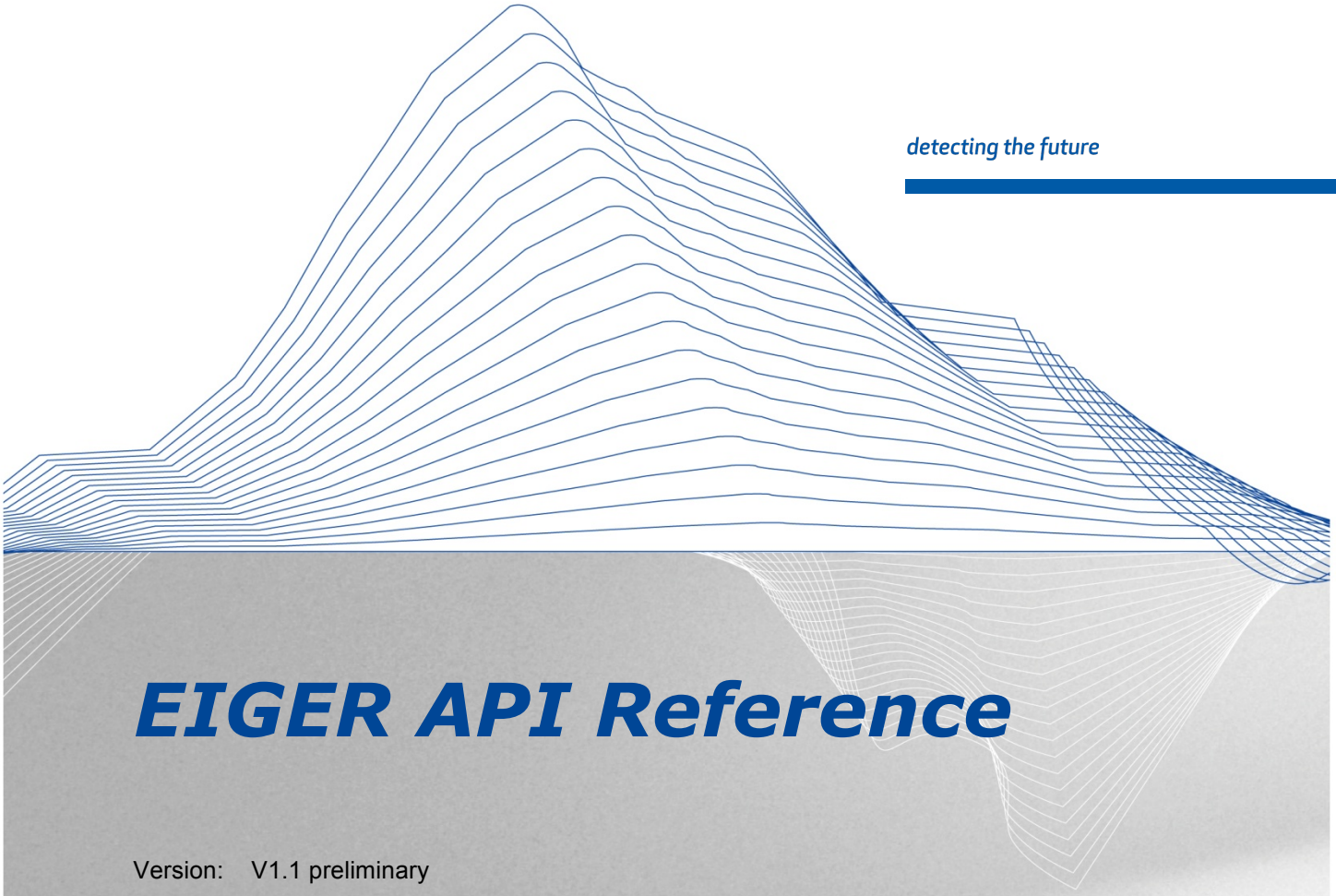
DECTRIS®

detecting the future



EIGER API Reference

Version: V1.1 preliminary

DECTRIS Ltd.
5400 Baden
Switzerland
www.dectris.com

Table of Contents

1. Introduction – RESTlike API 3
2. Operating the EIGER Detector System 5
   - 2.1. Interface: http/REST 6
     - 2.1.1. URLs 6
3. API 8
   - 3.1. Detector Subsystem 8
     - 3.1.1. Detector Configuration Parameters 8
     - 3.1.2. JSON Serialization 10
     - 3.1.3. Example – Setting photon_energy 12
   - 3.2. Detector Status Parameters 13
     - 3.2.1. JSON Serialization 13
     - 3.2.2. Status Information 13
   - 3.3. Detector Command Parameters 15
   - 3.4. Monitor interface 15
     - 3.4.1. Monitor configuration 15
     - 3.4.2. Data access 16
     - 3.4.3. Monitor Status Parameter 16
     - 3.4.4. Monitor control parameter 17
   - 3.5. File Writer 17

1. Introduction – RESTlike API

The main objective of the EIGER API is to provide platform independent access to the detector in a way, which is based on a well-established standardized protocol. Neither shall any special software be installed on the detector control unit nor shall the access to the detector be restricted to a special programming language. The HTTP server running on the detector control unit provides a control interface to the detector that fulfils all of these requirements. In order e.g. to define the state of the detector, trigger an exposure or request an image file a HTTP request needs to be transmitted to the server and the requested data may be received within the HTTP response.

Let's for instance consider the common detector control parameter count_time, which defines the time the detector is exposed to X-rays. You may request the current state of this parameter by entering its URL "`http://<IP_of_DCU>/detector/api/1.0.0/config/count_time`" in your favorite web browsers address field, where `<IP_of_DCU>` needs to be replaced by the IP address of the detector computer. Analogously, the URL of the frame time is "`http://<IP_of_DCU>/detector/api/1.0.0/config/frame_time`".

In particular we call this HTTP based API RESTlike because every detector resource is uniquely identified by its URL. A comprehensive definition of what is RESTful goes beyond the scope of this documentation. We confine ourselves to describing how to work with the EIGER API, which we indeed call RESTlike rather than RESTful because it does not fulfill all requirements of a RESTful API.

