

SIMPLON 1.6 API Reference

EIGER R/X Detector Systems

Document Version v1.0.0



CONTENT

| | |
|---|----|
| CONTENT | i |
| DOCUMENT HISTORY | ii |
| Current Document | ii |
| Changes | ii |
| 1 GENERAL INFORMATION | 1 |
| 1.1 Contact and Support | 1 |
| 1.2 Explanation of Symbols | 1 |
| 1.3 Warranty Information | 2 |
| 1.4 Disclaminer | 2 |
| 2 INTRODUCTION – RESTLIKE API | 3 |
| 3 OPERATING THE EIGER DETECTOR SYSTEM | 6 |
| 3.1 Interface: http/REST | 7 |
| 3.1.1 URLs | 8 |
| 3.1.2 Tree-Like View of Resources | 9 |
| 4 SIMPLON API | 11 |
| 4.1 Detector Subsystem | 11 |
| 4.1.1 Detector Configuration Parameters | 11 |
| 4.1.2 Detector Status Parameters | 16 |
| 4.1.3 Detector Command Parameters | 18 |
| 4.2 Monitor Interface | 18 |
| 4.2.1 Monitor Configuration | 19 |
| 4.2.2 Data Access | 19 |
| 4.2.3 Monitor Status Parameters | 20 |
| 4.2.4 Monitor Control Parameters | 21 |
| 4.3 FileWriter | 21 |
| 4.3.1 Filewriter Configuration | 21 |
| 4.3.2 Data Access | 23 |
| 4.3.3 Filewriter Status Parameters | 23 |
| 4.3.4 Filewriter Control Parameters | 24 |
| 4.4 Stream | 24 |
| 4.4.1 Stream Configuration | 24 |
| 4.4.2 Data Interface | 26 |
| 4.4.3 Stream Status Parameters | 28 |
| 4.4.4 Stream Control Parameters | 28 |
| 4.5 System | 29 |
| 4.5.1 System Control Parameters | 29 |

DOCUMENT HISTORY

Current Document

Table 1: Current Version of this Document

| Version | Date | Status | Prepared | Checked | Released |
|---------|------------|---------|----------|------------|------------|
| v1.0.0 | 2017-05-31 | release | AM, CE | MM, SB, DM | PH, SG, AM |

Changes

Table 2: Changes to this Document

| Version | Date | Changes |
|---------|------------|---|
| v1.0.0 | 2017-05-24 | Revision and redesign of the documentation. |

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 these publications 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

The main objective of the SIMPLON API is to provide platform-independent control of EIGER detector systems using a well-established standardized protocol. 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 request needs 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/1.6.0/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/1.6.0/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 an instruction 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.

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/1.6.0/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:

[]\$_ Python Code

```
{
  "min" : 0.000002999900061695371,
  "max" : 1800,
  "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/1.6.0/command//initialize*.

A *PUT* request cannot be sent from a web browser. For testing, you may either use the plugin *HttpRequester* 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/1.6.0/command/initialize*. Again, substitute *<address_of_dcu>* by the IP of the EIGER 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.

Now we want to set *count_time* to 1.0 seconds. To 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 *HttpRequester* we set the URL to *http://<address_of_dcu>/detector/api/1.6.0/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.

[]\$_ Python Code

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

HttpRequester 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:

[]\$_ Python Code

