EUROPEAN
SPALLATION
SOURCE

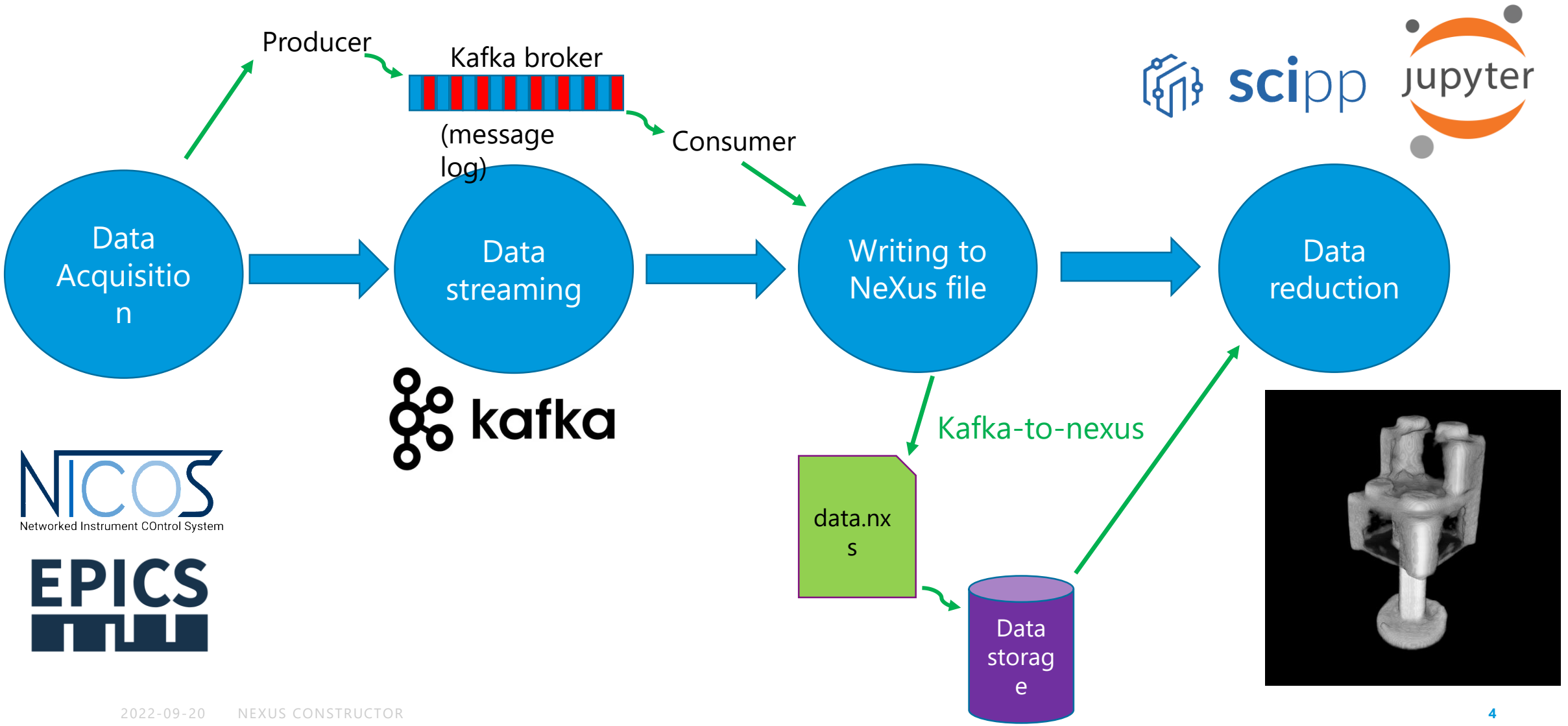
Taking the Pain out of Generating Complete and Compliant NeXus Files for Dynamic Setups

Kenan Murić, Tobias Richter
European Spallation Source

2022-09-20

Agenda

Motivation



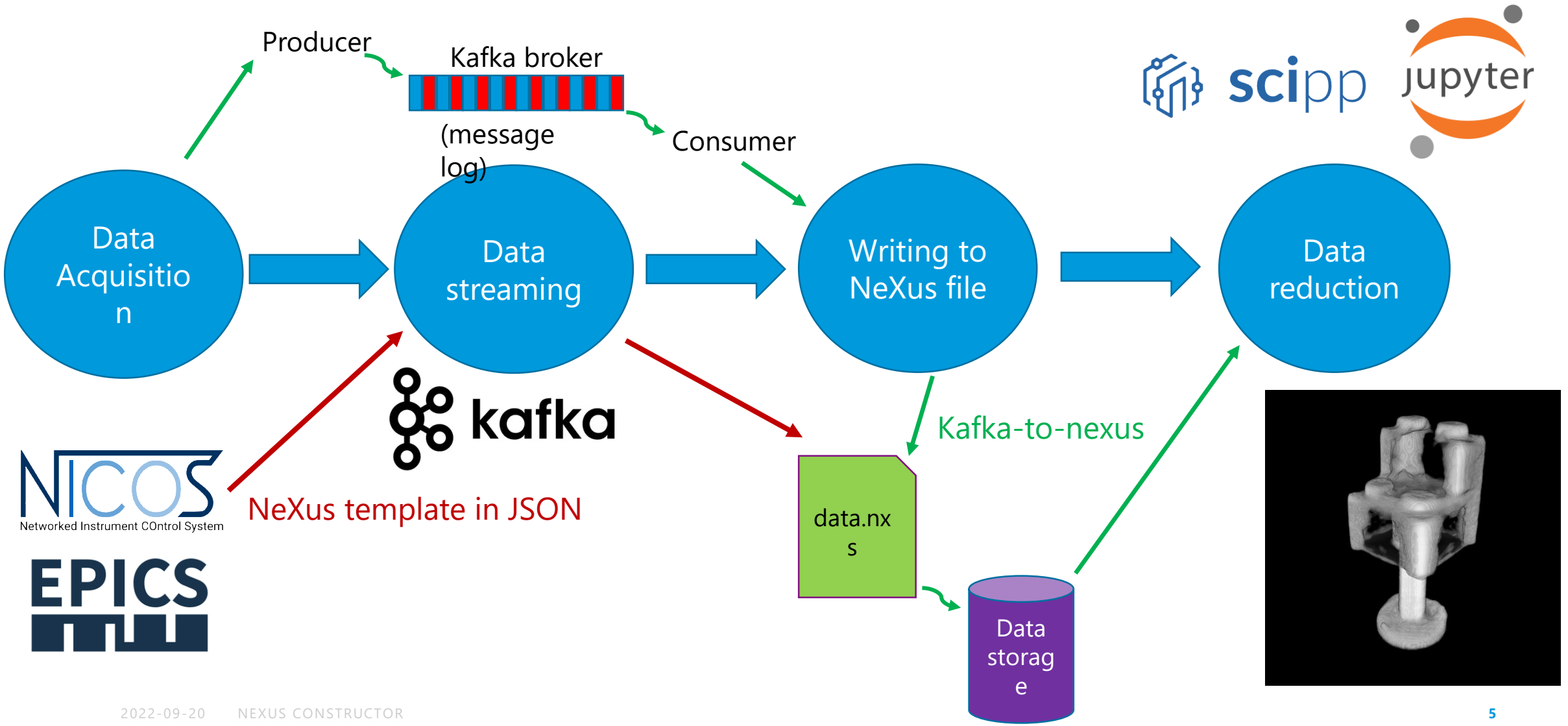
NICOS
Networked Instrument COntrol System

EPICS

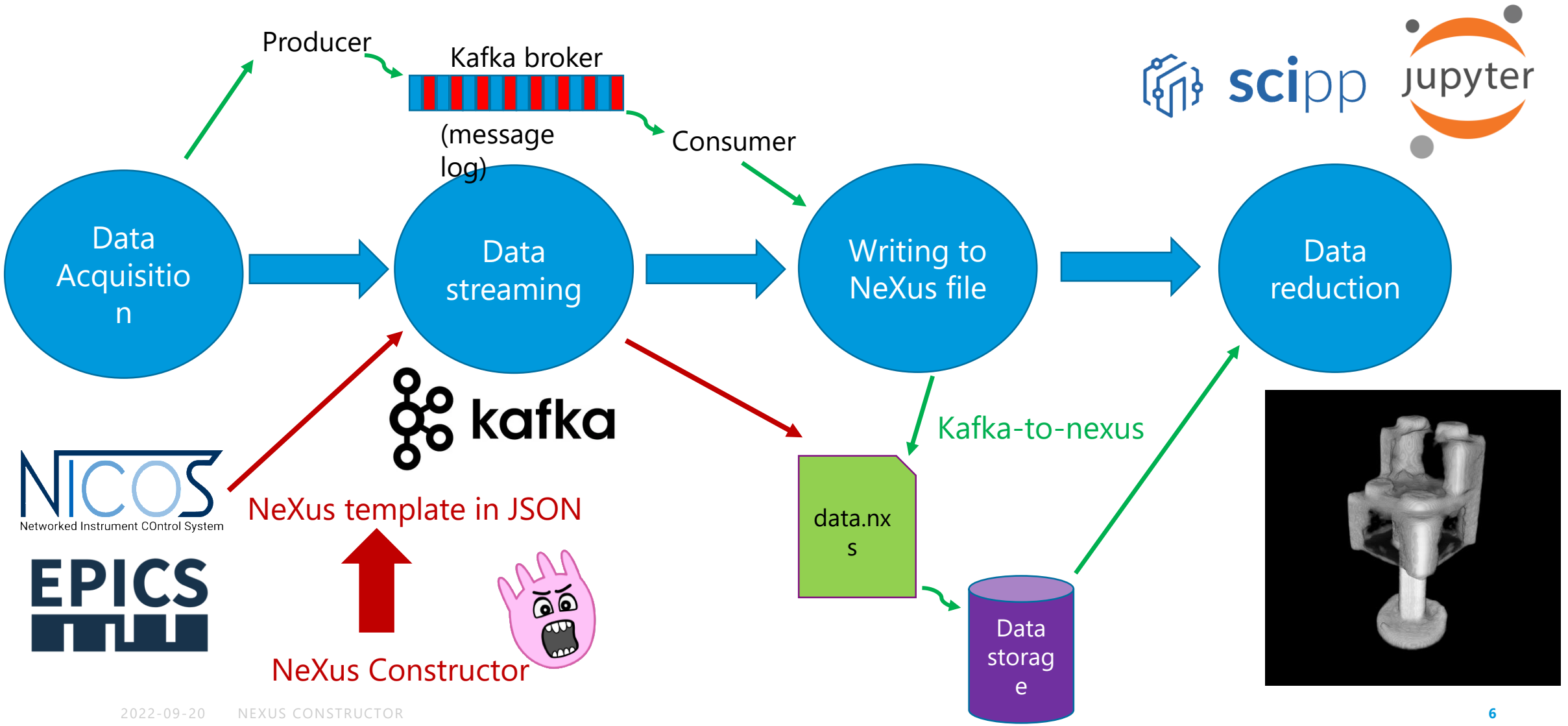
scipp

jupyter

Motivation


Producer
Kafka broker
(message log)
Consumer
Data Acquisition
Data streaming
Writing to NeXus file
Data reduction
kafka
NICOS
Networked Instrument COntrol System
NeXus template in JSON
EPICS
data.nxs
Kafka-to-nexus
Data storage
scipp
jupyter

Motivation


Producer
Kafka broker
(message log)
Consumer
Data Acquisition
Data streaming
Writing to NeXus file
Data reduction
scipp
jupyter
kafka
NICOS
Networked Instrument COntrol System
EPICS
NeXus template in JSON
NeXus Constructor
Kafka-to-nexus
data.nxs
Data storage

Motivation

The **NeXus** file will contain:

- Groups with corresponding NeXus classes.
- Static data (e.g. experiment title, users etc.).
- Dynamic data.
- Alarm information from instrument devices.
- Transformations (dynamic and static) that define instrument component locations and rotations.