Let's look at the sample request of count_time in more detail. We have learned that it is mapped to a unique URL, which is transferred via HTTP to the server. 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 EIGER API uses the verbs GET, PUT and DELETE. When entering `http://<IP_of_DCU>/detector/api/1.0.0/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 encoded in the message body of the HTTP request. The EIGER API chooses json as its main exchange format for message data. For instance your browser may display the following string:

```
{"min": 0.00032999899121932685, "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.

Note:

If your try this example and the browser prints instead "Parameter count_time does not exist", your detector may not have been initialized properly. You need to initialize the detector because the HTTP server needs to read a configuration from the detector in order to know to which kind of detector it is connected to. This is basically what is done when the detector is initialized. In order to initialize the detector you must send a PUT request to "`http://<IP_of_DCU>/detector/api/1.0.0/command/initialize`".

A PUT request cannot be sent from a web browser. For testing you may either use the plugin HttpRequester for Mozilla Firefox (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/1.0.0/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. You could also use cURL, in a terminal window (Linux, Mac, Windows). For further information on cURL refer to section 0.

Now we want to set count_time to 1.0 seconds. We just upload the float value 1.0, as count_time has the unit second (Note: There is no way to change the unit of a parameter). In HttpRequester we set the URL to `http://<IP_of_DCU>/detector/api/1.0.0/config/count_time`. Below there is a field "Content to Send". The content type must be changed to "application/json" and we paste into the content field the string `{"value": 1.0}`, i.e. we upload a json dictionary with its only key "value" set to 1.0. After pressing PUT we receive in the return window on the right hand the list:

