section we cover the classes that defines the user interface.

15.1.1 Hello, Lima!

Let's get started with a simple example of an image acquisition function using the simulator camera.

```
// A camera instance and its hardware interface
Simulator::Camera simu;
Simulator::Interface hw(simu);

// The control object
CtControl ct = CtControl(&hw);

// Get the saving control and set some properties
CtSaving *save = ct.saving();
save->setDirectory("./data");
save->setPrefix("test_");
save->setSuffix(".edf");
save->setNextNumber(100);
save->setFormat(CtSaving::EDF);
save->setSavingMode(CtSaving::AutoFrame);
save->setFramesPerFile(100);

// Set the binning or any other processing
Bin bin(2, 2);
CtImage *image = ct.image();
image->setBin(bin);

// Get the acquisition control and set some properties
CtAcquisition *acq = ct.acquisition();
acq->setAcqMode(Single);
acq->setAcqExpoTime(expo);
acq->setAcqNbFrames(nframe);

// Prepare acquisition (transfer properties to the camera)
ct.prepareAcq();
```

(continues on next page)

(continued from previous page)

```
// Start acquisition
ct.startAcq();
std::cout << "SIMUTEST: acq started" << std::endl;

//
long frame = -1;
while (frame < (nframe - 1))
{
      using namespace std::chrono;

      high_resolution_clock::time_point begin = high_resolution_clock::now();

      usleep(100000);

      CtControl::ImageStatus img_status;
      ct.getImageStatus(img_status);

      high_resolution_clock::time_point end = high_resolution_clock::now();

      auto duration = duration_cast<microseconds>(end - begin).count();

      std::cout << "SIMUTEST: acq frame nr " << img_status.LastImageAcquired
              << " - saving frame nr " << img_status.LastImageSaved << std::endl;

      if (frame != img_status.LastImageAcquired) {
              unsigned int nb_frames = img_status.LastImageAcquired - frame;

              std::cout << "  " << duration << " usec for " << nb_frames << " frames\n
↪";
              std::cout << "  " << 1e6 * nb_frames / duration << " fps" << std::endl;

              frame = img_status.LastImageAcquired;
      }
}
std::cout << "SIMUTEST: acq finished" << std::endl;

// Stop acquisition ( not really necessary since all frames where acquired)
ct.stopAcq();

std::cout << "SIMUTEST: acq stopped" << std::endl;
```

15.1.2 Control Interfaces

The control interface is the high level interface that controls an acquisition.

class lima::**CtControl**

Main client class which should be instantiated by the users in their acquisition software.

Advanced control accessors

CtAcquisition ***acquisition**()
Returns a pointer to the acquisition control.

CtSaving ***saving**()
Returns a pointer to the saving control.

CtImage ***image**()
Returns a pointer to the image control.

CtBuffer ***buffer**()
Returns a pointer to the buffer control.

CtAccumulation ***accumulation**()
Returns a pointer to the accumulation control.

CtVideo ***video**()
Returns a pointer to the video control.

CtShutter ***shutter**()
Returns a pointer to the shutter control.

CtEvent ***event**()
Returns a pointer to the event control.

Public Functions

void **abortAcq**()
stop an acquisition and purge all pending tasks.

void **stopAcqAsync**(*AcqStatus acq_status*, ErrorCode *error_code*, Data *&data*)
aborts an acquisiton from a callback thread: it's safe to call from a HW thread.

Creates a dummy task that calls stopAcq() and waits for all buffers to be released

void **abortAcq**(*AcqStatus acq_status*, ErrorCode *error_code*, Data *&data*, bool *ctrl_mutex_locked* = false)
This function is DEPRECATED.