At ESS, our data structures will follow the NeXus standard specification as much as possible!

Name	Description	Type	Shape
964281_00003931.hdf			
entry	["Unassigned experiments"]	NXentry	
data		NXdata	
definition	["NXtomo"]	string	1
experiment_identifier	["964281"]	string	1
instrument		NXinstrument	
flir_camera		NXdetector	
data		NXlog	
cue_index	[]	uint32	0
cue_timestamp_zero	[]	uint64	0
time	1D data	uint64	416
value	3D data	float64	416 × 480 × 830
image_key		NXlog	
alarm_severity	1D data	string	7
alarm_status	1D data	string	7
alarm_time	1D data	uint64	7
average_value	[1.71429]	float64	1
connection_status	[]	string	0
connection_status_time	[]	uint64	0
cue_index	[]	uint32	0
cue_timestamp_zero	[]	uint64	0
maximum_value	[3]	float64	1
minimum_value	[0]	float64	1
time	1D data	uint64	7
value	2D data	int16	7 × 1
laser_monitor		NXmonitor	
laser_source		NXsource	
light_source		NXsource	
probe	["visible light"]	string	1
mini_chopper		NXdisk_chopper	
sample	["lego"]	NXsample	
depends_on	["/entry/sample/transformations/rotation"]	string	1
name	["lego"]	string	1
transformations		NXtransformations	
title	["Unassigned experiments"]	string	1

Motivation

```
                    "name": "probe",
                    "values": "visible light",
                    "type": "string"
                  }
                }
              ]
            },
            {
              "name": "flir_camera",
              "type": "group",
              "attributes": [
                {
                  "name": "NX_class",
                  "dtype": "string",
                  "values": "NXdetector"
                }
              ],
              "children": [
                {
                  "name": "image_key",
                  "type": "group",
                  "attributes": [
                    {
                      "name": "NX_class",
                      "dtype": "string",
                      "values": "NXlog"
                    }
                  ],
                  "children": [
                    {
                      "module": "f142",
                      "config": {
                        "source": "flir_image_key",
                        "topic": "ymir_nicos_devices",
                        "dtype": "int16"
                      }
                    }
                  ]
                },
                {
                  "name": "data",
                  "type": "group",
                  "attributes": [
                    {
                      "name": "NX_class",
                      "dtype": "string",
                      "values": "NXlog"
                    }
                  ],
                  "children": [
                    {
                      "module": "ADAr",
                      "config": {
                        "source": "some_source",
                        "topic": "ymir_camera",
                        "array_size": "$AREADET$"
                      }
                    }
```

Configuration file the ESS file writer uses to write NeXus files

File writing

Name	Description	Type	Shape
964281_00003931.hdf			
entry	["Unassigned experiments"]	NXentry	
data		NXdata	
definition	["NXtomo"]	string	1
experiment_identifier	["964281"]	string	1
instrument		NXinstrument	
flir_camera		NXdetector	
data		NXlog	
cue_index	[]	uint32	0
cue_timestamp_zero	[]	uint64	0
time	1D data	uint64	416
value	3D data	float64	416 × 480 × 830
image_key		NXlog	
alarm_severity	1D data	string	7
alarm_status	1D data	string	7
alarm_time	1D data	uint64	7
average_value	[1.71429]	float64	1
connection_status	[]	string	0
connection_status_time	[]	uint64	0
cue_index	[]	uint32	0
cue_timestamp_zero	[]	uint64	0
maximum_value	[3]	float64	1
minimum_value	[0]	float64	1
time	1D data	uint64	7
value	2D data	int16	7 × 1
laser_monitor		NXmonitor	
laser_source		NXsource	
light_source		NXsource	
probe	["visible light"]	string	1
mini_chopper		NXdisk_chopper	
sample	["lego"]	NXsample	
depends_on	["/entry/sample/transformations/rotation"]	string	1
name	["lego"]	string	1
transformations		NXtransformations	
title	["Unassigned experiments"]	string	1

Motivation

Some problems to solve…

- Generating template files is a tedious work.

Why?

- Even the simplest cases result in files that are 1000s of lines.

- It is hard to verify the validity of structures with respect to the NeXus standard specification.

https://manual.nexusformat.org/classes/index.html

- Verifying instrument component layouts requires writing your own plotting tools.

- People use different libraries to create template files. Scripts used are usually not well-documented.

- Sometimes manual editing of the JSON file is needed.

NeXus Constructor

Seamlessly building configuration files

- NeXus Constructor comes with a proper and rather simple graphical user interface.

- Developed in *Python 3.7* using *PySide2* (upgrading to *PySide6*) for graphical components.

- You are able to see both the NeXus configuration structure and a 3D rendering of NeXus groups with geometric properties (e.g. choppers, detectors, monitors and slits).

- It is easy to load an existing configuration file and edit it in the NeXus constructor.

- Definitions and documentation of NeXus base classes is integrated in the application.

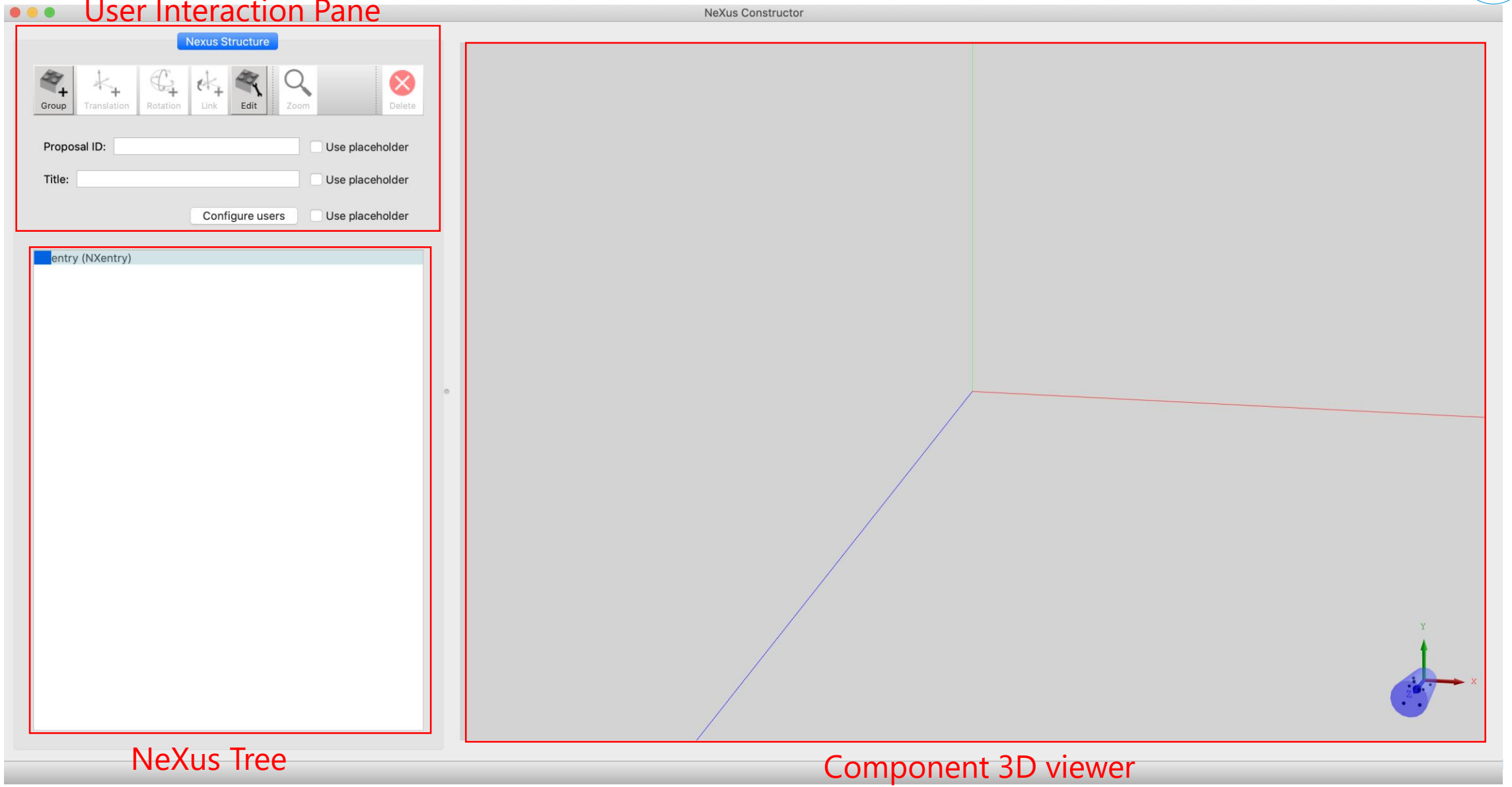
- NeXus constructor suggests group/field names, does verifications of inputs, suggests units etc.

- Works on macOS, Linux and Windows.