```
["count_time", "frame_count_time", "frame_period", "nframes_sum", "frame_time"]
```

This is the list of parameters that have been implicitly changed. The EIGER 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) needs to be changed as well, because obviously it must be greater than the count time. A GET request on `http://<IP_of_DCU>/detector/api/1.0.0/config/frame_time` tells us that frame_time is now slightly greater 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/1.0.0/config/<some parameter>`", detector commands are transferred via PUT requests "`http://<IP_of_DCU>/detector/api/1.0.0/command/<some command>`". Finally the status of the detector may be queried via "`http://<IP_of_DCU>/detector/api/1.0.0/status/<status parameter>`". Besides the detector interface (the URLs starting with /detector) there is a monitor interface ("`http://<IP_of_DCU>/monitor/api/1.0.0/...`") and a filewriter interface ("`http://<IP_of_DCU>/filewriter/api/1.0.0/...`"). The monitor interface lets you watch the currently running measurement, the filewriter interface lets you control how the data 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.

2. 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 power-up of either the detector or the detector control unit
  - Depending on System configuration, this may take up to 2 Minutes
  - Blocking operation, no other API operation has to be performed until successful completion
  - [See API → Commands → Initialize]

- Configuration
  - Although not resulting in error if not, the user should set the required parameters for the experiment
  - If nothing is configured, defaults will be used (chosen to acquire one image of 1 second at 6000 keV)
  - [See API → Configuration]

- Arm the detector (mandatory)
  - This uploads the configuration to the modules in the detector an prepares the system to be ready for data acquisition, but does not yet activate acquisition
  - Depending on System configuration, this may take up to 2 Minutes. If the configuration does not change, arm will be performed very fast.
  - [See API → Commands → Initialize]

- 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 API → Commands → Trigger]

- Disarm the detector
  - Disables the Trigger unit
  - Note: The last acquired image (or the image, if only one image was configured) will only be available after this command
  - [See API → Commands → Initialize]

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

configure	(optional)
arm	(mandatory)
trigger	(mandatory for internal trigger, omit for external trigger/enable)
disarm	(mandatory)

- The status of the EIGER detector system may be optionally checked during acquisition, but the values will only be updated if there is no actual acquisition active

- For receiving the data two options are available:
  - Downloading the HDF5 files

    See API: Filewriter for details.
  - Direct stream-like access on the images

    See API: Monitor interface for details

2.1. 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 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 a **get** or **put** operation 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 through a **get** operation.
- For every configurable parameter with numeric data type, the minimal and maximal value can be requested if available. For enumerated data types, the available values can be inquired.
- Any **put** request changing parameters may implicitly change depending 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.

2.1.1. URLs

Note:

Please replace <IP_of_DCU> with the IP-Address on which you can reach your detector control unit (e.g. 10.42.42.20 if you are using the preconfigured fixed address on interface em2).

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

detector: Used to configure the detector and the readout system, to control the data acquisition and request the detector status
`http://<IP_of_DCU>/detector/api/<version>`

monitor: Used to receive single frames at a low rate.
`http://<IP_of_DCU>/monitor/api/<version>`

filewriter: Configuration of the HDF5 filewriter.
`http://<IP_of_DCU>/filewriter/api/<version>`

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 small datasets, the http body contains a JSON string, described below. Larger datasets are returned/set as hdf5 files. 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 appropriately in all responses"

3. API

Note:

Please replace <IP_of_DCU> with the IP-Address on which you can reach your detector control unit (e.g. 10.42.42.20 if you are using the preconfigured fixed address on interface em2).

3.1. Detector Subsystem

The detector subsystem has the base URL:

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

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

3.1.1. 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/

configuration parameter	resource	datatype	Acc.	Remarks
	<base_path>/	configuration	rw	If the html header item accept=hdf5 is used, an hdf5 file containing the configuration is returned. **Put**ting an hdf5 file will set all values in the file recursively. For accept=JSON, the configuration is returned in the JSON format. Remark: Images will NOT be included (flatfield, pixel_mask) due to their size. Note:The flatfield and pixel_mask are included in the *master.h5 file of each acquisition
