



SIMPLON API Reference

SIMPLON API Version: 1.6.x
Document Version: 4

Table of Contents

1. Document History	4
1.1. Current Document	4
1.2. Changes	4
2. General Information	5
2.1. Contact	5
2.2. Explanation of Symbols	5
2.3. Disclaimer	5
3. Warnings	6
4. Introduction – RESTlike API	7
5. Operating the EIGER Detector System	10
5.1. Interface: http/REST	11
5.1.1. URLs	11
5.1.2. Tree view of resources (i.e. modules, tasks and parameters)	13
6. SIMPLON API	14
6.1. Detector Subsystem	14
6.1.1. Detector Configuration Parameters	14
6.1.2. Detector Status Parameters	19
6.1.3. Detector Command Parameters	20
6.2. Monitor Interface	21
6.2.1. Monitor Configuration	21
6.2.2. Data Access	21
6.2.3. Monitor Status Parameters	23
6.2.4. Monitor Control Parameters	23
6.3. File Writer	24
6.3.1. Filewriter Configuration	24
6.3.2. Data Access	25
6.3.3. Filewriter Status Parameters	25
6.3.4. Filewriter Control Parameters	25
6.4. Stream	26
6.4.1. Stream Configuration	26
6.4.2. Data Interface	26
6.4.3. Stream Status Parameters	29
6.5. System	29
6.5.1. System Control Parameters	29

1. Document History

1.1. Current Document

<i>Version</i>	<i>Date</i>	<i>Status</i>	<i>Prepared</i>	<i>Checked</i>	<i>Released</i>
4	11.07.2016	Released	AM/SB/MH/VP	SB/MM	SB

1.2. Changes

<i>Version</i>	<i>Date</i>	<i>Changes</i>
4	11.07.2016	Typo in JSON code, clarify metadata in stream & application of pixel mask, make bslz4 the recommended compression algorithm, stream documentation
3	10.02.2016	Typo in stream URL, add pixel mask.
2	28.01.2016	Minor changes
1	27.01.2016	First version

2. General Information

2.1. Contact

Address and Support:

DECTRIS Ltd.
Taefernweg 1
5405 Baden-Daettwil
Switzerland
Phone: +41 56 500 21 02
Fax: +41 56 500 21 01

Website:





- www.dectris.com → Support → Technical Notes → EIGER
- www.dectris.com → Support → FAQ
- www.dectris.com → Support → Problem Report

Email:

- support@dectris.com

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

2.2. Explanation of Symbols

Symbol	Description
	Caution. Please follow the instructions carefully to prevent equipment damage or personal injury.
	Important or helpful notice
	Code block
	Code output block. Contains the expected or an example output of a command or code block.

2.3. Disclaimer

DECTRIS has carefully compiled the contents on 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 published in these applications into other programs or other Web sites 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 application in whole or in part at any time, and is not obliged to update the contents of the manual.

3. *Warnings*



Please carefully read the full user documentation before operating the detector **system**.


4. Introduction – RESTlike API

The main objective of the SIMPLON API is to provide platform-independent access to the EIGER detector system 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 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://<IP_of_DCU>/detector/api/1.6.x/config/count_time` in your favorite web browser's address field, where `<IP_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://<IP_of_DCU>/detector/api/1.6.x/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 a 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://<IP_of_DCU>/detector/api/1.6.x/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:



```
{
  "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.



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://<IP_of_DCU>/detector/api/1.6.x/command/initialize`.

A PUT request cannot be sent from a web browser. For testing, you may either use the plugin HttpRequester for Mozilla Firefox (<https://addons.mozilla.org/de/firefox/addon/httprequester/>) or the command line tool cURL. HttpRequester (version 2.0) opens a window with a field "URL", where you have to enter "http://<IP_of_DCU>/detector/api/1.6.x/command/initialize". Again, substitute <IP_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.



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

In HttpRequester we set the URL to http://<IP_of_DCU>/detector/api/1.6.x/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.