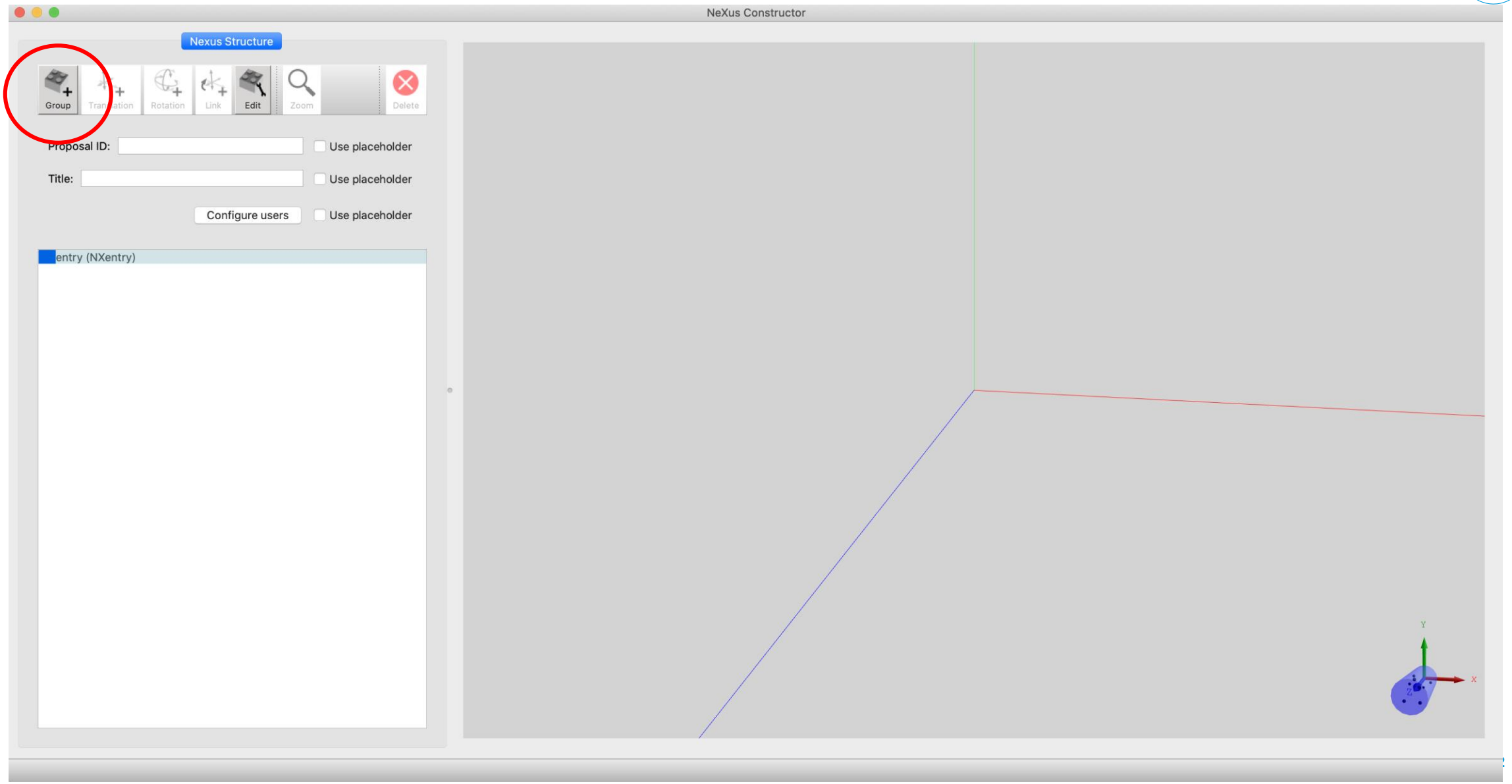
NeXus Constructor is open-source and can be found

https://github.com/ess-dmsc/nexus-constructor

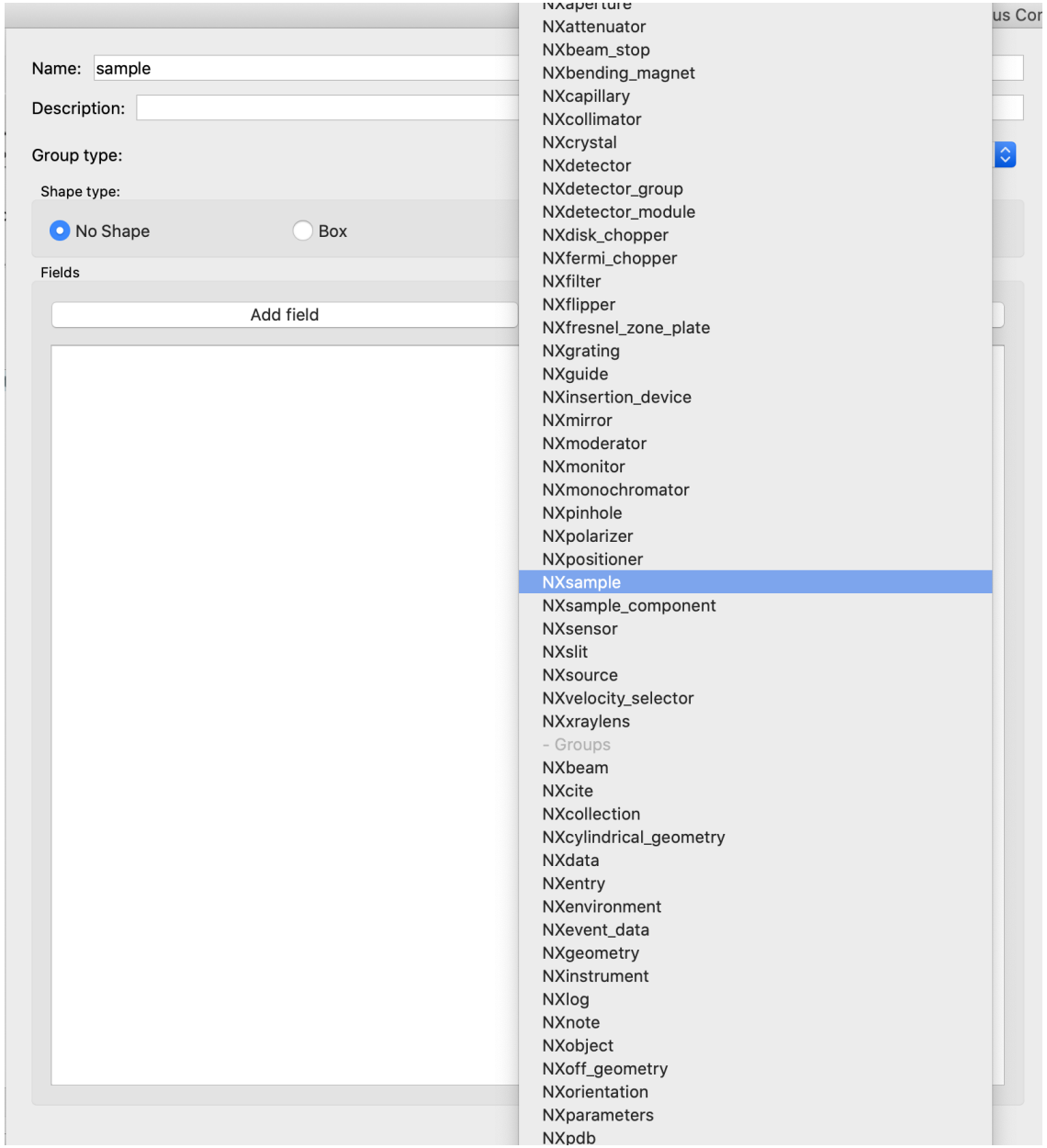
NeXus Constructor



Creating a NeXus group



Creating a NeXus group



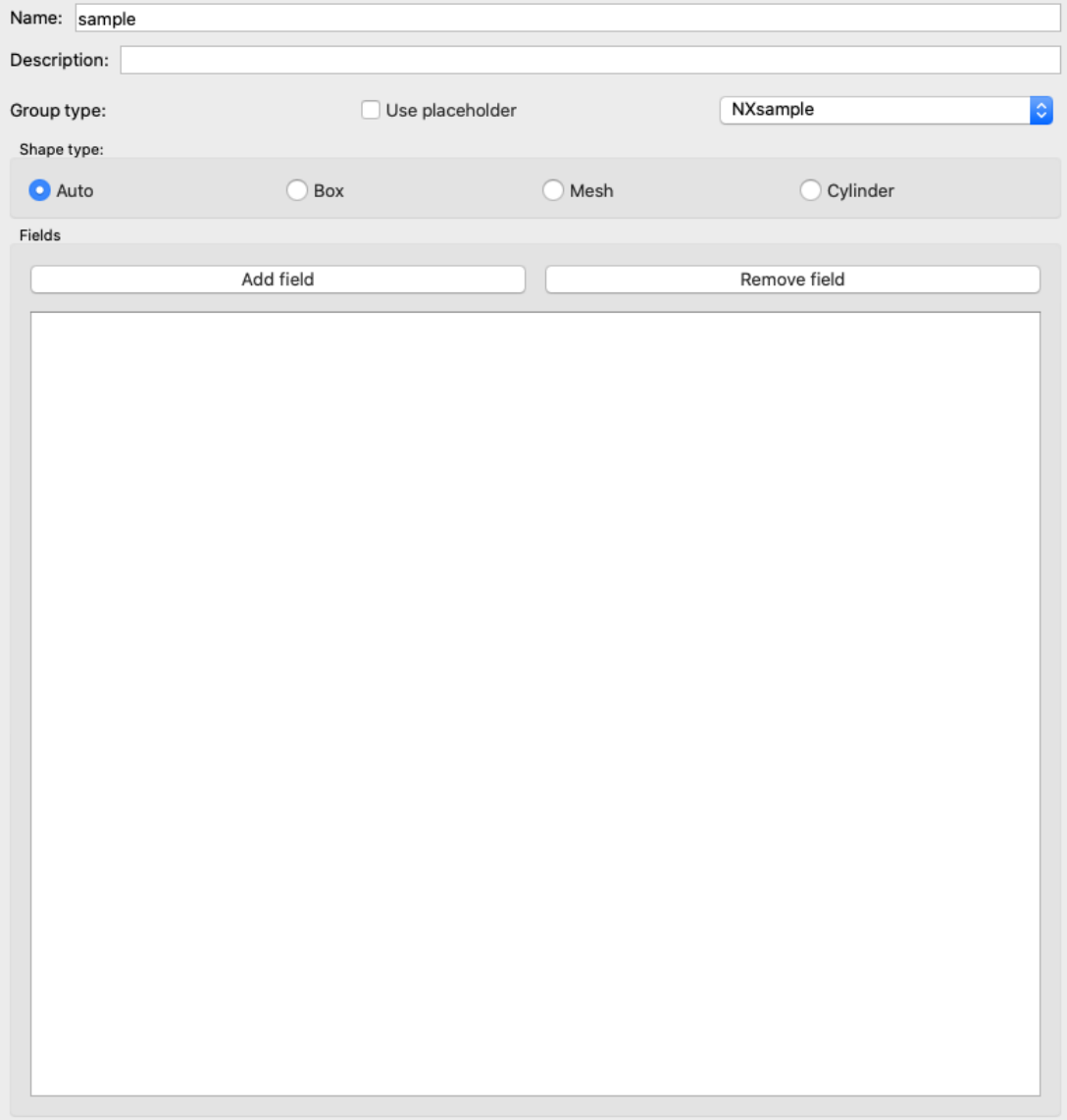
- Choose the desired NeXus class from the list.

