10th Silx Code Camp
June 17, 2019

This talk

- Introduction
  - Novelties (version 0.11.0)
- Status of silx (version 0.10.1)
- Goals of the code camp
  - For users
  - For core developers
- Hands on!

silx.opencl: Convolution

- A new module for Convolution (with OpenCL CPU-GPU acceleration):
  silx.opencl.convolution

- Designed to compute convolutions in the real space

- Many possible use cases
  - 1D, batched or separable 1D kernel on 2D/3D data
  - 2D, batched or separable 2D on 3D data
  - 3D
- Multiple padding modes, compatible with scipy:
  reflect, nearest, wrap, constant

- Input and/or output can be device arrays

silx.opencl: Convolution

Separable 1D convolution on 2D data

```
from silx.opencl.convolution import Convolution, gaussian_kernel
g = gaussian_kernel(1.0)
C = Convolution((512, 511), g, mode="wrap")
print(C.use_case_desc)
res = C(mydata)
```

Separable 2D convolution on 3D data

```
C = Convolution((512, 511, 513), kern2D, axes=(0, 1))
print(C.use_case_desc)
```

Batched 2D convolution on 3D data

```
C = Convolution((512, 511, 513), kern2D, axes=(0, ))
print(C.use_case_desc)
```

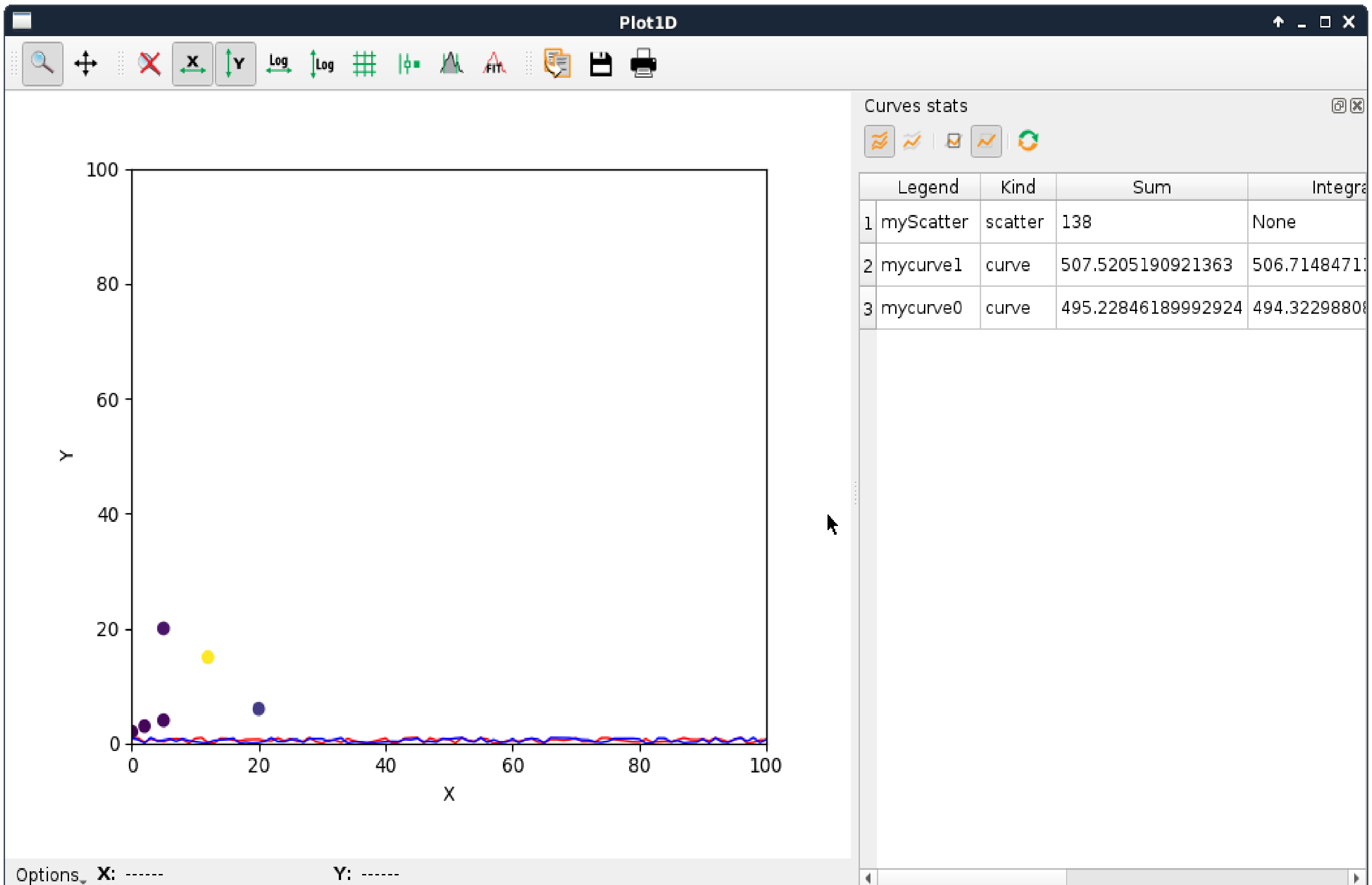
silx.opencl: Convolution

Separable convolution on 3D data

```python
from silx.opencl.convolution import Convolution, gaussian_kernel
from scipy.misc import ascent
img = ascent().astype("f")
data = np.tile(img[:, :511], (32, 1, 1))
g = gaussian_kernel(1.0)
C = Convolution(data.shape, g)
res = C(data)
# Compare with scipy
from scipy.ndimage import convolve
g3 = np.multiply.outer(np.outer(g, g), g)
res_scipy = convolve(data, g3)
print(np.max(np.abs(res - res_scipy)))
```

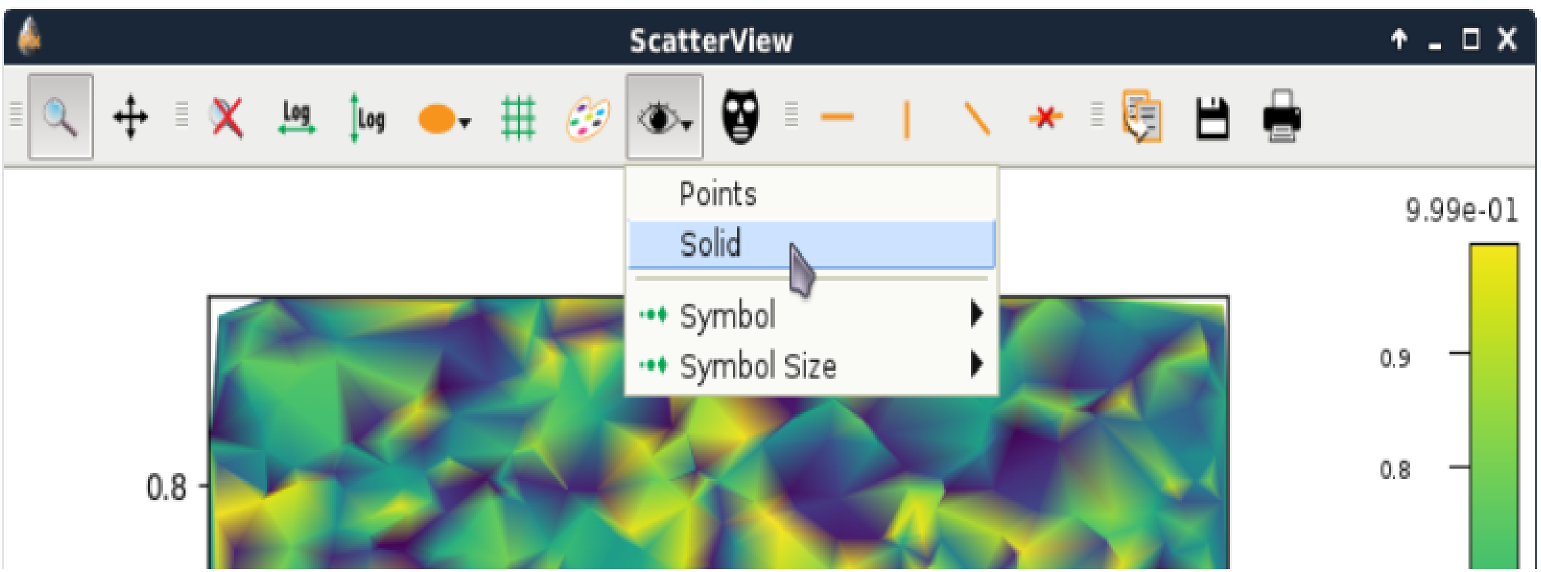
silx.gui.plot: StatsWidget

- Now managing Item visibility
- Added an update mode in `StatsWidget`
  - Modes can be manual or automatic (previous behavior)



silx.gui.plot: Scatter Plot Visualization

- Display scatter plots as a surface using Delaunay triangulation: `Scatter.get|setVisualization`
- `ScatterView`: Add a tool button to change scatter visualisation mode



silx.gui.plot

- `matplotlib` >=3.1 compatibility
- Change the handling of default Y axis limits
- Add option to provide an alpha image to a plotted image

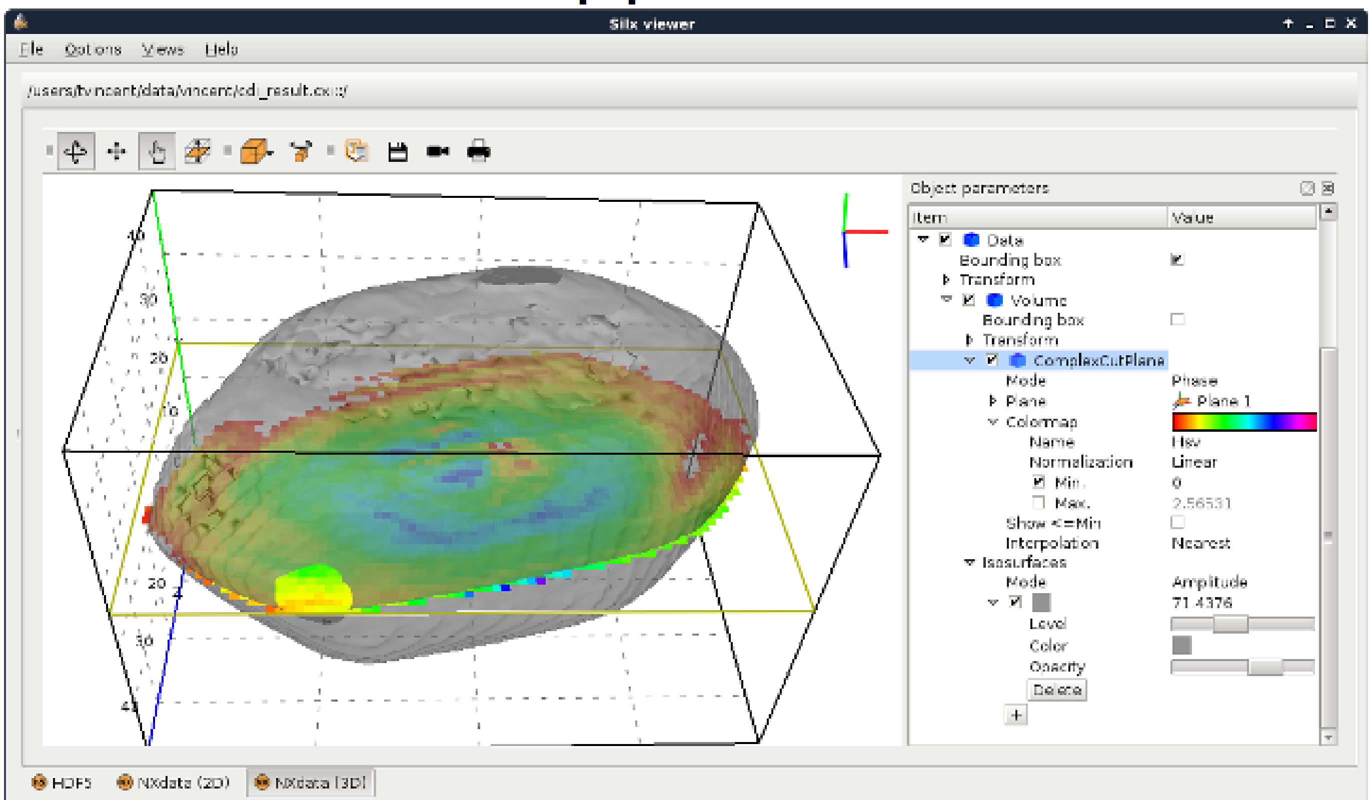
silx.gui.plot: OpenGL Backend

- Fix leak when creating/deleting widgets in a loop.
- Fix issue with large images

silx.gui.plot3d: Complex Data

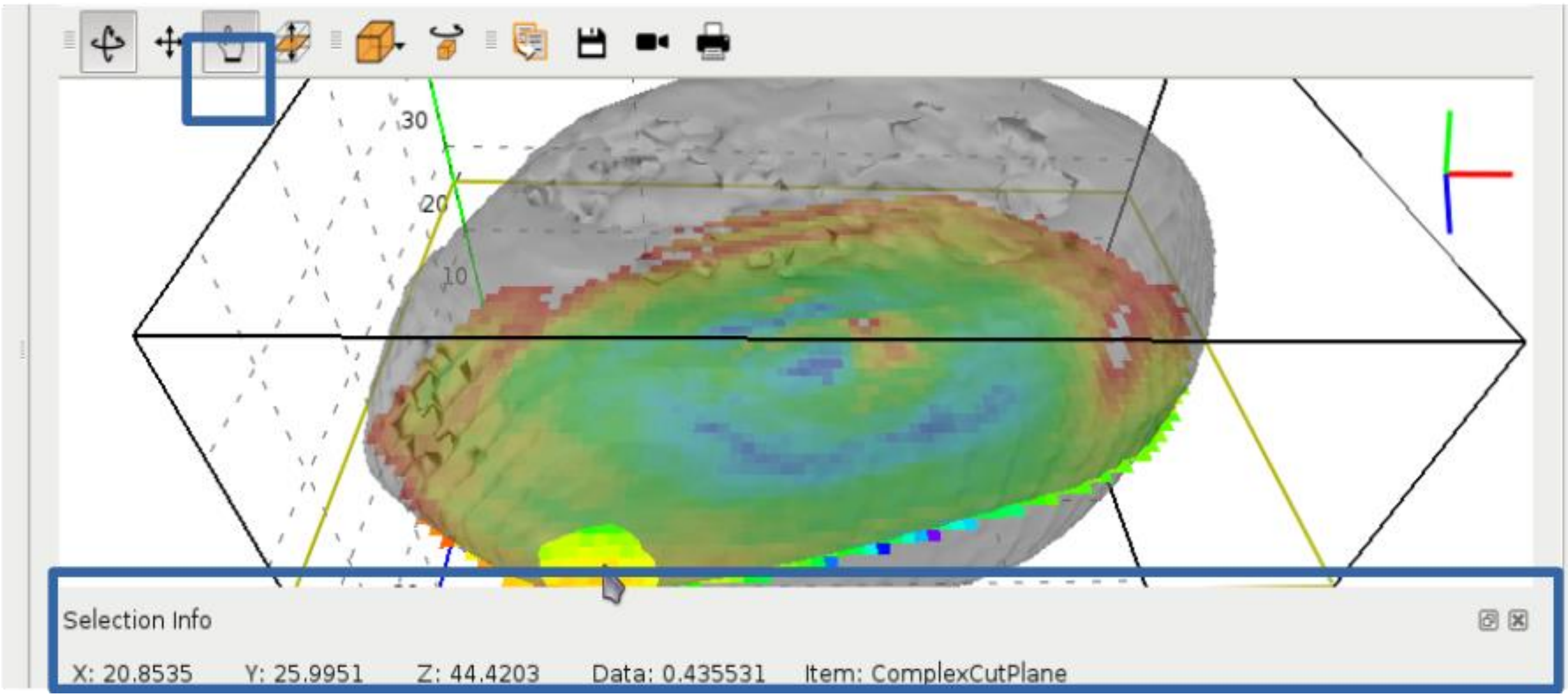
3D complex field visualization: `Complex3DField`

- Available from `silx view`.
- Still miss a colormapped isosurface



silx.gui.plot3d: Position and Picking

Add a `PositionInfoWidget` and a tool button to toggle the picking mode to `SceneWindow`.



silx.gui.plot3d: Other Changes

- 3D items parameter tree minor updates and optimizations
- !!! API break: 2D scatter items `supportedVisualizations` and `getVisualization` methods were returning `str` and now returns the corresponding `Enum` for consistency with `silx.gui.plot` API.

