DECTRIS
detecting the future

User Manual

DECTRIS EIGER®2

Document Version v1.7.1

DECTRIS Ltd.

5405 Baden-Daettwil
Switzerland
www.dectris.com

CONTENT

DOCUMENT HISTORY

Current Document

Table 1: Current Version of this Document

Version	Date	Status	Prepared	Checked	Released
v1.7.1	2021-10-20	released	LW	MM, SB, MB, TD, NP	LW

Changes

Table 2: Changes to this Document

Version	Date	Changes
v1.0.0	2017-04-09	First release.
v1.2.0	2017-09-04	EIGER2 R 500K integration.
v1.3.2	2017-09-04	PILATUS3 and EIGER2 R 500K API documentation integration.
v1.4.0	2018-06-19	EIGER2 Si detector series integration
v1.5.0	2020-03-02	New layout and rework of the technical specifications.
v1.5.1	2020-06-08	Fixed EIGER2 16M image of ground plate.
v1.6.0	2020-09-08	EIGER2 CdTe detector series integration.
v1.6.1	2020-11-27	EIGER2 S series integration.
v1.6.2	2021-01-14	EIGER2 XE 9M & 16M integration.
v1.7.0	2021-09-06	Introduced trademarks and general rework.
v1.7.1	2021-10-20	Update CdTe specifications table.

1. GENERAL INFORMATION

1.1. Contact and Support

Address:	DECTRIS Ltd. Taefernweg 1 5405 Baden-Daettwil Switzerland
Phone:	+41 56 500 21 02
Fax:	+41 56 500 21 01
Homepage:	http://www.dectris.com/
Email:	support@dectris.com

Should you have questions concerning the system or its use, please contact us via telephone, mail or fax.

1.2. Explanation of Symbols

Caution	#0
Caution blocks are used to indicate danger or risk to equipment.	

Information	#0
Information blocks are used to highlight important information.	

1.3. Warranty Information

Caution	#1
Do not ship the system back before you receive the necessary transport and shipping information.	

1.4. Disclaimer

DECTRIS® has carefully compiled the contents of this manual according to the current state of knowledge. Damage and warranty claims arising from missing or incorrect data are excluded.

DECTRIS® bears no responsibility or liability for damage of any kind, also for indirect or consequential damage resulting from the use of this system.

DECTRIS® is the sole owner of all user rights related to the contents of the manual (in particular information, images or materials), unless otherwise indicated. Without the written permission of DECTRIS® it is prohibited to integrate the protected contents in this publication into other programs or other websites or to use them by any other means.

DECTRIS® reserves the right, at its own discretion and without liability or prior notice, to modify and/or discontinue this publication in whole or in part at any time, and is not obliged to update the contents of the manual.

2. SAFETY INSTRUCTIONS

Caution	#2

Please read these safety instructions before operating the detector system.

- Before turning the power supply on, check the supply voltage against the label on the power supply. Using an improper main voltage will destroy the power supply and damage the detector.
- Power down the detector system before connecting or disconnecting any cable.
- Make sure the cables are connected and properly secured.
- Avoid pressure or tension on the cables.
- The detector system should have enough space for proper ventilation. Operating the detector outside the specified ambient conditions could damage the system.
- The detector is not specified to withstand direct beam at a synchrotron. Such exposure will damage the exposed pixels.
- Place the protective cover on the detector when it is not in use to prevent the detector from accidental damage.
- Opening the detector or the power supply housing without explicit instructions from DECTRIS® will void the warranty.
- Do not install additional software or change the operating system.
- Do not touch the entrance window of the detector.

3. SYSTEM DESCRIPTION

3.1. Components

The DECTRIS EIGER®2 detector system consists of the following components:

- EIGER2 detector
- Power supply for the detector[1]
- Detector control unit
- Thermal stabilization unit[2]
- Accessories
- Documentation

3.2. Hybrid Photon Counting (HPC) Technology

3.2.1. Basic Functionality

DECTRIS® X-ray detectors provide direct detection of X-rays with optimized solid-state sensors and CMOS readout ASICs in hybrid pixel technology. Well-proven standard technologies are employed independently for both the sensor and the CMOS readout ASIC. The EIGER2 hybrid pixel detector is composed of a sensor, a two-dimensional array of photodiodes or photoconductors processed in a high-resistivity semiconductor, connected to an array of readout channels designed in advanced CMOS technology. The X-ray detectors operate in single-photon counting mode and provide outstanding data quality. They feature very high dynamic range, zero dark signal and zero readout noise and hence achieve optimal signal-to-noise ratio at short readout time and high frame rates. Large-area detectors with dedicated active areas are built of multiple identical modules using a modular system concept.

Key Advantages

- Direct detection of X-rays
- Single-photon counting
- Excellent signal-to-noise ratio and very high dynamic range (zero dark signal, zero noise)
- Two energy discriminating thresholds
- Four 16 bit digital counters (two per energy discriminating threshold)
- Short readout time and high frame rates
- Shutterless operation
- Modular detectors enabling multi-module detectors with large active area

3.2.2. Continuous Readout

One of the hallmark features of EIGER2 is its continuous readout that enables high frame rates at high duty cycles. Every pixel of an EIGER2 ASIC features two digital counters for each of its two energy discriminating thresholds. After acquisition of a frame, the pixels switch from counting in one digital counter to the other. While one counter is being read out, data acquisition continues in the other counter.

[1] Some systems might be delivered without an external power supply unit. Please consult the Technical Specifications for more information.
[2] Some systems might be delivered without a thermal stabilization unit. Please consult the Technical Specifications for more information.

3.2.3. Auto-Summation

EIGER2 auto-summation mode is a further benefit of continuous readout with high duty cycle. While a single frame is limited to the 16 bit of the digital counter, auto-summation extends the data depth up to 32 bit, or more than 4.2 billion counts per pixel, depending on the number of summed frames in an image. At short exposure times and high frame rates, all counts are captured in the digital counter of a pixel and directly read out as an image. If long exposure times are requested, frames are still acquired at high rates on the pixel level, effectively avoiding any overflows. The detector system sums the frames to images on the fly, extending the bit depth of the data by the number of summed frames.

3.2.4. Instant Retrigger

The EIGER2 X-ray detector features an DECTRIS Instant Retrigger® capability, improved high-rate counting performance. DECTRIS INSTANT RETRIGGER® capability results in non-paralyzable counting and allows for enhanced count-rate correction. In addition, the EIGER2 detectors features various enhancements such as counter overflow handling, improved pixel uniformity and reduced crosstalk.

DECTRIS Instant Retrigger® technology with adjustable dead time is a photon counting imaging method that results in non-paralyzable counting and achieves an improved high-rate counting performance. In a conventional single-photon counting X-ray detector, the charge pulses generated by impinging photons are counted by digital circuits. Simultaneously generated pulses can pile up and as a result, photon counts can be lost. At high photon fluxes, pulse pile-up significantly affects the observed count rate and can lead to complete paralyzation of the counting circuit. In the first generation series of single-photon counting hybrid pixel X-ray detectors "EIGER2 ", count rate correction is applied in order to compensate for the counting loss at high count rates. However, counter paralyzation limits the maximum usable count rate. In EIGER2 detectors, the DECTRIS Instant Retrigger® technology re-evaluates the pulse signal after a predetermined dead-time interval after each count and is able to retrigger the counting circuit should a pulse pile-up occur. The dead time interval is adjustable and is equivalent to the width of one single photon pulse. This results in non-paralyzable counting and allows for enhanced count-rate correction so as to achieve improved data quality at high count rates.