```
{
  "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:



```
[
  "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://<IP_of_DCU>/detector/api/1.6.x/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://<IP_of_DCU>/detector/api/1.6.x/config/<parameter>", detector commands are transferred via PUT requests "http://<IP_of_DCU>/detector/api/1.6.x/command/<command>". Finally the status of the detector may be queried via "http://<IP_of_DCU>/detector/api/1.6.x/status/<status parameter>". 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://<IP_of_DCU>/monitor/api/1.6.x/...") and a filewriter interface ("http://<IP_of_DCU>/filewriter/api/1.6.x/..."). 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.

Protocol Prefix	Host	Resource				
		modules	version	tasks	parameters	
http://	<IP_of_DCU>/	<module>/	api/	<version>/	<task>/	<parameter>

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

Protocol Prefix	Host	Resource				
		modules	version	tasks	parameters	
http://	<IP_of_DCU>/	detector/	api/	1.6.x/	config/	count_time

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.

5. 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 modules in 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
 - *This 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
 - Disables the trigger unit
 - NOTE: The last acquired image (or the image, if only one image was configured) is available only after this command
 - [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 1.5.1 ¹)

- 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, see section 6.3
 - Retrieving single images via the monitor interface, see section 6.2
 - Retrieving the data as a stream via the stream interface, see section 6.4

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

5.1. 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 a 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.

5.1.1. URLs



Please replace <IP_of_DCU> with the IP-Address at which you can reach your detector control unit (e.g. 10.42.41.10 if you are using the preconfigured fixed address on interface em2).

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

detector:	Used to configure the detector and the readout system, to control data acquisition and request the detector status <code>http://<IP_of_DCU>/detector/api/<version></code>
monitor:	Used to receive single frames at a low rate. <code>http://<IP_of_DCU>/monitor/api/<version></code>
filewriter:	Configuration of the HDF5 filewriter. <code>http://<IP_of_DCU>/filewriter/api/<version></code>
stream:	Configuration of the stream interface. <code>http://<IP_of_DCU>/stream/api/<version></code>

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

```
http://<IP_of_DCU>/detector/api/<version>/config
http://<IP_of_DCU>/detector/api/<version>/command
http://<IP_of_DCU>/detector/api/<version>/status
```

A configuration parameter resource has the following URL:

```
http://<IP_of_DCU>/detector/api/<version>/config/<parameter_name>
```

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

`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.



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.

5.1.2. Tree view of resources (i.e. modules, tasks and parameters)

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.

Protocol Prefix	Host	Resource			
		modules	version	tasks	parameters
http://	<IP_of_DCU>/	<u>detector/</u>	api/	<version>/	<u>config/</u> 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
		<u>monitor/</u>	api/	<version>/	<u>status/</u> state , error , time , board 000/th0 temp , board 000/th0 humidity , builder/dcu buffer free
		<u>filewriter/</u>	api/	<version>/	<u>command/</u> initialize , arm , disarm , trigger , cancel , abort , status update
		<u>stream/</u>	api/	<version>/	<u>config/</u> mode , transfer mode , nimages per file , image nr start , name pattern , compression enabled
		<u>system/</u>	api/	<version>/	<u>status/</u> state , error , time , buffer free
		<u>monitor/</u>	api/	<version>/	<u>command/</u> clear , initialize
		<u>stream/</u>	api/	<version>/	<u>config/</u> mode , header detail , header appendix , image appendix
		<u>system/</u>	api/	<version>/	<u>status/</u> state , error , dropped
		<u>system/</u>	api/	<version>/	<u>command/</u> restart

6. SIMPLON API



Please replace <IP_of_DCU> with the IP-Address at which you can reach your detector control unit (e.g. 10.42.41.10 if you are using the preconfigured fixed address on interface em2).



Undocumented keys might be available in all modules. Using those keys is strongly dissuaded. 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.

6.1. Detector Subsystem

The detector subsystem has the base URL:

```
http://<IP_of_DCU>/detector/api/
```

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

6.1.1. Detector Configuration Parameters

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