Miscellaneous

- Python 3.8 compatibility
- Fix issues related to Debian packaging
- Numpy 1.12.1 required (i.e. Debian9 or Debian8 backports)

Bob the builder

Snapshots of `silx` are available as wheels for Windows and Linux (and Debian 9 packages) here: https://silx.gitlab-pages.esrf.fr/bob/silx/

Source of the machinery: https://gitlab.esrf.fr/silx/bob

"External" contributions

`@schooft`: For the plot profile tool window, use the same backend as the initial plot.

`@dnaudet`: Improve documentation

`@PiRK`:

- add a `is|setVisible` method to plot ROI objects
- ignore exotic mouse buttons in plot OpenGL backend

Good bye to `@vallsv`

Contributions:

- Allows to write HDF5 dataset already encoded on our side.
- Fixes, typos, sample code update

This talk

- Introduction
  - Novelties (version 0.9.0)
- Status of silx (version 0.10.1)
- Goals of the code camp
  - For users
  - For core developers
- Hands on!

Structure of silx

- gui: Graphical User Interface widgets
  - Plot, image display, masks, HDF5 tree view, fitting
- image: Image processing tools
  - Image interpolation, registration and drawing primitives
- io: Input / Ouput
  - Support for SPEC, HDF5 and image formats
- math:
  - least squares fit with constraints, isosurface calculations, histograms, fft,…
- opencl: Optimize the use of GPU (FBP, registration, median filter, fft, …)
- third-party: External utilities
- utils: Internal utilities
- sx: Convenience module for interactive use

Installation

- Easier installation of all dependencies:

```
pip install silx[full]
```

- Windows standalone application

silx.resources

Container of icons, opencl programs, …

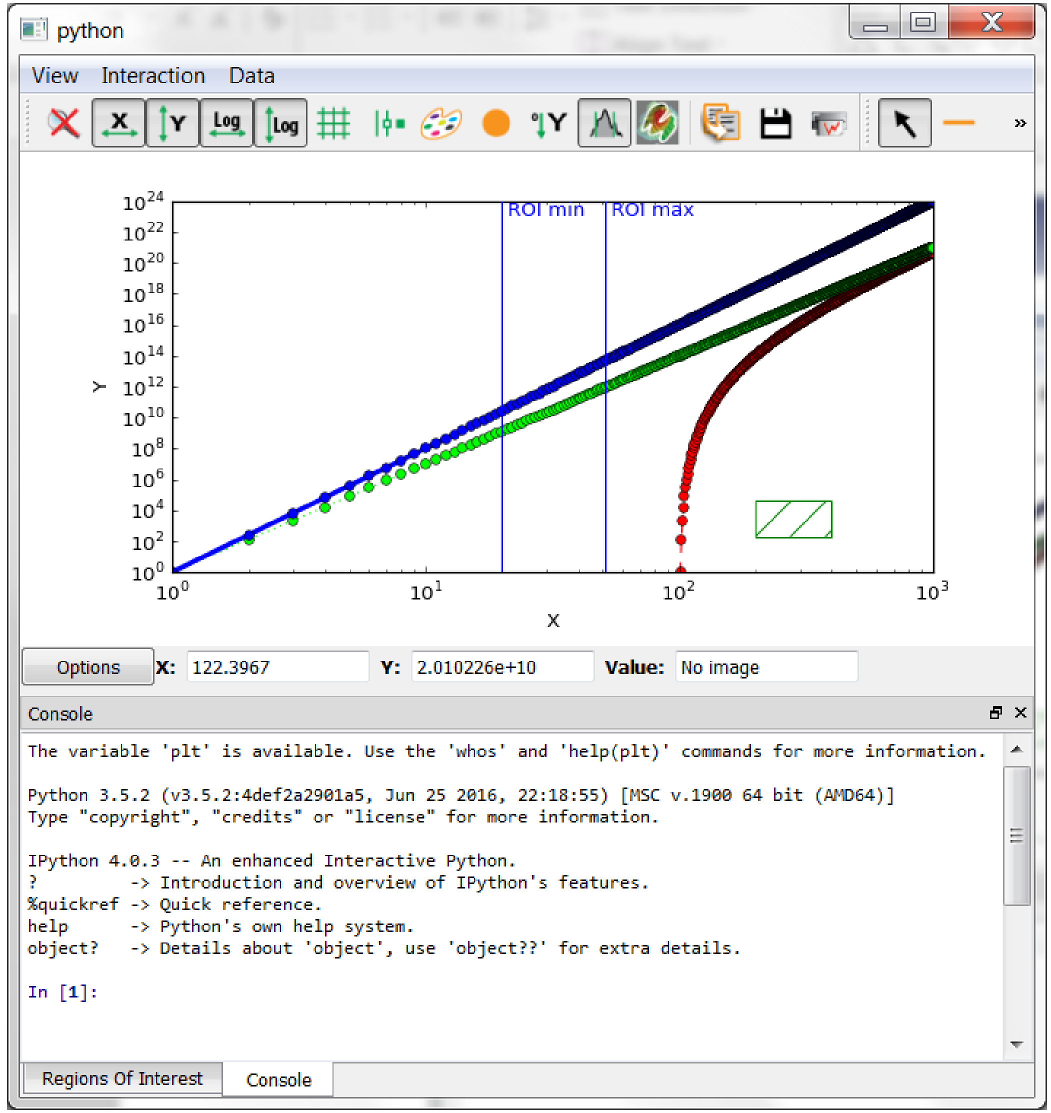
Provisions for simplifying handling of frozen binaries

A project can use silx as resource provider

```
import silx.resources

PYFAI_RESOURCE_DIR = None  # It has to be set for Debian package

silx.resources.register_resource_directory(
    "pyfai",
    pyFAI.resources,
    forced_path=PYFAI_RESOURCE_DIR)

filename = silx.resources.resource_filename("pyfai:calibrant/LaB6.C")

import silx.opencl.utils
filename = silx.opencl.utils.get_cl_file("pyfai:opencl/integrate")

import silx.gui.icons
icon = silx.gui.icons.getIcons("pyfai:icons/pyfai")
```

silx.gui: base widgets for scientific applications

- Browsing file contents
  - Single widget for HDF5, SPEC, Images
- Plotting curves
  - with ROI, fitting
- Display of images
  - with masks, profiles
- Interactive console



Plot: Object API

When getting a curve or an image from a Plot widget in silx, it used to return a list describing this item.

- Since v0.5.0 it returns an object:
  - Add support for updating items in the Plot:
    curve, image, markers...
  - Mostly backward-compatible with previous API

- Documentation:

http://www.silx.org/doc/silx/dev/modules/gui/plot/items.html

Plot: Object and Functional APIs

- Example: Getting image information:

```
from silx import sx
w = sx.imshow(img)
```

- Object API:

```
image = w.getActiveImage()
data = image.getData(copy=True)
scale = image.getScale()
```

- Legacy API:

```
image = w.getActiveImage()
data = image[0]
scale = image[4]['scale']
```

Plot: Object and Functional APIs

Example: Updating an image:

```
from silx import sx
w = sx.imshow(img)
```

- Object API:

```
image = w.getActiveImage()
image.setScale(2., 2.)
```

- Legacy API:

```
data, legend, info, pixmap, params = w.getActiveImage()
w.addImage(data,
           legend=legend,
           info=info,
           pixmap=pixmap,
           scale=(2., 2.))
```

silx.gui.plot API

- Signals on *PlotWidget* items (i.e. curves, images, markers,…) notifying updates: *sigItemChanged*

- Get all items in the plot: *getItems*

- Follow plot content update through signals:
  sigItemAdded and *sigItemAboutToBeRemoved*

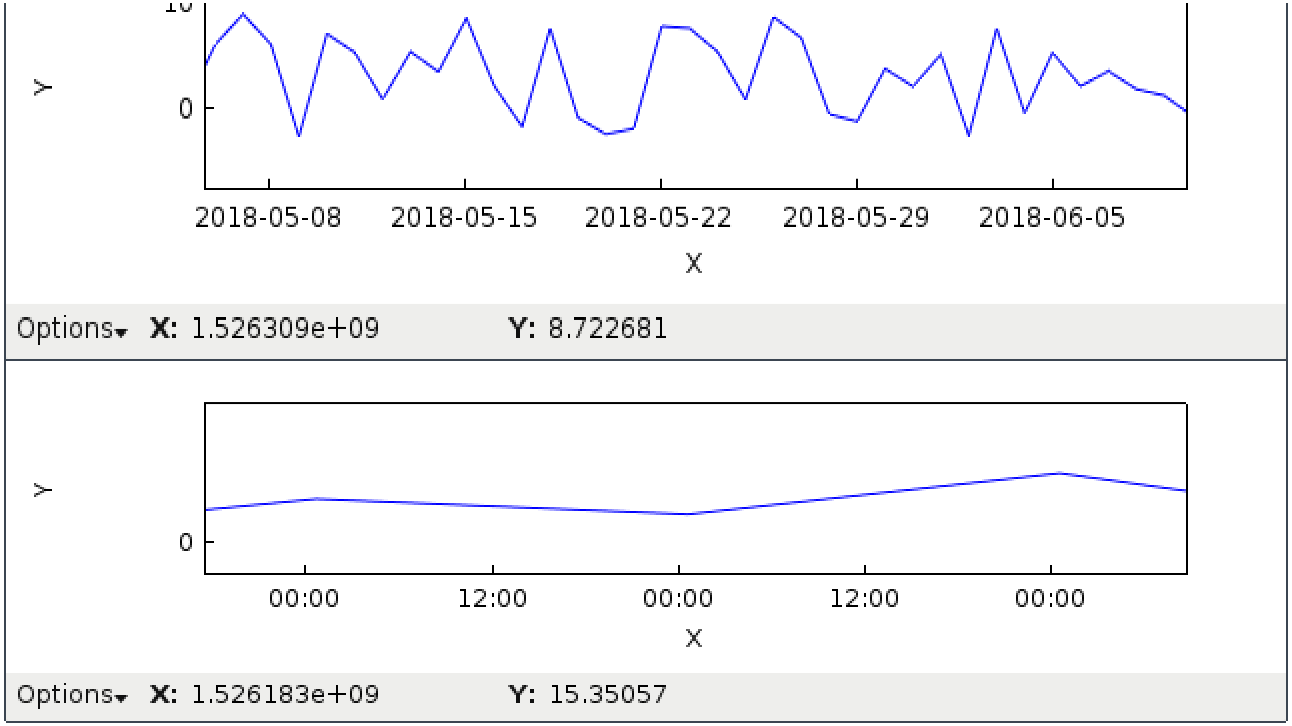
silx.gui: Plot 1D

- Visualize 1D data
- Apply ROIs on them
- Control the plot via an interactive console
- Fitting capabilities
- Object oriented API

silx.gui.plot Time series

- X axis labels displayed as dates or times depending on scale
- Thanks to Pepijn Kenter (SRON: Netherlands Institute for Space Research)

Doc: http://www.silx.org/doc/silx/dev/modules/gui/plot/items.html#silx.gui.plot.items.Axis.setTickMode



PlotWidget axis

- Provide a plot axis API

  axes =plot.getXAxis(), plot.getYAxis()

  - Provides getters, setters
  - Signals on limits, scale, label, direction

- Constraints on axes

  xaxis.setLimitsConstraints(minPos, maxPos)

  xaxis.setRangeConstraints(minRange, maxRange)

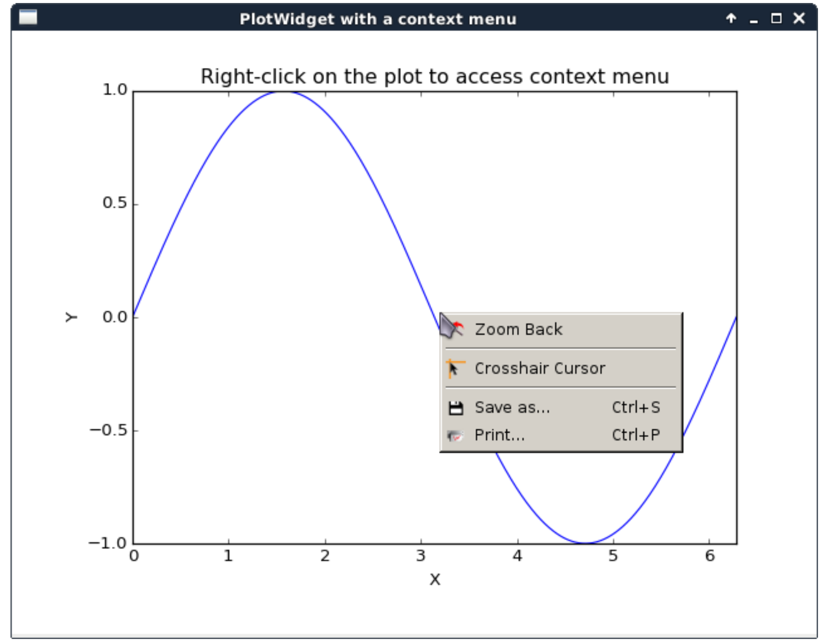
  - A demo is available at *examples/plotLimits.py*

