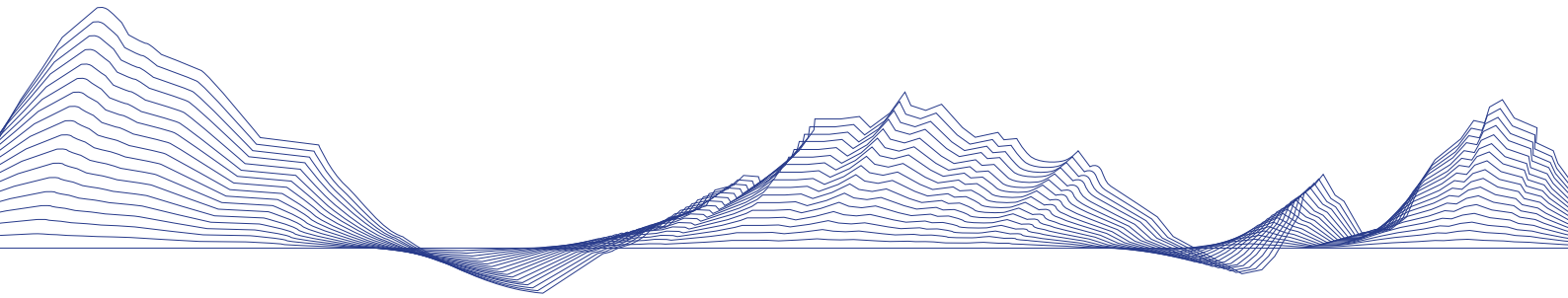
DECTRIS



SIMPLON 1.8

API documentation

Document Version v3.11

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
v3.11	2026-03-12	release	DG	DM	DG

Changes

Table 2: Changes to this Document

Version	SW version	Date	Changes
v3.4	>=v2024.2	2024-12-10	Clarify filewriter compression settings and threshold descriptions
v3.5	>=v2024.2	2024-12-10	Fix external gating mode description
v3.6	>=v2024.2	2025-01-23	Replace specific detector mentions with generic descriptors
v3.7	>=v2024.2	2025-02-24	Add missing EIES parameter (ntriggers_skipped)
v3.8	>=v2025.1	2025-03-11	Add lance operation keys (only available for certain detectors)
v3.9	>=v2025.1	2025-03-28	Minor corrections
v3.10	>=v2025.1	2025-07-01	Clarify auto_sum_strict
v3.11	>=v2025.1	2026-03-12	mentioned SW versions in the release history are the minimal affected SW Version for which the document is valid.

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. INTRODUCTION – RESTLIKE API

2.1. Motivation

The main objective of the SIMPLON API is to provide platform-independent control of DECTRIS detector systems using a well-established standardized API. The RESTlike API requires no additional software to be installed on the detector control unit nor is access to the detector restricted to a specific programming language. In order to define the state of the detector, trigger an exposure or request an image file, an HTTP requests need to be transmitted to the server and the requested data may be received within the HTTP response.

For instance consider the common detector control parameter *count_time*, which defines the duration of a frame (i.e. the time the detector is counting X-rays). You may request the current state of this parameter by entering its URL *http://<address_of_dcu>/detector/api/1.8.0/config/count_time* in your favorite web browser's address field, where *<address_of_dcu>* needs to be replaced by the IP address of the detector computer. The SIMPLON API will respond with a JSON dictionary containing information about the current setting, the limits and other useful information. Analogously, the URL of the frame time is *http://<address_of_dcu>/detector/api/1.8.0/config/frame_time*.

This HTTP-based API is RESTlike, because every detector resource is uniquely identified by its URL. A comprehensive definition of RESTful goes beyond the scope of this documentation. This documentation is confined to instructions on how to work with the SIMPLON API. The API is RESTlike rather than RESTful, because it does not fulfill all requirements of a RESTful API.

2.2. Usage

Let's have a further look at the sample request *count_time*. We have learned that the parameter is mapped to a unique URL, and that the request is transferred to the server via HTTP. Besides the URL, the HTTP request contains extra data. The HTTP verb or method defines which kind of action has to be performed on the server. The SIMPLON API uses the verbs *GET*, *PUT* and *DELETE*. When entering *http://<address_of_dcu>/detector/api/1.8.0/config/count_time* into your browser, your browser will send a *GET* request to the server, which is meant to return a representation of the resource but, by definition, must not change the resource itself. In our case, we receive the value of *count_time*. The value of *count_time* may be changed by a *PUT* request on the same URL. The data itself that is requested from the server or uploaded to the server, i.e. the value of *count_time*, is transferred in the message body of the HTTP request. The SIMPLON API relies on JSON as its default messaging data format. For instance, your browser may display the following string after you have issued a *GET* request to the *count_time* parameter:

[]$_ JSON Response example


```
{
  "min" : 0.01818171818181818,
  "max" : 3600,
  "value" : 0.5,
  "value_type" : "float",
  "access_mode" : "rw",
  "unit" : "s"
}
```


This is a JSON dictionary that contains the keys "*min*", "*max*", "*value*", "*value_type*", "*access_mode*" and "*unit*". From the value of the keys "*value*" and "*unit*" we find the value of the *count_time* to be 0.5 seconds.

Information #1

If you try this example and the browser prints instead "Parameter *count_time* does not exist", your detector may not have been initialized. The detector must be initialized beforehand because the SIMPLON API needs to obtain information on the detector's configuration. The initialization process reads the configuration back from the detector. In order to initialize the detector, you must send a *PUT* request to *http://<address_of_dcu>/detector/api/1.8.0/command/initialize*.

A *PUT* request cannot be sent from a web browser. For testing, you may either use the plugin HttpRequestMaker for Mozilla Firefox or the command line tool cURL. HttpRequester (version 2.0) opens a window with a field "*URL*", where you have to enter *http://<address_of_dcu>/detector/api/1.8.0/command/initialize*. Again, substitute *<address_of_dcu>* by the IP of the DECTRIS detector control unit. Press *PUT* and wait for the reply, which may take some time. The API will respond with status code 200 OK and an empty message body.
If an error occurs an HTTP error code is returned. In this case, please create a bug report, which can be accessed by the web interface of the DCU in the support subsection and contact support@dectris.com.

[]$_ URLs

```
<uri> = http://<ADDRESS_of_DCU>/#/support
```

Now we want to set *count_time* to 1.0 seconds. To set set the *count_time* you have to upload the value 1.0 (datatype *float*). The API assumes the value to be in seconds, because *count_time* has that unit.

Information #2

There is no way to change the unit of a parameter.

In HttpRequestMaker we set the URL to *http://<address_of_dcu>/detector/api/1.8.0/config/count_time*. Below you will find a field "Content to Send". The content type must be changed to "application/json" and the following string must be pasted into the content field.

[]$_ JSON Response

```
{
  "value" : 1
}
```

HttpRequestMaker will upload a JSON dictionary with its only key "*value*" set to 1.0. After pressing *PUT*, in the return window on the right hand, we receive the list:

[]$_ JSON Response

```
[
  "bit_depth_image",
  "count_time",
  "countrate_correction_count_cutoff",
  "frame_count_time",
  "frame_period"
]
```

This is the list of parameters that have been or may have been changed, explicitly or implicitly. The items in the list may vary depending on your detector model. The SIMPLON API always keeps the configuration in a consistent state. So if the *count_time* has been changed, the frame time (time between two successive images) might need to be changed as well, because frame time must be longer than the count time. A *GET* request on *http://<address_of_dcu>/detector/api/1.8.0/config/frame_time* tells us that *frame_time* is now slightly longer than *count_time*.

So far we have seen two examples of addressing a detector resource via a URL. Detector parameters are configured via *GET*/*PUT* requests on *http://<address_of_dcu>/detector/api/1.8.0/config/<parameter>*, detector commands are transferred via *PUT* requests *http://<address_of_dcu>/detector/api/1.8.0/command/<command>*. Finally the status of the detector may be queried via *http://<address_of_dcu>/detector/api/1.8.0/status/<statusparameter>*. The term "*task*" is used as the general term for "*configuration*", "*status*" and "*command*". In addition to the detector interface (i.e. module) there are for example a FileWriter interface (*http://<address_of_dcu>/filewriter/api/1.8.0/<task>*) and a stream interface (*http://<address_of_dcu>/stream/api/1.8.0/<task>*). A resource thus is composed of the module (e.g. detector, stream, filewriter etc.), the API and version references as well as the task (e.g. config, status, command) and the parameter.

		module		version	task	parameter
http://	*<address_of_dcu>/*	*<module>/*	*api/*	*<version>/*	*<task>/*	*<parameter>*

As an example the parameter *count_time* is child to the module detector and the task config.

		module		version	task	parameter
http://	*10.42.41.10/*	*detector/*	*api/*	*1.8.0/*	*config/*	*count_time*

The FileWriter interface lets you control how the data is stored in *hdf5* files. In addition the *hdf5* files may be received from *<ADDRESS_OF_DCU>/data/*. There are two *hdf5* files. The *master* file contains header data and links to the image data, which reside in *series_1_data_000001.h5*. Image series that contain more than one dataset may be distributed over multiple data files, each containing a block of (e.g. 1000) images.

2.3. API Versioning

The API version is at sequence of a least three numbers separated by dots. We use the semantic versioning scheme (MAJOR.MINOR.PATCH). With each change in the API a number is increased based on the conditions described in table 2.3.

Table 2.3: Severity of changes reflected in the API version.