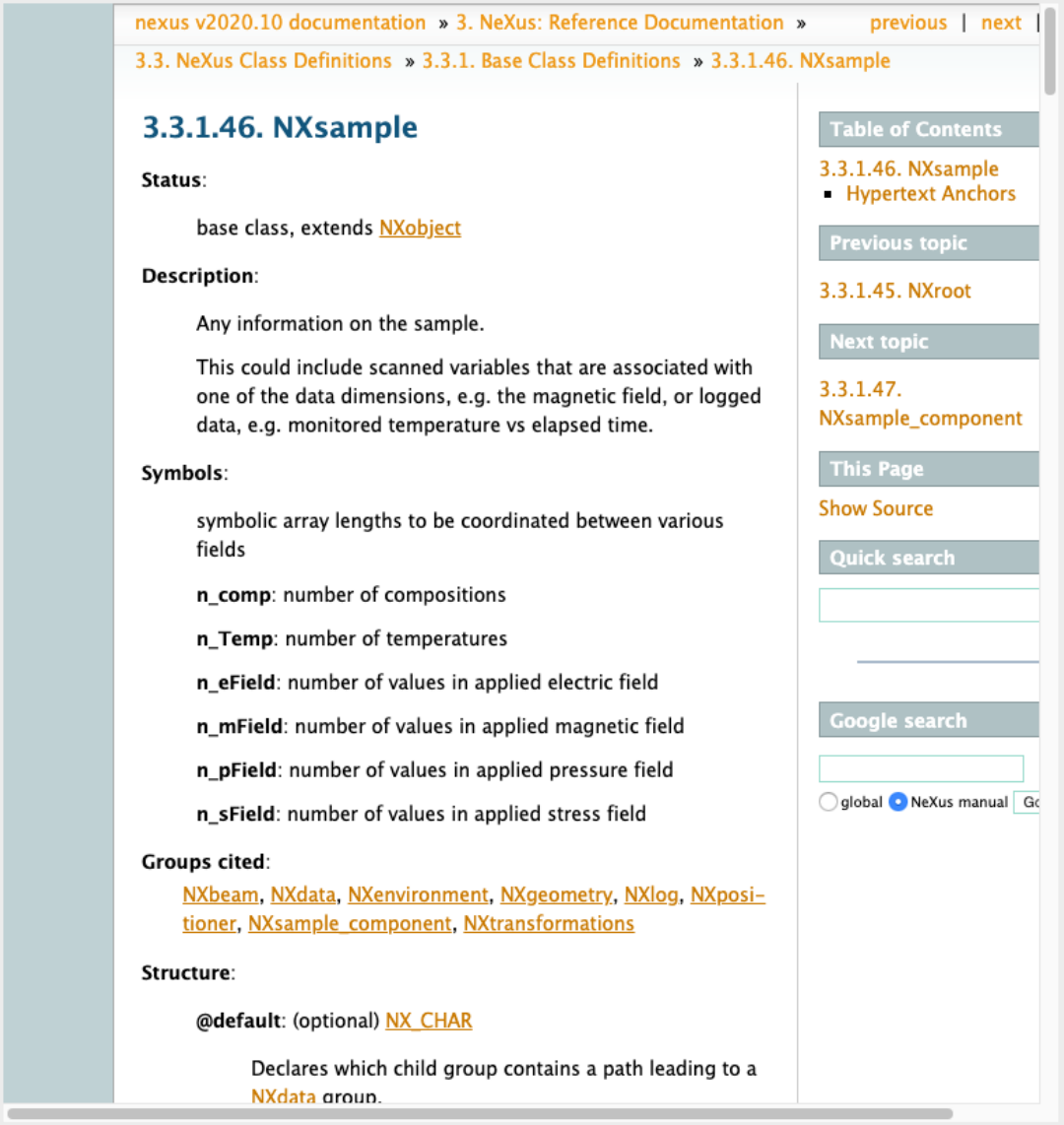
- NeXus classes in the list are split in two categories: components and groups.

- A component is something physical on the beam line, with geometric properties (dimensions, shape and location).

- Suggested NeXus classes are based on the NeXus standard specification.