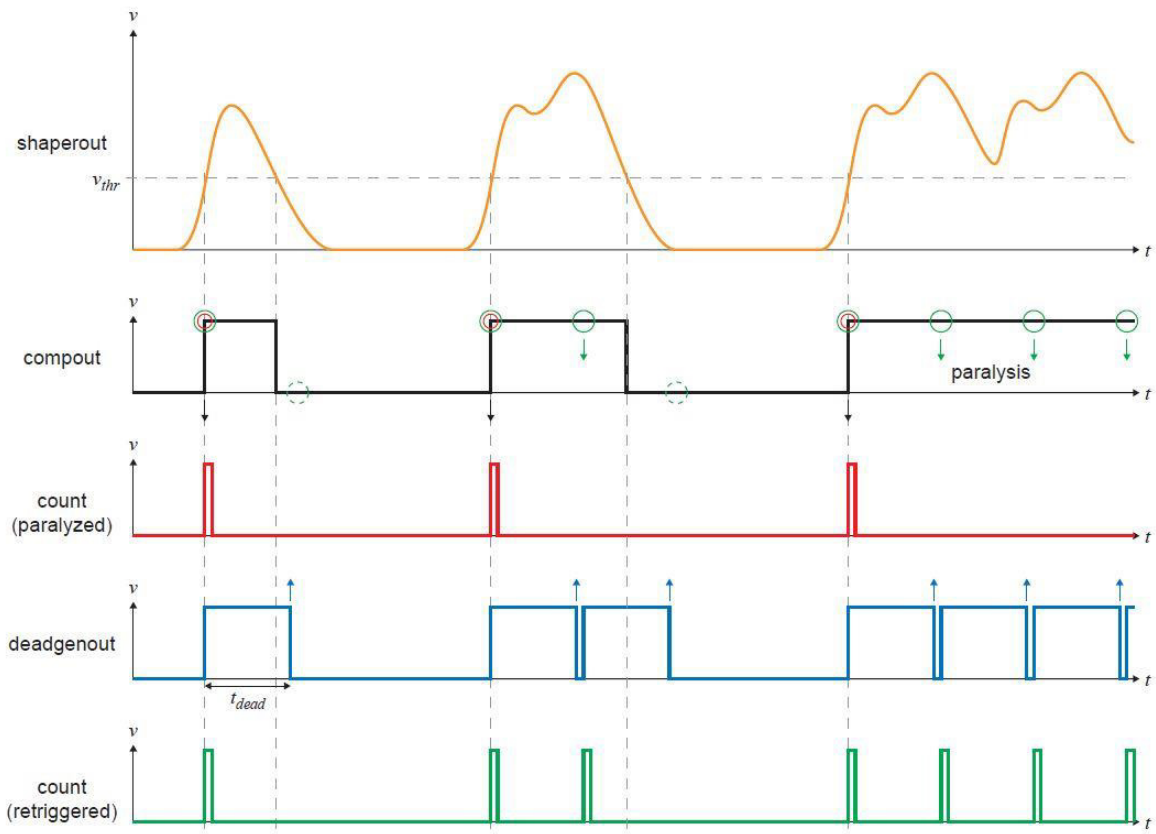

Figure 3.1: Signal Waveforms Illustrating DECTRIS Instant Retrigger® Technology

The principle of DECTRIS Instant Retrigger® technology is illustrated in figure 3.1. The first diagram shows the signal pulses generated by impinging photons and the effective discriminator threshold level for single photon counting. The pulse signal includes a series of one single pulse, a pile-up of two pulses and a pile-up of multiple pulses. The second diagram shows the corresponding digital discriminator output signal that triggers the counting circuit. The third diagram shows the corresponding counts being registered by a conventional single photon counting X-ray detector, clearly illustrating that counts are lost in the case of pulse pile-up and that this can lead to paralyzation. The fourth diagram shows the respective dead-time generator output signal provided by a single photon counting X-ray detector with DECTRIS Instant Retrigger® technology. Here, a predefined dead time interval is started whenever a count has been registered. The fifth diagram shows the corresponding counts being registered, including potential retriggering of the counting circuit after the dead-time interval after each count. This clearly illustrates that pulses are counted more accurately in the case of pile-up and that counting is non-paralyzable.

3.3. Software

3.3.1. Overview of SIMPLON

The EIGER2 detector system is controlled via the SIMPLON API, which relies on a http/REST interface. The API Reference is provided as a separate document "SIMPLON API Reference".
The detector's web interface (chapter 6) gives access to fundamental settings and status parameters and also enables a first test to see if the detector system has been set up properly (after installation and startup as described in chapters 4 and 5).

3.3.2. HDF5 file format and ALBULA image viewer

The EIGER2 detector writes images in the HDF5 file format (chapter 10). DECTRIS® provides the image viewer ALBULA, which is able to handle the HDF5 images, with the primary aim to display them. ALBULA is available free of charge for the platforms Linux and Windows (for download please go to http://www.dectris.com). The ALBULA version for Linux and Windows comes with a Python API for handling the HDF5 files that allows performing arithmetic operations on image data as well as basic analysis. Furthermore, the API enables seamless integration of the viewer into a beamline infrastructure. More information on HDF5 and ALBULA is given in chapter 10.

4. QUICK START GUIDE

4.1. Accessing the Detector Control Unit

The EIGER2 detector is controlled via the network interface of the detector control unit. Hence, the IP network address of the detector control unit has to be known to be able to connect to the API. Depending on the network structure, there are several ways of determining the IP network address, which are described below.

- See the Technical Specifications for the default network port configuration of your detector control unit.
- The default network port configuration may be changed through the detector's web interface (see section 6.2).

4.1.1. Using DHCP

If there is a DHCP server available on the network, plug the network cable into a port of the detector control unit pre-configured for DHCP. See the Technical Specifications for the default network port configuration of your detector control unit.

To determine the IP of your detector, plug in a keyboard and monitor to the detector control unit and power it up. Then follow the following steps:

- Once the detector control unit is fully booted, a command line login prompt will appear.
- Type `recovery` and press return.
- If a password prompt appears, leave it empty and press return. The login name was most likely misspelled, restart from point 1.
- Type `i` to select option `(i) show ip addresses`.
- The IP address will be displayed on the screen.
- If no IP is displayed, make sure that the DCU is properly connected to your network.

Alternatively, the IP network address can be retrieved by searching for the MAC address on the network. For network safety reasons, please ask the network administrator for assistance in obtaining the IP address. If you are the network administrator or have the required permission, the following Linux command can be used to retrieve the IP network address:

[]$_ Linux Command

```
sudo nmap -sP xxx.xxx.xxx.xxx/24 | awk '/^Nmap/{ip=$NF}/yy:yy:yy:yy:yy:zz/{print ip}'\\
```

where *xxx.xxx.xxx.xxx/24* is the network address range to be scanned (e.g. *192.168.0.1/24*) and *yy:yy:yy:yy:yy:zz* is the MAC address of the DHCP network port in the back of the detector control unit. You can find the MAC address of the port using the method described below.

Determining the MAC Address of a Port

The MAC address of the first network port can be found on the bottom of the service tag label (pull-out label in the front of the detector control unit, the correct address is the `EMBEDDED NIC 1 MAC ADDRESS`). The MAC addresses of the second, third and fourth ports are the same as the first one, but with the last two digits incremented by *zz+2*, *zz+4* and *zz+6*, respectively. E.g. if the first port is *01:23:45:67:89:ab*, then the second port is *01:23:45:67:89:ad* (make sure to use the hexadecimal system).

4.1.2. Using a Fixed IP

If you want to access the detector control unit using a fixed IP network address, plug the network cable into the service port of your detector control unit pre-configured for a fixed IP. See the Technical Specifications for the network port configuration of your detector control unit and configure your network accordingly.

If you use e.g., a laptop to access the detector control unit directly for the initial configuration, you can use the following network settings on the laptop:

Table 4.1: Network Settings

IP Adress	*169.254.254.100*
Subnet Mask	*255.255.0.0*
Default Gateway	not required

5. GETTING STARTED

Information	#1
Instructions on properly mounting and preparing the detector system can be found in the Technical Specifications. Before operating the detector, read the complete documentation.	