count_time	<base_path>/count_time	float	rw	Exposure time in seconds Default value: 0.5
frame_time	<base_path>/frame_time	float	rw	Exposure period in seconds Default value: 1.0
nimages	<base_path>/nimages	uint	rw	Number of images Default value: 1
photon_energy	<base_path>/photon_energy	float	rw	Photon energy in eV. Default value: 8000
wavelength	<base_path>/wavelength	float	rw	Sets the detector to photon wavelength in nm.
element	<base_path>/element	string	rw	Sets the detectors threshold to the fluorescence radiation energy of an element. Discrete Element (e.g. Cu, Mo, Au, etc.) _

threshold_energy	<base_path>/f_energy	float	rw	Sets the value of the threshold in eV This is set automatically by the photon energy.
flatfield	<base_path>/flatfield	float[][]	rw	Form DECTRIS generated flatfield correction coefficients for each pixel. Note: The flatfield is included in the hdf5 files of an acquisition.
flatfield_correction_applied	<base_path>/flatfield_correction_applied	bool	rw	Activates/Deactivated the flatfield correction. Default value: True
pixel_mask	<base_path>/pixel_mask	uint[][]	rw	Form DECTRIS generated mask for each pixel. Marks defective / disabled / non-active Pixels and how these will be represented in the data. Note: The pixelmask is included in the hdf5 files of an acquisition.
number_of_excluded_pixels	<base_path>/number_of_excluded_pixels	uint	r	Number of defective / disabled / non-active Pixels
countrate_correction_applied	<base_path>/countrate_correction_applied	bool	rw	Activated/deactivates the countrate corrections Default value: True
auto_summation	<base_path>/auto_summation	bool	rw	Activated/deactivates the auto summation. Default value: True Note: Should be always activated. If switched of the pixel counters may overflow very early (4096 Counts) and after a short time, depending on photon flux.
sub_image_count_time	<base_path>/sub_image_count_time	float	r	The count_time of a sub image. Note: For acquiring an image with the count time set by the user, the detector will divide the acquisition in several sub images and add them up internally. The effective “loss” of active time is below < 1%.
summation_nimages	<base_path>/summation_nimages	uint	r	The number of acquired sub images. See remark on sub_image_count_time.
trigger_mode	<base_path>/trigger_mode	string	rw	The active trigger mode. Default value: ints Available options: ints: internal (software programmable) trigger exts: external trigger exte: external enable signal
data_collection_date	<base_path>/data_collection_date	string	rw	
beam_center_x	<base_path>/beam_center_x	float	rw	
beam_center_y	<base_path>/beam_center_y	float	rw	

detector_distance	<base_path>/detector_distance	float	rw	
detector_origin	<base_path>/detector_origin	float[3]	rw	
detector_origin_rotated	<base_path>/detector_origin_rotated	float[9]	rw	
countrate_correction_count_cutoff	<base_path>/countrate_correction_count_cutoff	uint	r	
bit_depth_readout	<base_path>/bit_depth_readout	int	r	
detector_readout_time	<base_path>/detector_readout_time	float	r	
description	<base_path>/description	string	r	Type of detector
detector_number	<base_path>/detector_number	string	r	Serial number
software_version	<base_path>/software_version	string	r	
sensor_material	<base_path>/sensor_material	string	r	
sensor_thickness	<base_path>/sensor_thickness	float	r	
x_pixel_size	<base_path>/x_pixel_size	float	r	
y_pixel_size	<base_path>/y_pixel_size	float	r	
x_pixels_in_detector	<base_path>/x_pixels_in_detector	uint	r	
y_pixels_in_detector	<base_path>/y_pixels_in_detector	uint	r	

For a description of the parameters, see:

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

3.1.2. 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 as 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.
“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.

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.

Remarks:

- It is highly recommended to change a configuration by **put**ting 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.

3.1.3. Example – Setting photon_energy

As already mentioned, when setting a value, the DCU sends the names of the parameters, which were subsequently changed (to have a valid configuration) in the answer of the set request.
If the photon energy is e.g. set to 8040 keV and the answer tells which values were changed as well.

The following example uses some python libraries as a web client to send http requests:

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
payload = {'json_playload': data_json}
       # Set the request payload (the prepared JSON dictionary)
r = requests.put('http://<IP_of_DCU>/detector/api/<version>/config/photon_energy',
data=payload)
       # 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, also the threshold energy and the applied flatfield were changed.

3.2. 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 interaction due to operation conditions.