Changed Field	Description
1 MAJOR	incompatible API changes
8 MINOR	add functionality in a backwards-compatible manner
0 PATCH	currently unused, see release notes for changes

3. OPERATING THE DECTRIS DETECTOR SYSTEM

3.1. Acquiring Data

In order to acquire data with an DECTRIS detector system, these steps need to be performed:

Initialize the detector

- Mandatory only once after any of the following events: power-up of the detector; power-up of the detector control unit, restart of the DAQ service providing the SIMPLON API.
- Depending on system configuration, this may take up to 2 minutes
- Blocking operation, no other API operation may be performed until successful completion
- See Detector -> Commands -> Initialize

Configure the detector

- Although this does not result in error if not performed, the user should set the required parameters for the experiment
- If nothing is configured, defaults will be used
- See Detector -> Configuration

Configure the data interfaces

- In order for the acquired data to be written, at least one of following data interfaces needs to be activated.
- If no interface is enabled (default) no data will be written. Enable at least one of following interfaces.
- See FileWriter -> Configuration -> mode
- See Stream -> Configuration -> mode
- See Monitor -> Configuration -> mode

Arm the detector

- This uploads the configuration to the detector and prepares the system for data acquisition, but does not yet activate acquisition
- Depending on system configuration, this may take a few seconds. If the configuration is not changed the command will finish significantly faster on subsequent requests.
- See Detector -> Commands -> Arm

Trigger the detector

Information	#3
Sending a ”trigger” command is mandatory in software trigger mode (e.g ints, inte) and has to be omitted in external enabled modes (e.g exts, exte).	

- This activates the actual data acquisition.
- See Detector -> Commands -> Trigger

Disarm the detector

Information	#4

Depending on trigger mode the last acquired image (or the image, if only one image was configured) is available only after a disarm command has been issued.

- Disables the trigger unit
- See Detector -> Commands -> Disarm

Repeating Acquistions

If a new acquisition is required, repeat these steps in the given order:

Configure	(optional)
Arm	(mandatory)
Trigger	(mandatory for internal trigger, omit for external trigger/enable)
Disarm	(optional as of firmware > 1.5.1[1])

Receiving Data

For receiving data, multiple options are available:

- Writing and downloading *HDF5* files via the FileWriter interface (section 5.3)
- Retrieving the data as a stream via the stream interface (section 5.4)

3.2. Interface: http/REST

The interface to the DECTRIS detector system is defined through its protocol. The protocol is based on the http/REST framework. This definition helps to cleanly isolate the detector system. Thus, no DECTRIS software is needed on the user control computer. The main idea behind the http/RESTful interface is the following:

- A configuration parameter, a status message, a detector command, etc. correspond to a RESTful resource. Each resource has an URL.
- A user can perform **get** or **put** operations on the URL. Some resources, for example FileWriter files, can also be **deleted**. If a resource can be deleted it's mentioned in the specific key remarks.
  - A **get** request returns the current value of the configuration parameter.
  - A **put** request sets the value of a configuration parameter.
  - A **delete** request deletes the resource.
- For all commands that don't require any inputs (e.g. initialize) no JSON body should be sent. The only accepted empty body is {}.
- Every configuration parameter has a corresponding data type (e.g. float or string). The data type in a **put** request must agree. The type of the value of a parameter can be requested with a **get** operation.
- For every configurable parameter with numeric data type, the minimum and maximum value can be requested if available. For enumerated data types, the available values can be requested.

[1] The detector will disarm after any triggered (*trigger_mode*: ints, exts, ..) series has been completed.

- Any **put** request changing parameters may implicitly change dependent parameters. **Put** will always return a list of all parameters implicitly and explicitly changed, or that could have been changed.
- If an invalid resource is requested, an HTTP error code is returned.
- The serialization format of the configuration parameter values is, by default, in the JSON format. The syntax is described below. Larger datasets can be received in the hdf5 format.

3.2.1. URLs

To represent the resources of the SIMPLON API, URLs are used:

Table 3.2: API Modules and respective URLs

Detector (section 5.1)	Configuration of the detector and the readout system, control of data acquisition and requesting the detector status *http://<address_of_dcu>/detector/api/1.8.0/*
Monitor (section 5.2)	Receiving single frames at a low rate. *http://<address_of_dcu>/monitor/api/1.8.0/*
FileWriter (section 5.3)	Configuration of the HDF5 FileWriter. *http://<address_of_dcu>/filewriter/api/1.8.0/*
Stream (section 5.4)	Configuration of the stream interface. *http://<address_of_dcu>/stream/api/1.8.0/*
System (section 5.5)	Configuration and control of the system. *http://<address_of_dcu>/system/api/1.8.0/*

The URLs to configure the detector, to send a command to the detector and to request its status are:

```
[]$_ URLs

http://<ADDRESS_OF_DCU>/detector/api/<VERSION>/config
http://<ADDRESS_OF_DCU>/detector/api/<VERSION>/command
http://<ADDRESS_OF_DCU>/detector/api/<VERSION>/status
```

A configuration parameter resource has the following URL:

```
[]$_ URLs

http://<ADDRESS_OF_DCU>/detector/api/<VERSION>/config/<parameter_name>
```

For get requests, the image format can be chosen with the header item:

```
[]$_ URLs

accept=<format>
```

Possible formats are JSON and, for data arrays tiff. (MIME types *application/json* and *image/tiff*). The header item "*content-type*" is set respectively in all responses.

Information	#5

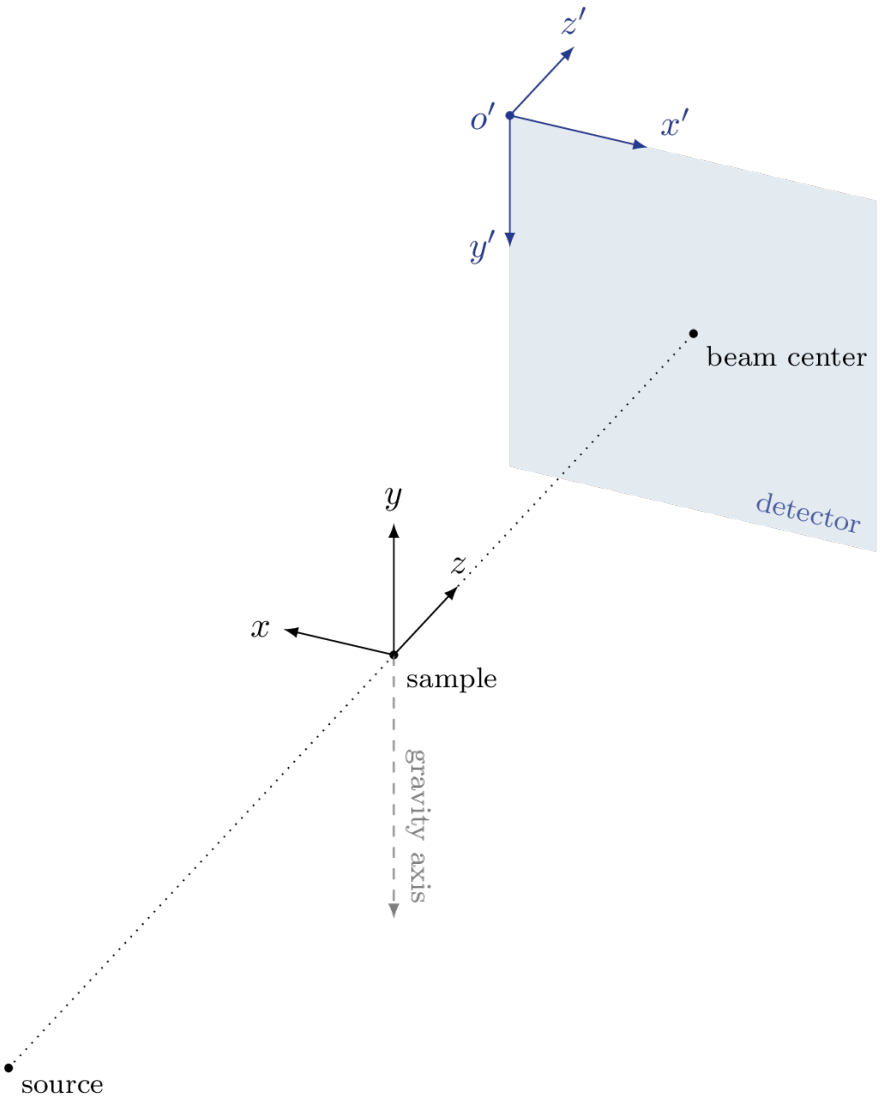
Default format is *application/json* for requests. For small datasets *application/json* is recommended. For larger datasets, in particular 2d arrays (ie. *flatfields* and *pixel_masks*), only *image/tiff* is supported, other MIME types may provide experimental access.

4. GEOMETRY

This section is about the geometry used in the software to describe the position of components of the experiment.

4.1. Coordinate Systems

There are two coordinate systems of interest, the Lab Coordinate System (x, y, z in the image below) and the Detector Coordinate System (x', y', z' in the image below).



4.1.1. Lab Coordinate System

For the lab coordinate system we use the same conventions that are used for the NeXus coordinate system and matches what is used by McStas.
The origin is positioned in the center of the sample, the direction of the z-axis is along the incident beam, the y-axis points upwards (in opposite gravitation direction) and the x-axis is in the horizontal plane pointing left as seen from the sample, thus completing a right-handed coordinate system, see image above.