```
<base_path> = http://<IP_of_DCU>/detector/api/<version>/config/
```

For a description of the parameters, see:

```
https://www.dectris.com/nexus.html#main_head_navigation
```

configuration parameter	resource	data type	access	remarks
	<base_path>/	config	rw	<p>If the html header item accept=hdf5 is used, an hdf5 file containing the configuration is returned. Putting an hdf5 file will set all values in the file recursively.</p> <p>For accept=JSON, the configuration is returned in the JSON format.</p> <p>REMARK: Images will NOT be included (flatfield, pixel_mask) due to their size.</p> <p>NOTE: The flatfield and pixel_mask are included in the *master.h5 file for each acquisition.</p>
auto_summation	<base_path>/auto_summation	bool	rw	Enables (True) or disables (False) auto-summation. Should always be enabled.
beam_center_x	<base_path>/beam_center_x	float	rw	Beam position on detector.
beam_center_y	<base_path>/beam_center_y	float	rw	Beam position on detector.
bit_depth_image	<base_path>/bit_depth_image	int	r	Bit depth of generated images
bit_depth_readout	<base_path>/bit_depth_readout	int	r	Bit depth of the internal readout.
chi_increment	<base_path>/chi_increment	float	rw	Chi increment per frame.
chi_start	<base_path>/chi_start	float	rw	Chi start angle (start angle of the first frame) for an exposure series.

configuration parameter	resource	data type	access	remarks
compression	<base_path>/compression	string	rw	Defines the compression algorithm used. Allowed options are lz4 and bslz4. The recommended compression algorithm is bslz4. For enabling and disabling compression see 6.3.1 Filewriter Configuration (compression enabled).
count_time	<base_path>/count_time	float	rw	Exposure time per image.
countrate_correction_applied	<base_path>/countrate_correction_applied	bool	rw	Enables (True) or disables (False) countrate correction. Should always be enabled. See the detector manual for details.
countrate_correction_count_cutoff	<base_path>/countrate_correction_count_cutoff	uint	r	Maximum number of possible counts after count rate correction.
data_collection_date	<base_path>/data_collection_date	string	rw	Date and time of data collection. Specifically this is the time when the ARM command was issued.
description	<base_path>/description	string	r	Detector model and type.
detector_distance	<base_path>/detector_distance	float	rw	Sample to detector distance.
detector_number	<base_path>/detector_number	string	r	Serial number of the detector system.
detector_readout_time	<base_path>/detector_readout_time	float	r	Readout dead time between consecutive detector frames.
element	<base_path>/element	string	rw	Sets parameter 'photon_energy' to the K-alpha fluorescence radiation energy of an element.
flatfield	<base_path>/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	<base_path>/flatfield_correction_applied	bool	rw	Enables (True) or disables (False) flatfield correction. Should always be enabled.
frame_time	<base_path>/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	<base_path>/kappa_increment	float	rw	Kappa increment per frame.
kappa_start	<base_path>/kappa_start	float	rw	Kappa start angle (start angle of the first frame).
nimages	<base_path>/nimages	uint	rw	Number of images. See the manual for details.
ntrigger	<base_path>/ntrigger	uint	rw	Number of triggers. See the manual for details.
number_of_excluded_pixels	<base_path>/number_of_excluded_pixels	uint	r	Total number of defective, disabled or inactive pixels.
omega_increment	<base_path>/omega_increment	float	rw	Omega increment per frame.
omega_start	<base_path>/omega_start	float	rw	Omega start angle (start angle of the first frame).
phi_increment	<base_path>/phi_increment	float	rw	Phi increment per frame.

configuration parameter	resource	data type	access	remarks
phi_start	<base_path>/phi_start	float	rw	Phi start angle (start angle of the first frame).
pixel_mask	<base_path>/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	<base_path>/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 $(2^{\text{bit_depth_image}})-1$. If disabled, the pixel mask needs to be applied at the point of data processing.
roi_mode ²	<base_path>/roi_mode	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 documentation for further details.
sensor_material	<base_path>/sensor_material	string	r	Material used for direct detection of X-rays in the sensor.
sensor_thickness	<base_path>/sensor_thickness	float	r	Thickness of the sensor material.
software_version	<base_path>/software_version	string	r	Software version used for data acquisition and correction.
threshold_energy	<base_path>/threshold_energy	float	rw	Threshold energy for X-ray counting. Photons with an energy below the threshold are not detected. See the detector manual for details.
trigger_mode	<base_path>/trigger_mode	string	rw	Mode of triggering image acquisition. See the manual for details.
two_theta_increment	<base_path>/two_theta_increment	float	rw	Two theta increment per frame.
two_theta_start	<base_path>/two_theta_start	float	rw	Two theta start angle (start angle of the first frame)
wavelength	<base_path>/wavelength	float	rw	Wavelength of incident X-rays. See the detector manual for details.
x_pixel_size	<base_path>/x_pixel_size	float	r	Size of a single pixel along x-axis of the detector.
x_pixels_in_detect or	<base_path> /x_pixels_in_detector	uint	r	Number of pixels along x-axis of the detector.
y_pixel_size	<base_path>/y_pixel_size	float	r	Size of a single pixel along y-axis of the detector.

² Only available on DECTRIS Ltd. EIGER X 9M / 16M systems.

configuration parameter	resource	data type	access	remarks
y_pixels_in_detector or	<base_path> /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.

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. 2 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.
"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. 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:

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. 2 dimensional arrays shall be uploaded as darrays (see text below)

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

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

JSON key	JSON value	description
None	<changed_parameters>	A list of all resources that are also affected by the put configuration parameter.

6.1.1.1. 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), the filters is always ["base64"], 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 7400keV) 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.

6.1.1.2. Example – Setting photon_energy

As already mentioned, when setting a value, the DCU sends the names of the parameters, which were implicitly changed to maintain a consistent detector configuration in the reply of the set request.

If the photon energy is, e.g., set to 8040 keV, then the reply contains a list with all implicitly changed parameters.

The following example uses some 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://<IP_of_DCU>/detector/api/<version>/config/photon_energy', data=data_json)
# Execute the request on the config value "photon_energy" (REPLACE
<IP_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)
```