Use stopAcqAsync instead

void **registerImageStatusCallback**(*ImageStatusCallback &cb*)
registerImageStatusCallback is not thread safe!!!

void **unregisterImageStatusCallback**(*ImageStatusCallback &cb*)
unregisterImageStatusCallback is not thread safe!!!

class _AbortAcqCallback : **public** TaskEventCallback

class _LastBaseImageReadyCallback : **public** TaskEventCallback

class _LastCounterReadyCallback : **public** TaskEventCallback

class _LastImageReadyCallback : **public** TaskEventCallback

class _LastImageSavedCallback : **public** TaskEventCallback

class _ReconstructionChangeCallback : **public** Callback

struct ImageStatus

class ImageStatusCallback

class ImageStatusThread : **public** Thread

class SoftOpErrorHandler : **public** EventCallback

struct Status

Acquisition Interface

class lima::**CtAcquisition**

This class control the acquisition of images given a hardware interface.

class _ValidRangesCallback : **public** ValidRangesCallback

struct Parameters

Saving Interface

class lima::**CtSaving**

Control saving settings such as file format and mode.

Saving modes

{

void **setSavingMode**(SavingMode *mode*)

set the saving mode for a saving stream

void **getSavingMode**(SavingMode &*mode*) **const**

get the saving mode for a saving stream

void **setOverwritePolicy**(OverwritePolicy *policy*, int *stream_idx* = 0)

set the overwrite policy for a saving stream

void **getOverwritePolicy**(OverwritePolicy &*policy*, int *stream_idx* = 0) **const**

get the overwrite policy for a saving stream

void **setFramesPerFile**(unsigned long *frames_per_file*, int *stream_idx* = 0)

set the number of frame saved per file for a saving stream

void **getFramesPerFile**(unsigned long &*frames_per_file*, int *stream_idx* = 0) **const**

get the number of frame saved per file for a saving stream

void **setManagedMode**(ManagedMode *mode*)

set who will manage the saving.

with this methode you can choose who will do the saving

- if mode is set to Software, the saving will be managed by Lima core
- if mode is set to Hardware then it's the sdk or the hardware of the camera that will manage the saving.

Parameters

– mode: can be either Software or Hardware

void **resetCommonHeader**()

}

clear the common header

void **setCommonHeader**(**const** HeaderMap &*header*)
set the common header.

This is the header which will be write for all frame for this acquisition

void **updateCommonHeader**(**const** HeaderMap &*header*)
replace/add field in the common header

void **getCommonHeader**(HeaderMap &*header*) **const**
get the current common header

void **addToCommonHeader**(**const** HeaderValue &*value*)
add/replace a header value in the current common header

void **updateFrameHeader**(long *frame_nr*, **const** HeaderMap &*header*)
add/replace several value in the current frame header

void **addToFrameHeader**(long *frame_nr*, **const** HeaderValue &*value*)
add/replace a header value in the current frame header

void **validateFrameHeader**(long *frame_nr*)
validate a header for a frame.

this mean that the header is ready and can now be save. If you are in AutoHeader this will trigger the saving if the data frame is available

void **getFrameHeader**(long *frame_nr*, HeaderMap &*header*) **const**
get the frame header.

Parameters

- `frame_nr`: the frame id
- `header`: the current frame header

void **takeFrameHeader**(long *frame_nr*, HeaderMap &*header*)
get the frame header and remove it from the container

void **removeFrameHeader**(long *frame_nr*)
remove a frame header

Parameters

- `frame_nr`: the frame id

void **removeAllFrameHeaders**()
remove all frame header

void **getStatistic**(std::list<double>&, std::list<double>&, std::list<double>&, std::list<double>&, int *stream_idx* = 0) **const**
get write statistic

void **setStatisticHistorySize**(int *aSize*, int *stream_idx* = 0)
set the size of the write time static list

int **getStatisticHistorySize**(int *stream_idx* = 0) **const**
get the size of the write time static list

void **clear**()
clear everything.

- all header

- all waiting data to be saved
- close all stream

void **writeFrame**(int *frame_nr* = -1, int *nb_frames* = 1, bool *synchronous* = true)
write manually a frame

Parameters

- `aFrameNumber`: the frame id you want to save
- `aNbFrames`: the number of frames you want to concatenate

void **setStreamActive**(int *stream_idx*, bool *active*)
activate/desactivate a stream

void **getStreamActive**(int *stream_idx*, bool &*active*) **const**
get if stream is active

void **getMaxConcurrentWritingTask**(int&, int *stream_idx* = 0) **const**
get the maximum number of parallel writing tasks

void **setMaxConcurrentWritingTask**(int, int *stream_idx* = 0)
get the maximum number of parallel writing tasks