Creating a NeXus group

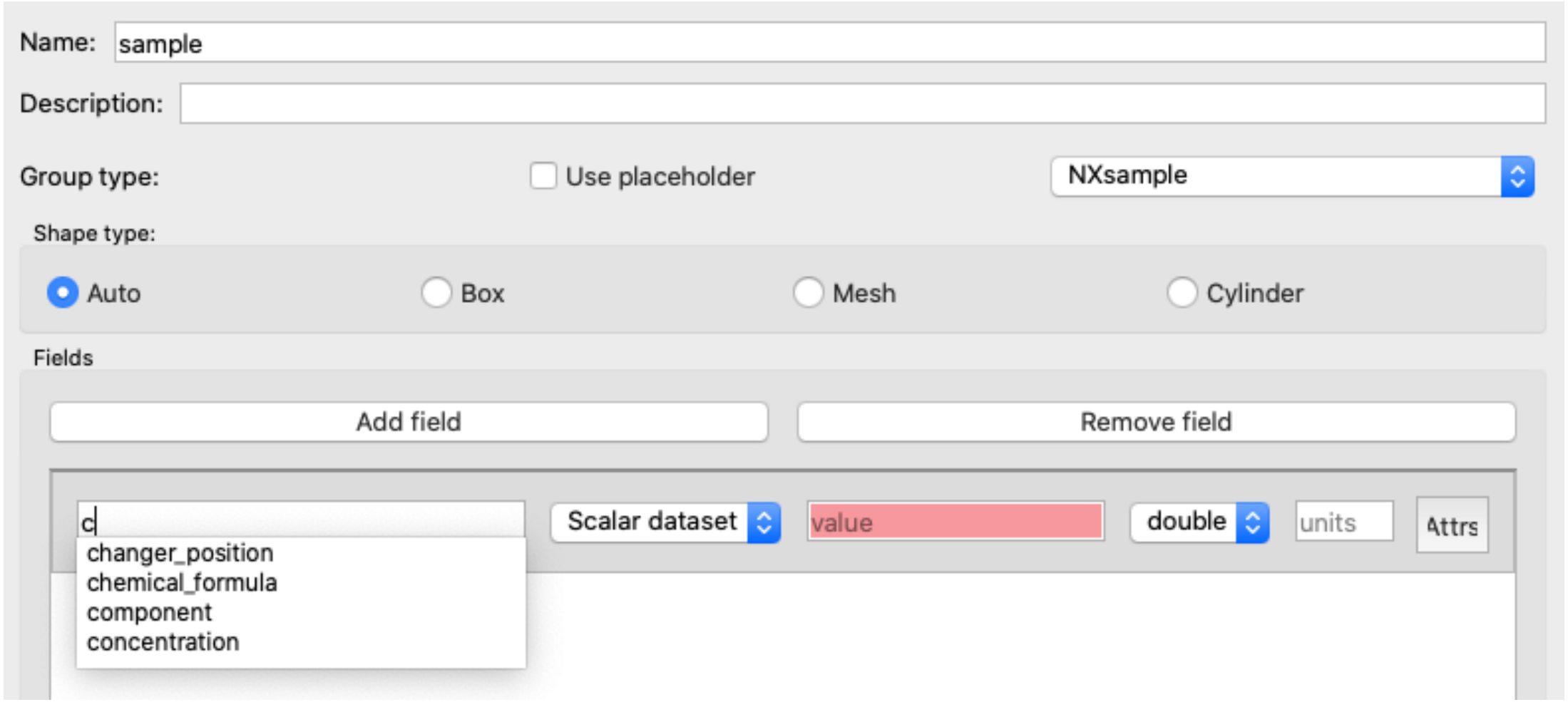




Add group | Cancel

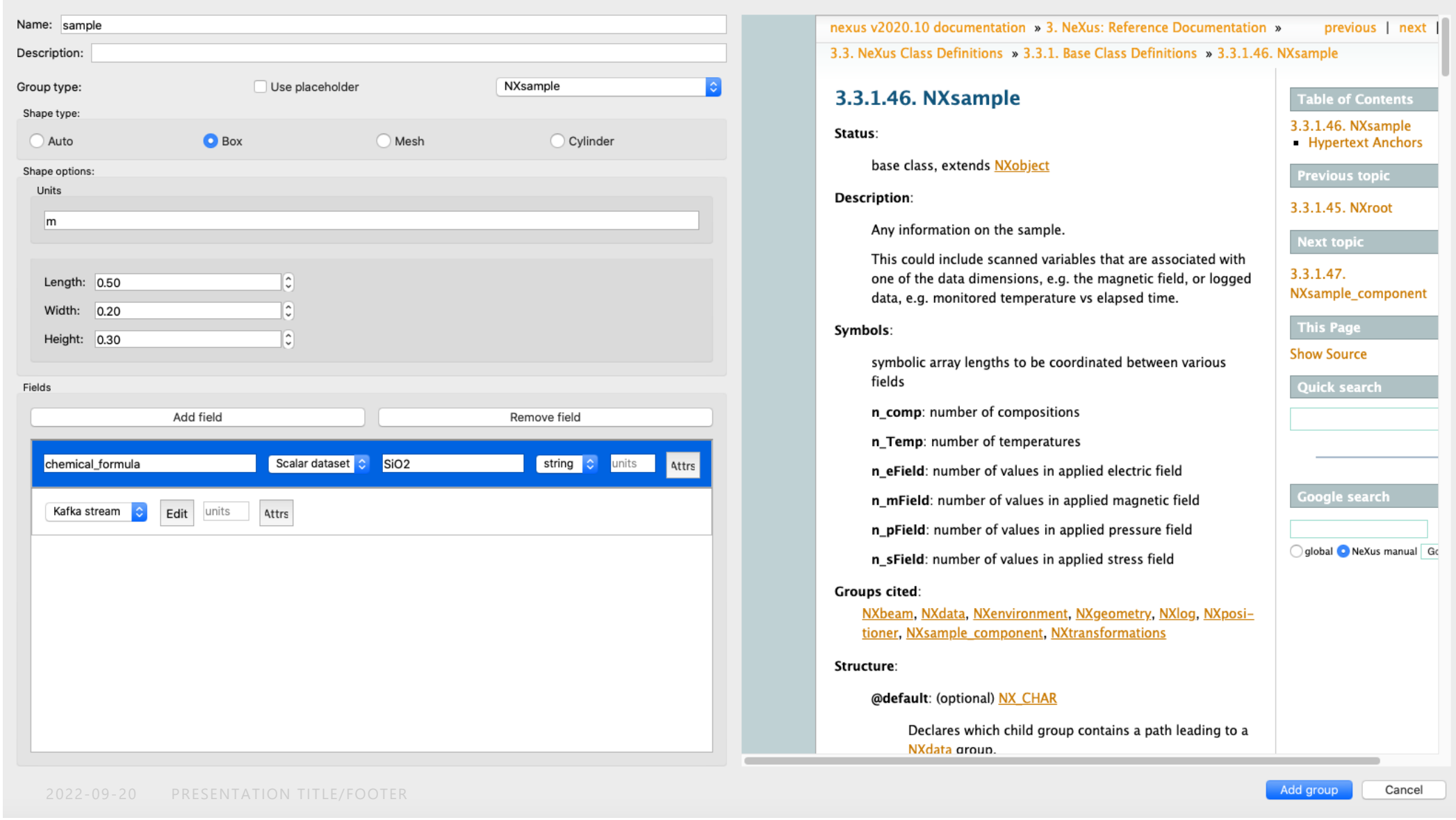
Creating a NeXus group

Creation of a sample using the NXsample class



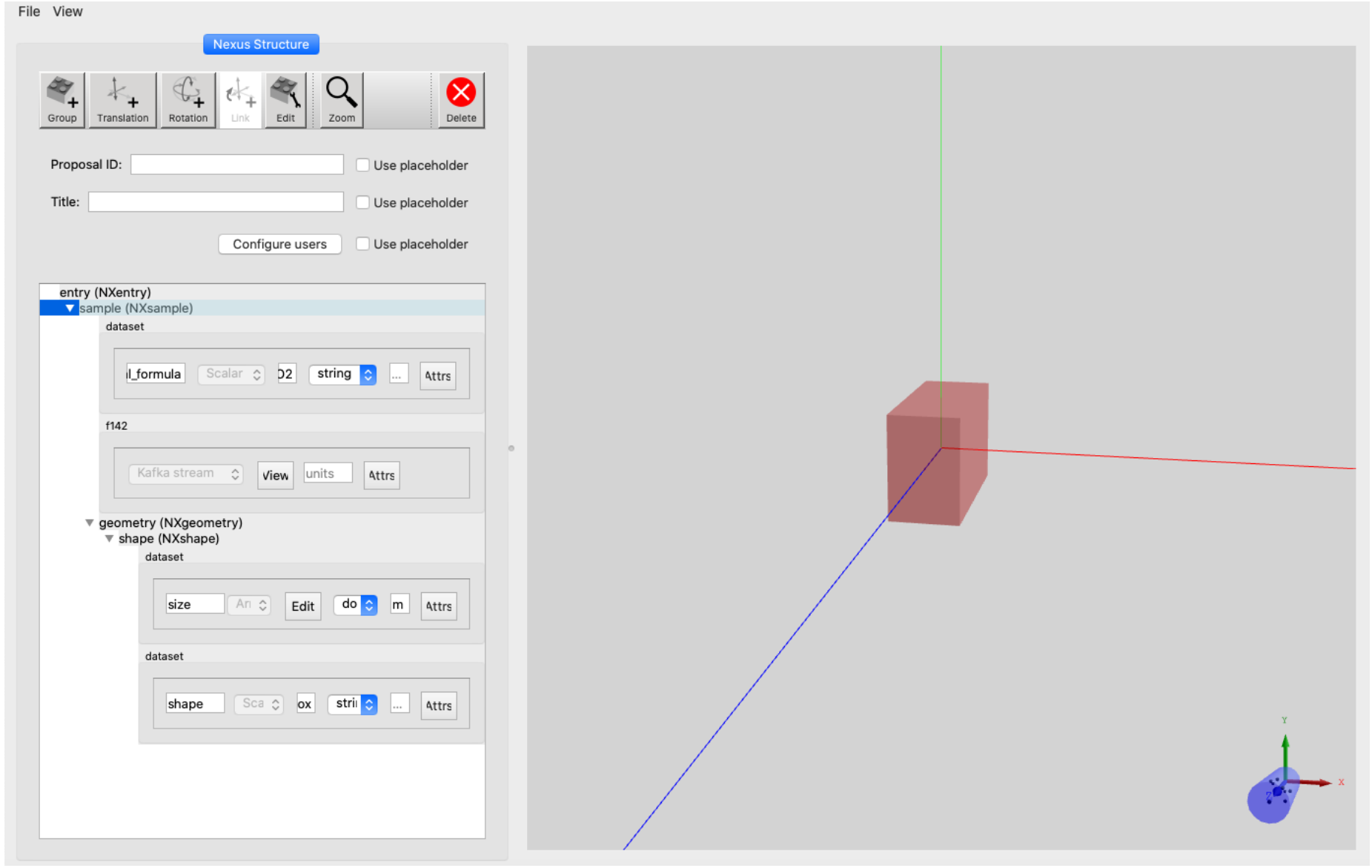
Creating a NeXus group

Creation of a sample using the NXsample class



Creating a NeXus group

Creation of a sample using the NXsample class



Components

Aspects of components in NeXus Constructor

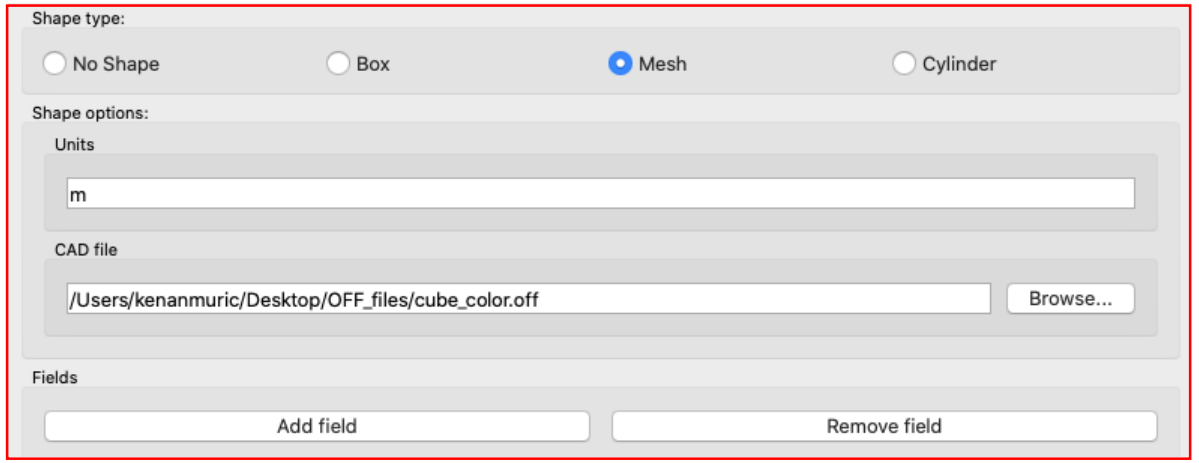
1. A component has dimensions, shape and location.

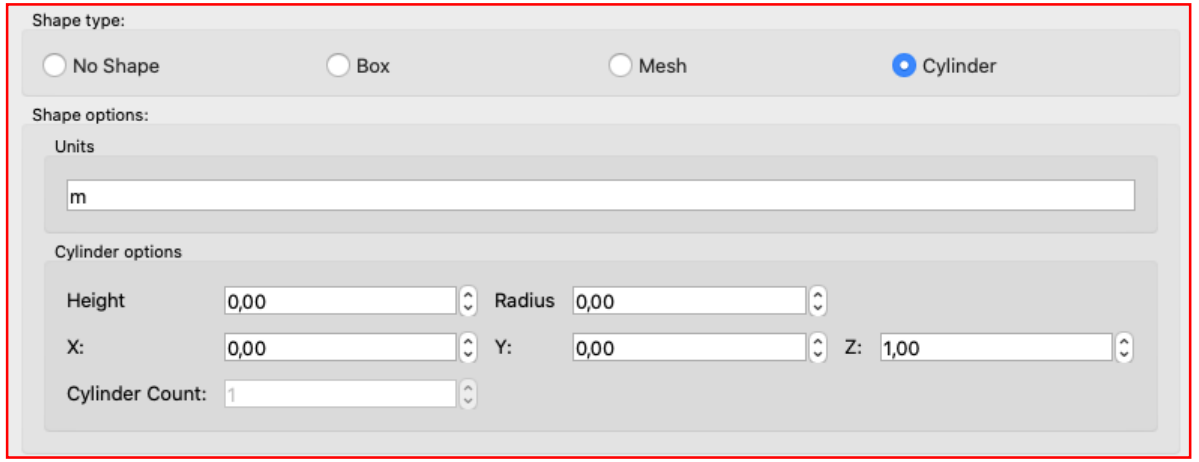
2. Shape type is at the moment limited to following set of shapes:
Auto, **box**, **mesh** and **cylinder**.

3. There are components with customized shapes, such as **NXslit**, **NXdetector** and **NXdisk_chopper**.

4. More shapes and customized components could be added in the future.

Note: the sample is a component!