```
[
  "bit_depth_image",
  "count_time",
  "count_rate_correction_count_cutoff",
  "frame_count_time",
  "frame_period",
  "nframes_sum"
]
```

This is the list of parameters that have been (implicitly) changed. 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/1.6.0/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. Parameters are configured via *GET/PUT* requests on *http://<address_of_dcu>/detector/api/1.6.0/config/<parameter>*, detector commands are transferred via *PUT* requests *http://<address_of_dcu>/detector/api/1.6.0/command/<command>*. Finally the status of the detector may be queried via *http://<address_of_dcu>/detector/api/1.6.0/status/<statusparameter>*. The term commonly used for the configuration, status and command subsystem is task. In addition to the detector interface (i.e. module) there is a monitor interface (*http://<address_of_dcu>/monitor/api/1.6.0/<module>*) and a filewriter interface (*http://<address_of_dcu>/filewriter/api/1.6.0/<module>*). 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.

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

| Protocol Prefix | Host | Resource | | | | |
|-----------------|--------------------------------|------------------------|-------------|-------------------------|----------------------|--------------------------|
| | | module | | version | task | parameter |
| <i>http://</i> | <i><address_of_dcu>/</i> | <i><module>/</i> | <i>api/</i> | <i><version>/</i> | <i><task>/</i> | <i><parameter></i> |

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

| Protocol Prefix | Host | Resource | | | | |
|-----------------|---------------------|------------------|-------------|---------------|----------------|-------------------|
| | | module | | version | task | parameter |
| <i>http://</i> | <i>10.42.41.10/</i> | <i>detector/</i> | <i>api/</i> | <i>1.6.0/</i> | <i>config/</i> | <i>count_time</i> |

Further modules are *stream* and *system*. The monitor interface lets you inspect images of the currently running measurement. The filewriter interface lets you control how the data is stored in *hdf5* files. In addition the *hdf5* files may be received from */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.

3. OPERATING THE EIGER DETECTOR SYSTEM

In order to acquire data with an EIGER 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

Configuration

- 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

Arm the detector (mandatory)

- 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 up to 17 seconds. If the configuration is not changed, arm will be performed very quickly.
- See Detector -> Commands -> Arm

Trigger the detector

Information

#3



Sending a "Trigger" command is mandatory in software trigger mode (ints) and has to be omitted in external enabled modes (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

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) |

Information

#5



The status of the EIGER detector system may be optionally checked during acquisition, but detector status parameters can only be updated while the system is not acquiring data.

For receiving data, three options are available:

- Writing and downloading *HDF5* files via the filewriter interface (section 4.3)
- Retrieving individual images via the monitor interface (section 4.2)
- Retrieving the data as a stream via the stream interface (section 4.4)

3.1. Interface: http/REST

The interface to the EIGER 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. Special resources in the filewriter and monitor subsystem can also be **deleted**.
- 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.
- 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.
- Any **put** request changing parameters may implicitly change dependent parameters. **Put** will always return a list of all parameters implicitly and explicitly 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.

¹ The detector will disarm after an internally triggered (*trigger_mode*: ints) series has been completed.

3.1.1. URLs

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

Table 3.2: Module API URLs

| | |
|------------------------------------|--|
| Detector (section 4.1) | Configuration of the detector and the readout system, control of data acquisition and requesting the detector status <i>http://<address_of_dcu>/detector/api/1.6.0/</i> |
| Monitor (section 4.2) | Receiving single frames at a low rate. <i>http://<address_of_dcu>/monitor/api/1.6.0/</i> |
| Filewriter (section 4.3) | Configuration of the HDF5 filewriter. <i>http://<address_of_dcu>/filewriter/api/1.6.0/</i> |
| Stream (section 4.4) | Configuration of the stream interface. <i>http://<address_of_dcu>/stream/api/1.6.0/</i> |
| System (section 4.5) | Configuration and control of the system. <i>http://<address_of_dcu>/system/api/1.6.0/</i> |

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, hdf5 and tiff as well. (MIME types *application/json*, *application/hdf5* and *application/tiff*). The header item "content-type" is set respectively in all responses.

Information

#6



Default format is *application/json*. The default format will be used if no specific format is requested. For small datasets *application/json* is recommended. For larger datasets, in particular 2d arrays (ie. *flatfields* and *pixel_masks*), only *application/tiff* is supported, other MIME types may provide experimental access.

3.1.2. Tree-Like View of Resources

Below table contains an interactive tree view of all resources. You may click on any field to get more information about the resource in question.

Table 3.3: Tree-Like View of Resources (i.e. modules, tasks and parameters)

| module | version | task | parameter |
|--|---------|------|--|
| http://<address_of_dcu>/detector/api/<version>/config/ | | | auto_summation beam_center_x beam_center_y bit_depth_image bit_depth_readout chi_increment chi_start compression count_time countrate_correction_applied countrate_correction_count_cutoff data_collection_date description detector_distance detector_number detector_readout_time element flatfield flatfield_correction_applied frame_time kappa_increment kappa_start nimages ntrigger number_of_excluded_pixels omega_increment omega_start phi_increment phi_start photon_energy pixel_mask pixel_mask_applied roi_mode sensor_material sensor_thickness software_version threshold_energy trigger_mode two_theta_increment two_theta_start wavelength x_pixel_size x_pixels_in_detector y_pixel_size y_pixels_in_detector |

Table 3.3: Tree-Like View of Resources (i.e. modules, tasks and parameters) - continued

| module | version | task | parameter |
|--------------------|-------------------------------------|-----------------|--|
| | | <i>status/</i> | <i>state</i> <i>error</i> <i>time</i> <i>board_000/th0_temp</i> <i>board_000/th0_humidity</i> <i>builder/dcu_buffer_free</i> |
| | | <i>command/</i> | <i>initialize</i> <i>arm</i> <i>disarm</i> <i>trigger</i> <i>cancel</i> <i>abort</i> <i>status_update</i> |
| <i>monitor/</i> | <i>api/<version>/config/</i> | | <i>mode</i> <i>buffer_size</i> |
| | | <i>status/</i> | <i>state</i> <i>error</i> <i>buffer_fill_level</i> <i>dropped</i> <i>next_image_number</i> <i>monitor_image_number</i> |
| | | <i>command/</i> | <i>clear</i> <i>initialize</i> |
| <i>filewriter/</i> | <i>api/<version>/config/</i> | | <i>mode</i> <i>transfer_mode</i> <i>nimages_per_file</i> <i>image_nr_start</i> <i>name_pattern</i> <i>compression_enabled</i> |
| | | <i>status/</i> | <i>state</i> <i>error</i> <i>time</i> <i>buffer_free</i> |
| | | <i>command/</i> | <i>clear</i> <i>initialize</i> |
| <i>stream/</i> | <i>api/<version>/config/</i> | | <i>mode</i> <i>header_detail</i> <i>header_appendix</i> <i>image_appendix</i> |
| | | <i>status/</i> | <i>state</i> <i>error</i> <i>dropped</i> |
| | | <i>command/</i> | <i>initialize</i> |
| <i>system/</i> | <i>api/<version>/command/</i> | <i>restart</i> | |

4. 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.

4.1. 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.

4.1.1. Detector Configuration Parameters

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

[]\$_ URLs