4.1.2. Detector Coordinate System

The second coordinate system of interest is the detector coordinate system, it is also used for certain metadata. One example is the beam center, which is given in pixel coordinates on the detector.
The detector coordinate system is defined as follows. The origin is in the upper left corner of the sensor area, the x-axis is pointing along the fast pixel direction, the y-axis along the slow pixel direction, and the z-axis is defined to complete the right-handed coordinate system.
The coordinate transformation from detector coordinates to lab coordinates is given by a rotation **R** (*detector_orientation* [1]) and a translation **t** (detector_translation). The defaults are what is shown in the image above, i.e. a rotation by 180 degrees around the z-axis, and a translation by detector_distance <detector_distance along the z-axis.
For a pixel with address (i, j), the corresponding point in detector coordinates can be determined as follows. Let $\mathbf{f}^{'}$ be the vector between the (upper left corners of the) two pixels $(0, 0)$ and $(1, 0)$, i.e. the fast direction, and $\mathbf{s}^{'}$ the vector between the (upper left corners of the) two pixels $(0, 0)$ and $(0, 1)$, the slow direction, both given in detector coordinates. Then the detector coordinates of a pixel (i, j) are given by

$$\mathbf{P}^{'} = i \cdot \mathbf{f}^{'} + j \cdot \mathbf{s}^{'}.$$

For a point $P^{'}$ in the detector coordinate system, the corresponding point P in lab coordinates is given by

$$\mathbf{P} = \mathbf{R} \cdot \mathbf{P}^{'} + \mathbf{t},$$

where $\mathbf{R} \in \mathbb{R}^{3x3}$ is the rotation matrix and $\mathbf{t} \in \mathbb{R}^3$ the translation vector mentioned above.
Hence, the position of a pixel in the lab can be calculated by combining the two steps above.

4.2. API Settings and Conventions

There are multiple settings available in the SIMPLON API that affect the geometry of an experiment. That implies that by changing one of those settings, other values will be (automatically) updated as well.
Namely, these settings are:

- *detector_orientation*
- *detector_orientation_axis*
- *detector_orientation_angle*
- *detector_translation*
- *detector_distance*
- *beam_center_x*
- *beam_center_y*

Let $\mathbf{C}^{'} = \begin{pmatrix} c_0 \\ c_1 \\ 0 \end{pmatrix}$ denote the beam center in detector coordinates and d the distance between the sample and the detector, then the above values are related through

$$\begin{pmatrix} 0 \\ 0 \\ d \end{pmatrix} = \mathbf{R} \cdot \mathbf{C}^{'} + \mathbf{t},$$

[1] Note that there are two different ways to set the rotation via API. The first one is *detector_orientation*, input are only the first two columns of the 3×3 rotation matrix **R**. The third column is already uniquely defined by the first two columns and the fact that we are using a right-handed system. The second option is an axis-angle representation, using the API keys *detector_orientation_axis* and *detector_orientation_angle*.

where **R** and **t** denote the detector_orientation and the detector_translation (see also Detector Coordinate System).

Let $\mathbf{R} = \begin{pmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{pmatrix}$ and $\mathbf{t} = \begin{pmatrix} t_0 \\ t_1 \\ t_2 \end{pmatrix}$, then the above expands to

$$
\begin{aligned}
0 &= r_{00}c_0 + r_{01}c_1 + t_0 \\
0 &= r_{10}c_0 + r_{11}c_1 + t_1 \\
d &= r_{20}c_0 + r_{21}c_1 + t_2.
\end{aligned}
$$

Whenever the beam center or the detector distance is changed, we use the convention that **R** is kept constant and **t** is computed from

$$
\begin{aligned}
t_0 &= -r_{00}c_0 - r_{01}c_1 \\
t_1 &= -r_{10}c_0 - r_{11}c_1 \\
t_2 &= d - r_{20}c_0 - r_{21}c_1.
\end{aligned}
$$

Whenever **R** or **t** changes, the beam center and the distance are updated as follows:

$$
\begin{aligned}
c_0 &= \frac{r_{01}t_1 - r_{11}t_0}{r_{00}r_{11} - r_{01}r_{10}} \\
c_1 &= \frac{r_{10}t_0 - r_{00}t_1}{r_{00}r_{11} - r_{01}r_{10}} \\
d &= t_2 + r_{20}c_0 + r_{21}c_1
\end{aligned}
$$

5. SIMPLON API

Caution #2

Undocumented keys might be available in all modules. Using those keys is strongly discouraged. Undocumented features are subject to change. No official support is provided for undocumented features and no warranties are provided for the functionality of such features.

5.1. Detector Subsystem

The detector subsystem has the base URL:

```
[]$_ URLs

http://<ADDRESS_OF_DCU>/detector/api/<VERSION>
```

It is used to configure the detector, to request its status and send control commands.

5.1.1. Detector Configuration Parameters

The user can set the parameters listed below. The base path to the resource is always:

```
[]$_ URLs

<uri> = http://<ADDRESS_OF_DCU>/detector/api/<VERSION>/config/<PARAMETER>
```

Table 5.1: Detector Config Parameters

Parameter	Type	Access	Remarks
auto_sum_strict	bool	rw	This setting is required for backward compatibility with EIGER1 systems. For EIGER1 the default setting is *False*, all other detectors the default setting is *True*. See release notes (release-2024.1) for more details.
auto_summation	bool	rw	Enables (*True*) or disables (*False*) autosummation. Should always be enabled.
beam_center_x	float	rw	Beam position on detector in pixels.
beam_center_y	float	rw	Beam position on detector in pixels.

Table 5.1: Detector Config Parameters - continued

Parameter	Type	Access	Remarks
binning_mode	string	rw	Defines how the physical pixels are being binned. This changes the behavior of the readout! Available binning modes can be requested by sending a GET request to the key *binning_mode*. [1]. **Caution #3** ⚠ The binning mode can only be changed if the already configured frame rate is within the allowed bounds.
bit_depth_image	uint	r	Bit depth of generated images.
bit_depth_readout	uint	r	Bit depth of the internal readout.
chi_axis	float[]	rw	Chi rotation axis given as a three-dimensional unit vector in a right-handed system.
chi_increment	float	rw	Chi increment per frame.
chi_start	float	rw	Chi start angle (start angle of the first frame) for an exposure series.
compression	string	rw	Defines the compression algorithm used. Allowed options are *lz4* and *bslz4*. **Information #6** To ensure highest stability at full frame rates, DECTRIS strongly advises using *bslz4* compression. For enabling and disabling compression see section 5.3.1 FileWriter Configuration *compression_enabled*).
count_time	float	rw	Exposure time per image.
counting_mode	string	rw	Switch between normal and retrigger counting mode. Other counting modes might be available for special applications.
countrate_correction_applied	bool	rw	Enables (*True*) or disables (*False*) countrate correction. Should always be enabled. See the User Manual for details.
countrate_correction_count_cutoff	uint	r	Maximum number of possible counts per image after count rate correction.

[1] Only available for certain detectors. Please refer to the User Manual and Technical specifications for further details.

Table 5.1: Detector Config Parameters - continued

Parameter	Type	Access	Remarks
countrate_correction_table	uint[]	r	Returns the countrate correction table.
data_collection_date	string	r	Date and time of data collection. This is the time when the ARM command was issued.
description	string	r	Detector model and type.
detector_distance	float	rw	Sample to detector distance.
detector_number	string	r	Detector serial number.
detector_orientation	float[]	rw	Together with *detector_translation*, this describes the coordinate transformation from detector coordinates to lab coordinates or the position of the detector in the lab. *detector_orientation* consists of the first two columns of the rotation matrix that describes how the detector is oriented in the lab. Alternatively, the orientation can be given in axis-angle representation via *detector_orientation_axis* and *detector_orientation_angle*.
detector_orientation_angle	float	rw	Together with *detector_orientation_axis*, this describes the rotation part of the coordinate transformation from detector coordinates to lab coordinates or the position of the detector in the lab, the angle is given in degrees, a right-handed system is assumed. The translation part of the coordinate transformation is given by *detector_translation*. Alternatively, the orientation can be given in matrix representation via *detector_orientation*.
detector_orientation_axis	float[]	rw	Together with *detector_orientation_angle*, this describes the rotation part of the coordinate transformation from detector coordinates to lab coordinates or the position of the detector in the lab, a right-handed system is assumed. The translation part of the coordinate transformation is given by *detector_translation*. Alternatively, the orientation can be given in matrix representation via *detector_orientation*.
detector_readout_time	float	r	Readout dead time between consecutive detector frames.
detector_translation	float[]	rw	Together with the *detector_orientation* or *detector_orientation_axis* and *detector_orientation_angle*, this describes the coordinate transformation from detector coordinates to lab coordinates or the position of the detector in the lab.

Table 5.1: Detector Config Parameters - continued

Parameter	Type	Access	Remarks
eiger_fw_version	string	r	Returns the currently running app-detector version.
element	string	rw	Sets parameter *photon_energy* to the K-alpha fluorescence radiation energy of an element.
extg_mode	string	rw	Can only be used when in *extg*[2] trigger mode. Values can either be *double* to make full use of both internal counters or *single* to acquire 32 bit images for high dynamic range measurements. See the User Manual for details.
flatfield	float[][]	rw	Flatfield correction factors used for flatfield correction. Pixel data are multiplied with these factors for calculating flatfield corrected data. This key is mapped to threshold/1/flatfield. **Information #7** Customer specific flatfields can be uploaded after setting the *threshold_energy* and *photon_energy*. The custom flatfield is lost after changing either of the mentioned settings. The default flatfield can also be restored with a delete request.
flatfield_correction_applied	bool	rw	Enables (*True*) or disables (*False*) flatfield correction. Should always be enabled.

[2] Only available for certain detectors. Please refer to the User Manual and Technical specifications for further details.

Table 5.1: Detector Config Parameters - continued

Parameter	Type	Access	Remarks
flux_type	string	rw	Type of flux, the flux value is set via flux_value. The following types can be set: • *"flux"*: Flux density incident on beam plane area in photons per second per unit area. • *"flux_area_integrated"*: Flux incident on beam plane in photons per second. • *"flux_time_integrated"*: Flux density incident on beam plane area in photons per unit area. • *"flux_area_and_time_integrated"*: Flux incident on beam plane in photons. An empty string will be interpreted as not set. This information is required by the NXmx application definition [3] **Information #8** Part of FileWriter metadata only if *format* is set to *"hdf5 nexus v2024.2 nxmx"*
flux_value	float	rw	Flux value of type *flux_type*. In case of a beam that varies in flux shot-to-shot, this is an array of values, one for each recorded shot. This information is required by the NXmx application definition [4] **Information #9** Part of FileWriter metadata only if *format* is set to *"hdf5 nexus v2024.2 nxmx"*
frame_count_time	float	r	Time interval between start of image acquisitions for sub-images when *auto_summation* is enabled
frame_time	float	rw	Time interval between start of image acquisitions. This defines the speed of data collection and is the inverse of the frame rate, the frequency of image acquisition.