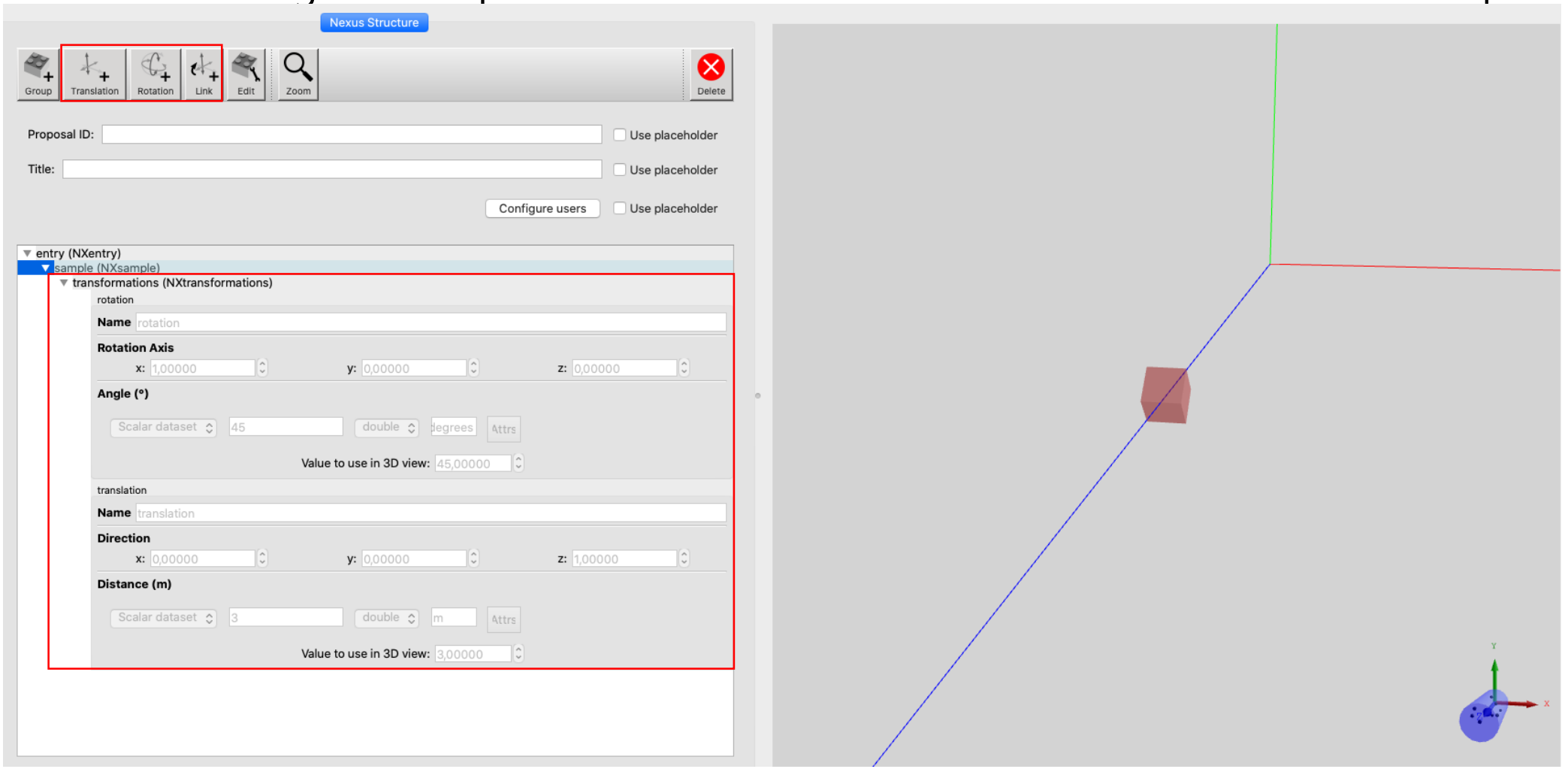




Components

Defining sample (or other component) position and rotation

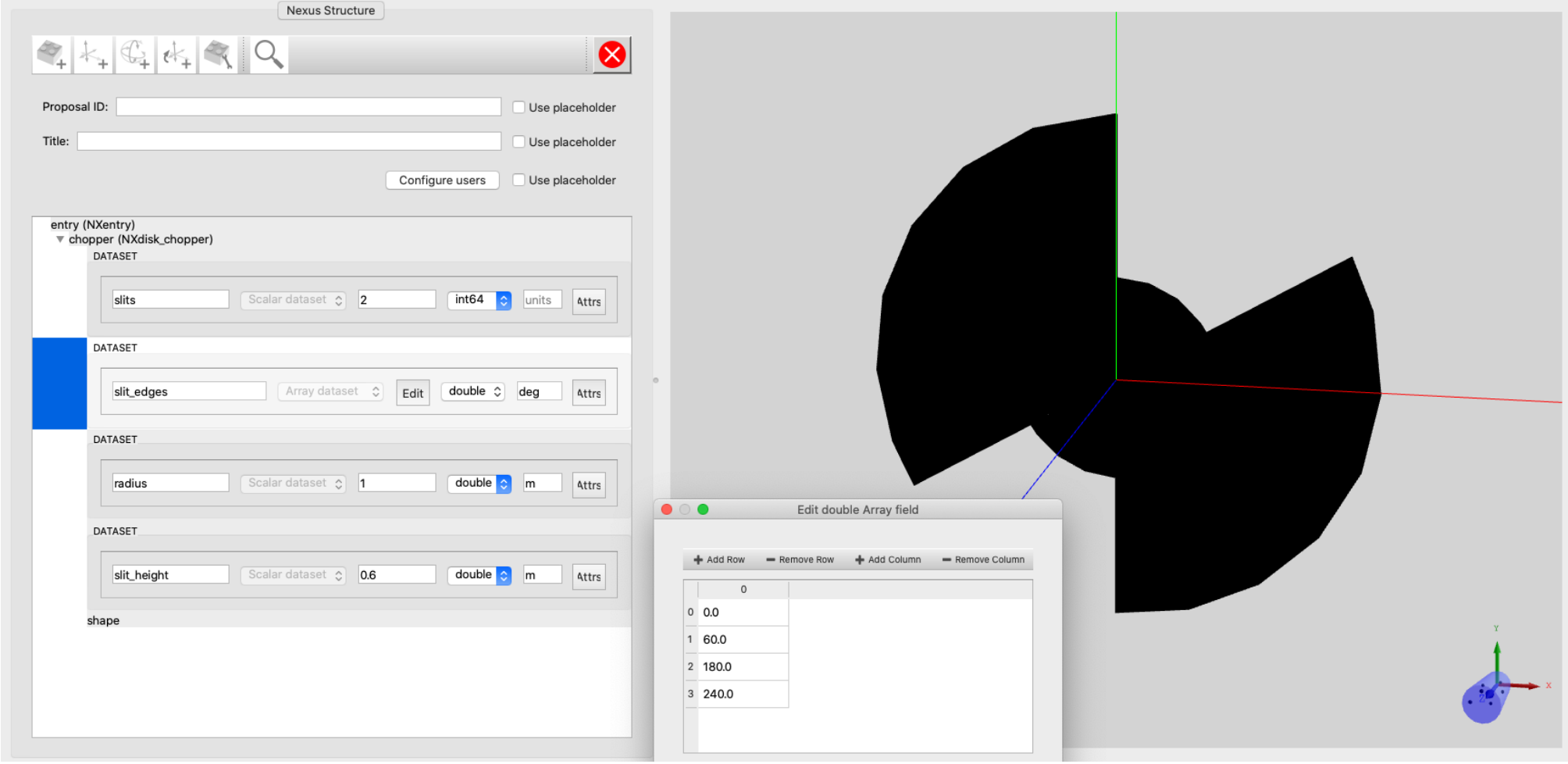
The NeXus standard uses the NeXus class **NXtransformations** to define translations and rotations for a given component. Link is used to link transformations to other components.



Components

Defining a chopper using NXdisk_chopper

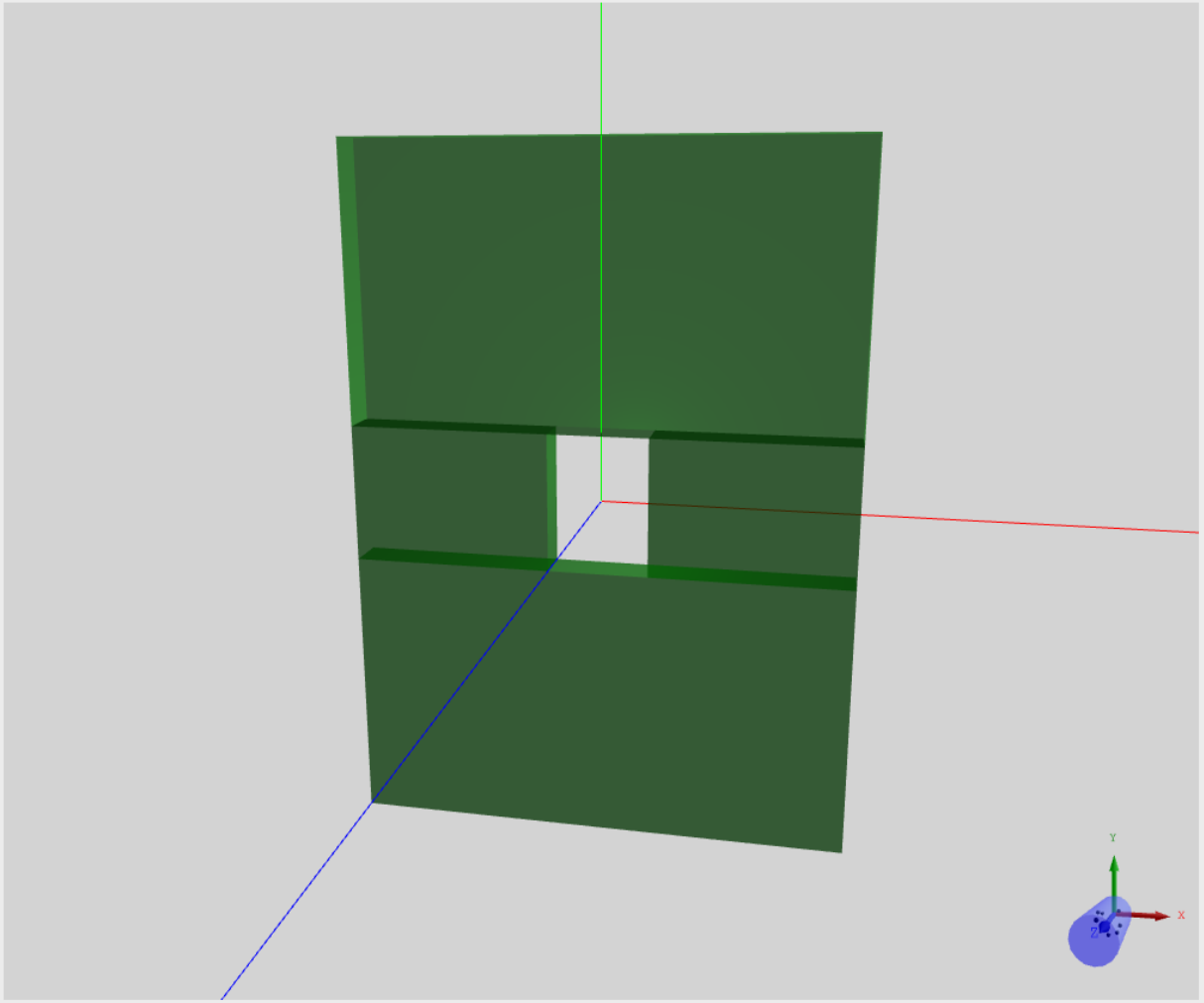
You need to provide radius, slits, slit_edges and slit_height to get a proper disk chopper.