- Helper to synchronize axes

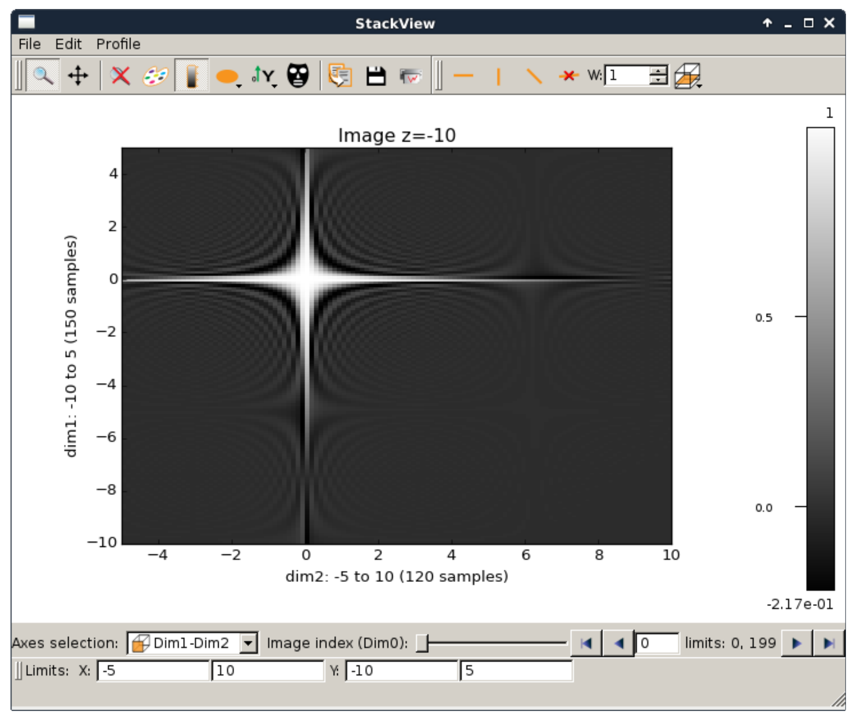
  ```
  from silx.gui.plot.utils.axis import SyncAxes
  sync = SyncAxes([plot1.getXAxis(),
                   plot2.getXAxis(),
                   plot3.getXAxis()])
  ```

  - A demo is available at *examples/syncaxis.py*

- PlotWidget: Add support for context menu:
  plotContextMenu.py

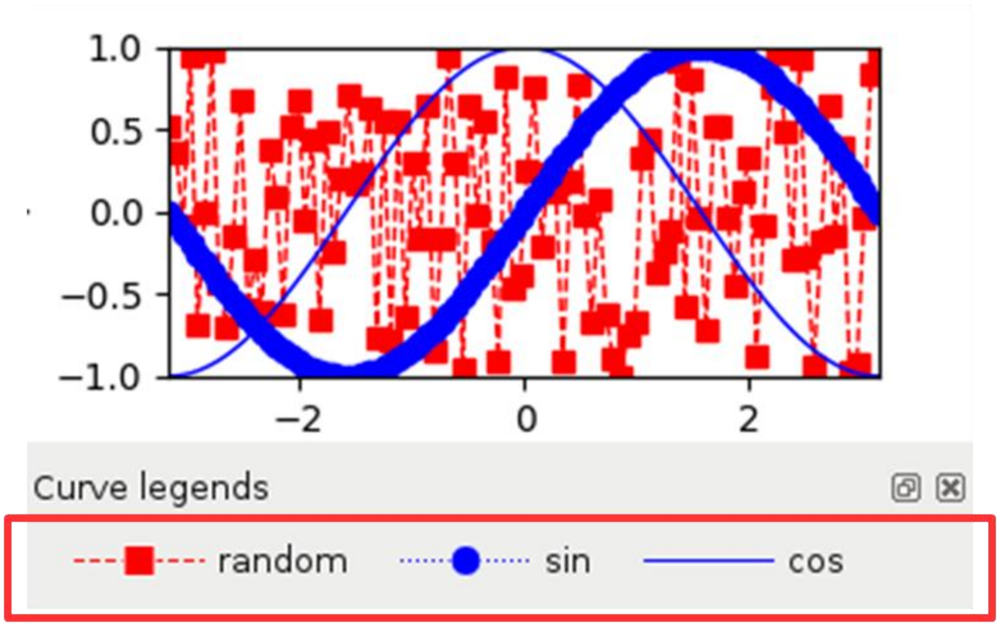


- PlotWindow, Plot2D
  - Add colorbar



silx.gui.plot: CurveLegendWidget

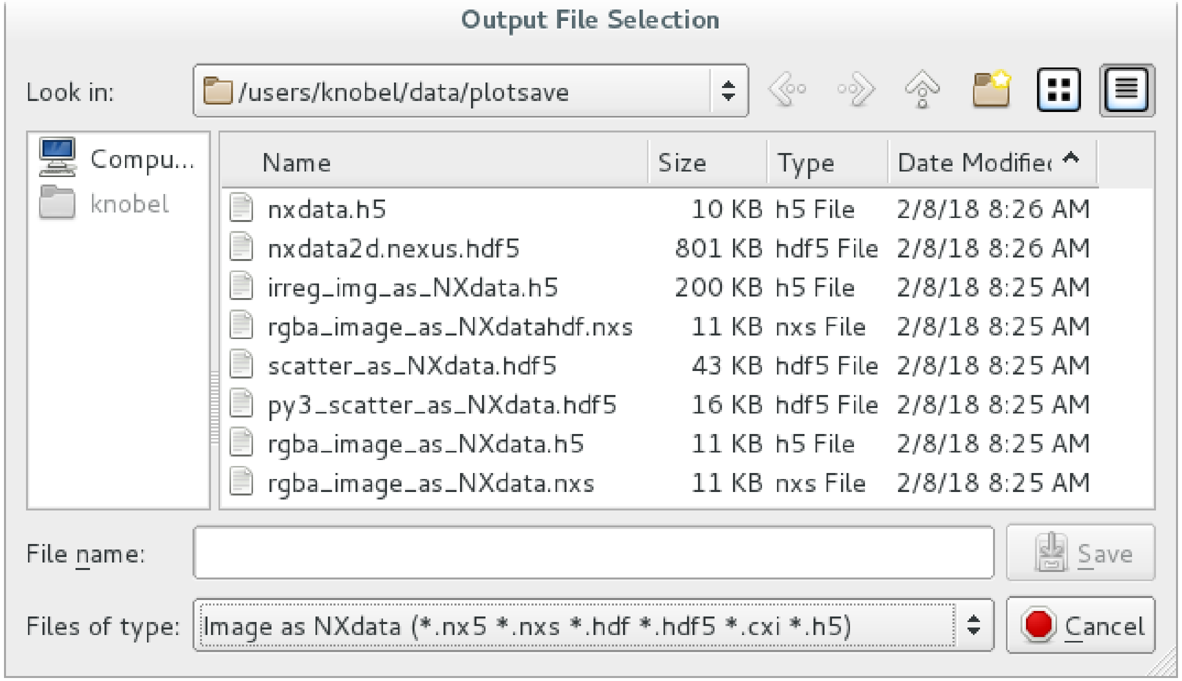
`silx.gui.plot.CurveLegendWidget:`



- Display legends of curves in a plot
- Compact alternative to `LegendSelector`.

Plot SaveAction : add save as NXdata

- Save active curve, active scatter or active image to *NXdata*



- Can save some parts of plot state (title, axis labels, active data...) but not all (no curve style, colormap info, additional data items...)
- Future improvements: add a dialog to specify output group in an existing HDF5 file

silx.gui: Plot 2D

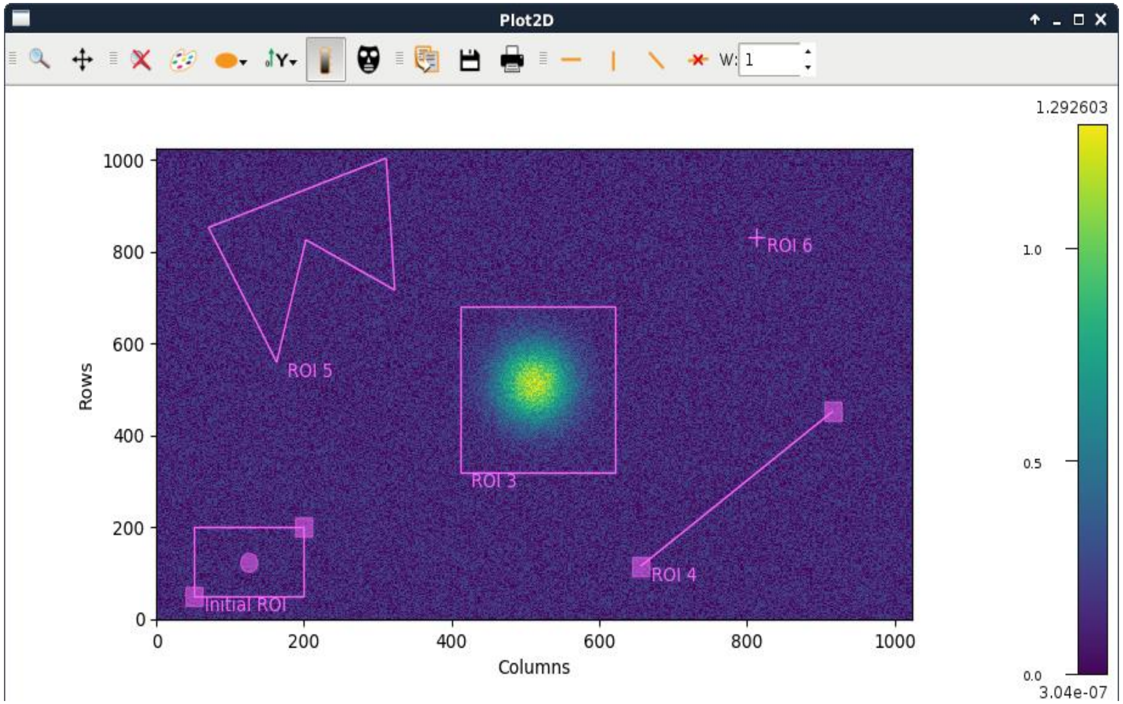
- Visualize 2D data (Images and Stacks of Images)
    - Support Median Filters, Profiles and Masks on them
- Visualize 3D data as scatter plots
    - Support Masks on them
- Apply different colormaps
- Plot an image with associated histograms
- Visualize 3D scalar fields (Isosurfaces)

Interactive Regions of Interest

- `silx.gui.plot.tools.roi:`
  - Regions of interest on a plot with different shapes
  - Editable interactively

Doc: http://www.silx.org/doc/silx/dev/modules/gui/plot/tools.html#module-silx.gui.plot.tools.roi

Sample code: plotInteractiveImageROI.py



Plot Widget Toolbars

- **Idea: Make plot widgets more modular:**
  - Allow to reuse `QAction` and `QToolBar`:

```
from silx.gui import qt
from silx.gui.plot import PlotWidget, tools
[...]
window = qt.QMainWindow()       # Create a window
plot = PlotWidget(window)       # Create a plot
window.setCentralWidget(plot)   # Place plot in window

# Add plot zoom/pan toolbar to the window
window.addToolBar(tools.InteractiveModeToolBar(parent=window, plot=plot))

# Add copy/save/print toolbar to the window
window.addToolBar(tools.OutputToolBar(parent=window, plot=plot))
[...]
window.show()
```

Colormap Object (silx.gui.plot.Colormap)

Colormaps are now defined as a ***Colormap*** object instead of a dictionary.

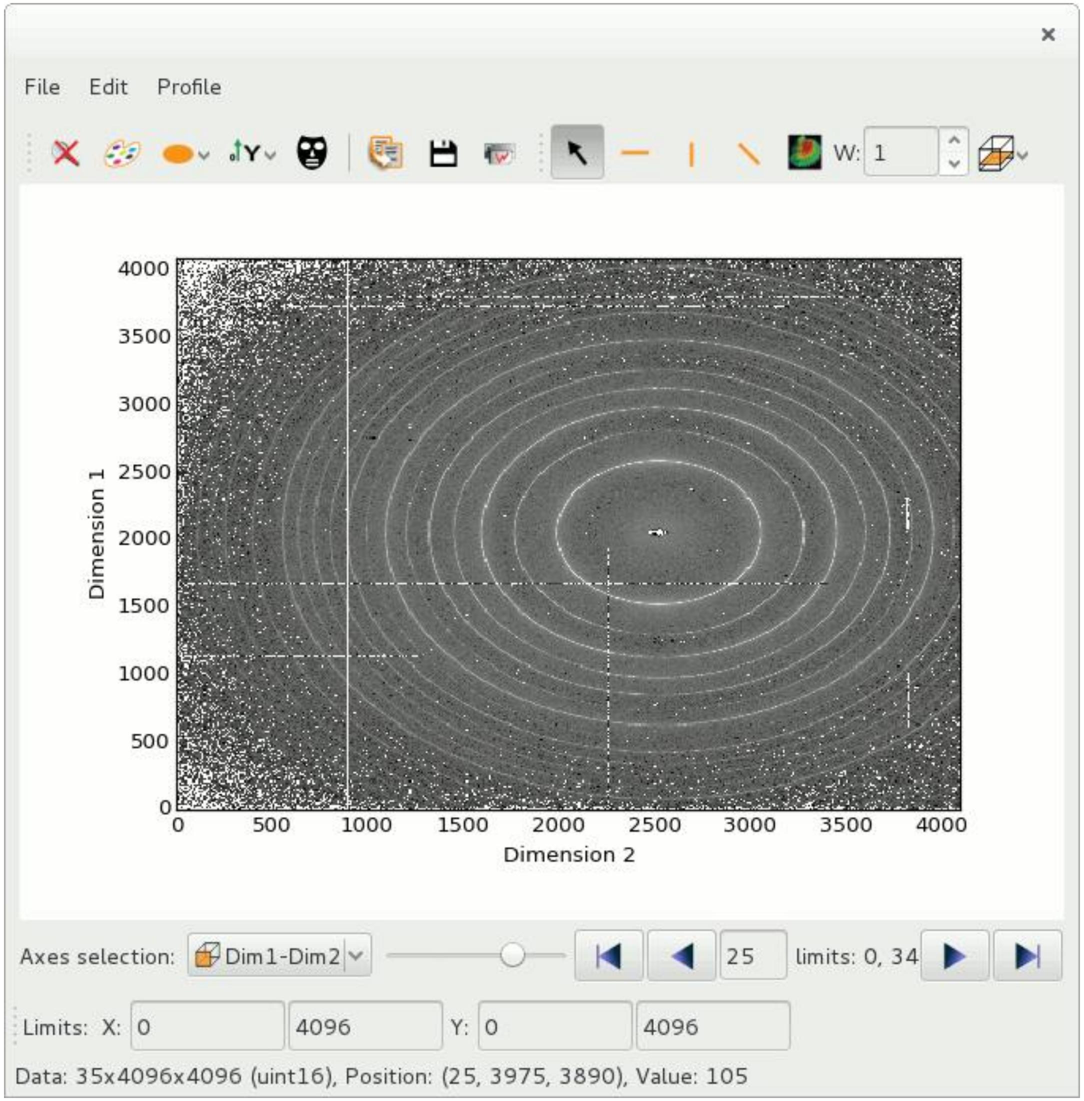
This allow modifications on colormaps objects to be managed by other classes such as ***PlotWidget*** or ***ColorBar*** (using Qt.Signal).

```
from silx.gui.plot.Colormap import Colormap

colormap = Colormap(name='temperature',
                    normalization=Colormap.LOGARITHM,
                    vmin=None,
                    vmax=None)
```

API with colormaps as a dictionary is kept but deprecated.