Public Functions

void **setParameters**(**const** *Parameters* &*pars*, int *stream_idx* = 0)
set saving parameter for a saving stream

Parameters

- `pars`: parameters for the saving stream
- `stream_idx`: the id of the saving stream

void **getParameters**(*Parameters* &*pars*, int *stream_idx* = 0) **const**
get the saving stream parameters

Parameters

- `pars`: the return parameters
- `stream_idx`: the stream id

void **setDirectory**(**const** std::string &*directory*, int *stream_idx* = 0)
set the saving directory for a saving stream

void **getDirectory**(std::string &*directory*, int *stream_idx* = 0) **const**
get the saving directory for a saving stream

void **setPrefix**(**const** std::string &*prefix*, int *stream_idx* = 0)
set the filename prefix for a saving stream

void **getPrefix**(std::string &*prefix*, int *stream_idx* = 0) **const**
get the filename prefix for a saving stream

void **setSuffix**(**const** std::string &*suffix*, int *stream_idx* = 0)
set the filename suffix for a saving stream

void **getSuffix**(std::string &*suffix*, int *stream_idx* = 0) **const**
get the filename suffix for a saving stream

void **setOptions**(**const** std::string &*options*, int *stream_idx* = 0)
set the additional options for a saving stream

void **getOptions**(std::string &*options*, int *stream_idx* = 0) **const**
get the additional options for a saving stream

void **setNextNumber**(long *number*, int *stream_idx* = 0)
set the next number for the filename for a saving stream

void **getNextNumber**(long &*number*, int *stream_idx* = 0) **const**
get the next number for the filename for a saving stream

void **setFormat**(FileFormat *format*, int *stream_idx* = 0)
set the saving format for a saving stream

void **getFormat**(FileFormat &*format*, int *stream_idx* = 0) **const**
get the saving format for a saving stream

void **setFormatAsString**(**const** std::string &*format*, int *stream_idx* = 0)
set the saving format as string for a saving stream

void **getFormatAsString**(std::string &*format*, int *stream_idx* = 0) **const**
get the saving format as string for a saving stream

void **getFormatList**(std::list<FileFormat> &*format_list*) **const**
get supported format list

void **getFormatListAsString**(std::list<std::string> &*format_list*) **const**
get supported format list as string

void **setFormatSuffix**(int *stream_idx* = 0)
force saving suffix to be the default format extension

void **getHardwareFormatList**(std::list<std::string> &*format_list*) **const**
return a list of hardware possible saving format

class _ManualBackgroundSaveTask : **public** SinkTaskBase
manual background saving

class _NewFrameSaveCBK : **public** Callback

class _SavingErrorHandler : **public** EventCallback

struct Parameters

Public Functions

Parameters()
Parameters default constructor.

Public Members

std::string **directory**
base path where the files will be saved

std::string **prefix**
prefix of the filename

std::string **suffix**
suffix of the filename

long **nextNumber**
next file number

FileFormat **fileFormat**
the saving format (EDF,CBF...)

SavingMode **savingMode**
saving mode (automatic,manual...)

OverwritePolicy **overwritePolicy**
how you the saving react it find existing filename

std::string **indexFormat**
ie: %.4d if you want 4 digits

long **framesPerFile**
the number of images save in one files

class SaveContainer
Subclassed by lima::SaveContainerCbf, lima::SaveContainerEdf, lima::SaveContainerFits, lima::SaveContainerHdf5, lima::SaveContainerNxs, lima::SaveContainerTiff

Public Functions

bool **needParallelCompression**() **const**
should return true if container has compression or havy task to do before saving if return is true, getCompressionTask should return a Task

See *getCompressionTask*

SinkTaskBase ***getCompressionTask**(**const** *CtSaving*::HeaderMap&)
get a new compression task at each call.

this method is not call if needParallelCompression return false
See *needParallelCompression*

struct Stat

class Stream

class _CompressionCBK : **public** TaskEventCallback
compression callback

class _SaveCBK : **public** TaskEventCallback
save callback

class _SaveTask : **public** SinkTaskBase
save task class

Image Interface

class CtImage
Control image processing settings such as ROI, binning and rotation.