[3] see https://manual.nexusformat.org/classes/applications/NXmx.html
[4] see https://manual.nexusformat.org/classes/applications/NXmx.html

Table 5.1: Detector Config Parameters - continued

Parameter	Type	Access	Remarks
incident_energy	float	rw	Energy of incident particles (e.g. X-rays) in eV.
instrument_name	string	rw	Name of the instrument or beamline. Consistency with the controlled vocabulary beamline naming [5] is highly recommended. This information is required by the NXmx application definition [6] , an empty string will be interpreted as not set. **Information #10** Part of FileWriter metadata only if *format* is set to *"hdf5 nexus v2024.2 nxmx"*
kappa_axis	float[]	rw	Kappa rotation axis given as a three-dimensional unit vector in a right-handed system.
kappa_increment	float	rw	Kappa increment per frame.
kappa_start	float	rw	Kappa start angle (start angle of the first frame).
mask_to_zero	bool	rw	If set to *True*, pixels marked in the *pixel_mask* will be set to zero instead of the maximum representable value.This may be preferred by some analysis software. Default: *False*.
nexpi	uint	rw	Number of exposures per image. Only applicable during *extg* trigger mode. See the User Manual for details.
nimages	uint	rw	Number of images. See the User Manual for details.
ntrigger	uint	rw	Number of triggers. See the User Manual for details.
ntriggers_skipped	uint	rw	Number of triggers to be skipped in eies trigger_mode. See the User Manual for details.
number_of_excluded_pixels	uint	r	Total number of defective, disabled or inactive pixels.
omega_axis	float[]	rw	Omega rotation axis given as a three-dimensional unit vector in a right-handed system.
omega_increment	float	rw	Omega increment per frame.

[5] see https://mmcif.wwpdb.org/dictionaries/mmcif_pdbx_v50.dic/Items/_diffrn_source.pdbx_synchrotron_beamline.html and https://mmcif.wwpdb.org/dictionaries/mmcif_pdbx_v50.dic/Items/_diffrn_source.type.html

[6] see https://manual.nexusformat.org/classes/applications/NXmx.html

Table 5.1: Detector Config Parameters - continued

Parameter	Type	Access	Remarks
omega_start	float	rw	Omega start angle (start angle of the first frame).
phi_axis	float[]	rw	Phi rotation axis given as a three-dimensional unit vector in a right-handed system.
phi_increment	float	rw	Phi increment per frame.
phi_start	float	rw	Phi start angle (start angle of the first frame).
photon_energy	float	rw	Energy of incident X-rays.
pixel_format	string	rw	Defines which data format is returned from the module. The maximum performance of the detector is depending on the selected pixel format [7]. Available pixel formats can be requested by sending a GET request to the key *pixel_format*. **Caution #4** ⚠ The pixel format can only be changed if the configured frame rate is within the allowed bounds of the new format.

[7] Only available for certain detectors. Please refer to the User Manual and Technical specifications for further details.

Table 5.1: Detector Config Parameters - continued

Parameter	Type	Access	Remarks
pixel_mask	uint[][]	rw	This key is mapped to threshold/1/pixel_mask. A bit mask that labels and classifies pixels which are either defective, inactive or exhibit non-standard behavior Bit 0: gap (pixel with no sensor) Bit 1: dead Bit 2: under responding Bit 3: over responding Bit 4: noisy Bit 5-31: -undefined- **Information #11** Please note that the actual integer value of a pixel in the mask depends on which bits are set, e.g. a dead pixel has the value 2^1=2 and an over responding pixel 2^3=8. The value can be summed up due to multiple classification. Any non-zero value in the mask has the same effect on the data. A custom pixel_mask can be persistently upload with a put request. Removing it requires a delete request, which restores the default *pixel_mask*.
pixel_mask_applied	bool	rw	Enables (*True*) or disables (*False*) applying the pixel mask on the acquired data. If *True* (default), pixels that have a corresponding bit set in the *pixel_mask* are flagged with (2^*bit_depth_image*)-1. If disabled, the pixel mask needs to be applied at the point of data processing.
roi_bit_depth	uint	rw	Bit depth of generated images when using *lines* ROI [8]. 8 Bit required for max frame rate acquisitions!

[8] Only available for certain detectors. Please refer to the User Manual and Technical specifications for further details.

Table 5.1: Detector Config Parameters - continued

Parameter	Type	Access	Remarks
roi_mode	string	rw	Selects the region of interest (ROI). ROI modes enable higher frame rates due to reduced readout area [9]. Available ROI modes can be requested by sending a GET request to the key *roi_mode*.When ROI is set to *"disabled"*, the entire active area is read out. **Caution #5** ⚠ Always configure the ROI mode first before setting any acquisition specific configurations. For example count_time and frame_time are reset to default after setting a ROI mode!
roi_y_size	uint	rw	Defines total height (y size) of output image when using *lines* ROI [10]
sample_name	string	rw	Name of the sample. This information is required by the NXmx application definition [11], an empty string will be interpreted as not set. **Information #12** Part of FileWriter metadata only if *format* is set to *"hdf5 nexus v2024.2 nxmx"*
sensor_material	string	r	Material used for direct detection of X-rays in the sensor.
sensor_movement_mode	string	rw	Controls sensor lance movement permission[12]. Default: *"insertion_allowed"* • *"insertion_allowed"*: Enables external control and permits insertion. • *"insertion_disallowed"*: Retracts the sensor if inserted and prevents insertion.
sensor_thickness	float	r	Thickness of the sensor material.
software_version	string	r	Software version used for data acquisition and correction.

[9] Only available for certain detectors. Please refer to the User Manual and Technical specifications for further details.
[10] Only available for certain detectors. Please refer to the User Manual and Technical specifications for further details.
[11] see https://manual.nexusformat.org/classes/applications/NXmx.html
[12] Only available for certain detectors. Please refer to the User Manual and Technical specifications for further details.

Table 5.1: Detector Config Parameters - continued

Parameter	Type	Access	Remarks
source_name	string	rw	Name of the neutron or x-ray storage ring/-facility. Consistency with the naming in synchrotoron controlled vocabulary [13] is highly recommended. This information is required by the NXmx application definition [14], an empty string will be interpreted as not set. **Information #13** Part of FileWriter metadata only if *format* is set to *"hdf5 nexus v2024.2 nxmx"*
test_image_mode	string	rw	A "non-empty" string value enables a test image readout and selects the type of the test image. The following test image modes are supported by all systems: "value": A simulated readout which takes the value from *test_image_value* and includes it in every pixel of the image data. The data is injected in the firmware on the MCB and will be transferred through the image pipeline as normal image data. In order to get images with the configured counts *auto_summation* has to be disabled. "cal_pulse": Calibration pulses are injected into the analog unit of the ROC, simulating an entering particle from the sensor. The signal will be transferred though the ROC just as a particle signal would do and be counted by the active counter. This test mode takes the *test_image_value* as the number of pulses which will be injected. When this mode is selected, the exposure period should be minimal and without radiation as the calibration pulses pre-expose the image during arm. *counting_mode* should be set to "normal" and *auto_summation* should be disabled. "mcb_id": A simulated readout with a unique pixel value for every MCB. This mode ignores *test_image_value*. Some detectors support other types of simulated readout like "chip", "chip_quadrant" and "pattern".

[13] see https://mmcif.wwpdb.org/dictionaries/mmcif_pdbx_v50.dic/Items/_diffrn_source.pdbx_synchrotron_site.html
[14] see https://manual.nexusformat.org/classes/applications/NXmx.html

Table 5.1: Detector Config Parameters - continued

Parameter	Type	Access	Remarks
test_image_value	uint	rw	Value to configure some test image modes.
threshold_energy	float	rw	Threshold energy for X-ray counting. Photons with an energy below the threshold are not detected. See the User Manual for details. **Information #14** For detectors with more than one threshold, this resource is mapped to the resource threshold/1/energy.[a] [a] Only available for certain detectors. Please refer to the User Manual and Technical specifications for further details.
threshold/n/energy	float	rw	Threshold energy for X-ray counting for threshold n. Energies have to be increasing with the threshold number n. Example: threshold/1 < threshold/2 < threshold/n-1 < threshold/n [15]
transformation_order	string[]	rw	Order of execution for goniometer rotation axes. Allowed values are *chi*, *kappa*, *omega* and *phi*. Let *transformation_order* = [*kappa*, *chi*] and R_{kappa}, R_{chi} the corresponding transformation matrices, then the combined transformation would be R_{chi} R_{kappa}. **Information #15** The corresponding fields *_axis*, *_increment* and *_start* should be set for all axes in *transformation_order*.

[15] Only available for certain detectors. Please refer to the User Manual and Technical specifications for further details.