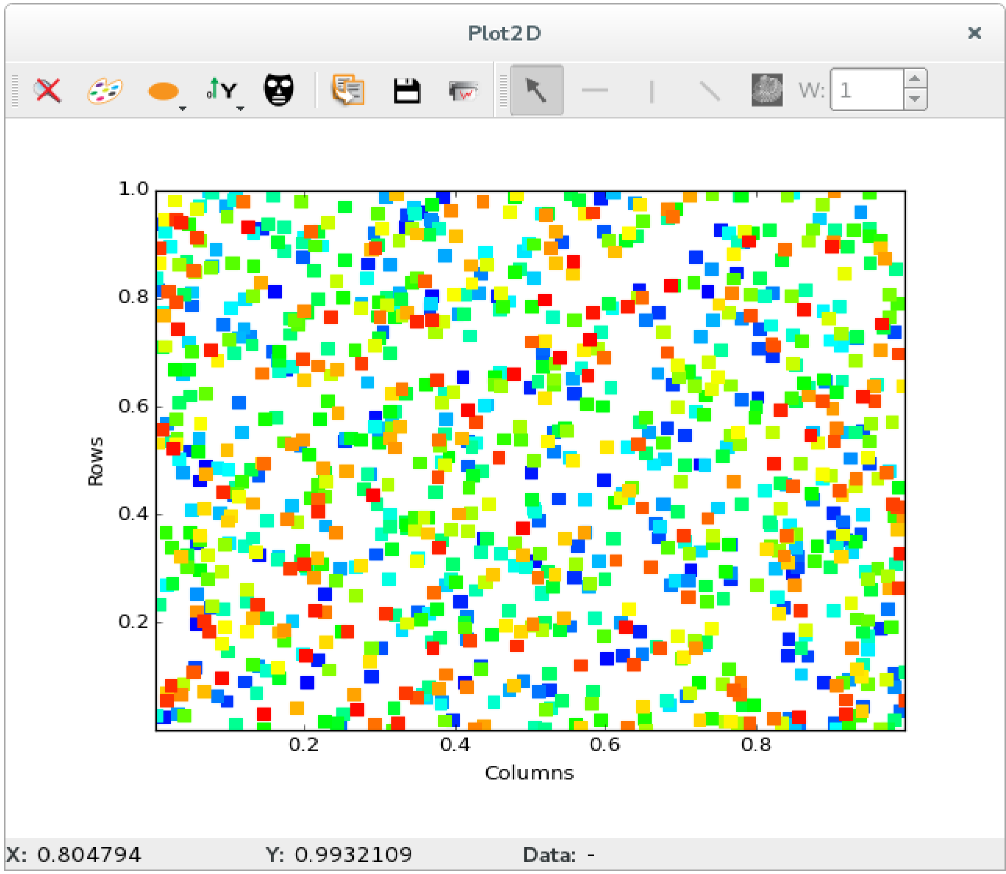
silx.gui.plot.StackView

- Viewing 3D arrays, 3D datasets or list of 2D arrays as a stack of images.
- Axes selection
- Profile tool to extract a 2D slice from the 3D stack
- Lazy loading for datasets (except when doing diagonal 3D profile)



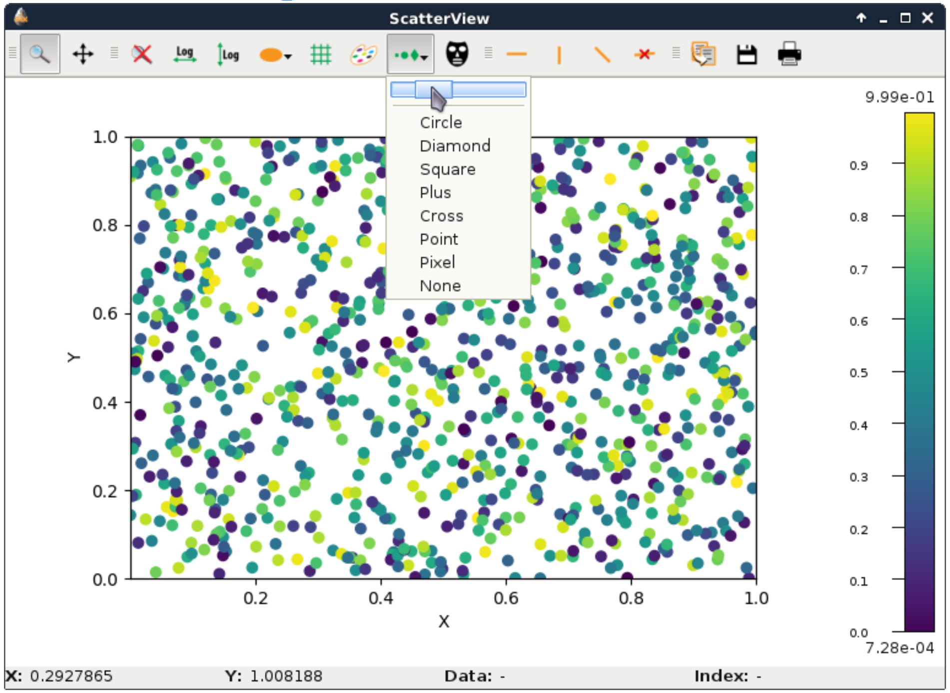
silx.gui.plot Scatter Objects

```python
import numpy
import sys
from silx.gui import qt
from silx.gui.plot import Plot2D

app = qt.QApplication([])
win = Plot2D()

win.addScatter(x=numpy.random.random(1000),
               y=numpy.random.random(1000),
               value=numpy.arange(1000),
               legend="my scatter")

sc = win.getScatter("my scatter")
sc.setSymbol("s")                    # square
sc.setSymbolSize(50)
sc.setColormap({'name': 'temperature',
                'normalization': 'linear',
                'autoscale': True,
                'vmin': 0.0, 'vmax': 1,})
win.resetZoom()
win.show()
sys.exit(app.exec_())
```



ScatterView: Features

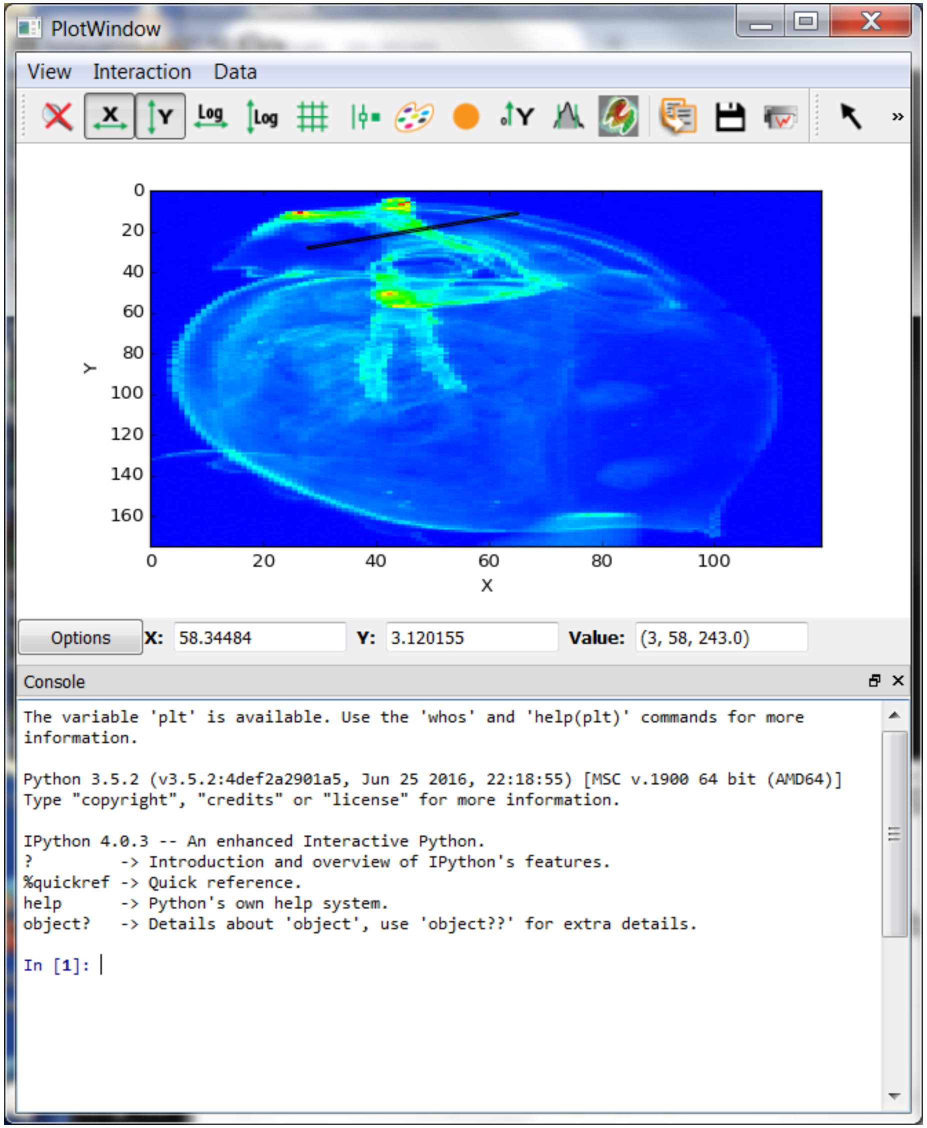
- Standard plot control, colorbar
- Points size/shape control
- Mask
- Profile

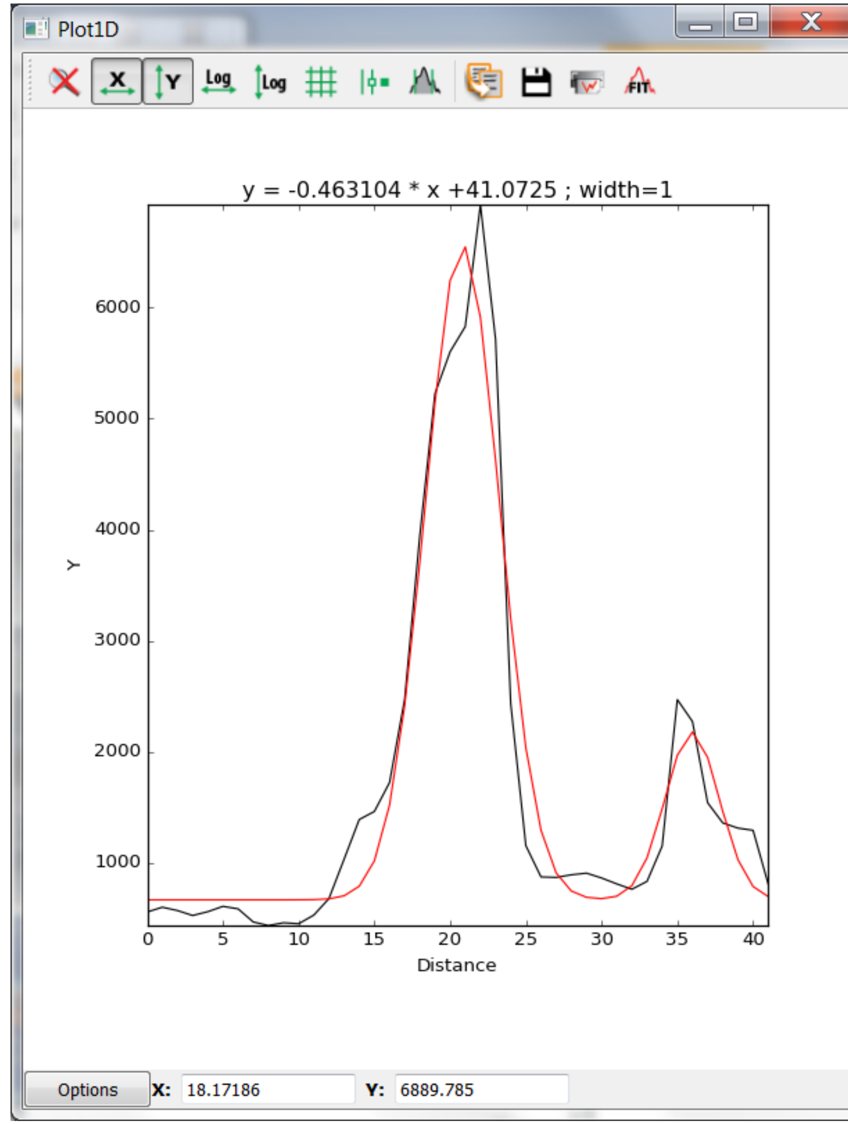



```
from silx.gui.plot.ScatterView import
ScatterView
```