5.1. Startup Procedure

- Turn on the nitrogen or dry air flow at least 30 min before turning on the detector.[1]
- Turn on the thermal stabilization unit and set the operation temperature as specified in the Technical Specifications. Please read the thermal stabilization unit manual, as some models must be powered and additionally activated in order to operate properly.[2]
- Connect all the required cables for power, data, and trigger signals (if applicable) to the detector and the detector control unit.
- Turn on the detector.
- Turn on the detector control unit. Please allow 6 minutes for the BIOS test procedures and startup of software to complete.
- Check the web interface for system status or start using the SIMPLON API.
- For detectors with a CdTe sensor, it is necessary to wait up to 30 minutes after initializing the detector in order to get the best data quality.

[1] Some detectors do not require a dry air connection. Consult the Technical Specifications for more information.
[2] Some detectors do not require water cooling. Consult the Technical Specifications for more information.

6. WEB INTERFACE

6.1. Overview

The EIGER2 web interface (figure 6.1) provides simple access to basic functions and settings of the detector system for installation, debugging, and system updates. For productive operation of the detector, please refer to the API Reference.

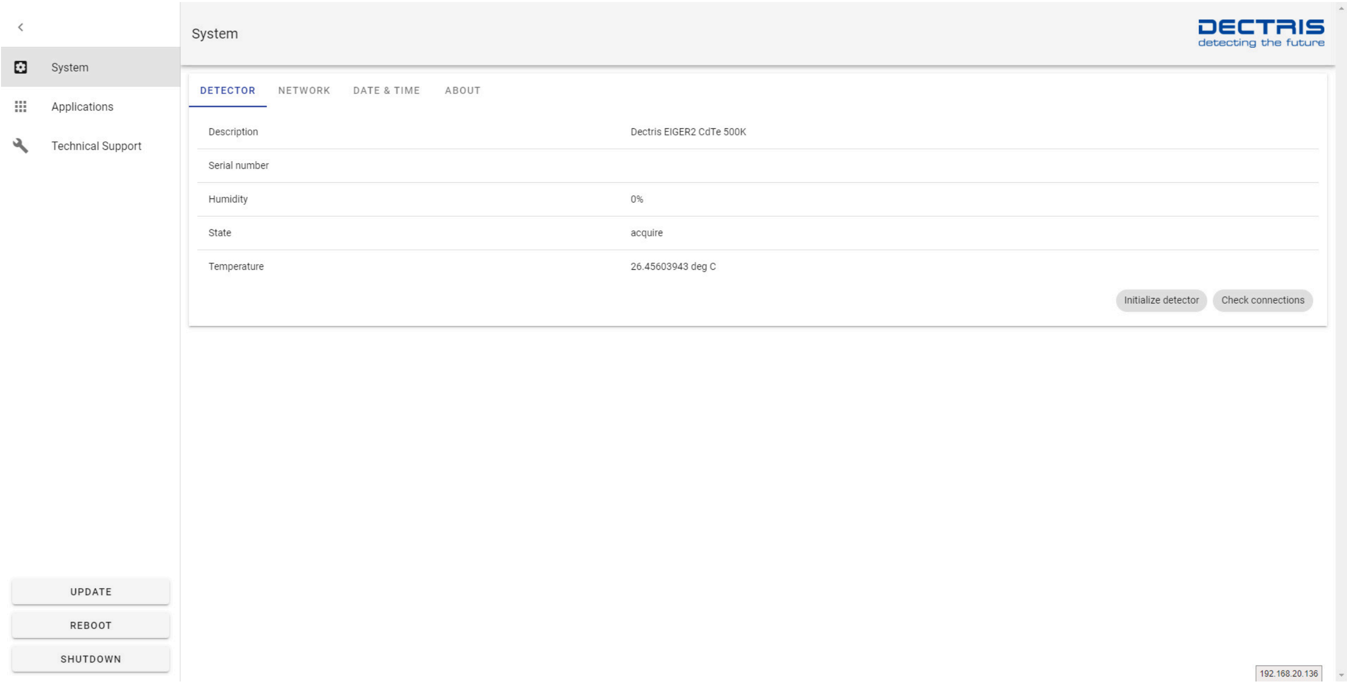

Figure 6.1: EIGER2 home page.

Table 6.1 summarizes the functions available through the left panel of the web system interface.

Table 6.1: Menu items of the EIGER2 web interface.

Menu Point	Content
System	View the system information and access the detector control unit system settings (see section 6.2)
Applications	Manage the detector and calibration applications. This tab provides functions to start and stop the detector software, change software versions and upgrade the detector software to a new version.
Technical Support	Simple interface to create a bug-report. The bug-report creates a tarball that can be downloaded and sent to DECTRIS® support at support@dectris.com. The bug report is not sent automatically.

6.2. System Settings and Administration

To access the EIGER2 system settings, click on the corresponding tab on the homepage. The system tab allows to configure the detector control unit network settings and get some status and system informations.

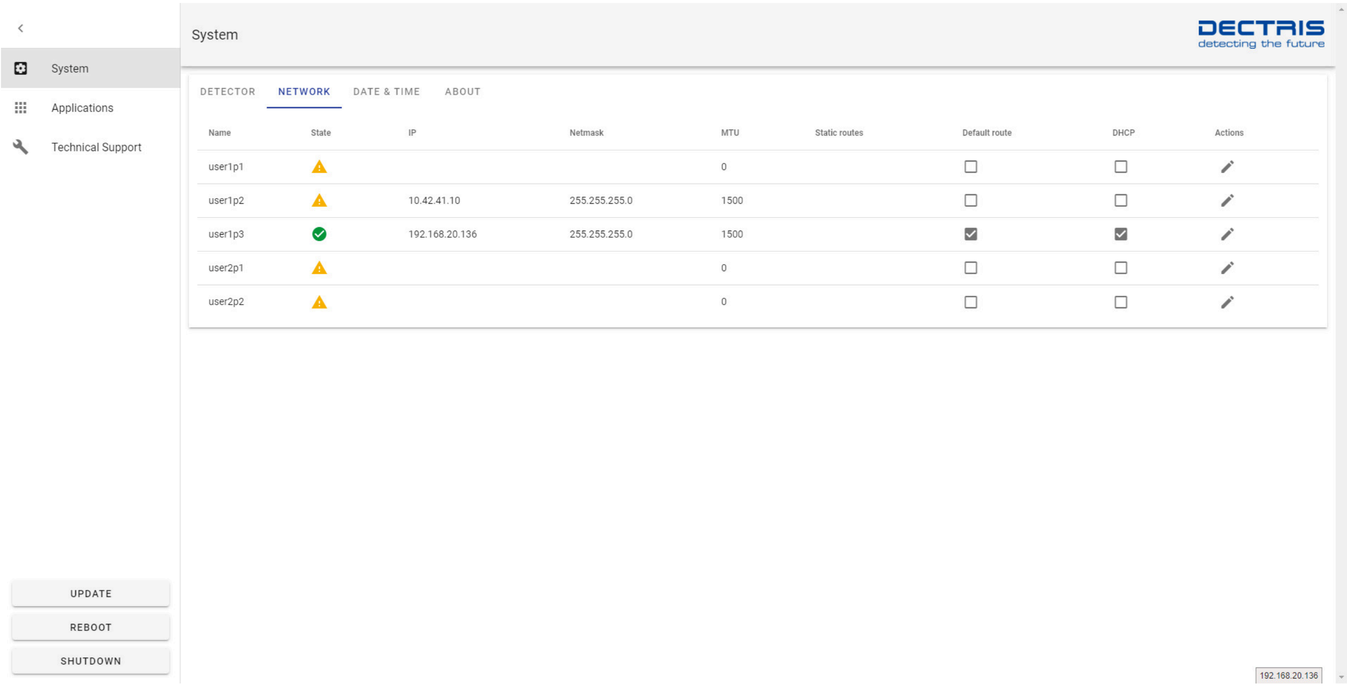

Figure 6.2: Screenshot of the system settings showing the network configuration page.

Table 6.2 summarises menu items available for configuration

Table 6.2: Menu Items for Configuration/Testing

Detector	See the detector status and initialize the detector. The detector status can only be read out when the detector is initialized and the detector application is running. This tab also provides a way to quickly check the quality of the detector connection using the "Check connections" button. Typical values are above 0.15 mW for the TX power and above 0.3 mW for the RX power.
Network	Configure the user accessible network interfaces.
Date & Time	Configure date and time on the detector control unit. Provides the possibility to add an NTP server.
About	View system information.