```
<base_path> = http://<address_of_dcu>/detector/api/<version>/config/  
<uri> = <base_path>/<parameter>
```

Table 4.1: Detector Config Parameters

| Parameter | Data Type | Access | Remarks |
|-------------------|-----------|--------|--|
| auto_summation | bool | rw | Enables (<i>True</i>) or disables (<i>False</i>) auto-summation. Should always be enabled. |
| beam_center_x | float | rw | Beam position on detector. |
| beam_center_y | float | rw | Beam position on detector. |
| bit_depth_image | int | r | Bit depth of generated images |
| bit_depth_readout | int | r | Bit depth of the internal readout. |
| 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. |

Table 4.1: Detector Config Parameters - continued

| Parameter | Data Type | Access | Remarks |
|-----------------------------------|-----------|--------|--|
| compression | string | rw | Defines the compression algorithm used. Allowed options are <i>lz4</i> and <i>bslz4</i> . The recommended compression algorithm is <i>bslz4</i> . For enabling and disabling compression see Section 4.3.1 Filewriter Configuration (<i>compression_enabled</i>). |
| count_time | float | rw | Exposure time per image. |
| countrate_correction_applied | bool | rw | Enables (<i>True</i>) or disables (<i>False</i>) countrate correction. Should always be enabled. See the User Manual for details. |
| countrate_correction_count_cutoff | uint | r | Maximum number of possible counts after count rate correction. |
| data_collection_date | string | rw | 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 | float | rw | Detector serial number. |
| detector_readout_time | float | r | Readout dead time between consecutive detector frames. |
| element | string | rw | Sets parameter <i>photon_energy</i> to the K-alpha fluorescence radiation energy of an element. |
| flatfield | float[][] | rw | Flatfield correction factors used for flatfield correction. Pixel data are multiplied with these factors for calculating flatfield corrected data. |
| flatfield_correction_applied | bool | rw | Enables (<i>True</i>) or disables (<i>False</i>) flatfield correction. Should always be 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. |
| kappa_increment | float | rw | Kappa increment per frame. |
| kappa_start | float | rw | Kappa start angle (start angle of the first frame). |
| nimages | uint | rw | Number of images. See the User Manual for details. |

Table 4.1: Detector Config Parameters - continued

| Parameter | Data Type | Access | Remarks |
|---------------------------|-----------|--------|--|
| ntrigger | uint | rw | Number of triggers. See the User Manual for details. |
| number_of_excluded_pixels | uint | r | Total number of defective, disabled or inactive pixels. |
| omega_increment | float | rw | Omega increment per frame. |
| omega_start | float | rw | Omega start angle (start angle of the first frame). |
| 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_mask | unit[][] | 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- 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 $2^3=8$. |
| pixel_mask_applied | bool | rw | Enables (<i>True</i>) or disables (<i>False</i>) applying the pixel mask on the acquired data. If <i>True</i> (default), pixels that have a corresponding bit set in the <i>pixel_mask</i> are flagged with $(2^{\text{bit_depth_image}})-1$. If disabled, the pixel mask needs to be applied at the point of data processing. |
| roi_mode ¹ | string | rw | Selects the region of interest (ROI). When ROI is disabled, the entire active area is read out. The "4M" ROI mode enables higher frame rates. Please refer to the User Manual for further details. |
| sensor_material | string | r | Material used for direct detection of X-rays in the sensor. |
| sensor_thickness | float | r | Thickness of the sensor material. |
| software_version | string | r | Software version used for data acquisition and correction. |

¹ Only available on DECTRIS EIGER X 9M / 16M systems.

Table 4.1: Detector Config Parameters - continued

| Parameter | Data Type | Access | Remarks |
|----------------------|-----------|--------|--|
| 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. |
| trigger_mode | string | rw | Mode of triggering image acquisition. See the User Manual for details. |
| 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). |
| 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. |
| 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. For hdf5 objects, the JSON metadata is stored with hdf5 attributes.

Table 4.2: 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) |
| "value_type" | <string> | Returns the data type of a parameter. Data types are bool, float, int, string or a list of float or int. |
| "min" | <minimal_parameter_value> | Returns the minimum of a parameter (for numerical datatypes). |
| "max" | <maximal_parameter_value> | Returns the maximum of a parameter (for numerical datatypes). |
| "allowed_values" | <list_of_allowed_values> | Returns the list of allowed values. An empty list indicates there are no restrictions. |

Table 4.2: Key Value Pairs for Detector Config Parameters (GET) - continued

| JSON Key | JSON Value | Description |
|---------------|------------|---|
| "unit" | <string> | The unit of the parameter. |
| "access_mode" | <string> | String, describing read, and/or write access to resource. When not available, the <i>access_mode</i> is "rw". |

Put requests send a body serialized in the JSON format. Arrays may be **put** as hdf5 objects. The HTTP header item *content-type* must be set appropriately. The JSON string may contain the following keywords:

Table 4.3: 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). |

Alternatively you can **put** larger datasets and images as hdf5 files.

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

Table 4.4: 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. |

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.

Remarks:

- It is highly recommended to change a configuration by **putting** each parameter separately. Do NOT upload a full configuration in one step to the config URL. This will only succeed if the configuration is valid and consistent. Uploading an inconsistent configuration (e.g., element is configured to Cu and energy should be set to 7400 eV) leads to undefined behavior.
- The order in which parameters are **put** is important, as parameters can influence each other.
- A base configuration can be stored on the user computer by using the HTTP header item *accept=application/hdf5* on the base URL. This configuration is consistent and valid and can be uploaded in one step.

Example – Setting *photon_energy*

As already mentioned, when setting a value, the DCU returns the names of the parameters that were 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:

[]>_ Return Value

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

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.

4.1.2. Detector Status Parameters

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

[]\$_ URLs