Table 5.1: Detector Config Parameters - continued

Parameter	Type	Access	Remarks
threshold/n/flatfield	float[][]	rw	Flatfield correction factors used for flatfield correction. Pixel data are multiplied with these factors for calculating flatfield corrected data. **Information #16** Customer specific flatfields can be uploaded after setting the *threshold_energy* and *photon_energy*. The custom flatfield is lost after changing either of the mentioned settings. The default flatfield can also be restored with a delete request.
threshold/n/mode	string	rw	Image data acquisition for threshold n, which can be 'enabled' or 'disabled'. When disabled, no image data from this threshold will be written to file or delivered from any other data interface.
threshold/n/number_of_excluded_pixels	uint	r	Total number of defective, disabled or inactive pixels for threshold n.
threshold/n/pixel_mask	uint[][]	rw	A bit mask that labels and classifies pixels which are either defective, inactive or exhibit non-standard behavior Bit 0: gap (pixel with no sensor) Bit 1: dead Bit 2: under responding Bit 3: over responding Bit 4: noisy Bit 5-31: -undefined- **Information #17** Please note that the actual integer value of a pixel in the mask depends on which bits are set, e.g. a dead pixel has the value 2^1=2 and an over responding pixel 2^3=8. The value can be summed up due to multiple classification. Any non-zero value in the mask has the same effect on the data. A custom pixel_mask can be persistently upload with a put request. Removing it requires a delete request.

Table 5.1: Detector Config Parameters - continued

Parameter	Type	Access	Remarks
threshold/difference/mode	string	rw	Calculation of difference images, which can be 'enabled' or 'disabled'. When disabled, no difference images will be written to file or delivered from any other data interface. At the moment the difference images is always calculated for threshold 1 and 2 (diff = th1 - th2). Only available for detectors with more than one threshold. **Information #18** If threshold/difference/mode is to be used, threshold 1 and 2 (see threshold/n/mode) have to be enabled beforehand.
threshold/difference/lower_threshold	uint	r	Lower threshold used for calculating difference images. Difference images are calculated by subtracting the counts of the upper threshold image from the counts of the lower threshold image. Only available for detectors with more than one threshold.
threshold/difference/upper_threshold	uint	r	Upper threshold used for calculating difference images. Difference images are calculated by subtracting the counts of the upper threshold image from the counts of the lower threshold image. Only available for detectors with more than one threshold.
trigger_mode	string	rw	Mode of triggering image acquisition. See the User Manual for details.
trigger_start_delay	float	rw	Delayed time from when the trigger comes until the configured series starts. Trigger Start Delay can be used in ints and exts mode.
two_theta_axis	float[]	rw	Two theta rotation axis given as a three-dimensional unit vector in a right-handed system.
two_theta_increment	float	rw	Two theta increment per frame.
two_theta_start	float	rw	Two theta start angle (start angle of the first frame).
virtual_pixel_correction_applied	bool	rw	Enables (*True*) or disables (*False*) applying the *virtual_pixel_correction* on the acquired data.
wavelength	float	rw	Wavelength of incident X-rays. See the User Manual for details.
x_pixel_size	float	r	Size of a single pixel along x-axis of the detector.

Table 5.1: Detector Config Parameters - continued

Parameter	Type	Access	Remarks
x_pixels_in_detector	uint	r	Number of pixels along x-axis of the detector.
y_pixel_size	float	r	Size of a single pixel along y-axis of the detector.
y_pixels_in_detector	uint	r	Number of pixels along y-axis of the detector.

JSON Serialization

Meta information in the body of the request and in the reply from the HTTP server are serialized in the JSON format and described in the table below.

The returned JSON of a **get** request string contains a subset of the fields below. Only fields which are applicable for a given resource are present in the returned JSON. For hdf5 objects, the JSON metadata is stored with hdf5 attributes.

Table 5.2: Key Value Pairs for Detector Config Parameters (GET)

JSON Key	JSON Value	Description
”value”	<parameter_value>	The value of the configuration parameter. Data type can be int, float, string or a list of int or float. Two-dimensional arrays are returned as darrays (see text below). Invalid or unknown values are represented as “null” or as empty string, list, or array.
”value_type”	<string>	Returns the data type of a parameter. Data types are bool, float, int, string or a list of float or int. Invalid or unknown values are represented as “null” or as empty string, list, or array.
”min”	<minimal_parameter _value>	Returns the minimum of a parameter (for numerical data types).
”max”	<maximal_parameter _value>	Returns the maximum of a parameter (for numerical data types).
”allowed_values”	<list_of_allowed _values>	Returns the list of allowed values. An empty list indicates there are no restrictions.
”unit”	<string>	The unit of the parameter.
”access_mode”	<string>	String, describing read, and/or write access to resource. When not available, the *access_mode* is ”rw”.

Put requests send a body serialized in the JSON format. The HTTP header item *content-type* must be set appropriately. The JSON string may contain the following keywords:

Table 5.3: Key Value Pairs for Detector Config Parameters (PUT)

JSON Key	JSON Value	Description
”value”	<parameter_value>	The value of the configuration parameter. Data type can be int, float, string or a list of int or float. Two-dimensional arrays are returned as darrays (see text below).

The return body of a **put** request is:

Table 5.4: Return Body of PUT Requests

JSON Key	JSON Value	Description
None	<changed_parameters>	A list of all resources that are also affected by the put configuration parameter.

3
darray

Two-dimensional arrays (*pixel_mask*, *flatfield*) are exchanged as darrays as defined below:

”__darray__”: <VERSION>, ”type”: <type>, ”shape”: [<width>,<height>], ”filters”:[”base64”],”data”: <base 64 encoded data>
where <*VERSION*> is the darray version ([major, minor, patch]), <*type*> is either ”<u4” or ”<f4” (little endian encoded 4 byte unsigned int or float) and <*base 64 encoded data*> contains the base 64 encoded data.

Example - Setting photon_energy

As already mentioned, when setting a value, the DCU returns the names of the parameters that were explicitly or implicitly changed to maintain a consistent detector configuration in the reply to the set request.
The following example of setting the photon energy to 8040 eV uses python libraries as a web client to send HTTP requests:

[]$_ Python Code

```
import json
# Imports "JSON" library
import requests
# Imports "requests" library
dict_data = {'value':8040.0}
# Prepare the dictionary (a "value" with the value 8040.0)
data_json = json.dumps(dict_data)
# Convert the dictionary to JSON
r = requests.put('http://<address_of_dcu>/detector/api/<version>/config/photon_energy', data
    =data_json)
# Execute the request on the config value "photon_energy" (REPLACE <ADDRESS_of_DCU> and <
    VERSION> with the values of YOUR system)
print(r.status_code)
# Print the http status code (NOTE: Only http code 200 is OK, everything else is an error)
print(r.json())
# Print the returned JSON string. (Containing the names of the subsequently changed values)
```

The code will return the following output (clipped for readability):

[]>_ Return Value

```
200
["threshold_energy", "flatfield", "element", ...]
```

The returned HTTP code ”200” indicates successful completion of the put request.

The JSON string ”[”threshold_energy”, ”flatfield”]” indicates that, resulting from the photon energy change, the threshold energy and the applied flatfield were also changed.

5.1.2. Detector Status Parameters

Status parameters are read only. The base path to the resource is:

[]$_ URLs

```
<uri> = http://<ADDRESS_OF_DCU>/detector/api/<VERSION>/status/<PARAMETER>
```

Status parameters are measured values which might change without user interaction. They represent the operational conditions.

Status Information

Table 5.5: Detector Status Parameters

Parameter	Type	Access	Remarks
board_000/th0_humidity	float	r	Caution #6 ⚠ This parameter is deprecated, please use *humidity*. Relative humidity reported by humidity sensor.
board_000/th0_temp	float	r	Caution #7 ⚠ This parameter is deprecated, please use *temperature*. Temperature reported by temperature sensor.
error	string[]	r	Always returns empty list. Only available for backwards compability

Table 5.5: Detector Status Parameters - continued

Parameter	Type	Access	Remarks
high_voltage/state	string	r	Returns high voltage state of target (*NA*, *OFF*, *RAMPING*, *READY*). **Information #19** For optimal data quality acquisitions should only be started when state is *READY*.
humidity	float	r	Humidity inside the detector module compartment.
sensor_movement_state	string	r	Sensor lance position state[16]. Possible values: *"retracted"*, *"inserted"*, *"moving"*, *"collision"*, *"unknown"*.
state	string	r	Possible states: na (not available), ready, initialize, configure, acquire, idle, test, error. **Information #20** State is "na", when the DCU is booted or the acquisition service was restarted.
temperature	float	r	Detector temperature.
time	string	r	Returns current system time. Formatted according to ISO 8601

JSON Serialization

Get requests have no body. The returned JSON of a **get** request string contains a subset of the fields below. Only fields which are applicable for a given resource are present in the returned JSON.

Table 5.6: Key Value Pairs for Detector Status Parameters (GET)

JSON Key	JSON Value	Description
"value"	<parameter_value>	The value of the configuration parameter. Data type can be single type or list of int, float or string. Invalid or unknown values are represented as "null" or as empty string, list, or array.
"value_type"	<string>	Returns the data type of a parameter.
"unit"	<string>	The unit of the parameter. The returned data type may only be valid if a valid value for this parameter is known.
"time"	<date>	Timestamp for when the value was updated.

[16] Only available for certain detectors. Please refer to the User Manual and Technical specifications for further details.

Table 5.6: Key Value Pairs for Detector Status Parameters (GET) - continued

JSON Key	JSON Value	Description
”state”	<state>	invalid, normal, critical, disabled
”critical_limits”	<list_containing_ minimal_and_maximal_ parameter_value>	Returns the minimum and maximum error threshold for a parameter if it is a numerical value type.
”critical_values”	<list_of_critical_ values>	Returns the list of values treated as error conditions. An empty list indicates there are no states causing an error condition.