Components

Defining a slit using NXslit

Provide x_gap and y_gap to render a proper slit.



Components

Defining a detector using NXdetector and cylinder shapes

Name: detector

Description:

Group type: NXdetector

Shape type:

Auto Box Mesh Cylinder

Shape options:

Units

mm

Cylinder options

Height 20,00 Radius 30,00

X: 0,00 Y: 0,00 Z: 1,00

Cylinder Count: 1

Pixel options

Pixel layout:

Repeated Single Pixel Shape Entire Shape No Pixels

Pixel grid:

Row: 20 Row height: 70,00

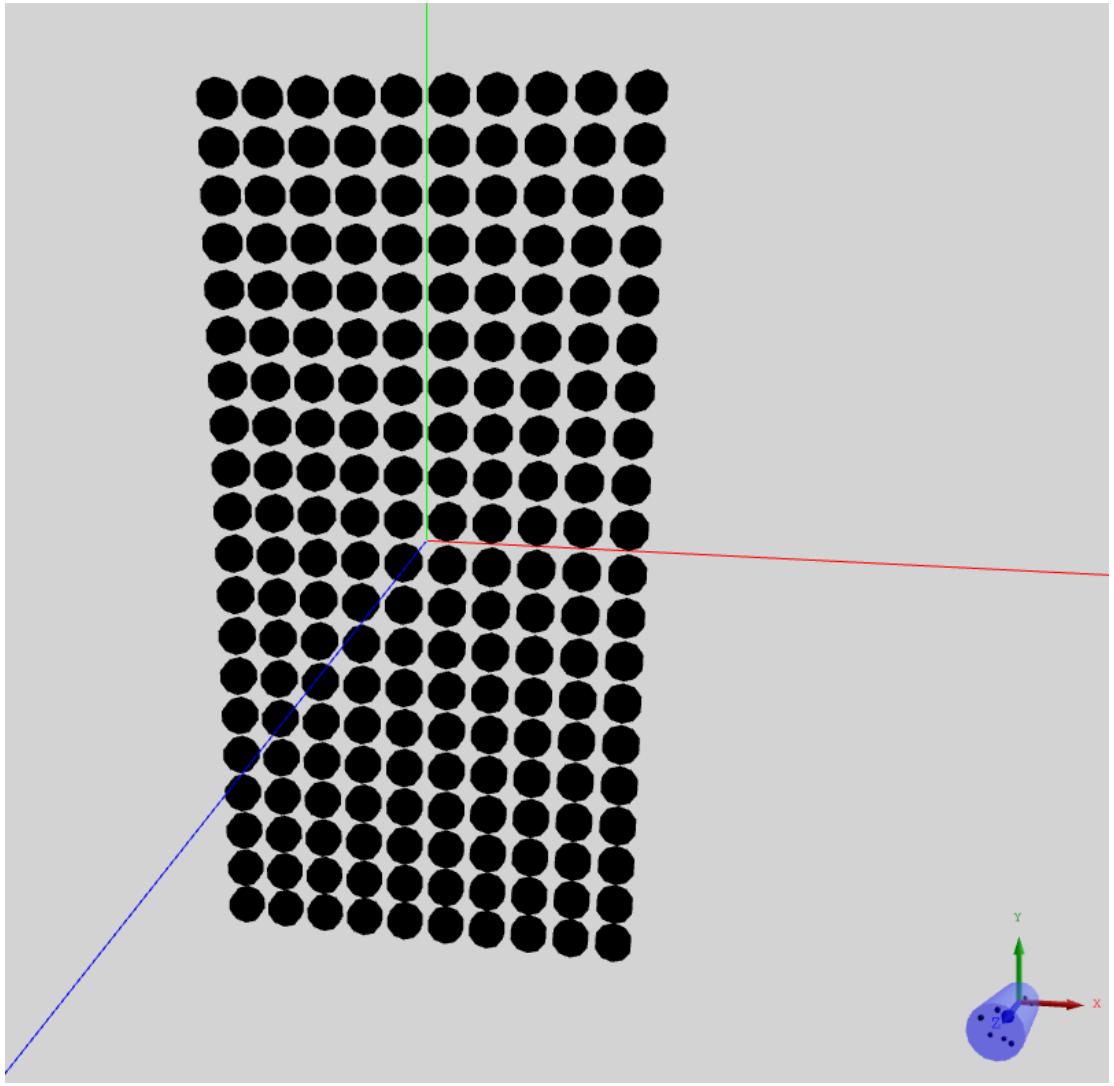
Columns: 10 Column width: 70,00

First ID: 0

Start counting from: Bottom Left

Fields

Add field Remove field

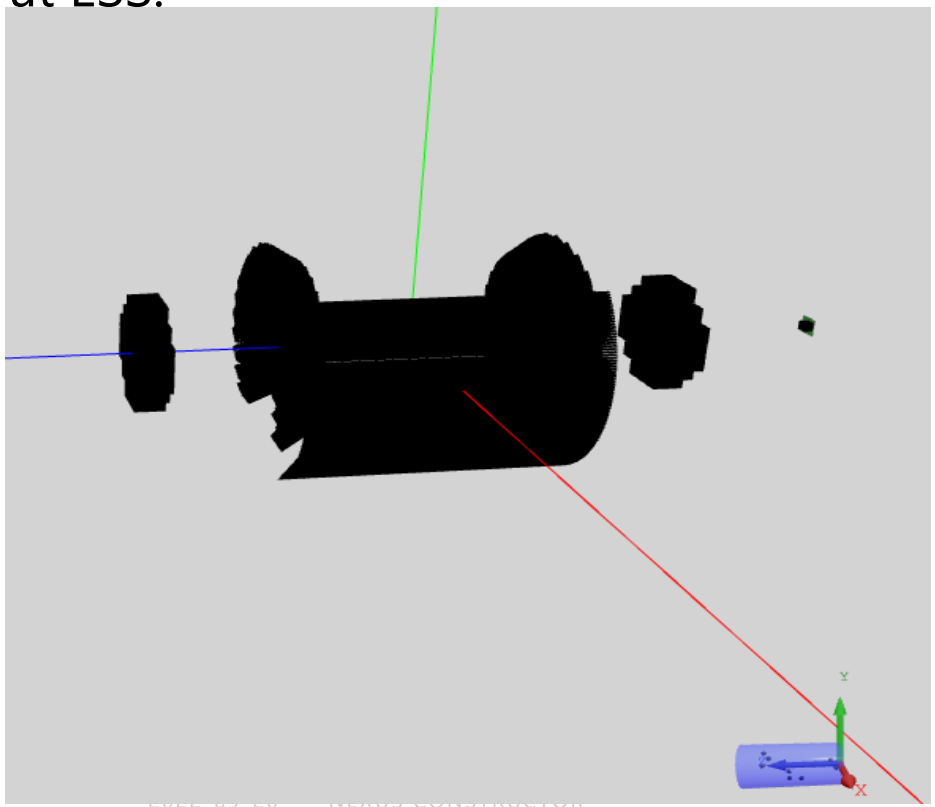


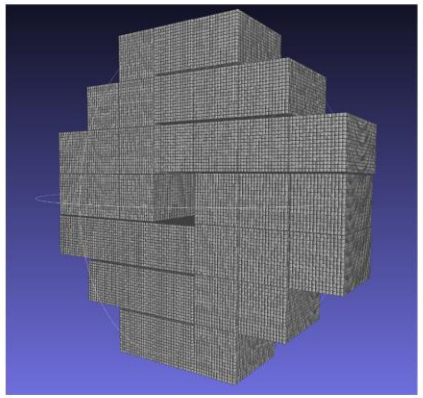
Components

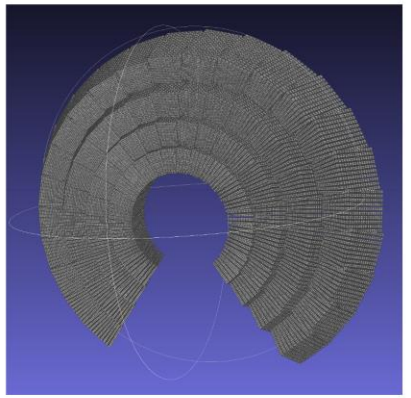
Defining a component with complicated geometry using an OFF file

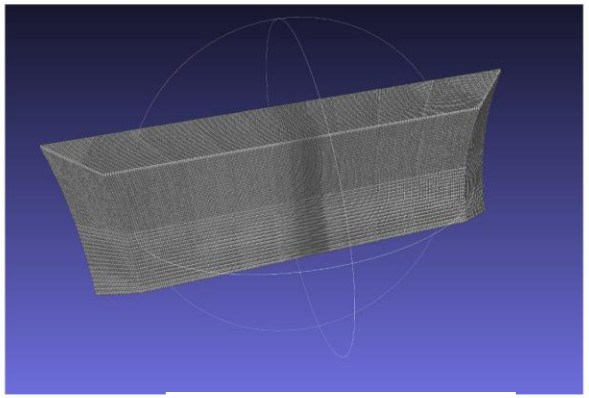
Geometry of the detector used in the different instruments can be defined using an OFF file if the geometry does not conform to the shapes in NeXus Constructor.

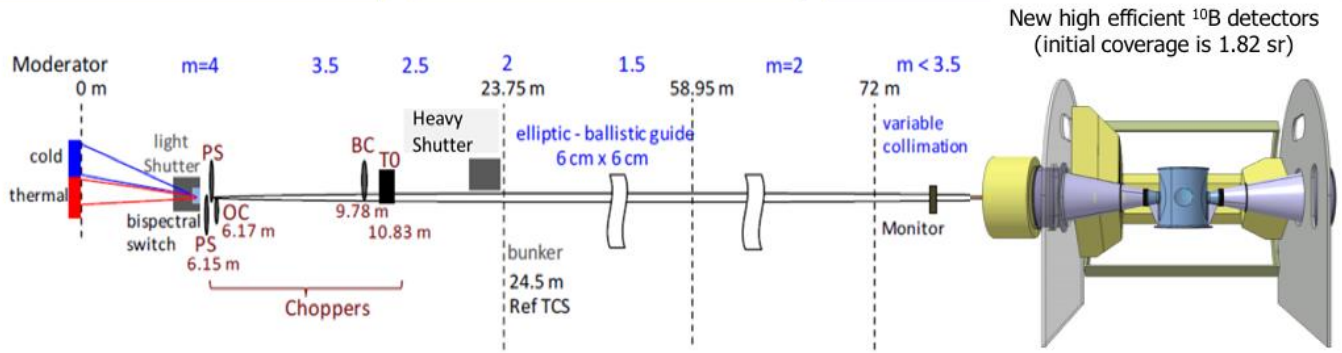
Example of such a case is the DREAM (Bi-spectral Powder Diffractometer) instrument at ESS.