```
<base_path> = http://<address_of_dcu>/detector/api/<version>/status
<uri> = <base_path>/<parameter>
```

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

Status Information

Table 4.5: Detector Status Parameters

| Parameter | Data Type | Access | Remarks |
|-------------------------|-----------|--------|---|
| state | string | r | <p>Possible states: na (not available), ready, initialize, configure, acquire, test, error</p> <p>NOTE: State is "na", when the DCU is booted or the acquisition service was restarted.</p> |
| error | string[] | r | Returns list of status parameters causing error condition. |
| time | date | r | Returns actual system time. |
| board_000/th0_temp | float | r | Temperature reported by temperature sensor. |
| board_000/th0_humidity | float | r | Relative humidity reported by humidity sensor. |
| builder/dcu_buffer_free | float | r | Percentage of available buffer space on the DCU. |

JSON Serialization

Get requests have no body. The returned JSON string contains the fields below.

Table 4.6: 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. |
| "value_type" | <string> | Returns the data type of a parameter. |
| "unit" | <string> | The unit of the parameter. |
| "time" | <date> | Timestamp for when the value was updated. |
| "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. |

4.1.3. Detector Command Parameters

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

[]\$_ Python Code

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

Table 4.7: Detector Command Parameters

| Parameter | Return Value | Access | Remarks |
|---------------|------------------|--------|--|
| initialize | - | w | Initializes the detector. |
| arm | sequence_id: int | w | Loads configuration to the detector and arms the trigger unit. |
| disarm | sequence_id: int | w | Writes all data to file and disarms the trigger unit. |
| trigger | - | w | Starts data acquisition with the programmed trigger sequence. |
| cancel | sequence_id: int | w | Stops the data acquisition, but only after the next image is finished. |
| abort | sequence_id: int | w | Aborts all operations and resets the system immediately . All data in the pipeline will be dropped. |
| status_update | - | w | Update detector status. |

If an error occurs, the HTTP error code "400" is returned. In this case, please download the API log, which can be accessed by the web interface of the DCU and contact support@dectris.com.

The trigger command can also accept an argument in the put request – the *count_time* – if used in *trigger_mode* inte (internal enable).

4.2. Monitor Interface

The monitor interface is used to inspect single frames. This is a low performance and low bandwidth interface, and thus should only be used at low frame rates. For high frame rates (>10 Hz), usage of either the filewriter or the streaming interface is advised. In order to use the monitor interface, it must first be configured. The base URL for the Monitor API is:

[]\$_ Python Code