Shutter Interface

class lima::**CtShutter**
Control shutter settings such as open and close time.

struct Parameters

Buffer Interface

class lima::**CtBuffer**
Controls buffer settings such as number of buffers, binning and rotation.

class _DataDestroyCallback : **public** Callback

struct Parameters

15.1.3 Statuses

enum lima::**AcqStatus**
The global acquisition status.

Values:

enumerator AcqReady
Acquisition is Ready.

enumerator AcqRunning
Acquisition is Running.

enumerator AcqFault
An error occured.

enumerator AcqConfig
Configuring the camera.

enum lima::**DetStatus**
Compound bit flags specifying the current detector status.

Values:

enumerator DetIdle

enumerator DetFault

enumerator DetWaitForTrigger

enumerator DetShutterOpen

enumerator DetExposure

enumerator DetShutterClose

enumerator DetChargeShift

enumerator DetReadout

enumerator DetLatency

15.2 Camera Plugin API

15.2.1 Hardware Interface

The Hardware Interface is the low level interface that must be implemented by detector plugins.

class lima::**HwInterface**

As an interface to the Control Layer, this class exports the capabilities provided by the hardware.

It is implemented by every camera plugins.

Public Types

typedef struct lima::*HwInterface*::*Status* **StatusType**

A tuple of status with acquisition and detector status / mask.

Public Functions

void **getCapList**(CapList&) **const** = 0

Returns a list of capabilities.

void **reset**(ResetLevel *reset_level*) = 0

Reset the hardware interface.

void **prepareAcq**() = 0

Prepare the acquisition and make sure the camera is properly configured.

This member function is always called before the acquisition is started.

void **startAcq**() = 0

Start the acquisition.

void **stopAcq**() = 0

Stop the acquisition.

void **getStatus**(*StatusType* &*status*) = 0

Returns the current state of the hardware.

int **getNbAcquiredFrames**()

Returns the number of acquired frames.

int **getNbHwAcquiredFrames**() = 0

Returns the number of acquired frames returned by the hardware (may differ from getNbAcquiredFrames if accumulation is on)

struct Status

A tuple of status with acquisition and detector status / mask.

Public Types

enum Basic
Basic detector states (some detectors may have additional states)

Values:

enumerator Fault
Fault.

enumerator Ready
Ready for acquisition.

enumerator Exposure
Counting photons.

enumerator Readout
Reading data from the chip.

enumerator Latency
Latency between exposures.

enumerator Config
Fault.

Public Members

AcqStatus **acq**
Global acquisition status.

DetStatus **det**
Compound bit flags specifying the current detector status.

DetStatus **det_mask**
A mask specifying the detector status bits that are supported by the hardware.

15.2.2 Capabilities interfaces

class lima::**HwDetInfoCtrlObj**
Provides static information about the detector and the current image dimension.

Public Functions

void **getMaxImageSize**(Size &*max_image_size*) = 0
Return the maximum size of the image.

void **getDetectorImageSize**(Size &*det_image_size*) = 0
Return the size of the detector image, it is always equal or greater than the MaxImageSize.

void **getDefImageType**(ImageType &*def_image_type*) = 0
Returns the default data type of image (ushort, ulong, ...)

void **getCurrImageType**(ImageType &*curr_image_type*) = 0
Returns the current data type of image (ushort, ulong, ...).

void **getPixelSize**(double &*x_size*, double &*y_size*) = 0
Physical size of pixels (in mm)

void **getDetectorType**(std::string &*det_type*) = 0
Returns the type of the detector (Frelon, Maxipix, ...)

void **getDetectorModel**(std::string &*det_model*) = 0
Returns the model of the detector.

void **registerMaxImageSizeCallback**(HwMaxImageSizeCallback &*cb*) = 0
Register a callback called when the detector is reconfigured with a different geometry.

void **unregisterMaxImageSizeCallback**(HwMaxImageSizeCallback &*cb*) = 0
Unregister a callback previsouly registered with registerMaxImageSizeCallback.

void **setUserDetectorName**(**const** std::string &*username*)
Set a detector user name.

void **getUserDetectorName**(std::string &*username*)
Get a detector user name.

class lima::**HwBufferCtrlObj**
This interface controls the image memory buffer allocation and management.