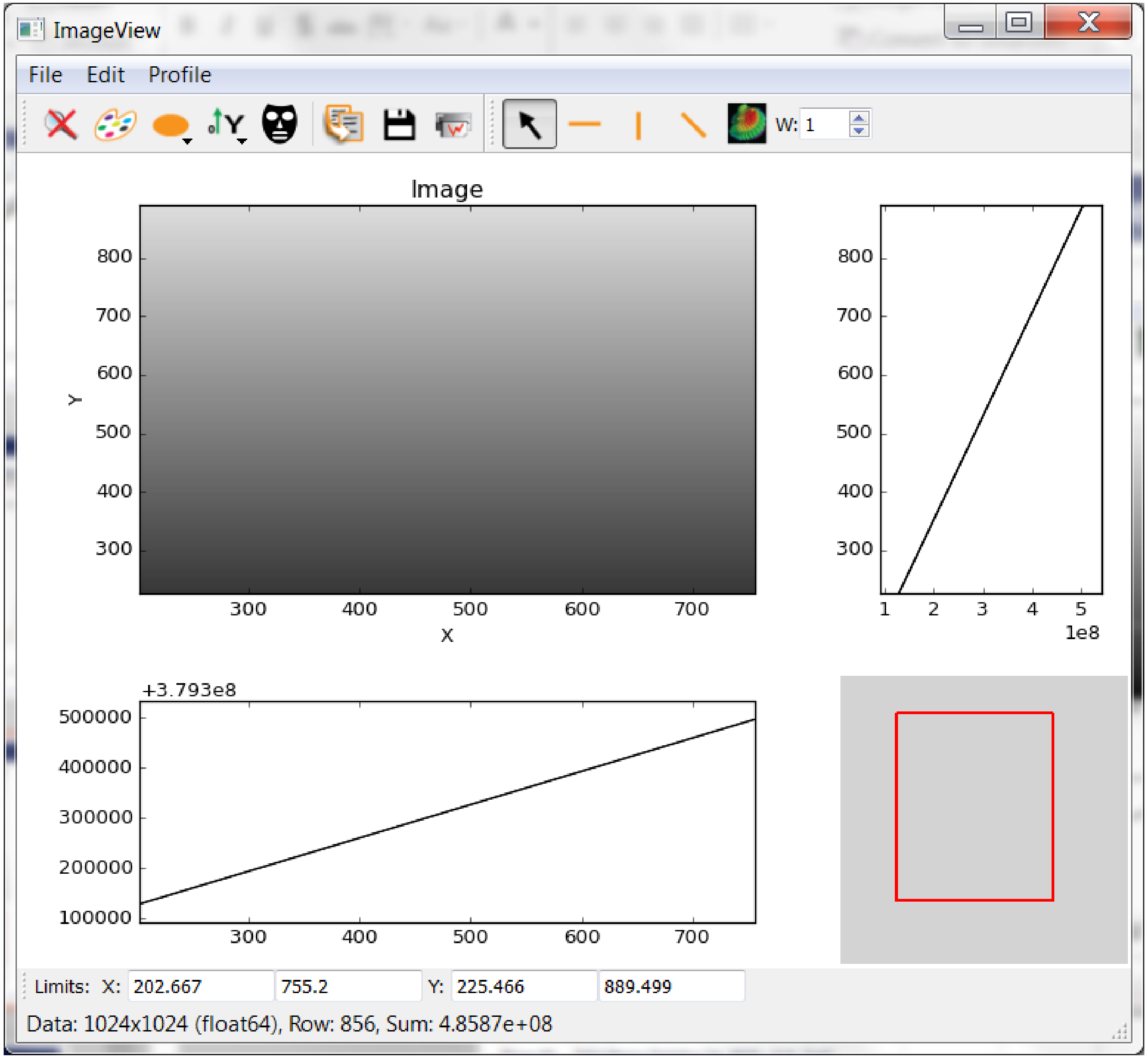
Doc: http://www.silx.org/doc/silx/dev/modules/gui/plot/scatterview.html

Full-featured widgets





Full-featured Widgets

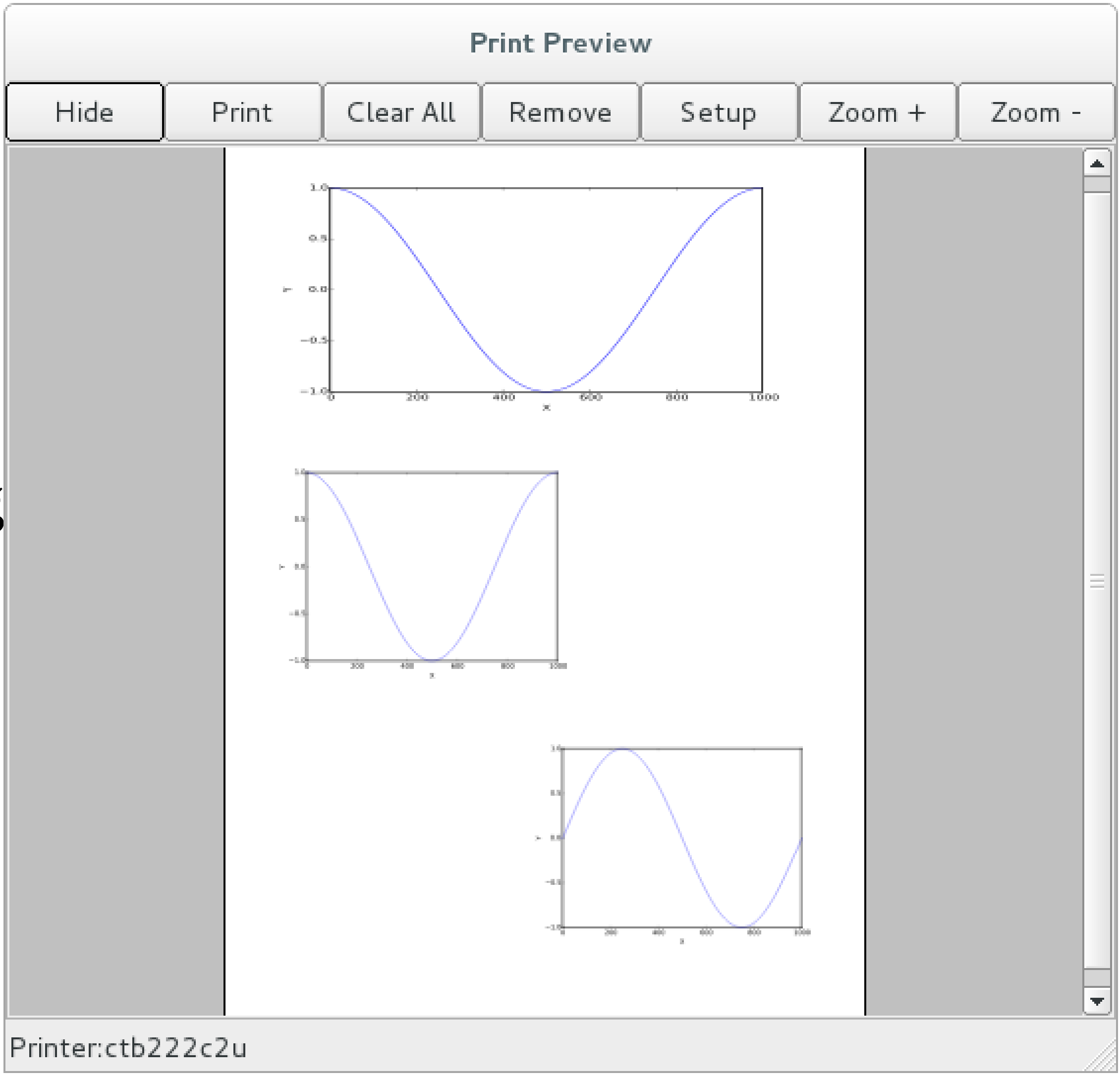


Print Preview

- Print preview dialog (with addImage, addPixmap and addSvgItem methods)

- Tool button for a plot widg *(to send the plot as an SVG item)*

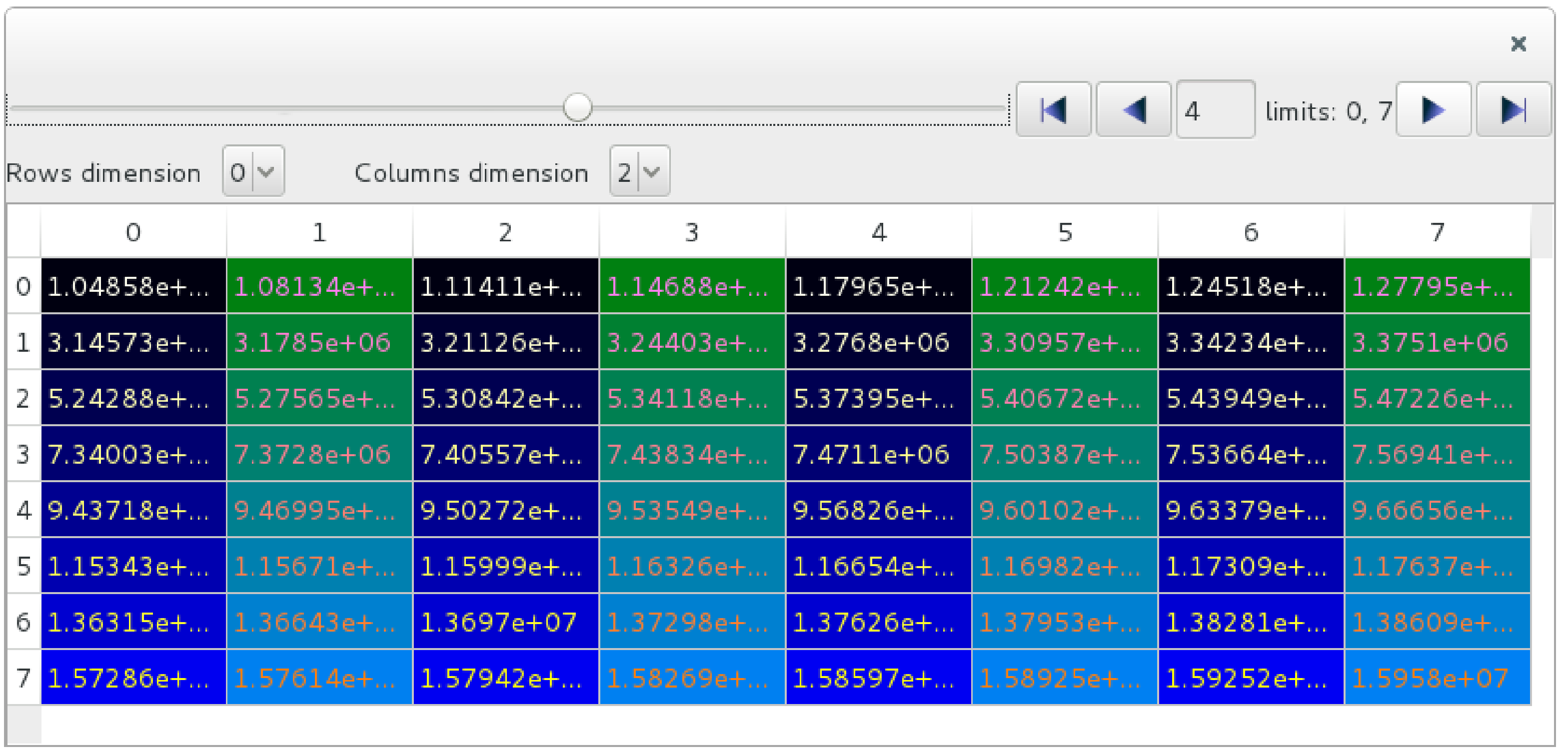
- Items can be dragged and resized. (*Geometry can be configured prior to send the plot)*.



silx.gui.data.ArrayTableWidget

- Display arrays and datasets of any number of dimensions in a TableView

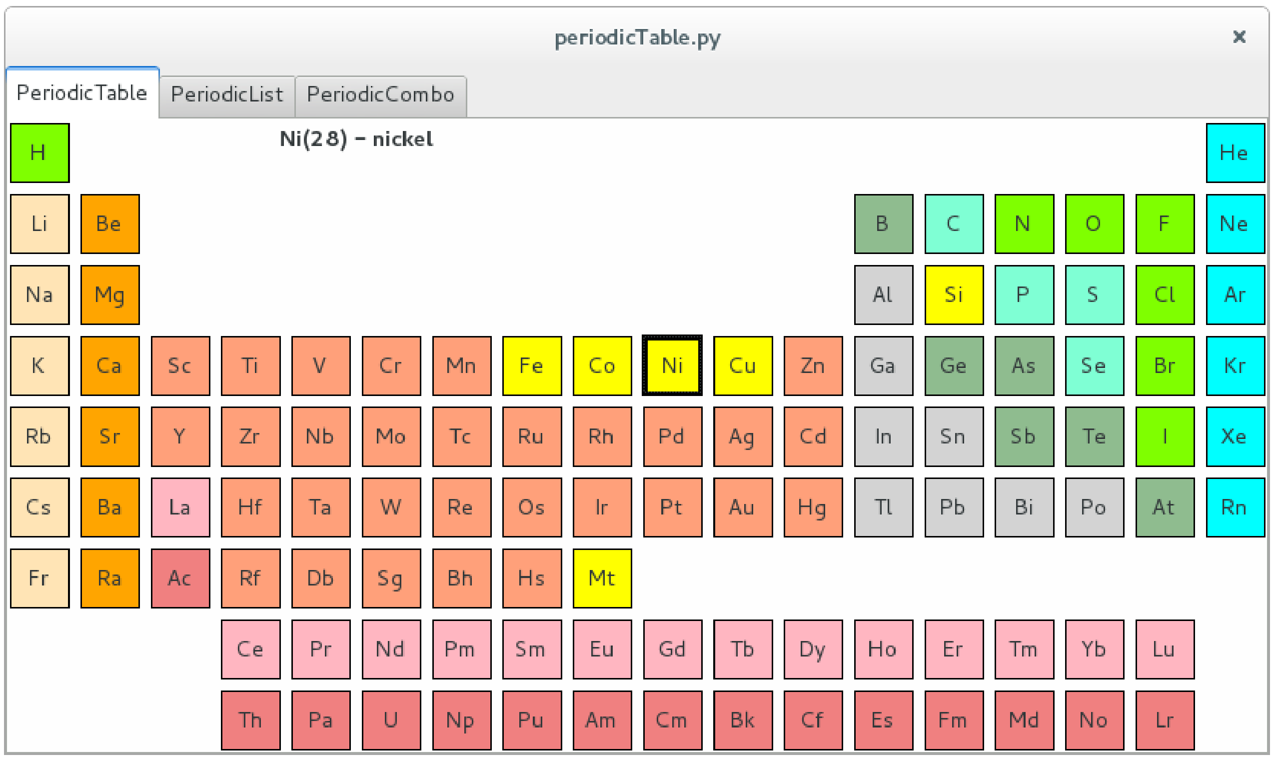
- Lazy loading for datasets: only the currently displayed 2D slice is read from HDF5 file



	0	1	2	3	4	5	6	7
0	1.04858e+...	1.08134e+...	1.11411e+...	1.14688e+...	1.17965e+...	1.21242e+...	1.24518e+...	1.27795e+...
1	3.14573e+...	3.1785e+06	3.21126e+...	3.24403e+...	3.2768e+06	3.30957e+...	3.34234e+...	3.3751e+06
2	5.24288e+...	5.27565e+...	5.30842e+...	5.34118e+...	5.37395e+...	5.40672e+...	5.43949e+...	5.47226e+...
3	7.34003e+...	7.3728e+06	7.40557e+...	7.43834e+...	7.4711e+06	7.50387e+...	7.53664e+...	7.56941e+...
4	9.43718e+...	9.46995e+...	9.50272e+...	9.53549e+...	9.56826e+...	9.60102e+...	9.63379e+...	9.66656e+...
5	1.15343e+...	1.15671e+...	1.15999e+...	1.16326e+...	1.16654e+...	1.16982e+...	1.17309e+...	1.17637e+...
6	1.36315e+...	1.36643e+...	1.3697e+07	1.37298e+...	1.37626e+...	1.37953e+...	1.38281e+...	1.38609e+...
7	1.57286e+...	1.57614e+...	1.57942e+...	1.58269e+...	1.58597e+...	1.58925e+...	1.59252e+...	1.5958e+07