5.1.3. Detector Command Parameters

Command parameters are write only. The base path to the resource is:

[]$_ URLs

```
<uri> = http://<ADDRESS_of_DCU>/detector/api/<VERSION>/command/<PARAMETER>
```

Table 5.7: Detector Command Parameters

Parameter	Return Value	Access	Remarks
abort	sequence_id: uint	w	Aborts all operations and resets the system **immediately**. All data in the pipeline will be dropped.
arm	sequence_id: uint	w	Loads configuration to the detector and arms the trigger unit.
cancel	sequence_id: uint	w	Stops the data acquisition, **but only after the next image is finished**.
check_connections	Array	w	Caution #8 ⚠ The detector needs to be re-initialized in order to resume operation! Issuing this command returns the state of the DATA interfaces along with additional readings where available.
disarm	sequence_id: uint	w	Writes all data to file and disarms the trigger unit.
hv_enabled	-	w	Turn high voltage on (*true*) or off (*false*). Information #21 Only available on specific detector types.
hv_reset	-	w	Resets the module high voltage for number of supplied seconds (range 1 - 600 seconds). Reset time is 30s if no time is supplied. Information #22 Should only be used with CdTe detectors. See the User Manual for details.

Table 5.7: Detector Command Parameters - continued

Parameter	Return Value	Access	Remarks
initialize	-	w	Initializes the detector. **Caution #9** ⚠ Before initializing the detector only the detector state is available!
insert_sensor	-	w	**Information #23** The *sensor_movement_mode* must be set to *insertion_allowed* and the *sensor_movement_state* must be checked before issuing this command. Inserts the sensor lance into the detection volume. [17]
retract_sensor	-	w	**Information #24** The *sensor_movement_state* must be checked before issuing this command. Retracts the sensor lance from the detection volume. [18] Can be issued to clear the *"collision"* state.
trigger	-	w	Starts data acquisition with the programmed trigger sequence. The trigger command can also accept an argument in the put request – the *count_time* - if used in *trigger_mode* inte (internal enable).

For parameters without a return value one should check whether a HTTP code 200 is returned. If an error occurs a corresponding HTTP error code is returned instead. In this case please create a bug report on the web interface of the DCU (Technical Support - Create Bug Report) and contact support@dectris.com.

[17] Only available for certain detectors. Please refer to the User Manual and Technical specifications for further details.
[18] Only available for certain detectors. Please refer to the User Manual and Technical specifications for further details.

5.2. Monitor Subsystem

The Monitor subsystem provides access to recent exposures over HTTP. When enabled, incoming exposures are inserted into a buffer (as space allows). Buffered images may either be retrieved directly (i.e. using their identifier) or in FIFO (first in, first out) order.

The buffer capacity is based on the *buffer_size* configuration parameter as well as the amount of internal allocated memory. The buffer is full when the number of exposures reaches *buffer_size* or there is not enough available buffer memory to store the next exposure (see *buffer_free*). In this case, incoming exposures will be dropped or cause older exposures to be dropped from the buffer as specified by *discard_new*. When exposures are dropped, the dropped status parameter is incremented. Note that the most recent exposure is always accessible through images/monitor, regardless if it is in the buffer or not.

The Monitor must be enabled before it can be used.

Note: The Monitor interface should be avoided in scenarios requiring high throughput. For frame rates above 10 Hz, use of either the FileWriter or Stream interface is advised.

The base URL for the Monitor API is:

[]$_ Python Code

```
http://<ADDRESS_OF_DCU>/monitor/api/<VERSION>
```

5.2.1. Monitor Configuration Parameters

The configuration is applied at the URL:

[]$_ URL

```
<uri> = http://<ADDRESS_OF_DCU>/monitor/api/<VERSION>/config/<PARAMETER>
```

with the following commands:

Table 5.8: Monitor Config Parameters

Parameter	Type	Access	Remarks
buffer_size	uint	rw	Specifies the maximum number of exposures that may be buffered by the Monitor. The effective size of the buffer may be smaller depending on the active configuration and amount of allocated memory (see *buffer_free*). Default: 100.
discard_new	bool	rw	When *true*, new exposures are dropped when the Monitor buffer is full. Shrinking the size of the buffer will evict (drop) newer exposures first. When *false*, older exposures will be evicted (dropped) when the buffer is full or shrunk. Default: *true*.

Table 5.8: Monitor Config Parameters - continued

Parameter	Type	Access	Remarks
mode	string	rw	Operational mode of the Monitor. When set to *enabled*, the Monitor processes incoming exposures. When set to *disabled*, incoming exposures are ignored and do not affect the Monitor. Default: *disabled*.

5.2.2. Data Access

A **get** request to the URL:

```
[]$_ URL

http://<ADDRESS_OF_DCU>/monitor/api/<VERSION>/images/
```

returns a list of all available frames.

```
[]$_ URLs

[[series, [id, id, ...]], ...]
```

During data taking, the frames can be accessed with a **get** operation at the URL:

```
[]$_ URL

http://<ADDRESS_of_DCU>/monitor/api/<VERSION>/images/<series>/<id>/<threshold_id>
```

Information #25

Whether multiple threshold IDs are available depends on the amount of thresholds enabled using *threshold/n/mode*

The images will be returned in .tif format. If a requested image is not available, the request will return *HTTP 404 Not Found* error. The following parameters allow special frames to be accessed in a similar way.

```
[]$_ URL

<uri> = http://<ADDRESS_OF_DCU>/monitor/api/<VERSION>/images/<PARAMETER>
```

Table 5.9: Monitor Images Parameters

Parameter	Type	Access	Remarks
monitor	tif	r	Gets the last (newest) exposure. Default waits for 500 ms for image. Timeout adjustable by adding query string *?timeout=<ms>* to URI. Returns 408 if no image is available.
next	tif	r	Gets the oldest image (next in order) and removes it from the buffer. Default waits for 500 ms for image. Timeout via *?timeout=<ms>* adjustable. Returns 408 if no image available.

Tiff Header Data

The TIFF images include meta data listed in the table below. The information is available by accessing the DECTRIS private TIFF tag at 51192 (*hex: C7F8*).
Following Image File Directory (IFD) Tags are currently available:

Table 5.10: DECTRIS IFD Tags

Code (Hex)	Name	Type	Count	Description
0000	IfdVersion	Long	1	Indicates the Dectris IFD version. Currently 0.
0001	SeriesUniqueId	ASCII	N	The unique identifier of the series. Example: 01DC3XQ8ENM14PYCDHQ9XVZZJ8
0002	SeriesNumber	LONG	1	The series number (id).
0003	ImageNumber	LONG	1	The image number (id) in the series.
0004	ImageDateTime	ASCII	N	The date and time when the original image data was acquired. The format is RFC 3339. Example: 2019-05-14T10:59:56.000000000Z
0005	ThresholdId	SHORT	N	The threshold identifier(s). For difference mode there are two threshold ids (N=2).
0006	ThresholdEnergy	DOUBLE	N	Threshold energy in eV. For difference mode, the two threshold energies used (N=2). Otherwise, the single threshold energy corresponding to the threshold id (N=1).
0007	ExposureTime	DOUBLE	1	The elapsed exposure (counting) time of the image. This field has the same meaning as Nexus NXDetector.count_time.
0009	IncidentEnergy	DOUBLE	1	The configured energy of incident particles (e.g. X-rays) in eV.
000A	IncidentWavelength	DOUBLE	1	The configured wavelength of incident photons in Angstrom. [OPTIONAL]

Table 5.10: DECTRIS IFD Tags - continued

Code (Hex)	Name	Type	Count	Description
0012	LostPixelCount	LONG	1	The number of pixels lost due transmission (network) errors.
0016	BeamCenter	DOUBLE	2	Center of the beam in pixels (BeamCenterX, BeamCenterY).
0017	DetectorDistance	DOUBLE	1	Distance from the sample to the detector. [OPTIONAL]

5.2.3. Monitor Status Parameters

The status of the monitor can be requested at the address:

[]$_ URLs

```
<uri> = http://<ADDRESS_of_DCU>/monitor/api/<VERSION>/status/<PARAMETER>
```

Table 5.11: Monitor Status Parameters

Parameter	Type	Access	Remarks
buffer_fill_level	uint[2]	r	An array(2) containing the current and maximum number of exposures in the buffer (see *buffer_size*).
buffer_free	uint	r	The remaining buffer space in bytes. Note: The total available space in bytes is based on the amount of allocated memory and is unaffected by *buffer_size*.
dropped	uint	r	Number of images which were dropped as not requested.
error	string[]	r	Unused.
state	string	r	The value *overflow* if images were dropped due to a full buffer, otherwise the value *normal*.

5.2.4. Monitor Command Parameters

To clear the buffer of images, the following command can be executed at the address *http://<address_of_dcu>/monitor/api/1.8.0/command/clear*

```
[]$_ URLs

<uri> = http://<ADDRESS_of_DCU>/monitor/api/<VERSION>/command/<PARAMETER>
```

Table 5.12: Monitor Command Parameters

Parameter	Return Value	Access	Remarks
clear	-	w	Clears the image buffer. *dropped* is reset to zero and state is reset to *normal*. Note: The last exposure is still accessible via *images/monitor*.
initialize	-	w	Resets the Monitor to its default state. All exposures are discarded from the buffer.

5.3. FileWriter Subsystem

The Filewriter subsystem writes series data to files which are accessible through the SIMPLON API. Files are written as HDF5 and follow the NeXus/HDF5 NXmx application definition which is compatible with major data-processing software.
The base URL for the FileWriter is:

```
[]$_ URLs

http://<ADDRESS_OF_DCU>/filewriter/api/<VERSION>/
```

5.3.1. FileWriter Configuration Parameters

To configure the FileWriter, this URL is used:

```
[]$_ URL

<uri> = http://<ADDRESS_OF_DCU>/filewriter/api/<VERSION>/config/<PARAMETER>
```

Table 5.13: Filewriter Config Parameters

Parameter	Type	Access	Remarks
compression_enabled	bool	rw	Optionally disable (*false*) compression of image data. **Information #26** Compression is required for full detector performance, disabling compression may lead to data loss at high frame rates. For compression modes see section 5.1.1 Detector Configuration Parameters (compression).
format	string	rw	Select the format of the files written by FileWriter. • *"hdf5 nexus legacy nxmx"*: HDF5 files where the metadata of the master file is following the NXmx application definition of the NeXus 3.2 (2016) standard. • *"hdf5 nexus v2024.2 nxmx"*: HDF5 files where the metadata of the master file is following the NXmx application definition of the NeXus v2024.2 standard.
image_nr_start	uint	rw	Sets the *image_nr_low* metadata parameter in the first HDF5 data file *<name_pattern>_data_000001.h5*. This parameter is useful when a data set is collected in more than one HDF5 file structures. If you collect image number *m* to *n* in the first file structure, you can set *image_nr_start* to *n+1* in the subsequent file structure.
mode	string	rw	Operation mode of the FileWriter, which can be *enabled* or *disabled*. During an active acquisition the interface remains enabled and will be disabled after receiving the end of series message.