Buffers are used:

- As temporary frame storage before saving, allowing disk / network speed fluctuations.
- To permanently hold images that can be read by the user after the acquisition is finished. These buffer functionalities may be implemented by the hardware layer (kernel driver in the case of the Espia). If not, an auxiliary buffer manager class will be provided to facilitate (and unify) its software implementation. The buffer management parameters are :

Subclassed by *lima::SoftBufferCtrlObj*

Public Functions

void *__getBufferPtr__(int *buffer_nb*, int *concat_frame_nb* = 0) = 0
Returns a pointer to the buffer at the specified location.

void *__getFramePtr__(int *acq_frame_nb*) = 0
Returns a pointer to the frame at the specified location.

void **getStartTimestamp**(Timestamp &*start_ts*) = 0
Returns the start timestamp.

void **getFrameInfo**(int *acq_frame_nb*, HwFrameInfoType &*info*) = 0
Returns some information for the specified frame number such as timestamp.

class Callback

class lima::**HwSyncCtrlObj**

Public Functions

bool **checkTrigMode**(TrigMode *trig_mode*) = 0
Check wether a given trigger mode is supported.

void **setTrigMode**(TrigMode *trig_mode*) = 0
Set the triggering mode.

void **getTrigMode**(TrigMode &*trig_mode*) = 0
Get the current triggering mode.

void **setExpTime**(double *exp_time*) = 0
Set the frame exposure time.

void **getExpTime**(double &*exp_time*) = 0
Get the current frame exposure time.

bool **checkAutoExposureMode**(AutoExposureMode *mode*) **const**
Check wether a given auto exposure mode is supported.

void **setHwAutoExposureMode**(AutoExposureMode *mode*)
this method should be redefined in the subclass if the camera can managed auto exposure

void **setLatTime**(double *lat_time*) = 0
Set the latency time between frames.

void **getLatTime**(double &*lat_time*) = 0
Get the current latency time between frames.

class ValidRangesCallback

struct ValidRangesType

15.2.3 Callbacks

class HwFrameCallback

15.2.4 Implementations Helpers

class lima::**SoftBufferCtrlObj** : **public** lima::*HwBufferCtrlObj*
This class is a basic *HwBufferCtrlObj* software allocation implementation, It can be directly provided to the control layer as a *HwBufferCtrlObj*.

Public Functions

void *__getBufferPtr__(int *buffer_nb*, int *concat_frame_nb* = 0)
Returns a pointer to the buffer at the specified location.

void *__getFramePtr__(int *acq_frame_nb*)
Returns a pointer to the frame at the specified location.

void **getStartTimestamp**(Timestamp &*start_ts*)
Returns the start timestamp.

void **getFrameInfo**(int *acq_frame_nb*, HwFrameInfoType &*info*)
Returns some information for the specified frame number such as timestamp.

class Sync : **public** Callback

CHAPTER

SIXTEEN

PYTHON API

Most of the previous sections about the user interface routines applies to the Python binding. Naturally, some specifics concerning Python come into play.

This documentation is very much a work in progress. Stay tuned!

16.1 Hello, pyLima!

Let's start with a simple example of an image acquisition function using the simulator camera.

```
from Lima import Core
from Lima import Simulator
import time


def test_mode_generator(cam, nb_frames_prefetched=0):
    if nb_frames_prefetched:
        cam.setMode(Simulator.Camera.MODE_GENERATOR_PREFETCH)
        fb = cam.getFrameGetter()
        fb.setNbPrefetchedFrames(nb_frames_prefetched)
        test = fb.getNbPrefetchedFrames()
    else:
        cam.setMode(Simulator.Camera.MODE_GENERATOR)
        fb = cam.getFrameGetter()

    # Add a peak
    p1 = Simulator.GaussPeak(10, 10, 23, 1000) # peak at 10,10 fwhm=23 and max=1000
    fb.setPeaks([p1])


def test_mode_loader(cam, nb_frames_prefetched=0):
    if nb_frames_prefetched:
        cam.setMode(Simulator.Camera.MODE_LOADER_PREFETCH)
        fb = cam.getFrameGetter()
        fb.setNbPrefetchedFrames(nb_frames_prefetched)
        test = fb.getNbPrefetchedFrames()
    else:
        cam.setMode(Simulator.Camera.MODE_LOADER)
        fb = cam.getFrameGetter()

    # Set file pattern
    fb.setFilePattern(b'input\\test_*.edf')
```