3.2.1. 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 the value got updated
“state”	<state>	invalid, normal, critical, disabled
“critical_limits”	<list containing minimal and maximal parameter_value>	Returns the minimal and maximal 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.

3.2.2. 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 of 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	Returns the temperature value of the detector

board_000/th0_humidity	<base_path>/ board_000/th0_humidity	float	r	Returns the humidity value of the detector

3.3. 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/

command parameter	resource	Return Value	Access	Remarks
initialize	<base_path>/initialize	-	w	Initializes the detector
arm	<base_path>/configure	sequence_id: int	w	Loads configuration to the detector and arms the trigger unit
disarm	<base_path>/start	sequence_id: int	w	Flushes all images 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, but after the next image was finished
abort	<base_path>/abort	sequence_id: int	w	Aborts all operation and resets the system right away

If an error occurs the http error code “400” is returned. (In this case, please consult the API log which can be accessed by the web interface of the DCU **Error! Hyperlink reference not valid.**)

3.4. Monitor interface

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

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

3.4.1. Monitor configuration

The configuration is applied at the URL:

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

with the following commands:

configuration parameter	resource	data type	Access	Remarks
mode	<base_path>/mode	bool	rw	enabled (keep old, drop new), disabled

buffer_size	<base_path>/buffer_size	int	rw	number of images getting buffered, default: 1 Note:

3.4.2. 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`

data	resource	data type	Access	Remarks
	<base_path>	Json List	r	List of available images [[seriesId, [imfId, imgId, ...]], ...]
monitor	<base_path>/monitor	tif	r	Gets latest image and discards it from the buffer Default waits for 500 ms for image Timeout via ?timeout=[ms] adjustable Returns 408 if no image available
next	<base_path>/next	tif	r	Gets next image and discards it from the buffer Default waits for 500 ms for image Timeout via ?timeout=[ms] adjustable Returns 408 if no image available
<series>/<id>	<base_path>/<series>/<id>	tif	r	Gets corresponding image, if not available return http 404 Not Found

3.4.3. Monitor Status Parameter

At the address

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

the status of the monitor can be requested.

status parameter	resource	data type	Access	Remarks
state	<base_path>/state	string	r	("normal", "overflow") Overflow if images got dropped (status/dropped > 0)
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 got 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

3.4.4. Monitor control parameter

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`

control parameter	resource	data type	Access	Remarks
clear	<base_path>/clear	-	-	Drops all buffered images and resets status/dropped to zero

3.5. File Writer

The data itself, the frames, are by default written to HDF5 files, where the meta data is stored in the NeXus compliant metadata standard. These files can be accessed through the EIGER API. Frames can also be obtained through the EIGER API with the monitor interface, but at a low performance (see point 3.4). This is mostly used in order to monitor the data.

In a later implementation, the data stream can be also accessed directly through its own API.

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

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

To configure the filewriter, this url is used:

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

configuration parameter	resource	data type	Access	Remarks
mode	<base_path>/mode	string		disabled, enabled
transfer_mode	<base_path>/transfer_mode	string	rw	The mode the files are transferred to the user system. Currently we only support “http”
nimages_per_file	<base_path>/nimages_per_file	int	rw	Max. number of frames stored in one HDF5 file.
name_pattern	<base_path>/name_pattern	string	rw	The basename of the file. The pattern $id will include the sequence number in the file name. As default the following name pattern is used: series_$id. Then the master file will be called series_{sequence_nr}_master.h5, the data files: series_{sequence_nr}_data_{filenr}.h5,
compression_enabled	<base_path>/compression_enabled	bool	rw	enable/disable LZ4 compression

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.

The filewriter is started automatically when the data taking is started. Thus, there is no need for specific filewriter commands. 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:

status parameter	resource	data type	Access	Remarks
state	<base_path>/state	string	r	disabled, ready, acquire, error
error	<base_path>/error	string	r	returns list of status parameters causing error condition
time	<base_path>/time	date	r	returns current system time
buffer_free	<base_path>/buffer_free	int	r	The buffer space left in KB.