7. GENERAL USAGE OF THE DETECTOR SYSTEM

7.1. Detector Control and Output

The EIGER2 detector system is controlled through the SIMPLON API, an interface to the detector that is based on the http protocol and implemented on the detector control unit. The API Reference supplied with the system describes this interface in detail and allows for easy integration of detector control into instrument control or similar software. Please refer to the API Reference for details.

The data recorded by the detector can be accessed in different ways. Images can be stored by the filewriter on the detector control unit as HDF5 files (see chapter 10). HDF5 files include meta-data in a NeXus-compatible format. Buffered files have to be regularly fetched and subsequently deleted on the detector control unit as buffer space is limited[1]). Data can also be fetched through the stream API interface. The stream interface relies on ZeroMQ[2], a distributed messaging protocol. The stream has a low latency and offers utmost flexibility. The meta-data is transferred as part of the header. Streamed data is not buffered and will be lost if not fetched or incompletely fetched.

7.2. Recording an Image or an Image Series

Caution	#3
!	Data might have to be fetched concurrently to a running image series. The lifespan of the data on the detector control unit is dependent on the configuration of your system as well as the interface used for collecting data. Data not fetched within this lifespan is permanently lost.

To record images or image series, the following steps need to be performed through the SIMPLON API (see the API Reference for details).

1. Make sure the detector has been set up according to the steps described in chapter 5.
2. Initialize the detector if this has not yet been done. The detector does not need to be reinitialized unless the detector entered an error status.
3. Set the detector parameters for data acquisition and specify and configure the desired output interface (file writer and/or stream interface). A list of essential configuration parameters can be found in section 7.3.1.
4. Arm the detector
5. Record the image or image series.
   - Send trigger(s) to record the image or image series as previously configured.
   - Fetch data through the data interface(s).
6. Disarm the detector (to ensure files are finalized and closed).
7. Repeat from step 2 for further data acquisition with different settings or to step 3 for identical settings.

1 Buffer space varies dependent on the configuration of your system, buffer overflow will cause loss of data. See API Reference for further details.
2 ZeroMQ distributed messaging (http://zeromq.org/)

7.3. Control of the Detector from a Specific Environment

Integrating the detector into a specific environment requires understanding of the necessary detector functions. The API reference will list all possible commands and features, but it does not give an explanation of the required functionality. Sections 7.3.1 and 7.3.2 cover a selection of essential and situational parameters respectively.

7.3.1. Main Configuration Parameters

The parameters described in this section allow control of the detector and data acquisition. Data will be acquired, however further configuration of the interface for data retrieval might be necessary depending on your set up. For starting a data acquisition, only the following parameters need to be adjusted.

- detector | config | *count_time*
- detector | config | *frame_time*
- detector | config | *photon_energy*

The detector configuration parameter *photon_energy* has to be set to the X-ray energy used for the experiment. *Count_time* refers to the actual time the detector counts photons and *frame_time* is the interval between acquisitions of subsequent frames (i.e. period).

- detector | config | *ntrigger*
- detector | config | *nimages*

The *ntrigger* Parameter configures the number of expected triggers for one acquisition series. Setting values greater than 1 for *ntrigger* allows several trigger commands or external trigger pulses per *arm/disarm* sequence. This mode allows recording several series of *nimages* with the same parameters.
The parameter *nimages* configures the number of images that will be acquired for one trigger. This parameter is only used in "ints" and "exts" mode. In "inte" and "exte" mode *nimages* has to be set to one, as every trigger will always only trigger/gate one image. In internally and externally triggered series, the detector always considers a trigger as the start of a series of *n* images. For example a single image is considered as a series of images containing 1 image. Once the detector has been armed a series can be started by issuing a trigger command or triggering the detector using an electronic pulse on the external trigger input (ExtIn). To switch between the trigger modes (see chapter 9) one can use the configuration parameter *trigger_mode*.

- detector | config | *trigger_mode*

There is a convenience function for setting the *photon_energy*, called *element*.

- detector | config | *element*

The detector configuration parameter *element* accepts the chemical symbols for elements, e.g., "Cu", "Mo", as argument and sets *photon_energy* to the K_{α_1} emission line for that element.

Information	#2
Please note that data be retrieved in different ways. For details, please see the API Reference.	

With the filewriter enabled, the acquired data is written into HDF5 files. The filewriter has the following important configuration parameters:

- filewriter | config | *name_pattern*
- filewriter | config | *nimages_per_file*
- filewriter | config | *compression_enabled*

The filewriter parameter *name_pattern* sets the name template/pattern for the HDF5 files. The pattern "*$id*" is replaced with a sequence identification number and therefore can be used to discriminate between subsequent series. The sequence identification number is reset after initializing the detector. The parameter *nimages_per_file* sets the number of images stored per data file. A value of 1000 (default) means that for every 1000th image, a data file is created. If for example, 1800 images are expected to be recorded, the *arm*, *trigger*, *disarm* sequence means that a master file is created in the data directory after arming the detector. The trigger starts the image series and after 1000 recorded images one data container is made available on the buffer of the detector control unit. No further files will made available until the series is finished either by completing the nth image (*nimages*) of the nth trigger (*ntrigger*) or by ending the series using the detector command *disarm*. As soon as either criteria is met the second data container is closed and made available for fetching.

7.3.2. Additional Configuration Parameters

Information	#3
	The following parameters are for special conditions and should be set with care and with understanding of the consequences. Changing these parameters to non-default values can have a substantial negative impact on data quality!

- detector | config | *threshold/n/energy*
- detector | config | *threshold/n/mode*
- detector | config | *threshold/difference/mode*

The EIGER2 detectors provide two independent thresholds and it is possible to read out either one threshold, all of them at once or the difference of the thresholds. To read out more than one threshold at once, it is necessary to use the monitor interface. The thresholds can be enabled or disabled using the *threshold/n/mode* configuration parameter, where *n* is the threshold number. The difference mode is enabled with the *threshold/difference/mode* parameter. Enabling the difference mode will automatically disable the *threshold/n/mode* parameters.

The threshold energy of the lower threshold (n=1) is set automatically to 50 % of the *photon_energy*. *Threshold/1/energy* should only be changed in cases where the suppression of fluorescence is a necessity in the experiment. The *threshold/1/energy* must be kept within 50 % to 80 % of the incoming *photon_energy*. The API incorporates no sanity check on the *threshold/n/energy*.

Corrections are enabled by default, but can be turned off using the following detector configuration parameters. In the vast majority of experiments data quality benefits from the data corrections. Therefore, disabling either correction will likely result in inferior data quality.

- detector | config | *countrate_correction_applied*
- detector | config | *flatfield_correction_applied*
- detector | config | *pixel_mask_applied*

Further parameters, represented by the following selection, allow to enrich the meta-data of the image (series) with experimental data.

- *beam_center_x*
- *beam_center_y*
- *detector_distance*
- *detector_orientation*
- *detector_translation*
- *wavelength* (see section 7.4.1 for dependency with *photon_energy*)

Further parameters and their function are described in the API Reference.

7.4. Interdependency of Configuration Parameters

7.4.1. Interdependency of Calibration Parameters

The following calibration parameters have an implied or direct dependency. Changing either of the parameters might influence other parameters in the list.

detector | config | *photon_energy*

Changing *photon_energy* sets *element* to an empty string and sets *wavelength* to its corresponding value. The *threshold/1/energy* is set to half of *photon_energy*, which is the optimal threshold energy in most cases. The *threshold/2/energy* is set to $1.4*$*photon_energy* in order to suppress higher energy photons and cosmic radiation. In accordance with above rule the *threshold/n/energy* has to be explicitly set after setting *photon_energy* to operate the detector with a threshold energy not equal to half the photon energy. The *flatfield* is recalculated whenever a calibration relevant parameter is changed.

detector | config | *element*

Setting the *element* is equivalent to setting *photon_energy* to the energy of the K_{α_1} emission line of the element. Hence, *photon_energy*, *wavelength* and all parameters that depend on *photon_energy* are changed accordingly.

detector | config | *wavelength*

Setting the *wavelength* is equivalent to setting *photon_energy* to the equivalent energy of the wavelength. Hence, *photon_energy*, *element* and all parameters that depend on *photon_energy* are changed accordingly.

detector | config | *threshold/n/energy*

Changing *threshold_energy* for a threshold causes the *flatfield* for this threshold to be recalculated.

detector | config | *flatfield*

The *flatfield* applied for a given *photon_energy* and *threshold_energy* is an artefact of the detector calibration. During the calibration a multitude of flatfields at different settings have been recorded to ensure optimal data quality of the flatfield for all common settings.