```
http://<address_of_dcu>/monitor/api/<version>
```

4.2.1. Monitor Configuration

The configuration is applied at the URL:

[]\$_URL

```
<base_path> = http://<address_of_dcu>/monitor/api/<version>/config
<uri> = <base_path>/<parameter>
```

with the following commands:

Table 4.8: Monitor Config Parameters

| Parameter | Data Type | Access | Remarks |
|-------------|-----------|--------|---|
| mode | bool | rw | Operation mode of the monitor, which can be <i>enabled</i> or <i>disabled</i> . When enabled, a number of <i>buffer_size</i> images are stored in the monitor buffer. The monitor keeps old and drops new images if the buffer is running full. |
| buffer_size | int | rw | Number of images that can be buffered by the monitor interface. |

4.2.2. Data Access

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

[]\$_URL

```
<base_path> = http://<ADDRESS_of_DCU>/monitor/api/<VERSION>/images
<uri> = <base_path>/<parameter>
```

Table 4.9: Monitor Data Access (Images) Parameters

| Parameter | Data Type | Access | Remarks |
|-------------|-----------|--------|--|
| <base_path> | JSON Dict | r | List of available images [[seriesId, [imfId, imgId, ...]], ...]. |
| monitor | tif | r | Gets latest image from the buffer. Default waits for 500 ms for image. Timeout via ?timeout=[ms] adjustable. Returns 408 if no image is available. |
| next | tif | r | Gets next image and removes it from the buffer. Default waits for 500 ms for image. Timeout via ?timeout=[ms] adjustable. Returns 408 if no image available. |

Table 4.9: Monitor Data Access (Images) Parameters - continued

| Parameter | Data Type | Access | Remarks |
|---------------|-----------|--------|---|
| <series>/<id> | tif | r | Gets corresponding image, if not available, returns HTTP 404 Not Found. |

After an image has been requested using the monitor API commands <base_path>/monitor, <base_path>/next or <base_path>/<series>/<id> a Json dictionary is returned.

[]\$_ URLs