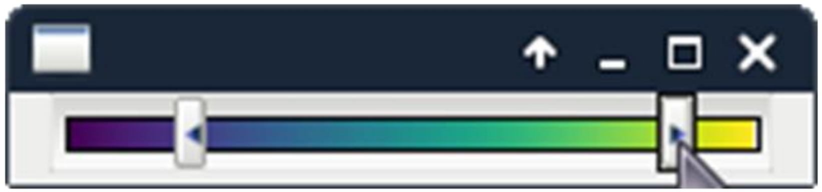
- Periodic table, list (QTreeView) and combo/dropdown list providing minimal data for elements: symbol, name, atomic number, mass

- Selectable elements, signals for element clicked and selection changed events



silx.gui.widgets.RangeSlider

`silx.gui.widgets.RangeSlider:`



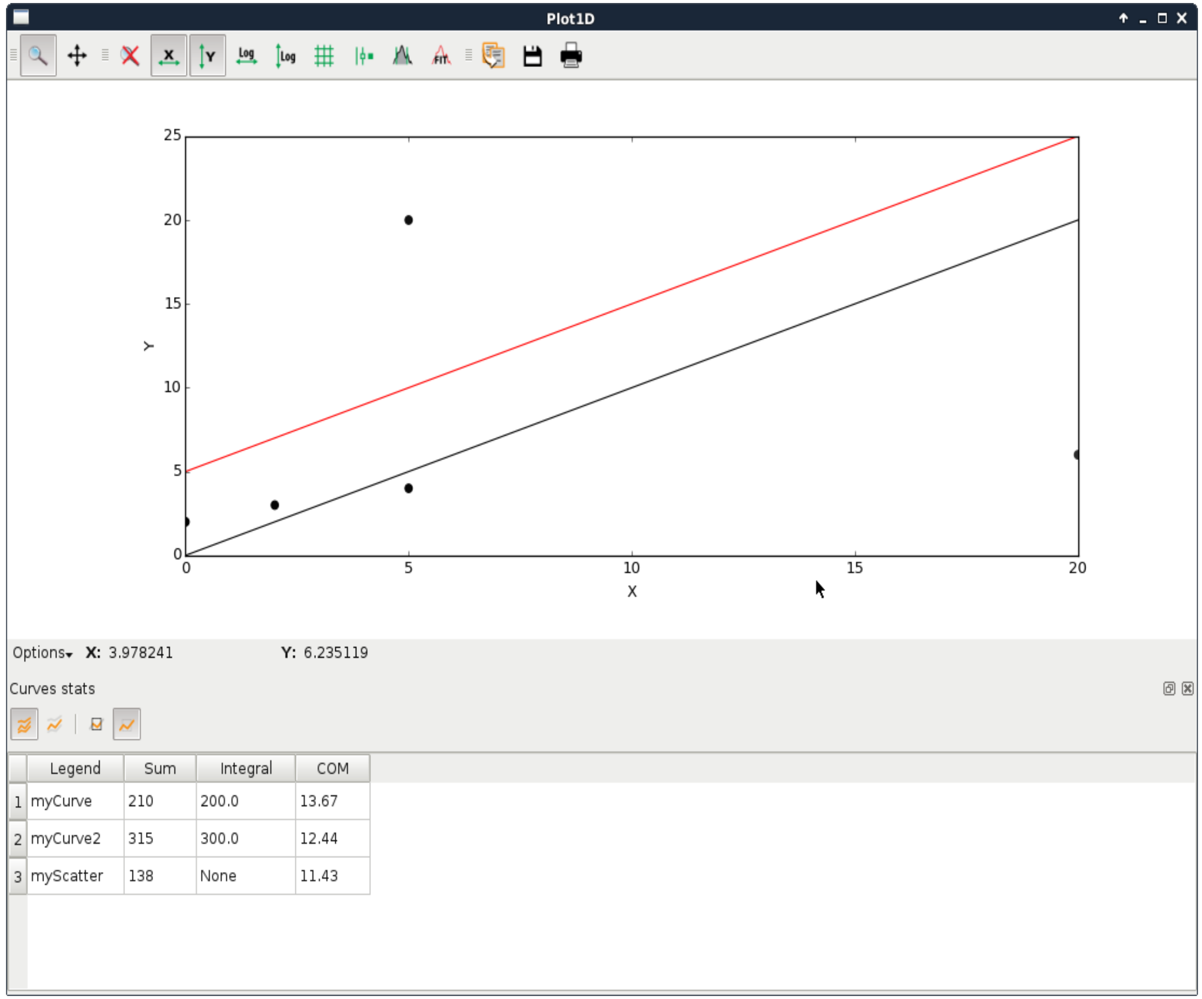
- 2 sliders defining a range with settable color-mapped background.
- Initial version developed by Damien Naudet in XSocs application.

Stats Widget

Deal with:
- curves
- Images
- Scatters
- Histograms

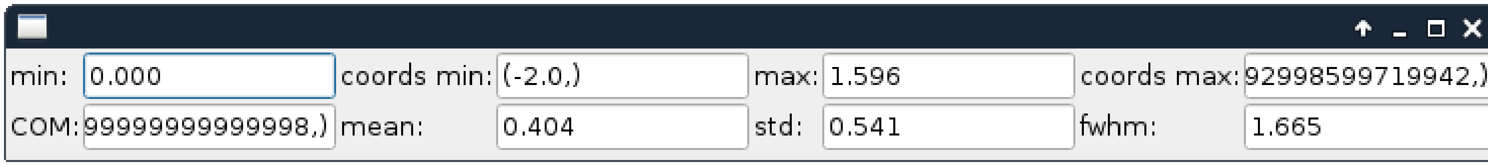
Can calculate on:
- All items or active items
- Full data range or visible one (no interpolation !!!)

Example:
examples/plotStats.py

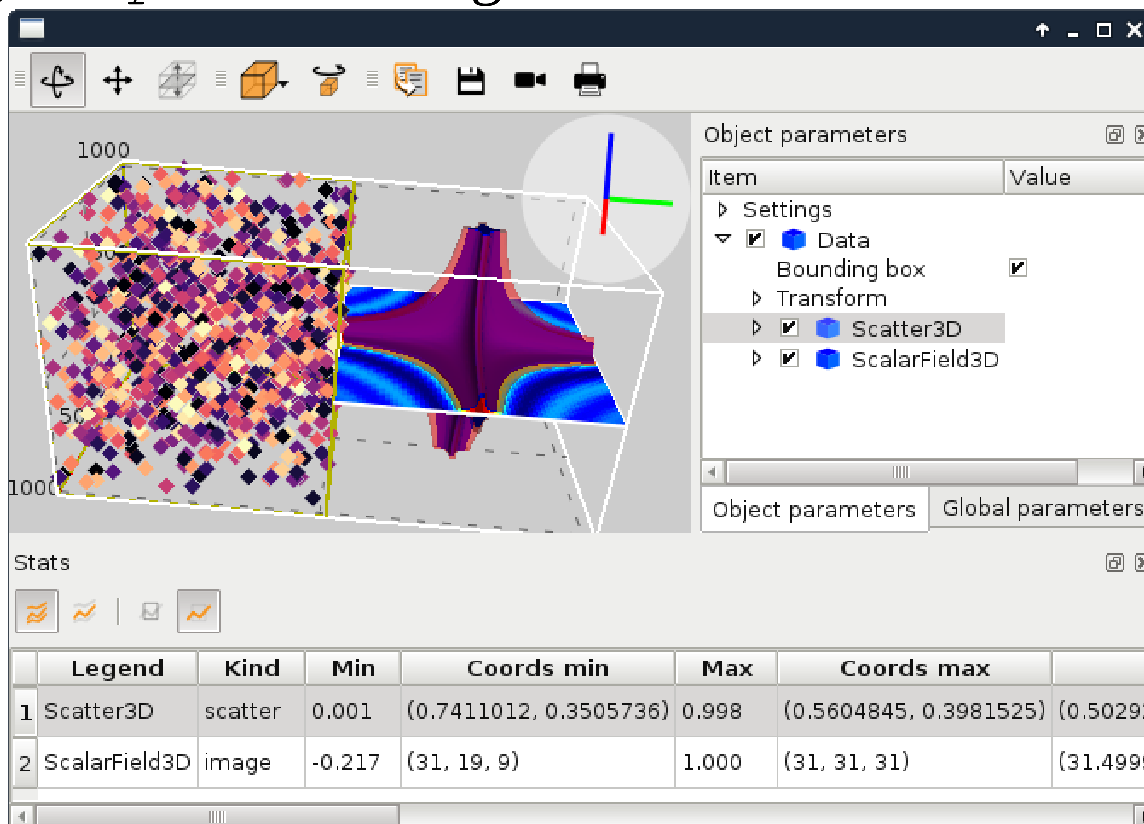


silx.gui.plot.StatsWidget

- Add *BasicGridStatsWidget*



- Add support of *silx.gui.plot3d* widgets



- Improvements
  - On-demand mode
  - Optimizations

OpenGL in *plot3d* and *plot*

- Support for Qt >= 5.4 OpenGL Widgets *(QOpenGLWidget)*

- Better support of OpenGL context issues (i.e. missing QtOpenGL, ssh GLX forwarding disabled,…) : display an error message rather than raising exceptions.

- First steps of Continuous Integration for OpenGL-based widgets

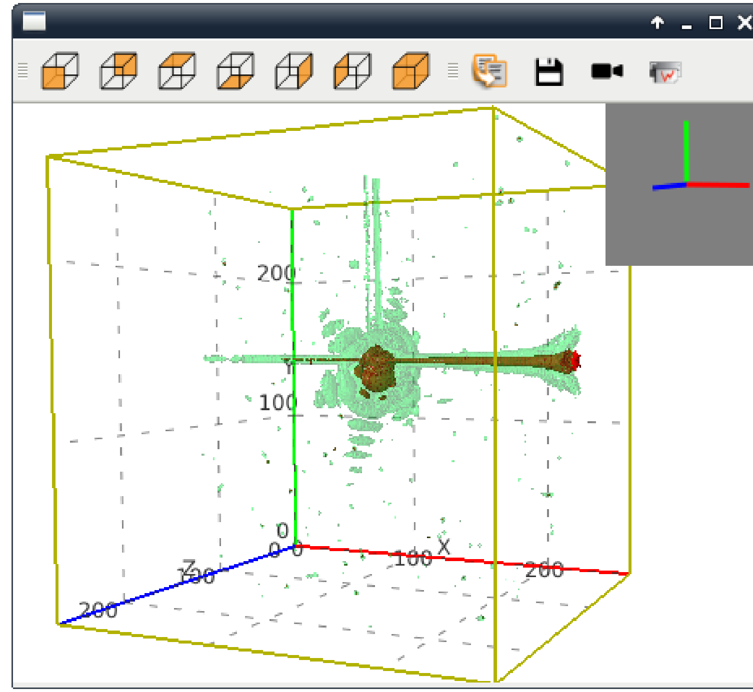
silx.gui.plot OpenGL Rendering

Matplotlib and OpenGL rendering backends in silx.gui.plot widgets:

- Usage: Set argument backend='gl‘ in widget constructor for: PlotWidget, PlotWindow, Plot1D, Plot2D, StackView, ImageView
- Example:

```
from silx import sx
plot = sx.Plot2D(backend='gl')
plot.show()
```

silx.gui.plot3d

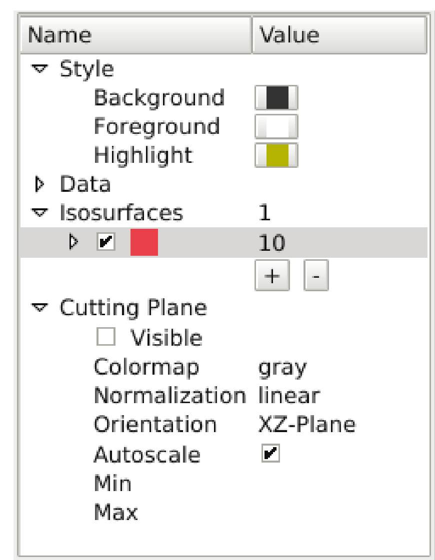


Silx 3D Visualization

- Dependencies

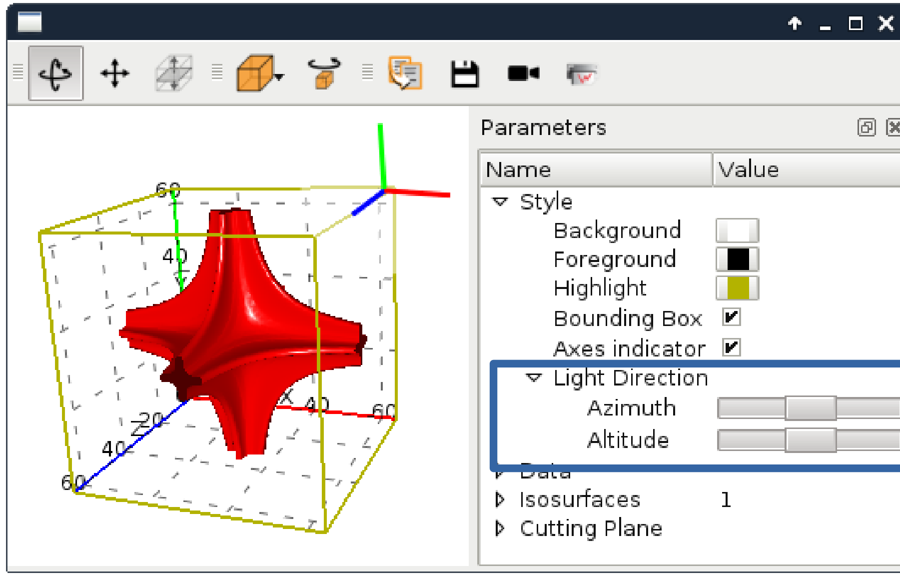
  - PyQt.QtOpenGL
  - PyOpenGL 3.x
  - OpenGL 2.1 subset

- Qt widgets for 3D plotting
  - ScalarFieldView (scalar field visualization)
  - Iso-surfaces
  - Cutting plane
- Based on an internal 3D scene structure