7.4.2. Interdependency of Timing Parameters

The following parameters are essential for exposure timing. Changing either values might influence other values in the list.

detector | config | *frame_time*

If frame_time conflicts with the current *count_time*, *count_time* is set to the difference of *frame_time* and *detector_readout_time*. The auto summation parameters *frame_count_time*, *frame_period* and *nframes_sum* may be updated as well.

detector | config | *count_time*

If count_time conflicts with *frame_time*, *frame_time* is set to the sum of *count_time* and *detector_readout_time*. The auto summation parameters *frame_count_time*, *frame_period* and *nframes_sum* may be updated as well. To acquire images with a certain frame rate and best possible duty cycle, a simple procedure is to first set *count_time* to the inverse of the frame rate and subsequently *frame_time* to the inverse of the frame rate.

8. REGION OF INTEREST (ROI)

The Region Of Interest (ROI) feature enables the user to read out a reduced area of an EIGER2 X 16M or EIGER2 X 9M detector at higher frame rates. The ROI mode is also available on the EIGER2 S 16M or EIGER2 S 9M without the increase in framerate. Refer to the Technical Specifications for the ROI framerate specifications.

Information	#4
Please consult the API reference for further details concerning the usage of the region of interest detector configuration parameter (*roi_mode*).	

Information	#5
Please consult the Technical Specifications and User Manual for details about the ROI capability of your detector.	

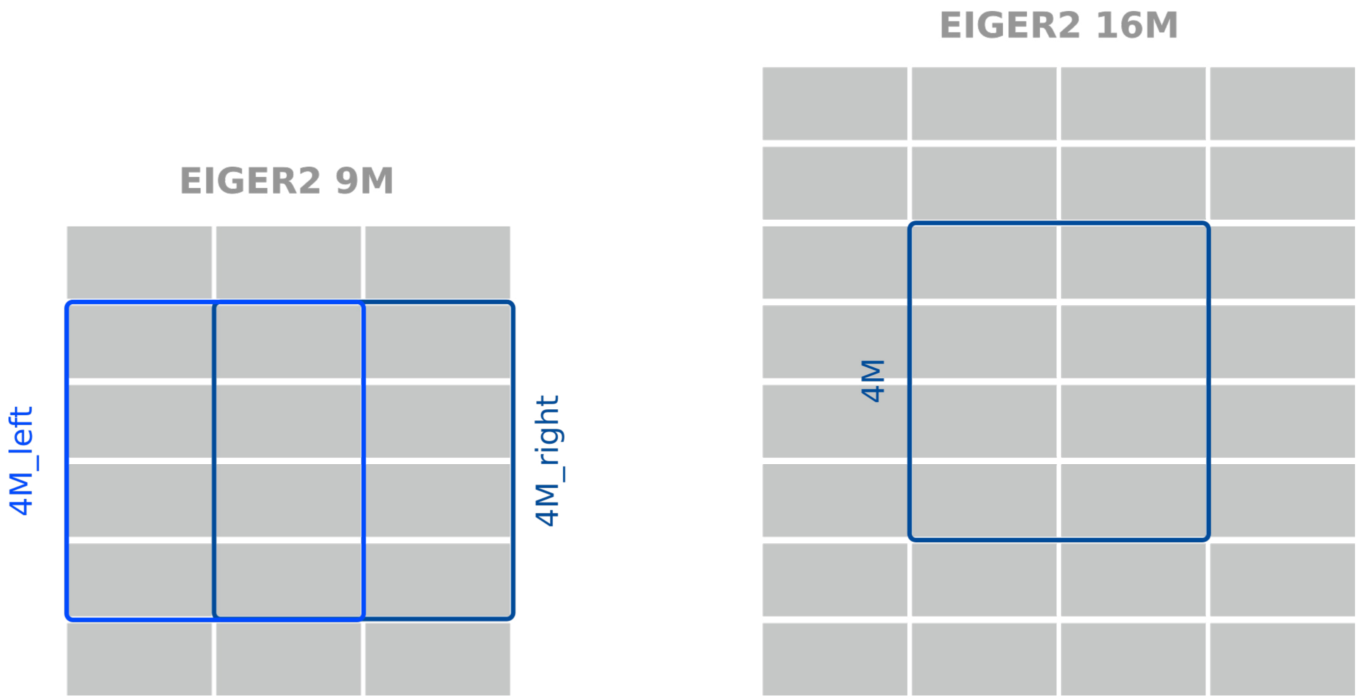

Figure 8.1: The different ROI modes for EIGER2 X 9M (left) and EIGER2 X 16M (right) as seen from the front of the detector.

The ROI mode is disabled by default and the full active area is read out (figure 8.1, grey and blue areas). Changing the ROI mode to another value (eg. "4M", "4M_left", "4M_right", figure 8.1, blue frame) from the list of allowed values will cause the detector to only read out the selected modules. In the EIGER2 X detectors the selected ROI can be read out using an increased frame rate compared to a full detector read out.

9. TRIGGER USAGE

9.1. Introduction

Information	#6
	Depending on the type of EIGER2 system, the valid ranges for the trigger parameter differ. Please consult the Technical Specifications for your system. The values presented in the examples below should work on every EIGER2 system. If the settings or the external trigger/enable pulses applied are out of specification, acquisitions will not be performed and the measurement obtained with the detector might be incomplete. All values used in the example are for demonstrational purposes only and should be adapted to meet the requirements of your application.

In order to record an image or a series of images, the EIGER2 detector has to be initialized, configured, armed, and the exposure(s) started by a trigger signal. The steps necessary to record an image series are comprehensively described in chapter 5 and section 7.2. The detector can be triggered through software (internal trigger) or by an externally applied trigger signal (external trigger). Four different trigger modes are available and described in the following sections 9.2 to 9.5.

9.2. INTS - Internal (Software) Triggering

An exposure (series) can be triggered by using a software trigger. This is the default mode of operation.

Example detector configuration for internally triggered exposure series:

detector | config | *trigger_mode*	{"value": "ints"}
detector | config | *frame_time*	{"value": 1}
detector | config | *count_time*	{"value": 0.7}
detector | config | *nimages*	{"value": 10}

The detector starts the first exposure after the trigger command has been received and processed[1]. All subsequent frames are triggered according to the configuration of the *frame_time* and *count_time* parameters. The detector records *nimages* frames per trigger and stays armed until *ntrigger* are received. Figure 9.1 depicts an internally triggered series defined by *frame_time*, *count_time* and *nimages*.