Output:

```
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.

6.1.2. Detector Status Parameters

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

<base_path> = http://<IP_of_DCU>/detector/api/<version>/status/

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

6.1.2.1. JSON Serialization

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

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>	Returns 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.

6.1.2.2. Status Information

status parameter	resource	data type	access	remarks
state	<base_path>/state	string	r	Possible states: na (not available), ready, initialize, configure, acquire, test, error NOTE: State is "na", when the DCU is booted or the acquisition service was restarted.
error	<base_path>/error	list of strings	r	Returns list of status parameters causing error condition.
time	<base_path>/time	date	r	Returns actual system time.
board_000/th0_temp	<base_path>/board_000/th0_temp	float	r	Temperature reported by temperature sensor.
board_000/th0_humidity	<base_path>/board_000/th0_humidity	float	r	Relative humidity reported by humidity sensor.
builder/dcu_buffer_free	<base_path>/builder/dcu_buffer_free	float	r	Percentage of available buffer space on the DCU.

6.1.3. Detector Command Parameters

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

<base_path> = http://<IP_of_DCU>/detector/api/<version>/command/

<i>command parameter</i>	<i>resource</i>	<i>return value</i>	<i>access</i>	<i>remarks</i>
initialize	<base_path>/initialize	-	w	Initializes the detector.
arm	<base_path>/arm	sequence_id: int	w	Loads configuration to the detector and arms the trigger unit.
disarm	<base_path>/disarm	sequence_id: int	w	Writes all data to file and disarms the trigger unit.
trigger	<base_path>/trigger	-	w	Starts data taking with the programmed trigger sequence.
cancel	<base_path>/cancel	sequence_id: int	w	Stops taking data, <u>but only after the next image is finished.</u>
abort	<base_path>/abort	sequence_id: int	w	Aborts all operations and resets the system <u>immediately.</u>
status_update	<base_path>/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).

6.2. 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 (>10Hz), 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:

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

6.2.1. Monitor Configuration

The configuration is applied at the URL:

`<base_path> = http://<IP_of_DCU>/monitor/api/<version>/config`

with the following commands:

<i>configuration parameter</i>	<i>resource</i>	<i>data type</i>	<i>access</i>	<i>remarks</i>
mode	<code><base_path>/mode</code>	bool	rw	Operation mode of the monitor, which can be 'enabled' or 'disabled'. When enabled, a number of 'buffer_size' images are stored in the monitor buffer. The monitor keeps old and drops new images if the buffer is running full.
buffer_size	<code><base_path>/buffer_size</code>	int	rw	Number of images that can be buffered by the monitor interface.

6.2.2. Data Access

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

`<base_path> = http://<IP_of_DCU>/monitor/api/<version>/images`

<i>data</i>	<i>resource</i>	<i>data type</i>	<i>access</i>	<i>remarks</i>
	<code><base_path></code>	Json List	r	List of available images [[seriesId, [imfId, imgId, ...]], ...].
monitor	<code><base_path>/monitor</code>	tif	r	Gets latest image from the buffer. Default waits for 500 ms for image. Timeout via <code>?timeout=[ms]</code> adjustable. Returns 408 if no image is available.
next	<code><base_path>/next</code>	tif	r	Gets next image and removes it from the buffer. Default waits for 500 ms for image. Timeout via <code>?timeout=[ms]</code> adjustable. Returns 408 if no image available.
<code><series>/<id></code>	<code><base_path>/<series>/<id></code>	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.