(continues on next page)

(continued from previous page)

```
cam = Simulator.Camera()

#test_mode_generator(cam)
#test_mode_generator(cam, 10)
#test_mode_loader(cam)
test_mode_loader(cam, 100)

# Get the hardware interface
hwint = Simulator.Interface(cam)

# Get the control interface
control = Core.CtControl(hwint)

# Get the acquisition control
acq = control.acquisition()

# Set new file parameters and autosaving mode
saving = control.saving()

pars=saving.getParameters()
pars.directory = b'output'
pars.prefix = b'testsimul_'
pars.suffix = b'.edf'
pars.fileFormat = Core.CtSaving.EDF
pars.savingMode = Core.CtSaving.AutoFrame
saving.setParameters(pars)

acq = control.acquisition()

# now ask for 2 sec.  exposure and 10 frames
acq.setAcqExpoTime(0.1)
acq.setAcqNbFrames(10)

control.prepareAcq()
control.startAcq()

# wait for last image (#9) ready
status = control.getStatus()
lastimg = status.ImageCounters.LastImageReady
while lastimg != 9:
  time.sleep(0.1)
  lastimg = control.getStatus().ImageCounters.LastImageReady
  status = control.getStatus()
  lastimg = status.ImageCounters.LastImageReady

# read the first image
im0 = control.ReadImage(0)
```

CHAPTER

SEVENTEEN

PREREQUISITE

For collaborative development, we use the "Fork & Pull" model from Github. So anyone who wants to contribute needs an account on Github. Then you need to fork the project you want to contribute.

Note: If you want to contribute with a new camera plug-in you should first request us (by email @ lima@esrf.fr) to get the new plug-in camera sub-module created. We will provide:

- a default structure of directories (<mycamera>/src /include sip/ doc/ python/ test/)
- the build system file (<mycamera>/CMakeLists.txt)
- templates files (src and include) for the mandatory classes:
- <MyCamera>Interface
- <MyCamera>DetInfoCtrlObj
- <MyCamera>SyncCtrlObj
- a standard .gitignore file
- a template index.rst for the documentation

As above do not forget to fork the new sub-module project.

17.1 Create a github account

This is an easy task, you just have to Sign up, it's free!

17.2 Fork a project

Check out the Github doc, it is far better explained than we could do ;)

CHAPTER

EIGHTEEN

CONTRIBUTE GUIDELINE

It is very simple to contribute, you should follow the steps below.

1. Branch

   First of all you have to create a branch for a new feature or for a bug fix, use an explicit branch name, for instance “soleil_video_patch” .

2. Code/patch

   If it’s a patch from an existing module, respect and keep the coding style of the previous programmer (indentation,variable naming,end-line. . .).

   If you’re starting a new camera project, you’ve just to respect few rules:

   - Class member must start with ‘**m_**’
   - Class method must be in **CamelCase**
   - You must define the camera’s namespace

3. Commit

   Do as many commit as you need with clear comments. Prefer an atomic commit with a single change rather than a huge commit with too many (unrelated) changes.

4. Pull Request

   Then submit a Pull Request

At this stage you have to wait, we need some time to accept or reject your request. So there are two possible issues:

1. The Pull-request is accepted, congrat!

   We merge your branch with the the main project master branch, then everything is fine and you can now synchronize your forked project with the main project and go on with your next contribution.

2. The pull-request is rejected:

   The pull request could be rejected if:

   - the new code doesn’t compile
   - it breaks backward compatibility
   - the python wrapping is missing or not updated
   - the commit log message doesn’t describe what you actually do

   In case of a new camera plug-in sub-module the first pull request will be rejected if:

   - as above

- the documentation is missing or if it does not fit with the guidelines (e.i *Understand the plugin architecture*)

We will tell you (code review on Github and/or email) about the reason and we will give some advises to improve your next tentative of pull-request.

So at this point you have to loop to item 2 (Code/Patch) again. Good luck !

INDEX

L