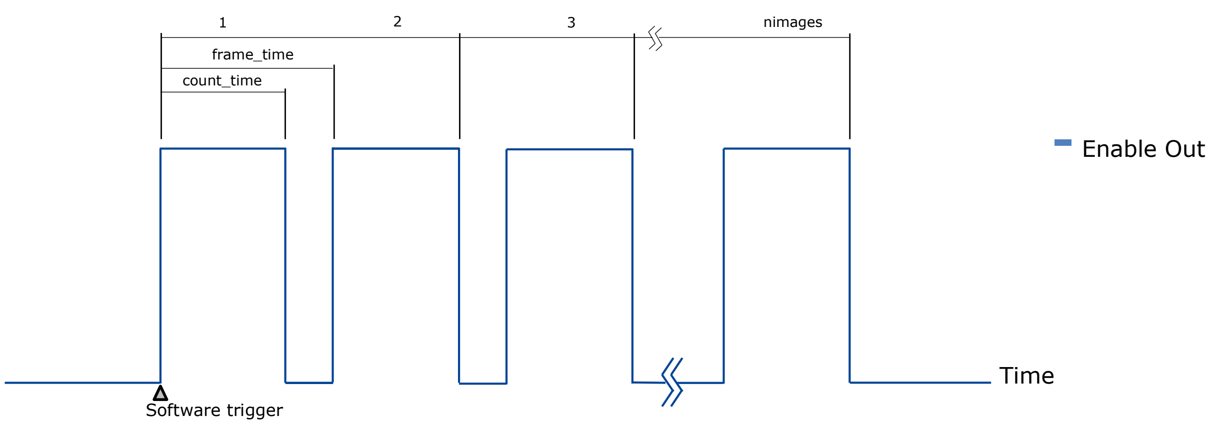

Figure 9.1: Series of exposures, defined by *frame_time*, *count_time* and *nimages*, triggered by a software trigger.

[1] As the trigger command is sent over an TCP/IP connection the exact latency of the start of the exposure is hard to predict.

9.3. INTE – Internal (Software) Enable

In the *trigger_mode* 'inte' a single exposure or series of individual exposures can be started by issuing a number of (*ntrigger*) *trigger* commands. Unlike *trigger* commands in the *trigger_mode* 'ints', 'inte' *trigger* commands take an (optional) argument containing the *count_time* for the subsequent frame. In all enable modes the detector configuration parameter *nimages* is implied to be 1. The number of frames in a series therefore is solely based on the value of the parameter *ntriggger*.

Information	#7
The configured *count_time* and *frame_time* should be close to the count time and frame time of the shortest expected exposure in the configured series. The set *count_time* will be used to calculate internal auto-summation configuration values (section 3.2.3). In most situations a reasonable estimate of these values is sufficient.	

Example detector configuration for an internally enabled exposure series:

detector | config | *trigger_mode*	{"value": "inte"}
detector | config | *nimages*	{"value": 1}
detector | config | *ntrigger*	{"value": 3}
detector | config | *frame_time*	{"value": 1.0} (see 7)
detector | config | *count_time*	{"value": 0.7} (see 7)

The detector starts the first exposure after a *trigger* command has been received and processed[2]. All subsequent frames have to be triggered by individual *trigger* commands with an (optional) argument containing the *count_time* of the triggered frame. The detector stays armed until *ntrigger* are issued or the detector is disarmed. Figure 9.2 depicts an internally enabled exposure series defined by *count_time* (payload of the trigger command) and *ntrigger*. Table 9.3 summarises the commands issued to record the same series.

Table 9.3: Command sequence for an internally enabled (inte) series.

Method	Parameter	Payload
PUT	detector | command | *arm*	
PUT	detector | command | *trigger*	{"value": 0.7}
PUT	detector | command | *trigger*	{"value": 2.1}
...		
PUT	detector | command | *trigger*	{"value": 0.7}

[2] As the trigger command is sent over an TCP/IP connection the exact latency of the start of the exposure is hard to predict.

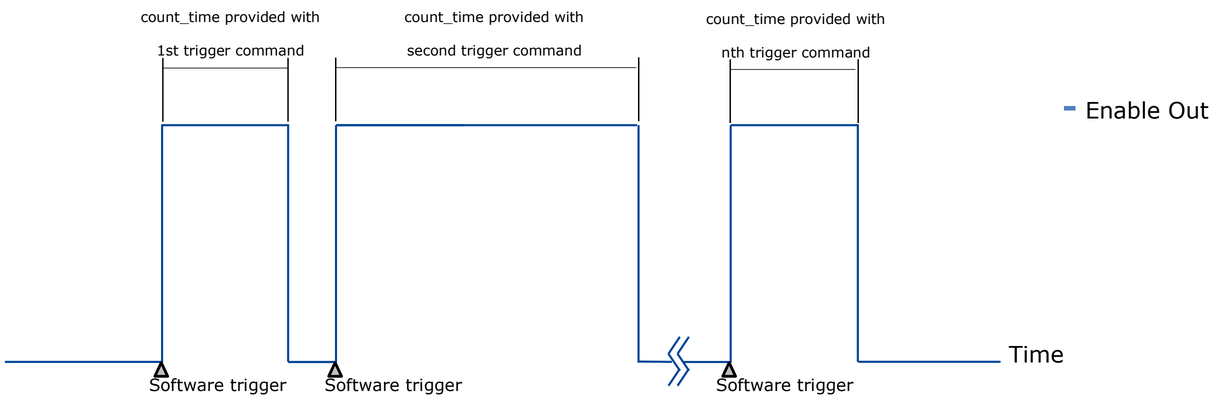

Figure 9.2: Series of exposures, defined by *count_time* (payload of the trigger command) and *ntrigger*, triggered by a software trigger.

9.4. EXTS - Externally Triggered Exposure Series

Caution	#4
⚠	Consult the Technical Specifications for details about the required electrical characteristics of the trigger signal.

The EIGER2 detector systems also support external triggering. In the *trigger_mode* 'exts', *nimages* are recorded per trigger until *ntrigger* are received. Both *count_time* as well as *frame_time* are defined by the configuration.
Example detector configuration for externally triggered exposure series:

detector | config | trigger_mode	{"value": "exts"}
detector | config | frame_time	{"value": 1.0}
detector | config | count_time	{"value": 0.7}
detector | config | nimages	{"value": 10}
detector | config | ntrigger	{"value": 1}

After the detector has been initialized, configured, and armed the acquisition can be triggered by a single external trigger pulse. The detector starts exposing after the (electrical) trigger signal has been issued. All subsequent frames are internally triggered according to the information previously configured by the *frame_time* and *count_time* parameters. The detector records *nimages* frames and stays armed until *ntrigger* are received. Figure 9.3 depicts an externally triggered series defined by *frame_time*, *count_time* and *nimages*.

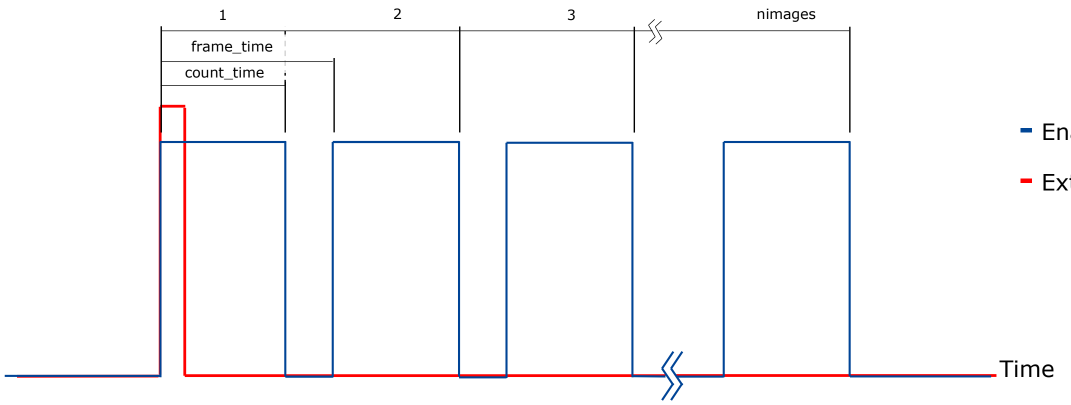

Figure 9.3: Exposure series defined by *frame_time*, *count_time* and *nimages*, triggered by a single external trigger pulse. Note that the periods are not drawn true to scale.

9.5. EXTE - Externally Enabled Exposure Series

Caution #5

Consult the Technical Specifications for details about the required electrical characteristics of the trigger signal.

The EIGER2 detector systems also support external enabling. In the external enable mode 'exte' a series of *ntrigger* frames can be recorded. The count time as well as the period of individual frames of a series are defined by the duration of the high state of the external trigger/enable signal. In all enable modes the detector configuration parameter *nimages* is implied to be 1. The number of frames in a series therefore is solely based on the value of the parameter *ntriggger*.