```
{
  "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.

6.2.3. Monitor Status Parameters

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

<base_path> = http://<IP_of_DCU>/monitor/api/<version>/command

<i>control parameter</i>	<i>resource</i>	<i>data type</i>	<i>access</i>	<i>remarks</i>
state	<base_path>/state	string	r	State can be 'normal' or 'overflow' if images have been dropped.
error	<base_path>/error	string	r	Returns list of status parameters causing error condition.
buffer_fill_level	<base_path>/buffer_fill_level	int	r	Returns a tuple with current number of images and maximum number of images in buffer.
dropped	<base_path>/dropped	Int	r	Number of images which were dropped as not requested.
next_image_number	<base_path>/next_image_number	int	r	seriesId, imageId of the last image requested via images/next.
monitor_image_number	<base_path>/monitor_image_number	int	r	seriesId, imageId of the last image requested via images/monitor.

6.2.4. Monitor Control Parameters

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

<base_path> = http://<IP_of_DCU>/monitor/api/<version>/command

<i>command parameter</i>	<i>resource</i>	<i>return value</i>	<i>access</i>	<i>remarks</i>
clear	<base_path>/clear	-	w	Drops all buffered images and resets status/dropped to zero.
initialize	<base_path>/initialize	-	w	Resets the monitor to its original state.

6.3. File Writer

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 6.2). 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:

`http://<IP_of_DCU>/filewriter/api/<version>/`

6.3.1. Filewriter Configuration

To configure the filewriter, this url is used:

`<base_path> = http://<IP_of_DCU>/filewriter/api/<version>/config`

<i>configuration parameter</i>	<i>resource</i>	<i>data type</i>	<i>access</i>	<i>remarks</i>
mode	<code><base_path>/mode</code>	string	rw	Operation mode of the filewriter, which can be 'enabled' or 'disabled'. When disabling the filewriter, data loss may occur if data is not retrieved via the stream or the monitor.
transfer_mode	<code><base_path>/transfer_mode</code>	string	rw	Transfer mode for files written by the filewriter. Currently, only http is supported.
nimages_per_file	<code><base_path>/nimages_per_file</code>	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	<code><base_path>/image_nr_start</code>	int	rw	Sets the 'image_nr_low' 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 <i>m</i> to <i>n</i> in the first file structure, you can set <code>image_nr_start</code> to <i>n</i> +1 in the subsequent file structure.
name_pattern	<code><base_path>/name_pattern</code>	string	rw	<p>The basename of the file. The pattern <code>\$id</code> 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:</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	<base_path>/compression_enabled	bool	rw	Enables (True) or disables (False) 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 6.1.1 Detector Configuration Parameters (compression).
---------------------	---------------------------------	------	----	--

6.3.2. 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:

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

and the data files at:

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

A **get** request to the URL:

`http://<IP_of_DCU>/filewriter/api/<version>/files/`

returns a list with all available files.