Table 5.13: Filewriter Config Parameters - continued

Parameter	Type	Access	Remarks
name_pattern	string	rw	The basename of the file. The pattern *$id* will include the sequence number in the file name. *series_$id* is the default name pattern, resulting in the following names of the HDF5 file structure created by the FileWriter: *series_<sequence_nr>_master.h5*, *series_<sequence_nr>_data_<filenr>.h5* **Caution #10** ⚠ The FileWriter will overwrite existing files with identical names of the files to be written.
nimages_per_file	uint	rw	Maximum number of images stored in each *<name_pattern>_data_<file_nr>.h5* file in the HDF5 file structure created by the FileWriter. Data files are only created when *<nimages_per_file>* is reached or when detector command *disarm* is sent. No data files are created and all images are stored in *<name_pattern>_master.h5* when this parameter is set to 0. **Caution #11** ⚠ Only set to 0 when collecting a small number of images.

5.3.2. Data Access

The files are created locally on the detector server and have to be transferred to the user computer. The master file is accessible at the URL:

```
[]$_ URL

http://<ADDRESS_OF_DCU>/data/<name_pattern>_master.h5
```

and the data files at:

```
[]$_ URL

http://<ADDRESS_OF_DCU>/data/<name_pattern>_data_<filenr>.h5
```

A **get** request to the URL:

```
[]$_ URL

http://<ADDRESS_OF_DCU>/filewriter/api/<VERSION>/files/
```

returns a list of all available files.

5.3.3. FileWriter Status Parameters

The FileWriter is automatically started when data taking is started. The status of the FileWriter can be accessed at:

```
[]$_ URL

<uri> = http://<ADDRESS_OF_DCU>/filewriter/api/<VERSION>/status/<PARAMETER>
```

The following FileWriter status variables are accessible:

Table 5.14: Filewriter Status Parameters

Parameter	Type	Access	Remarks
buffer_free	uint	r	The remaining buffer space in Bytes.
error	string[]	r	Returns list of status parameters causing error condition.
files	string[]	r	Returns list of available files on the DCU.
state	string	r	Possible states: *disabled*, *ready*, *acquire*, *error*.

5.3.4. FileWriter Command Parameters

Command parameters are write only. The base path to the resource is:

```
[]$_ URL

<uri> = http://<ADDRESS_OF_DCU>/filewriter/api/<VERSION>/command/<PARAMETER>
```

Table 5.15: Filewriter Command Parameters

Parameter	Return Value	Access	Remarks
clear	-	w	Drops all data (image data and directories) on the DCU.
initialize	-	w	Resets the FileWriter to its original state.

5.4. Stream Subsystem

The SIMPLON API lets you configure and read out the status of the stream.

The base URL for the stream is:

[]$_ URL

```
http://<ADDRESS_OF_DCU>/stream/api/<VERSION>
```

5.4.1. Stream Configuration Parameters

To configure the stream, this URL is used:

[]$_ URL

```
<uri> = http://<ADDRESS_OF_DCU>/stream/api/<VERSION>/config/<PARAMETER>
```

Table 5.16: Stream Config Parameters

Parameter	Type	Access	Remarks
format	string	rw	Select the format of the event messages. • *legacy*: Use legacy multi-part JSON messages. • *cbor*: Use CBOR messages. See also https://github.com/dectris/documentation/tree/main/stream_v2
header_appendix	string	rw	User data to be included in start event messages. If *legacy* stream format is selected, data will be visible as string in *header_appendix* JSON message.

Table 5.16: Stream Config Parameters - continued

Parameter	Type	Access	Remarks
header_detail	string	rw	Select which fields are sent in the start messages: • *all*: All fields • *basic*: Exclude flatfield and pixel mask • *none*: No meta data **Caution #12** ⚠ If header_detail "*none*" is selected, no experimental meta-data is transferred, complicating processing and archiving of the data. Therefore, usage of header_detail "*none*" is discouraged.
image_appendix	string	rw	User data to be included in start event messages. If *legacy* stream format is selected, data will be visible as string in *image_appendix* JSON message.
mode	string	rw	State of the stream interface, either *enabled* or *disabled*.

Table 5.17: Stream Version Comparison

Version	Configuration	Transport	Port	Multipart	Encoding
V1	SIMPLON	ZeroMQ PUSH	9999	Yes	JSON and Binary
V2	SIMPLON	ZeroMQ PUSH	31001	No	CBOR

5.4.2. Stream V2 Messages

The latest StreamV2 documentation, along with examples, can be found on our github:

- https://github.com/dectris/documentation/tree/main/stream_v2/

Each Stream V2 message is sent as one ZeroMQ message. Messages are sent to clients using a ZeroMQ PUSH socket. Clients should connect to the server using a ZeroMQ PULL socket over TCP port *31001*. Multiple clients may connect to the same server simultaneously. However, each message is only ever sent to a single client using a round-robin strategy.

The table below describes the available messages:

Table 5.18: Stream V2 Messages

Message	Type	Description
Start	start	Sent at the start of each series (arm)
Image	image	Sent once per exposure
End	end	Sent at the end of each series (disarm)

Message Encoding

Stream V2 messages are encoded as CBOR (RFC 8949). Each message is serialized as a top-level CBOR map of text string keys to values of varying types (see "Map" below). Each such top-level map begins with the key type that specifies the *type* of message.

Data Model

The Stream V2 data model is based on the CBOR extended generic data model . The following table summarizes the types used by Stream V2:

Table 5.19: Data Model

Type	CBOR Encoding	Description
Any	(any)	Any valid data item
Array(T)	array	A sequence of zero or more data items of type T
Boolean	false/true	A boolean value
DateTime	tag(0) + text string	A standard date / time string defined in RFC 8949 section 3.4.1
Float	float	A floating-point value representable by IEEE 754 binary64
Integer	unsigned integer	An unsigned integer representable in 64 bits
Map	map	A mapping from text string keys each to a data item
Map(K -> V)	map	A mapping from keys of type K to values of type V
MultiDimArray	tag(40) + array	A multi-dimensional typed array defined in RFC 8746 section 3.1
Rational	array of two unsigned integers	A number expressed as the ratio of two integers. The numerator and denominator are represented as the first and second elements of an array, respectively. The rational number need not be in lowest terms

Table 5.19: Data Model - continued

Type	CBOR Encoding	Description
TextString	text string	A sequence of zero or more Unicode code points
TypedArray	tag + byte string	A typed array defined in RFC 8746 section 2

Compression

Compression is realized using CBOR tag 56500. This tag is equivalent to a byte string of the decompressed data. It may appear anywhere a byte string is expected, including in tags that require data items of major type 2.

Start Message

Table 5.20: Start Message

Field	Type	Description
type	TextString	The value “start“.
arm_date	DateTime	Date and time the ARM command was issued.
beam_center_x	Float	x-position where the direct beam would hit the detector.
beam_center_y	Float	y-position where the direct beam would hit the detector.
channels	Array(TextString)	List of active channel names. Image channel data is serialized in this order.
count_time	Float	Exposure time per image
countrate_correction_enabled	Boolean	*true* if count rate correction is enabled.
countrate_correction_lookup_table	TypedArray	The lookup table used for count rate correction.
detector_description	TextString	Detector model and type.
detector_serial_number	TextString	Serial number of the detector.
detector_translation	Array(Float)	Translation vector of the detector in meters.
flatfield	Map(TextString -> MultiDimArray)	Map from channel name to flatfield.
flatfield_enabled	Boolean	*true* if flatfield correction is enabled.

Table 5.20: Start Message - continued

Field	Type	Description
frame_time	Float	Time interval between start of image acquisitions.
goniometer	Map	Goniometer parameters.
AXIS	Map	Any goniometer axis
increment	Float	Step increment for AXIS.
start	Float	Starting value for AXIS.
image_size_x	Integer	The x-size (width) of each image in pixels.
image_size_y	Integer	The y-size (height) of each image in pixels.
incident_energy	Float	Configured energy of incident particles (e.g. X-rays) in eV.
incident_wavelength	Float	Wavelength of incident particles (e.g. X-rays) in angstroms.
number_of_images	Integer	Number of images in the series.
pixel_mask	Map(TextString -> MultiDimArray)	Map from channel name to 32-bit pixel mask.
pixel_mask_enabled	Boolean	*true* if pixel masking is enabled.
pixel_size_x	Float	Physical size of a pixel in the x-direction.
pixel_size_y	Float	Physical size of a pixel in the y-direction.
saturation_value	Integer	Maximum valid sample value (excluding the NODATA value).
sensor_material	TextString	Material used for direct detection in the sensor.
sensor_thickness	Float	Thickness of the sensor material.
series_id	Integer	The series number.
series_unique_id	TextString	The unique ID of the series.
threshold_energy	Map(TextString -> Float)	Map from channel name to energy (eV).
user_data	Any	Additional data specified by the user.
virtual_pixel_interpolation_enabled	Boolean	*true* if virtual-pixel interpolation is enabled.

Image Message

This message is sent for every exposure. Each message contains all active channel (threshold) data of the exposure.