Information #8

The configured *count_time* and *frame_time* should be close to the count time and frame time of the shortest expected exposure in the configured series. The set *count_time* will be used to calculate internal auto-summation configuration values (section 3.2.3). In most situations a reasonable estimate of these values is sufficient.

Example detector configuration for externally enabled exposure series:

detector | config | trigger_mode	{"value": "exte"}
detector | config | nimages	{"value": 1}
detector | config | ntrigger	{"value": 10}
detector | config | frame_time	{"value": 1.0} (see 8)
detector | config | count_time	{"value": 0.7} (see 8)

After arming the detector, the acquisition can be enabled by an external signal. The value *ntrigger* defines how often this can be repeated. The detector starts exposing the first image after the rising edge and stops after the falling edge of the external trigger signal. In the same manner, all subsequent frames are externally enabled. The count time and period are therefore solely determined by the external enable signal and the limitations of your detector system. The detector records as many frames as valid (according to the specifications) enable pulses are received until the value set for *ntrigger* is reached. Figure 9.4 illustrates a externally enabled series.

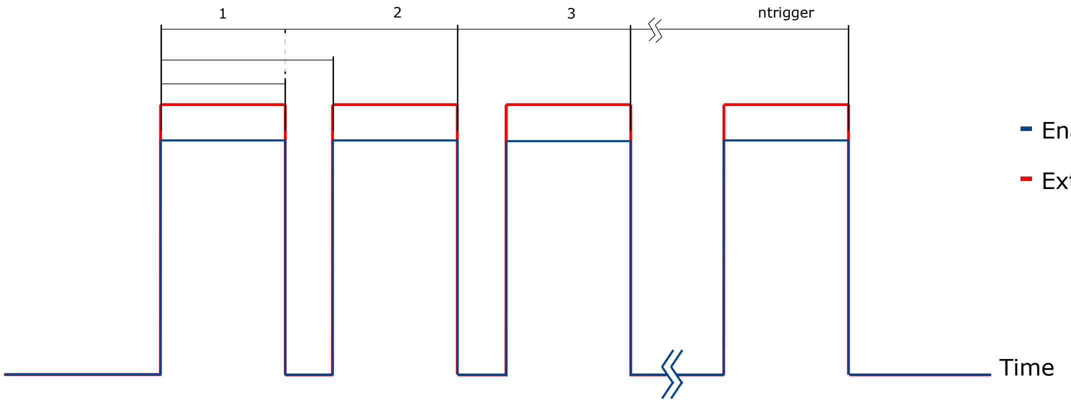

Figure 9.4: Exposures defined by external enable

10. HDF5 AND ALBULA

10.1. ALBULA Overview

ALBULA is a cross-platform image viewer developed and maintained by DECTRIS®. The Linux and Windows versions also provide an image library for the Python language.

ALBULA can be downloaded for free[1] at www.dectris.com. Scripts written in Python using ALBULA can be used to read, display and store data taken by the EIGER2 detectors.

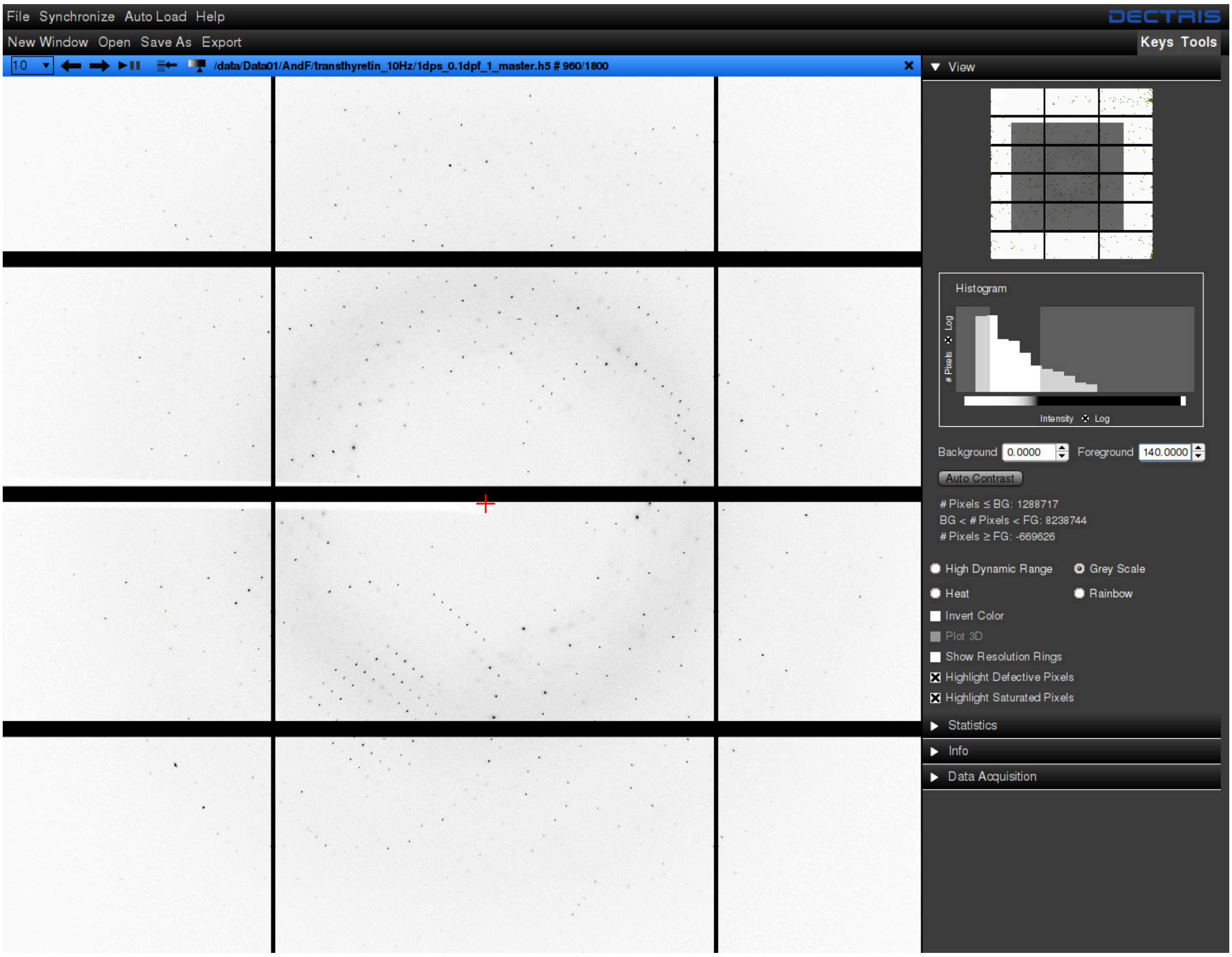

Figure 10.1: Screenshot ALBULA showing crystallographic data acquired by an EIGER X 9M

[1] Registration required

10.2. ALBULA HDF5 Python Library (Linux and Windows only)

The following examples illustrate how the data stored in HDF5 files by the EIGER2 detector can be manipulated with ALBULA.