6.3.3. Filewriter Status Parameters

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

`<base_path> = http://<IP_of_DCU>/filewriter/api/<version>/status`

The following filewriter status variables are accessible:

<i>status parameter</i>	<i>resource</i>	<i>data type</i>	<i>access</i>	<i>remarks</i>
state	<base_path>/state	string	r	Possible states: disabled, ready, acquire, error
error	<base_path>/error	string	r	List of status parameters causing error state.
time	<base_path>/time	date	r	Current system time.
buffer_free	<base_path>/buffer_free	int	r	The remaining buffer space in KB.

6.3.4. Filewriter Control Parameters

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

`<base_path> = http://<IP_of_DCU>/filewriter/api/<version>/command`

<i>command parameter</i>	<i>resource</i>	<i>return value</i>	<i>access</i>	<i>remarks</i>
clear	<base_path>/clear	-	w	Drops all data (image data and directories) on the DCU.
initialize	<base_path>/initialize	-	w	Resets the filewriter to its original state.

6.4. Stream

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

The base URL for the stream is:

`http://<IP_of_DCU>/stream/api/<version>`

6.4.1. Stream Configuration

To configure the stream, this url is used:

`<base_path> = http://<IP_of_DCU>/stream/api/<version>/config`

<i>configuration parameter</i>	<i>resource</i>	<i>data type</i>	<i>access</i>	<i>remarks</i>
mode	<code><base_path>/mode</code>	string	rw	Operation mode of the stream, which can be 'enabled' or 'disabled'. When disabling the stream, data loss may occur if data is not retrieved via the filewriter or monitor.
header_detail	<code><base_path>/header_detail</code>	string	rw	Detail of header data to be sent: Either "all" (all header data), "basic" (no flatfield nor pixel mask, default setting) or "none" (no header data). NOTE: Choosing "basic" is recommended only if pixel_mask_applied = True in the detector configuration. WARNING: Choosing "none" will result in the loss of experiment metadata. This will complicate processing and archiving and is not recommended.
header_appendix	<code><base_path>/header_appendix</code>	string	rw	Data that is appended to the header data as zeromq submessage
image_appendix	<code><base_path>/image_appendix</code>	string	rw	Data that is appended to the image data as zeromq submessage

Please see section 6.1.1 for details about the GET and PUT requests.

6.4.2. 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.

6.4.2.1. 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_appendix contains non-empty string): Content of API parameter header_appendix

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)

6.4.2.2. 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}
```

6.4.2.3. End of Series

Zeromq message consisting of one part containing the json string:

```
{"htype": "dseries_end-1.0", "series": <id>}
```

6.4.3. Stream Status Parameters

The status of the stream can be accessed at:

<base_path> = http://<IP_of_DCU>/stream/api/<version>/status

The following stream status variables are accessible:

<i>status parameter</i>	<i>resource</i>	<i>data type</i>	<i>access</i>	<i>remarks</i>
state	<base_path>/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.
error	<base_path>/error	string	r	Returns list of status parameters causing error condition (currently only "state").
dropped	<base_path>/dropped	int	r	Number of images that got dropped as not requested. After "arm" this number is reset to zero.

6.5. System

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

6.5.1. System Control Parameters

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

<base_path> = http://<IP_of_DCU>/system/api/<version>/command

<i>command parameter</i>	<i>resource</i>	<i>return value</i>	<i>access</i>	<i>remarks</i>
restart	<base_path>/restart	-	w	Restarts the service providing the SIMPLON API.