Table 5.21: Image Message

Field	Type	Description
type	TextString	The value “image”.
data	Map(TextString → MultiDimArray)	The map of channel data.
image_id	Integer	The image number.
real_time	Rational	Real exposure time of the image in seconds. The denominator is the time base frequency which is constant for the duration of the series.
series_date	DateTime	Date and time of the start of the series.
series_id	Integer	The series number.
series_unique_id	TextString	The unique ID of the series.
start_time	Rational	Start time of the image in seconds, relative to series_date. The denominator is the time base frequency which is constant for the duration of the series.
stop_time	Rational	Stop time of the image in seconds, relative to series_date. The denominator is the time base frequency which is constant for the duration of the series.
user_data	Any	Additional data specified by the user.

End Message

This message signifies the end of a series.

Table 5.22: End Message

Field	Type	Description
type	TextString	The value “end“
series_id	Integer	The series number
series_unique_id	TextString	The unique ID of the series

5.4.3. Legacy Stream (V1) Data Access

Image and header data are transferred via *zeromq sockets*. The port is 9999, the scheme is Push/Pull, i.e. the server opens a *zeromq push socket*, whereas the client needs to open a *zeromq pull socket*.

There are 3 types of messages, which are defined below in more detail: **Global Header Data**, **Image Data** and **End of Series**. After passing the "*arm*" command to the detector one message containing *Global Header Data* is sent over the *zeromq socket*. After passing "*trigger*" one messages per image containing *Image Data* is sent. After passing "*disarm*", "*cancel*" or "*abort*", one message containing *End of Series* is sent.

Global Header Data

Zeromq multipart message consisting of the following parts:

- **Part 1**: Json Dictionary, reading {*"htype":"dheader-1.0", "series": <id>, "header_detail": "all"* | *"basic"* | *"none"*}. *<id>* denotes the series id of the present image series.
- **Part 2**: (only if *header_detail* is "*all*" or "*basic*"): Detector configuration as json dictionary, reading {*<config parameter>: <value>*}. The keys are the configuration parameters as defined in the detector API. The values are the current configuration values. There are maximum 1 dim arrays, which are stored as json array. *Flatfield* and *Pixelmask* and *countrate_correction_table* are not part of the dictionary.
- **Part 3**: (only if *header_detail* is "*all*"): Flatfield Header. Json Dictionary reading {*"htype": "dflatfield-1.0", "shape": [x,y], "type": <data type>*}. *<data type>* is always "*float32*" (32 bit float) for a flatfield.
- **Part 4**: (only if *header_detail* is "*all*"): Flatfield data blob.
- **Part 5**: (only if *header_detail* is "*all*"): Pixel Mask Header. Json Dictionary reading {*"htype": "dpixelmask-1.0", "shape": [x,y], "type": <data type>*}. *<data type>* is always "*uint32*" (32 bit unsigned integer) for a pixel mask.
- **Part 6**: (only if *header_detail* is "*all*"): Pixel Mask data blob.
- **Part 7**: (only if *header_detail* is "*all*"): Countrate Table Header. Json Dictionary reading {*"htype": "dcountrate_table-1.0", "shape": [x,y], "type": <data type>*}. *<data type>* is always "*float32*" (32 bit float).
- **Part 8**: (only if *header_detail* is "*all*"): Countrate Table data blob.
- **Appendix** (only if *header_detail* is "*all*"): Content of API parameter header_appendix if not empty.

Example:

{"htype":"dheader-1.0", "series": 1, "header_detail": "all"}
{"auto_summation": true, "photon_energy": 8000, ...}

{"htype": "dflatfield-1.0", "shape": [1030,1065], "type": "float32"}
DATA BLOB (Flatfield)

{"htype": "dpixelmask-1.0", "shape": [1030,1065], "type": "uint32"}
DATA BLOB (Pixel Mask)

{"htype": "dcountrate_table-1.0", "shape": [2,1000], "type": "float32"}
DATA BLOB (countratecorrection table)

Image Data

Zeromq multipart message consisting of the following parts:

- **Part 1**: Json Dictionary, reading {*"htype":"dimage-1.0","series": <series id>, "frame": <frame id>, "hash": <md5>*}, *<series id>* is the number identifying the series, *<frame id>* is the frame id, i.e. the image number. *<md5>* is the md5 hash of the next message part.
- **Part 2**: {*"htype":"dimage_d-1.0", "shape":[x,y,(z)], "type": <data type>, "encoding": <encoding>, "size": <size of data blob>*}.
  - *<data type>*: "uint8", "uint16" or "uint32".
  - *<encoding>*: String of the form "[bs<BIT>][[-]lz4][<|>]". bs<BIT> stands for bit shuffling with <BIT> bits, lz4 for lz4 compression and < (>) for little (big) endian. E.g. "bs8-lz4<" stands for 8bit bitshuffling, lz4 compression and little endian. lz4 data is written as defined at https://code.google.com/p/lz4/ without any additional data like block size etc.
  - *<size of data blob>*: Size in bytes of the following data blob
- **Part 3**: Data Blob
- **Part 4**: {*"htype":"dconfig-1.0", "start_time": <start_time>, "stop_time", <stop_time>, "real_time": <real_time>*}. Begin, end and duration of the exposure of the current image in nano seconds. The start time of first image of the series is by definition zero. The value of start_time and stop_time is set to zero when initializing the detector system.

  Information #27

  - The time values are based on the clock quartz on the detector control board.
  - Time values are returned in ns.

- **Appendix** (only if *image_appendix* contains non-empty string): Content of API parameter *image_appendix*

Example:

{"htype":"dimage-1.0", "series": 32, "frame": 324, "hash": "fc67f000d08fe6b380ea9434b8362d22"}
{"htype":"dimage_d-1.0", "shape":[1030,1065], "type": "uint32", "encoding": "lz4<", "size": 47398247}

DATA BLOB (Image Data)
{"htype":"dconfig-1.0", "start_time": 834759834260, "stop_time": 834760834280, "real_time": 1000000}

End of Series

Zeromq message consisting of one part containing the json string:
{"htype": "dseries_end-1.0", "series": <id>}

5.4.4. Stream Status Parameters

The status of the stream can be accessed at:

[]$_ URL

```
<uri> = http://<ADDRESS_OF_DCU>/stream/api/<VERSION>/status/<PARAMETER>
```

The following stream status variables are accessible:

Table 5.23: Stream Status Parameters

Parameter	Type	Access	Remarks
dropped	uint	r	Number of images that got dropped as not requested. After "*arm*" this number is reset to zero.
state	string	r	*disabled*, *ready*, *acquire* or *error*. After the detector has been armed the state becomes acquire, after disarm, abort or cancel the state becomes ready. There are currently no error conditions.

5.4.5. Stream Command Parameters

Command parameters are write only. The base path to the resource is:

Table 5.24: Stream Command Parameters

Parameter	Return Value	Access	Remarks
initialize	-	w	Resets the stream to its original state (i.e reset dropped images and errors, mode is set to disabled). **Caution #13** ⚠ If issued during acquisition the stream interface will stop sending data.

5.5. System Subsystem

The SIMPLON API lets you control the DAQ service providing the SIMPLON API.

5.5.1. System Configuration Parameters

The status of the system can be accessed at:

```
[]$_ URL

<uri> = http://<ADDRESS_OF_DCU>/system/api/<VERSION>/config/<PARAMETER>
```

Table 5.25: System Config Parameters

Parameter	Type	Access	Remarks
datetime/date	string	rw	Returns or sets the current date.
datetime/ntp	string	rw	Enables (on) or disables (off) Network Time Protocol (NTP).
datetime/ntp_server	string	rw	Returns or sets the comma seperate list of Network Time Protocol (NTP) servers.
datetime/time	string	rw	Gets or sets the current time (HH:MM:SS).
datetime/timezone	string	rw	Gets or sets the current timezone. Also returns list of available timezones on a GET request.
network/n/addr	string	rw	Returns or sets the network address (IP) of the detector control unit network interface "n". **Information #28** Please note that the fall-back interface configurations are read-only. **Information #29** Please note that "n" needs to be replaced with the network interface name (e.g. user1p1). Available interfaces can be requested with a GET request to *<system config URI>/keys*
network/n/dhcp	string	rw	Enables (on) or disables (off) DHCP for detector control unit network interface "n".
network/n/dns1	string	rw	Returns or sets the primary domain name server (DNS) for detector control unit network interface "n".

Table 5.25: System Config Parameters - continued

Parameter	Type	Access	Remarks
network/n/dns2	string	rw	Returns or sets the secondary domain name server (DNS) for detector control unit network interface ”n”.
network/n/mtu	string	rw	Returns or sets the maximum transmission unit (MTU) for detector control unit network interface ”n”.
network/n/netmask	string	rw	Return or sets the netmask for detector control unit network interface ”n”.
network/n/static_routes	string	rw	Return or sets the static routes for detector control unit network interface ”n”. **Information #30** Example: ”value”:[”gateway”:”192.168.50.2”, ”destination”:”10.20.1.1/32”, ”gateway”:”192.168.50.3”, ”destination”:”10.30.0.0/16”]
system_info/service_tag	string	r	Returns the DELL service tag of the detector control unit.

5.5.2. System Command Parameters

Command parameters are write only. The base path to the resource is:

[]$_ URL

```
<uri> = http://<ADDRESS_OF_DCU>/system/api/<VERSION>/command/<PARAMETER>
```

Table 5.26: System Command Parameters

Parameter	Return Value	Access	Remarks
create_log_tarball	URI of the log tarball	w	Creates a tarball of all logfiles relevant for support.
delete_logs	-	w	Deletes all customer logs on the detector control unit.
reboot	-	w	Reboots the detector control unit.
restart	-	w	Restarts the service providing the SIMPLON API.
shutdown	-	w	Shuts down (halts) the detector control unit.