```
{
  "state": "normal",
  "critical_values": [],
  "value": [
    8,
    9,
    614078893378,
    614083892870,
    4969946
  ],
  "value_type": "int",
  "time": "rw"
}
```

The value array is built from the values *series_id*, *frame_id*, *start_time*, *end_time*, *real_time*. In above example *series_id* is 8, *frame_id* is 9, *start_time* is 614078893378 and so forth.

4.2.3. Monitor Status Parameters

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

[]\$_ URLs

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

Table 4.10: Monitor Status Parameters

| Parameter | Data Type | Access | Remarks |
|-------------------|-----------|--------|---|
| state | string | r | State can be <i>normal</i> or <i>overflow</i> if images have been dropped. |
| error | string | r | Returns list of status parameters causing error condition. |
| buffer_fill_level | int | r | Returns a tuple with current number of images and maximum number of images in buffer. |

Table 4.10: Monitor Status Parameters - continued

| Parameter | Data Type | Access | Remarks |
|----------------------|-----------|--------|---|
| dropped | int | r | Number of images which were dropped as not requested. |
| next_image_number | int | r | seriesId, imageId of the last image requested via images/next. |
| monitor_image_number | int | r | seriesId, imageId of the last image requested via images/monitor. |

4.2.4. Monitor Control Parameters

To clear the buffer of images, the following command can be executed at the address *http://<address_of_dcu>/monitor/api/1.6.0/command/clear*

Table 4.11: Monitor Control Parameters

| Parameter | Return Value | Access | Remarks |
|------------|--------------|--------|--|
| clear | - | w | Drops all buffered images and resets status/dropped to zero. |
| initialize | - | w | Resets the monitor to its original state. |

4.3. FileWriter

The data itself, the frames, are by default written to HDF5 files, where the metadata is stored in the NeXus compliant metadata standard. These files can be accessed through the SIMPLON API. Individual frames can also be obtained through the SIMPLON API with the monitor interface, but at low performance (see Section 4.2). This is usually used to monitor and follow data acquisition.

The filewriter subsystem writes the frames and the metadata in the NeXus format to an HDF5 file. The base URL for the filewriter is:

[]\$_URLs

http://<address_of_dcu>/filewriter/api/<version>/

4.3.1. Filewriter Configuration

To configure the filewriter, this URL is used:

[]\$_URL

*<base_path> = http://<address_of_dcu>/filewriter/api/<version>/config
<uri> = <base_path>/<parameter>*

Table 4.12: FileWriter Config Parameters

| Parameter | Data Type | Access | Remarks |
|---------------------|-----------|--------|--|
| mode | string | rw | Operation mode of the filewriter, which can be <i>enabled</i> or <i>disabled</i> . When disabling the filewriter, data loss may occur if data is not retrieved via the stream or the monitor. |
| transfer_mode | string | rw | Transfer mode for files written by the filewriter. Currently, only HTTP is supported. |
| nimages_per_file | int | rw | <p>Maximum number of images stored in each <code><name_pattern>_data_<file_nr>.h5</code> file in the HDF5 file structure created by the filewriter.</p> <p>No data files are created and all images are stored in <code><name_pattern>_master.h5</code> when this parameter is set to 0. WARNING: Only set to 0 when collecting a small number of images.</p> |
| image_nr_start | int | rw | Sets the <code>image_nr_low</code> metadata parameter in the first HDF5 data file <code><name_pattern>_data_000001.h5</code> . 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 <code>image_nr_start</code> to $n+1$ in the subsequent file structure. |
| name_pattern | string | rw | <p>The basename of the file. The pattern <code>\$id</code> will include the sequence number in the file name. <code>series_\$id</code> is the default name pattern, resulting in the following names of the HDF5 file structure created by the filewriter:</p> <p><code>series_<sequence_nr>_master.h5</code>, <code>series_<sequence_nr>_data_<filenr>.h5</code></p> <p>WARNING: The filewriter will overwrite existing files with identical names of the files to be written.</p> |
| compression_enabled | bool | rw | Enables (<i>True</i>) or disables (<i>False</i>) compression of detector data written to HDF5 files. Compression is required for full detector performance, disabling compression may lead to data loss at high frame rates. For compression modes see Section 4.1.1 Detector Configuration Parameters (compression). |

4.3.2. 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/
<uri> = <base_path>/<parameter>`

returns a list of all available files.

4.3.3. Filewriter Status Parameters

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

[]\$_ URL

`<base_path> = http://<address_of_dcu>/filewriter/api/<version>/status
<uri> = <base_path>/<parameter>`

The following filewriter status variables are accessible:

Table 4.13: FileWriter Status Parameters

| Parameter | Data Type | Access | Remarks |
|-------------|-----------|--------|---|
| state | string | r | Possible states: <i>disabled, ready, acquire, error</i> . |
| error | string | r | List of status parameters causing error state. |
| time | date | r | Current system time. |
| buffer_free | int | r | The remaining buffer space in KB. |

4.3.4. Filewriter Control Parameters

To clear the buffer of images, the following command can be executed at the address:

[]\$_ URL