File writer modules

Different fields to add in a group

1. When creating the sample, we added a static dataset containing the chemical formula of the sample.

2. This dataset is added as a field to the instance of the NXsample called sample.

3. However, there are in total 3 types of fields that can be added to a group (or component).

4. These are: datasets, links and streams.

Datasets:

Datasets contain static data that we know the value of beforehand. Added directly to NeXus structure.

Streams:

Streams define how dynamic data is written by the file writer using predefined flatbuffer schemas:

Event data would use **ev42** schema.

Chopper rotation speed would use **f142** schema.

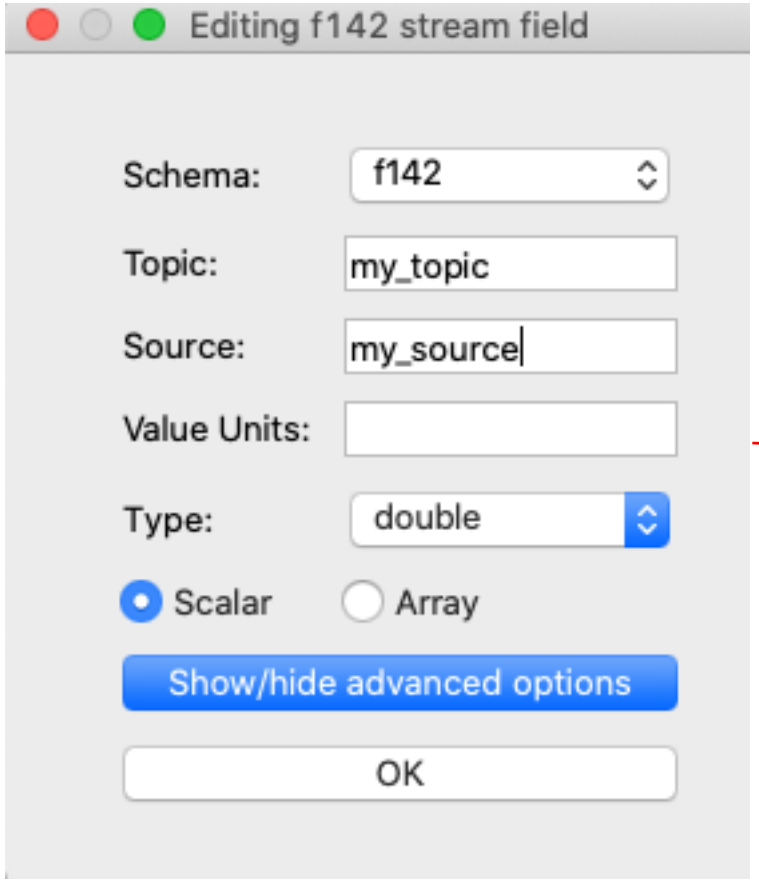
Links:

These are “shortcuts” to other parts of the NeXus file.

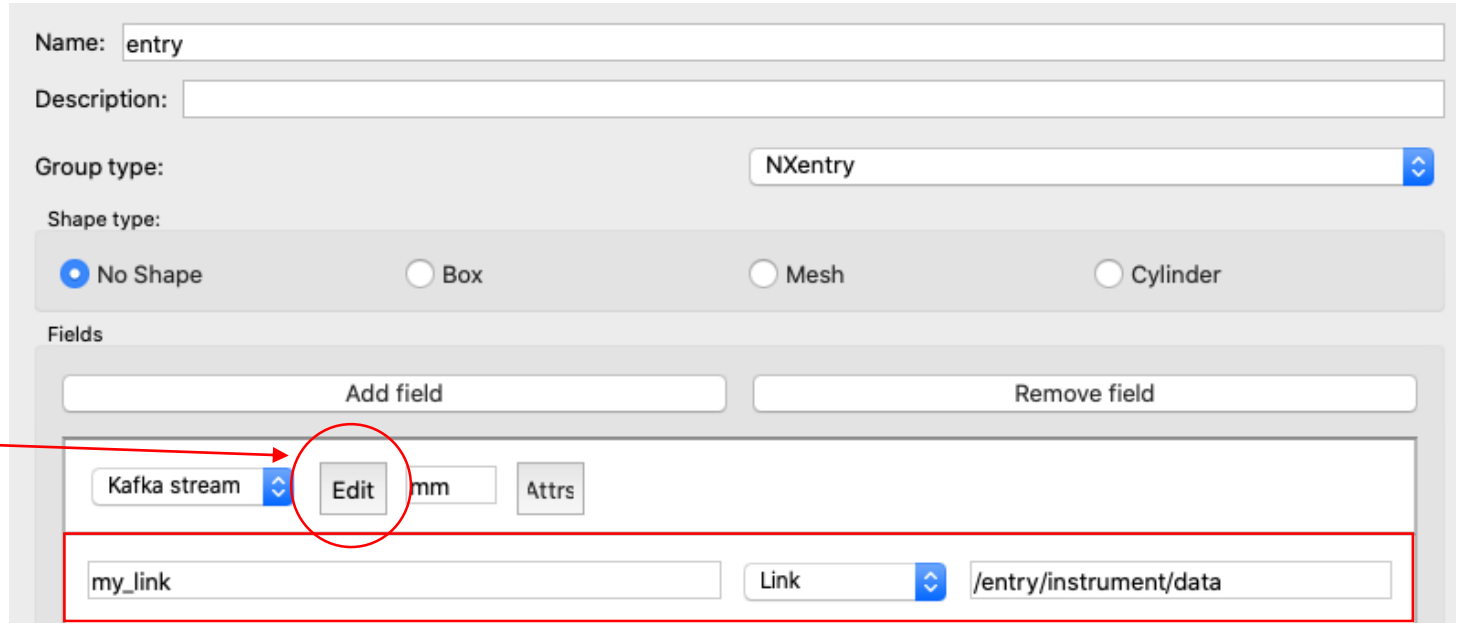
File writer modules

Different fields to add in a group

Configuring a f142 stream

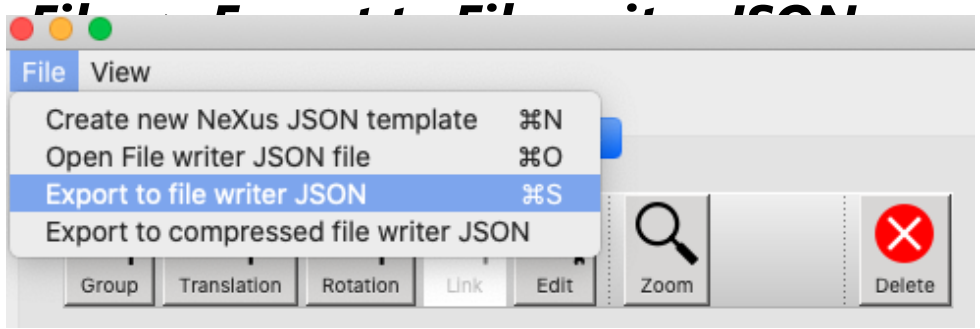


Configuring a link



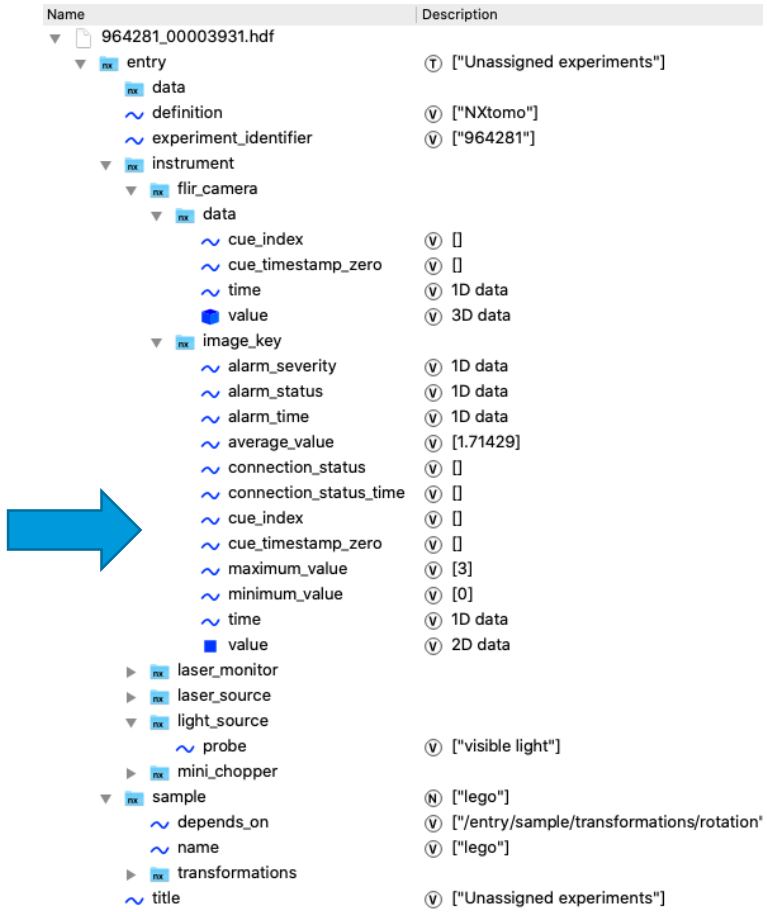
Saving the NeXus template file is easy

1. Once the desired NeXus structure has been produced, it can export it to a JSON file.



2. The saved JSON file can be provided to the file writer to write a NeXus data file.

3. You can also load an existing and valid JSON file and edit it by:
File –> Open File writer JSON file



Summary

NeXus Constructor

1. As can be seen, the graphical user interface is simple.
2. The 3D component viewer enables the user to verify component geometries and locations while building the NeXus structure.
3. Complicated geometries can be imported as OFF files.
4. It is easy to save, load and share templates between people.
5. NeXus class documentation in the UI, field name suggestions and error checking helps the user to avoid the most obvious mistakes.

NeXus Constructor can be found on:

https://github.com/ess-dmsc/nexus-constructor

Thank you for your attention!

Questions?

Kenan Murić, Data Acquisition Scientist, ESS ERIC
Contact: kenan.muric@ess.eu