10.2.1. Getting Started

[]$_ Example ALBULA

```
#!/usr/bin/python

### import the dectris.albula image library ###
import sys
sys.path.insert(0,"/usr/local/dectris/python")
import dectris.albula as albula
def iterateChildren(node,nodeList=[]):
    """ iterates over the children of a neXus node """
    if node.type() == albula.GROUP:
        for kid in node.children():
            nodeList = iterateChildren(kid,nodeList)
    else:
        nodeList.append(node)
    return nodeList
### open the albula viewer ###
m = albula.openMainFrame()
s = m.openSubFrame()
```

10.2.2. Reading data

[]$_ Example ALBULA

```
### read the compressed (or uncompressed) container through the master file ###
h5cont = albula.DImageSeries("series_16_master.h5")

### loop over the frames and read out optional data ###
for i in range(h5cont.first(), h5cont.last() + 1):
    img = h5cont[i]
    ### read header items using convenience functions ###
    optData = img.optionalData()
    ## e.g. wavelength ##
    wavelength = optData.wavelength()
    ## threshold energy ##
    threshold_energy = optData.threshold_energy()

### Read the header item directly without convenience functions ###
neXusHeader = h5cont.neXus()
### print all header item names with path ###
neXusRoot = neXusHeader.root()

def iterateChildren(parent):
    if not hasattr(parent, 'children'):
        return set([parent.path()])
    else:
        paths = set()
        for child in parent.children():
            paths.update(iterateChildren(child))
        return paths

for kid in iterateChildren(neXusRoot):
    print kid.neXusPath()

## extract wavelength ##
wavelength = neXusRoot.childElement('/entry/instrument/monochromator/wavelength')
## print value ##
print "wavelength value: ",wavelength.value()
## extract threshold ##
threshold_energy = neXusRoot.childElement('/entry/instrument/detector/threshold_energy')
## print value ##
print "threshold_energy value: ",threshold_energy.value()
```

10.2.3. Writing Data

[]$_ Example ALBULA

```
### write the (uncompressed) images and the neXus header to a new HDF5 file ###
HDF5Writer = albula.DHdf5Writer("testContainer.h5",1000, neXusHeader)
for i in range(h5cont.first(), h5cont.last() + 1):
    img = h5cont[i]
    HDF5Writer.write(img)
### flushing closes the master and the data files ###
HDF5Writer.flush()

### write the images in the cbf format. Careful: Information from the header will be lost!
    ###
for i in range(h5cont.first(), h5cont.last() + 1):
    img = h5cont[i]
    albula.DImageWriter.write(img, "testImage_{0:05d}.cbf".format(i))

### write the images in the tif format. Careful: Information from the header will be lost!
    ###
for i in range(h5cont.first(), h5cont.last() + 1):
    img = h5cont[i]
    albula.DImageWriter.write(img, "testImage_{0:05d}.tif".format(i))
```

10.3. Third Party HDF5 Libraries

The EIGER2 HDF5 data can also be directly read with programs using the HDF5 library. By default the EIGER2 data is compressed using the BSLZ4[2] algorithm. In order to decompress the data, the HDF5 plug-in filter[3] can be used, see https://github.com/dectris/HDF5Plugin. By setting the environment variable *HDF5_PLUGIN_PATH* to the path where the compiled plug-in filter can be found, the HDF5 library will decompress the data compressed with LZ4 by itself. If you want to use proprietary software like Matlab, IDL or similar, make sure that the HDF5 library version used by this software is at least v1.8.11 in order for the plug-in mechanism to work.

For developers using C++, example code can be found on the DECTRIS® website, after registration and login.

[2] See: https://code.google.com/p/lz4/, https://github.com/kiyo-masui/bitshuffle
[3] In order to use the filter plug-in mechanism, HDF5 v1.8.11 or greater must be used. See also http://www.hdfgroup.org/HDF5/doc/Advanced/DynamicallyLoadedFilters/HDF5DynamicallyLoadedFilters.pdf

11. PIXEL MASK

11.1. Applying the pixel mask

The detector configuration parameter *pixel_mask_applied* enables (*True*) or disables (*False*) applying the pixel mask on the acquired data. Every threshold has its own pixel mask. These masks may differ from one another. If *true* (default), pixels which have any bit set in the *pixel_mask* are flagged with $(2^{\text{image bit depth}} - 1)$. Please consult the API Reference for details on the detector configuration parameter *pixel_mask*.

11.2. Updating the pixel mask

11.2.1. Overview

Updating the pixel mask of an EIGER2 detector system involves four basic steps:

1. Retrieving the current pixel mask from the detector system via the SIMPLON API.
2. Manipulating the pixel mask to add or update pixels.
3. Uploading the updated pixel mask to the detector system via the SIMPLON API.
4. Persistently storing the updated pixel mask on the detector system by sending the detector command *arm*.

11.2.2. Retrieving the current mask from the detector system

The pixel mask can be retrieved from the detector system by a GET request on the detector configuration parameter *threshold/n/pixel_mask* where *n* is the threshold number. The data of the pixel mask is retrieved either as tiff or in JSON serialization by choosing *application/tiff* or *application/json* in the *get* request accordingly.

11.2.3. Manipulating the pixel mask

Information	#9
For details about the pixel values in the pixel mask and their meaning, consult the API Reference.	

TIFF

If the pixel mask is retrieved and stored as *tiff*, the uint32 data in the *tiff* file can be manipulated with ALBULA API.

JSON

If the pixel mask is retrieved as *JSON*, the HTTP reply has to be parsed correctly into an array. Please see the example below and the API Reference for details. The values in this array can then be manipulated to reflect the required updates of the pixel mask. After updating the array, it has to be serialized again in JSON according to the specifications in the API Reference.

11.2.4. Uploading and storing the pixel mask

The pixel mask is uploaded by sending a PUT request on the detector configuration parameter *pixel_mask* with the new mask as data. After sending the detector command *arm*, the updated pixel mask is permanently stored on the detector system.

11.2.5. Python Example

The following Python code using common libraries provides a simple example for updating the pixel mask:

[]$_ Python Code

```python
import json
import numpy
import requests

from base64 import b64encode, b64decode

IP = '169.254.254.1'
PORT = '80'

def get_mask(ip, port):
    """
    Return the pixel mask of host EIGER system as numpy.ndarray
    """
    url = 'http://{}:{}/detector/api/1.8.0/config/pixel_mask'.format(ip, port)
    reply = requests.get(url)
    darray = reply.json()['value']
    return numpy.frombuffer(b64decode(darray['data']),
        dtype=numpy.dtype(str(darray['type']))).reshape(darray['shape'])

def set_mask(ndarray, ip, port):
    """
    Put a pixel mask as ndarray on host EIGER system and return its reply
    """
    url = 'http://%s:%s/detector/api/1.8.0/config/pixel_mask' % (ip, port)
    data_json = json.dumps({'value': {
                                '__darray__': (1,0,0),
                                'type': ndarray.dtype.str,
                                'shape': ndarray.shape,
                                'filters': ['base64'],
                                'data': b64encode(ndarray.data).decode('ascii') }
                            })
    headers = {'Content-Type': 'application/json'}
    return requests.put(url, data= data_json, headers=headers)

if __name__ == '__main__':
        # initialize the detector
        url = 'http://{}:{}/detector/api/1.8.0/command/initialize'.format(IP, PORT)
        assert (requests.put(url).status_code == 200), 'Detector could not be initialized'
    # get the mask
    mask = get_mask(ip=IP, port=PORT)
    # copy the mask to writeable buffer, necessary for numpy>=1.16.0
    mask = numpy.copy(mask)
    # set a new dead pixel [y,x]
    mask[123, 234] = 2
    # set a new noisy pixel [y,x]
    mask[234, 123] = 8
    # upload the new mask
    reply = set_mask(mask, ip=IP, port=PORT)
    # reply.status_code should be 200, then arm and disarm to test the system
    if reply.status_code == 200:
        for command in ('arm', 'disarm'):
            url = 'http://{}:{}/detector/api/1.8.0/command/%s'.format(IP, PORT, command)
            requests.put(url)
    else:
        print reply.content
```

TRADEMARKS AND PATENTS

Trademarks

Registered trademarks:

“DECTRIS”: EU, JP, CN, KR (IR0911969), USA (5,253,168)
“detecting the future”: CN, KR, JP, EU, AUS (IR1191333), USA (4,607,800)
“DECTRIS Instant Retrigger”: CN, KR, JP, EU, AUS (IR1224728), USA (4,797,363)
”DECTRIS PILATUS”: JP, KR (IR1350411), USA (5,415,156)
”DECTRIS EIGER”: EU, JP, CN, KR, CH, AUS (IR1350410), USA (5,415,155)
”DECTRIS MYTHEN”: JP, KR, CH, EU, AUS (IR1350261), USA (5,499,834)
”DECTRIS SÄNTIS”: JP, KR, CH, EU, AUS (IR1350408), USA (5,415,153)
”DECTRIS QUADRO” CH, EU (722 238), CN (IR1437238)