```
<base_path> = http://<address_of_dcu>/filewriter/api/<version>/command
<uri> = <base_path>/<parameter>
```

Table 4.14: FileWriter Control 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. |

4.4. Stream

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>
```

4.4.1. Stream Configuration

To configure the stream, this URL is used:

[]\$_ URL

```
<base_path> = http://<address_of_dcu>/stream/api/<version>/config
<uri> = <base_path>/<parameter>
```

Table 4.15: Stream Config Parameters

| Parameter | Data Type | Access | Remarks |
|-----------|-----------|--------|---|
| mode | string | rw | Operation mode of the stream, which can be <i>enabled</i> or <i>disabled</i> . When disabling the stream, data loss may occur if data is not retrieved via the filewriter or monitor. |

Table 4.15: Stream Config Parameters - continued

| Parameter | Data Type | Access | Remarks |
|-----------------|-----------|--------|---|
| header_detail | string | rw | <p>Detail of header data to be sent: Either <i>"all"</i> (all header data), <i>"basic"</i> (no flatfield nor pixel mask, default setting) or <i>"none"</i> (no header data).</p> <p>NOTE: Choosing <i>"basic"</i> is recommended only if <i>pixel_mask_applied = True</i> in the detector configuration.</p> <p>WARNING: Choosing <i>"none"</i> will result in the loss of experiment metadata. This will complicate processing and archiving and is not recommended.</p> |
| header_appendix | string | rw | Data that is appended to the header data as <i>zeromq</i> submessage. |
| image_appendix | string | rw | Data that is appended to the image data as <i>zeromq</i> submessage. |

Information

#7



Please see Section 4.1.1 for details about the GET and PUT requests.

4.4.2. Data Interface

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*.

| | |
|-----------------------------|---|
| Protocol | Zeromq |
| Port | 9999 |
| Scheme | Push/Pull |
| Direction Connection | Receiver connects to detector (this enables automatic load balancing if more than 1 client is required to receive/process the data) |

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"): Countrate Table data blob.

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

Example:

```
{ "htype": "dimage-1.0", "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> }
```

4.4.3. Stream Status Parameters

The status of the stream can be accessed at:

```
[ ]$_URL
```

```
<base_path> = http://<address_of_dcu>/stream/api/<version>/status
<uri> = <base_path>/<parameter>
```

The following stream status variables are accessible:

Table 4.17: Stream Status Parameters

| Parameter | Data Type | Access | Remarks |
|-----------|-----------|--------|--|
| state | string | r | <i>disabled, ready, acquire</i> or <i>error</i> . After the detector has been armed the state becomes <i>acquire</i> , after disarm, abort or cancel the state becomes <i>ready</i> . There are currently no error conditions. |
| error | string | r | Returns list of status parameters causing error condition (currently only " <i>state</i> "). |
| dropped | int | r | Number of images that got dropped as not requested. After " <i>arm</i> " this number is reset to zero. |

4.4.4. Stream Control Parameters

Table 4.18: Monitor Control Parameters

| Parameter | Return Value | Access | Remarks |
|------------|--------------|--------|--|
| initialize | - | w | Resets the stream to its original state. |

4.5. System

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

4.5.1. System Control Parameters

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

[]\$_URL

<base_path> = http://<address_of_dcu>/system/api/<version>/command
<uri> = <base_path>/<parameter>

Table 4.19: System Command Parameter

| Parameter | Return Value | Access | Remarks |
|-----------|--------------|--------|---|
| restart | - | w | Restarts the service providing the SIMPLON API. |