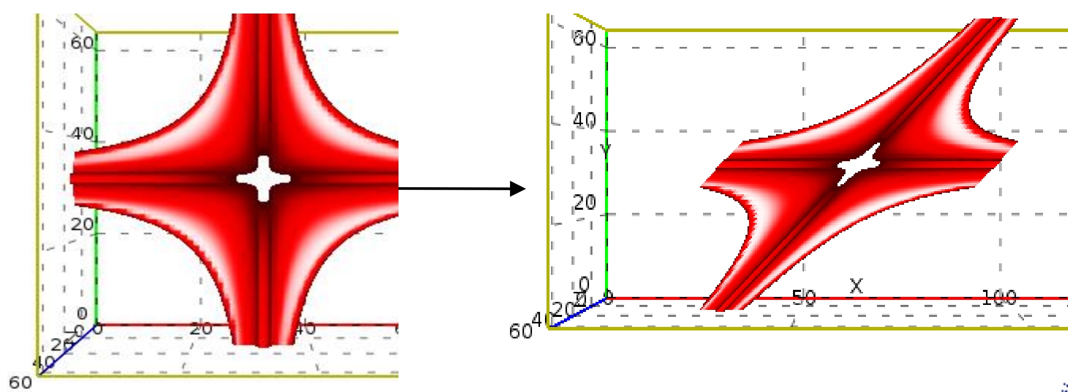
silx.gui.plot3d: ScalarFieldView

- Add light control



- Support of 3x3 matrix transform (for non-orthogonal axes support) to 3D scalar field visualization widget (`ScalarFieldView`):

```
scalarFieldView.setTransformMatrix((
     (1., 1., 0.),
     (0., 1., 0.),
     (0., 0., 1.)))
```

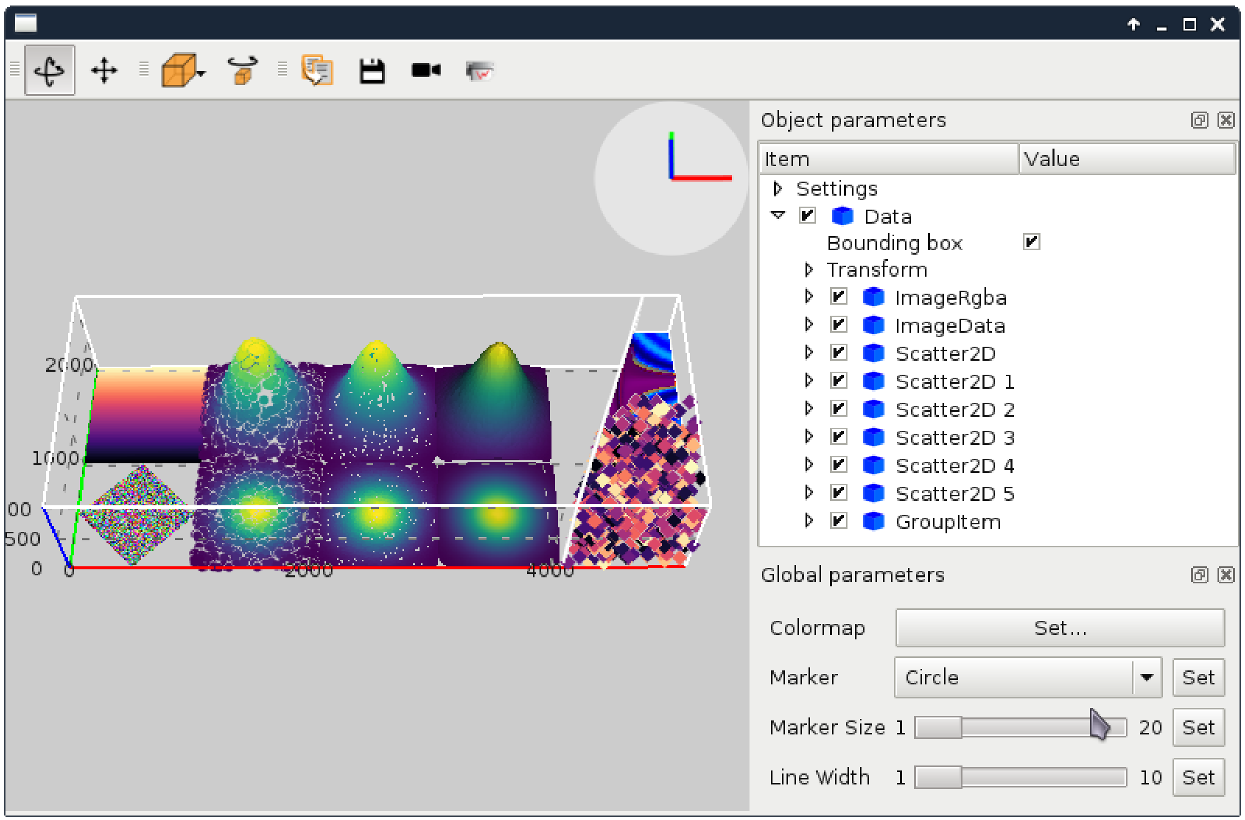


The European Synchrotron | ESRF

silx.gui.plot3d: Scene widgets

General purpose 3D visualization widget and associated tools:

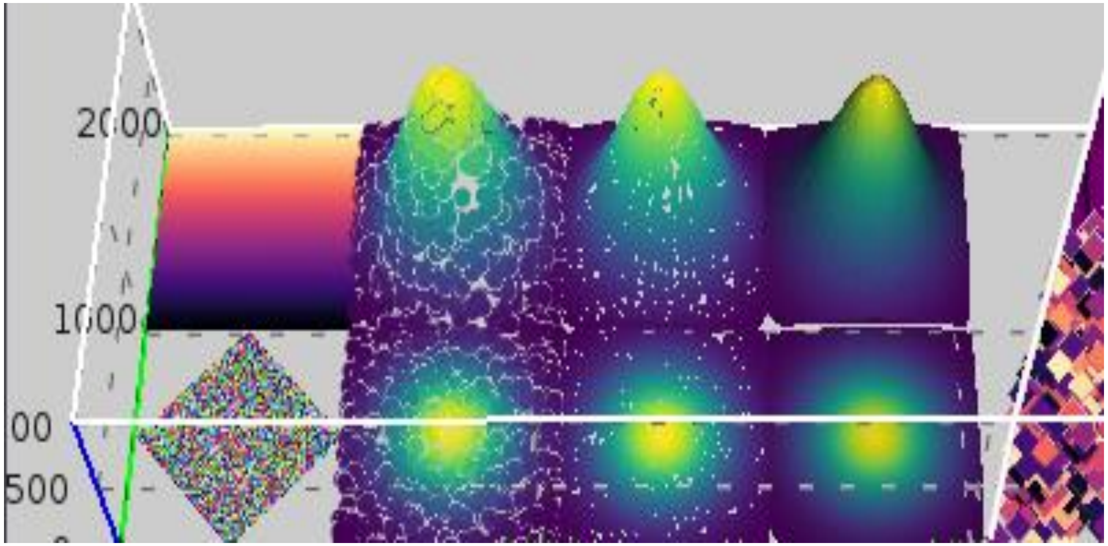
- Goal: Replacement candidate for PyMca OpenGL tab



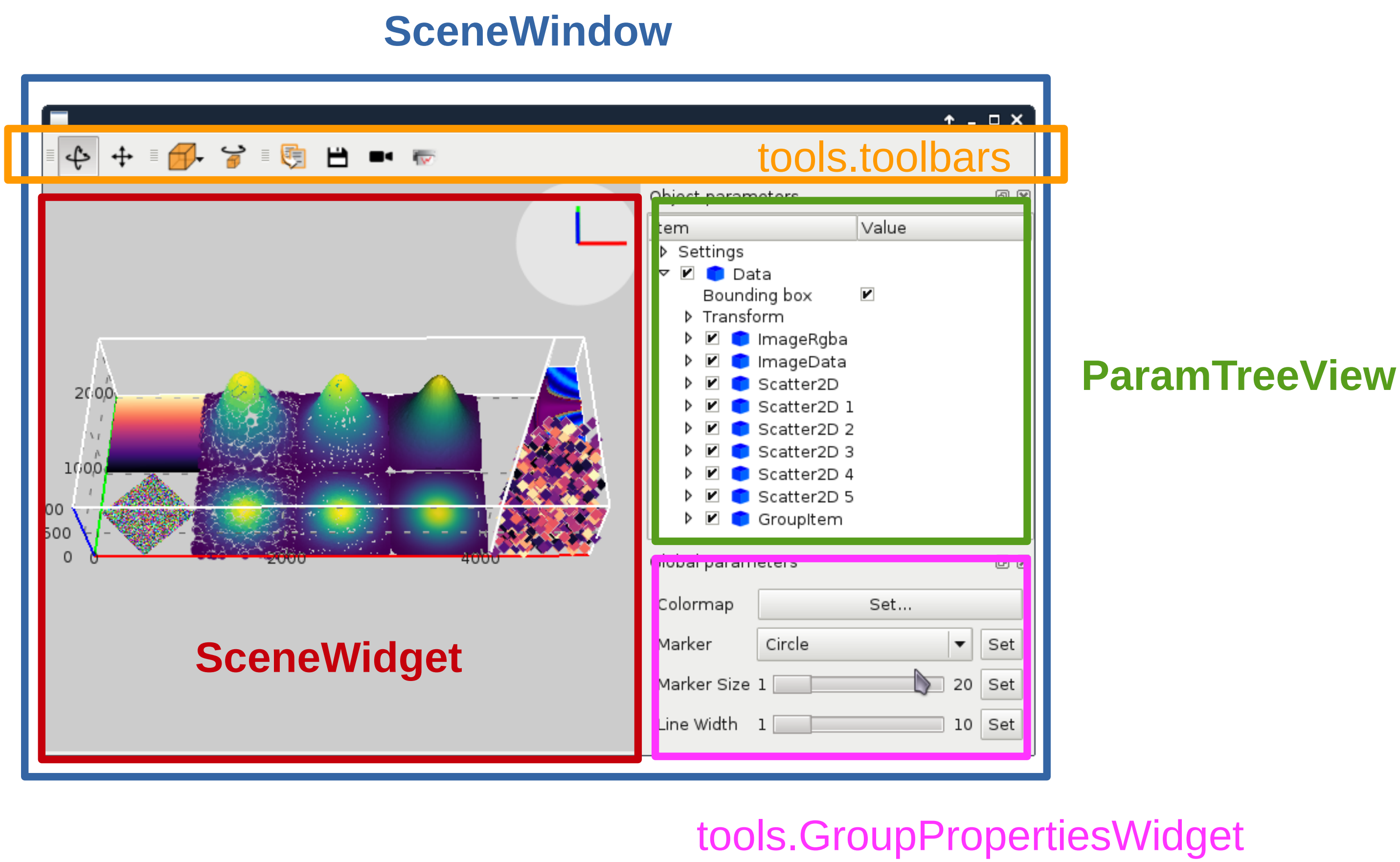
silx.gui.plot3d: Scene available items

`silx.gui.plot3d.items`:

- Images: `ImageData`, `ImageRgba`
- Scatter plots: `Scatter2D`, `Scatter3D`
- Scalar fields (with a cut plane and isosurfaces): `ScalarField3D`
- A clipping plane: `ClipPlane`
- 3D meshes: `Mesh`
- Groups: `GroupItem`, `GroupWithAxesItem`



silx.gui.plot3d: Scene widgets structure



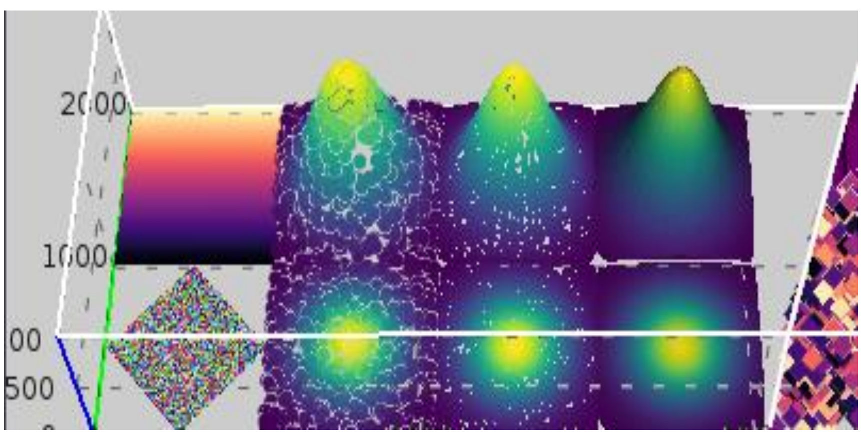
silx.gui.plot3d: ParamTreeView

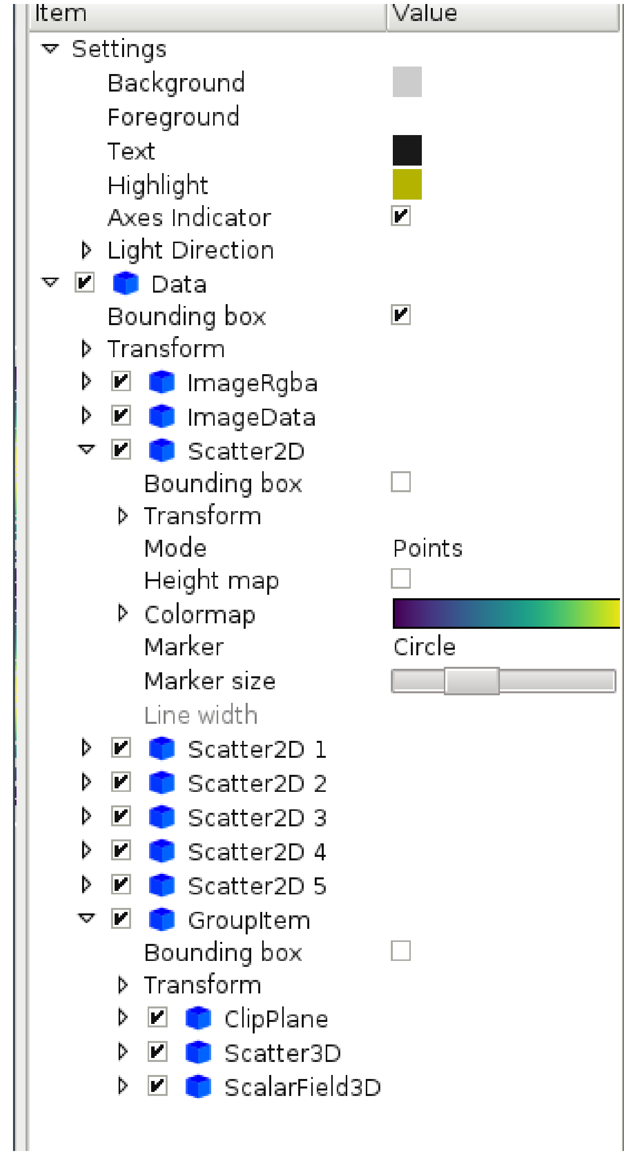
Content/Parameter tree based on:

- `silx.gui.plot3d.ParamTreeView`
- `SceneWidget.model()`

- If there is interest,

this can be adapted to 1D, 2D `PlotWidget`

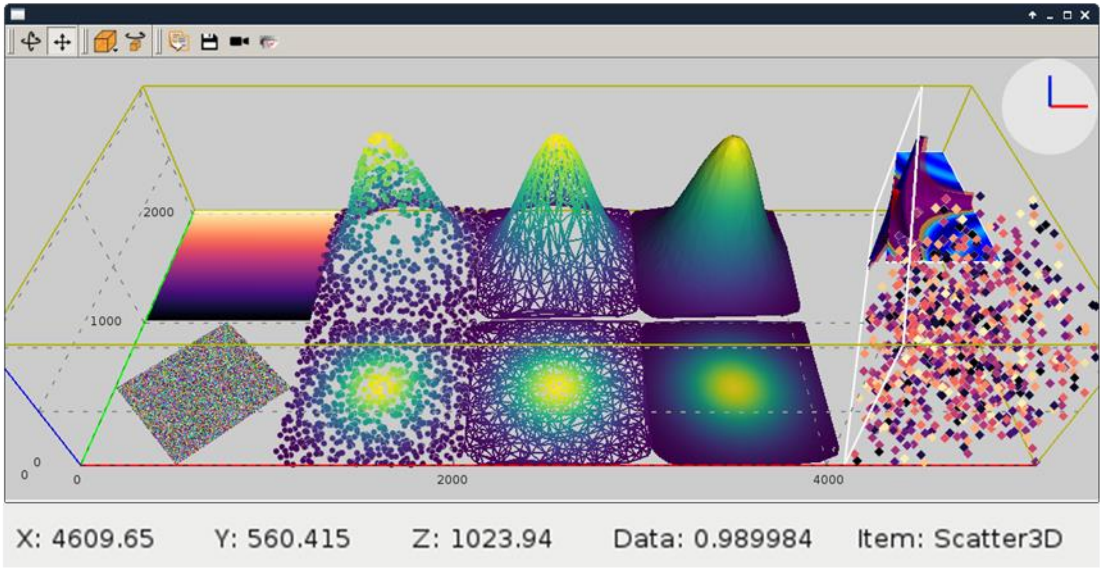




silx.gui.plot3d: PositionInfoWidget

`silx.gui.plot3d.tools.PositionInfoWidget:`

- Widget displaying data at mouse position on double-click.



silx.gui.plot3d: Picking

`silx.gui.plot3d.SceneWidget`: Add picking of 3D items at a position on the screen:

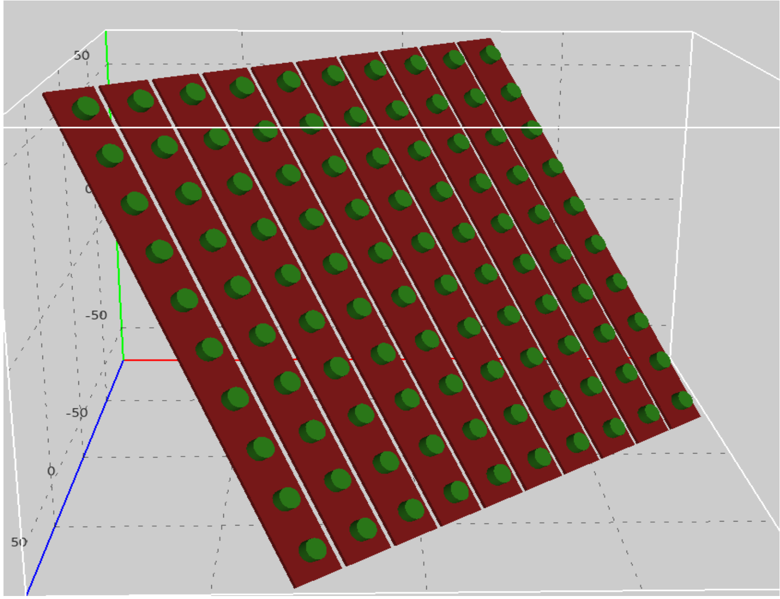
```
pickItems(x, y, condition=None)
```

Implementation choices:

- CPU-based ray-casting
- No preprocessing (e.g., space partitioning)
- Pure Python/numpy implementation

silx.gui.plot3d: Simple 3D Shapes

- Simple shapes: Cubes, cylinders, hexagons
- Allows to render many similar shapes at once
- Thanks to Guillaume Communie (ISDD/Detector & Electronics)



silx.math: miscellaneous mathematical functions

- Non-linear least squares with constraints on fitting parameters
    - Has a configuration widget for easy integration into GUIs
- 1D peak search
- Isosurface calculations with Marching Cubes algorithm
    - For 4D visualization (visualization of scalar fields)
- N-dimensional histograms based on look-up tables
- Fitting functions with automatic estimation of initial parameters
- 1D and 2D median filters
- FFT

silx.math.fft: silx FFT

- A new module for Fast Fourier Transform: *`silx.math.fft`*

- One unique interface with 4 backends:
  - numpy, fftw, OpenCL, CUDA.

- 1D, 2D, 3D ; possibly batched

- R2C, C2C

- For GPU transforms, input and/or output can be device arrays.

- Currently not supported :
  - In-place transforms
  - Hermitian transforms
  - Automatic zero-padding (ex. fft(data, size=2048))

silx.math.fft: silx FFT

Simple FFT with numpy

```
import numpy as np
from scipy.misc import ascent
from silx.math.fft import FFT
img = ascent().astype(np.float32)

F = FFT(data=img, backend="numpy") # automatically chooses R2C transform
img_f = F.fft(img)
```

Using FFTW

```
F = FFT(data=img, backend="fftw", num_threads=4)
img_f = F.fft(img)
# do some operation of img_f ...
F.ifft(img_f)
```

silx.math.fft: silx FFT

Using OpenCL

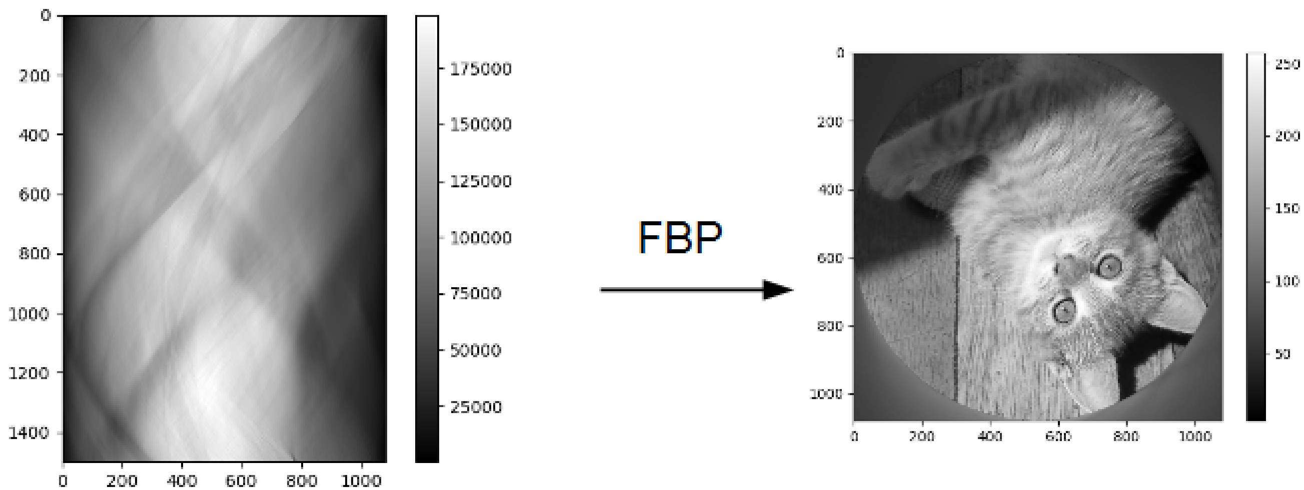
```
import pyopencl.array as parray
F = FFT(data=img, backend="opencl")
# All the Host <-> Device copies are handled under the hood
img_f = F.fft(img) # by default, result is a numpy array
# Input and/or output can be device array as well
d_in = parray.to_device(F.queue, img)
d_out = parray.zeros(F.queue, F.shape_out, dtype=F.dtype_out)
F.fft(d_in, output=d_out)
```

Using CUDA

```
import pycuda.autoinit
import pycuda.gpuarray as gpuarray
F = FFT(data=img, backend="cuda")
d_in = gpuarray.to_gpu(img)
d_out = gpuarray.zeros(F.shape_out, F.dtype_out)
F.fft(d_in, output=d_out) # CUFFT is twice faster than clfft for R2C transforms
```

Filtered Back Projection in silx

- Filtered Back-Projection (**FBP**) is the usual reconstruction method in (parallel) tomography
- silx now provides a FBP module
- The filtering can be omitted if the data is already filtered
- Works on both GPU and CPU (Mac OS is not supported)



sinogram

slice

silx.opencl.backprojection: silx backprojector

- New features in *`silx.opencl.backprojection`*:
- Filtering is done with *`silx.math.fft`* (possibly on GPU)
- User can choose other built-in and custom filters
- Input and/or output of backproj/FBP can be numpy and pyopencl arrays

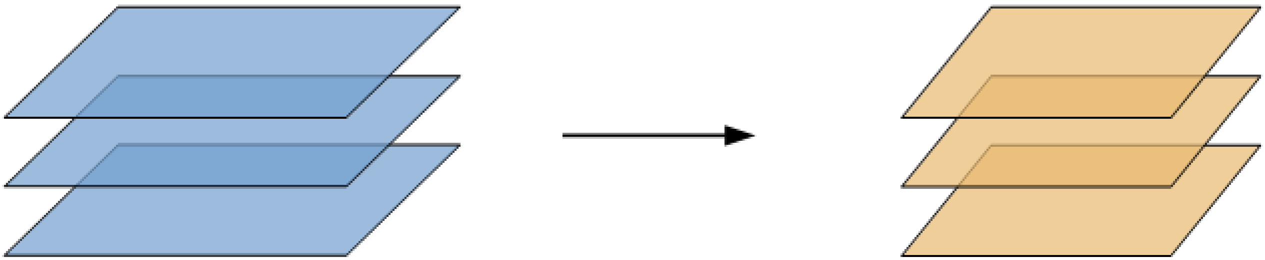
silx.opencl.backprojection: silx backprojector

```
import numpy as np
import pyopencl.array as parray
from silx.opencl.backprojection import Backprojection
from silx.test.utils import utilstest
from silx.gui import qt
from silx.gui.plot.CompareImages import CompareImages

sino = np.load(utilstest.getfile("sino500.npz"))["data"]

B1 = Backprojection(sino.shape)
rec1 = B1(sino)

B2 = Backprojection(
    sino.shape,
    filter_name="hamming",
    extra_options={"cutoff": 0.7}
)
d_sino = parray.to_device(B2.queue, sino)
d_rec2 = parray.zeros(B2.queue, B2.slice_shape, "f")
B2(d_sino, output=d_rec2)

app = qt.QApplication([])
C = CompareImages()
C.setData(rec1, d_rec2.get())
C.show()
app.exec_()
```

silx.opencl : Filtered Back Projection in silx

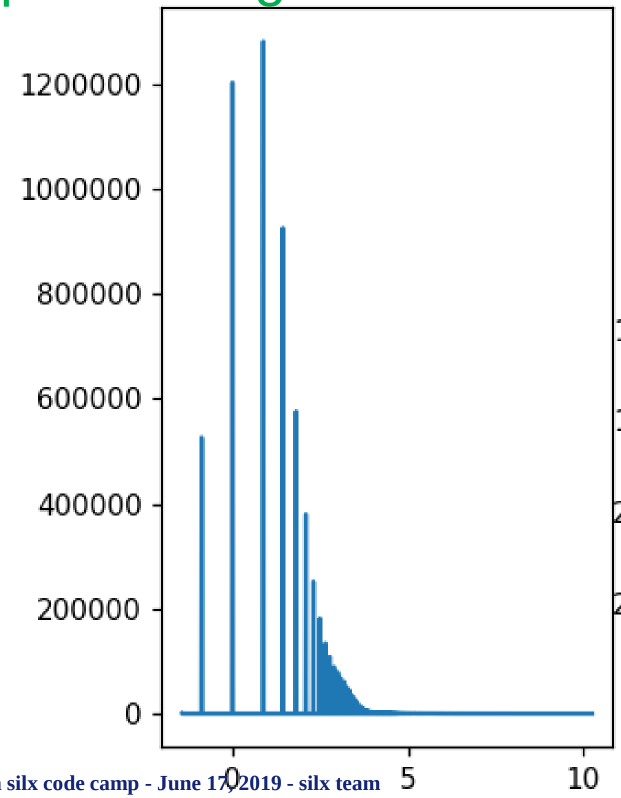
- Principle : define a geometry and use it to reconstruct one or several sinograms.
- Geometry = sinogram shape, [series of angles, slice shape, rotation center position]

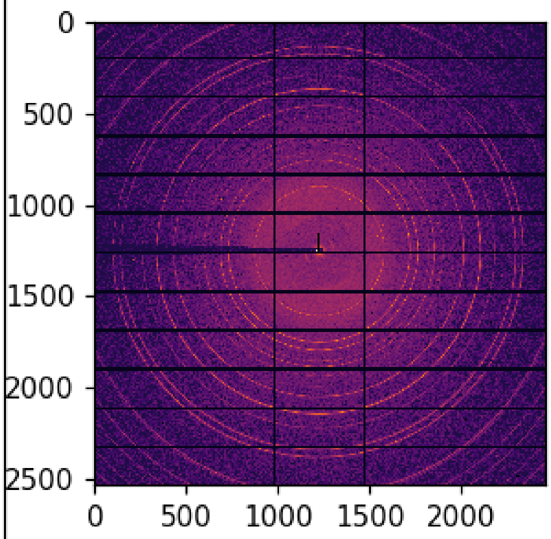
```
from  silx.opencl.backprojection import Backprojection
# Compute the tomography geometry
tomo_geometry = Backprojection(sinograms_stack.shape[1:],
                               axis_position=1337,
                               devicetype='GPU')

# Allocate the memory for volume reconstruction
num_sinos = sinograms_stack.shape[0]
reco = np.zeros((num_sinos,) + tomo_geometry.shape)
# Reconstruct
for i in range(num_sinos):
    reco[i] = tomo.fbp(sinograms_stack[i])
```



CoDec : Byte offset for CBF processing on GPU

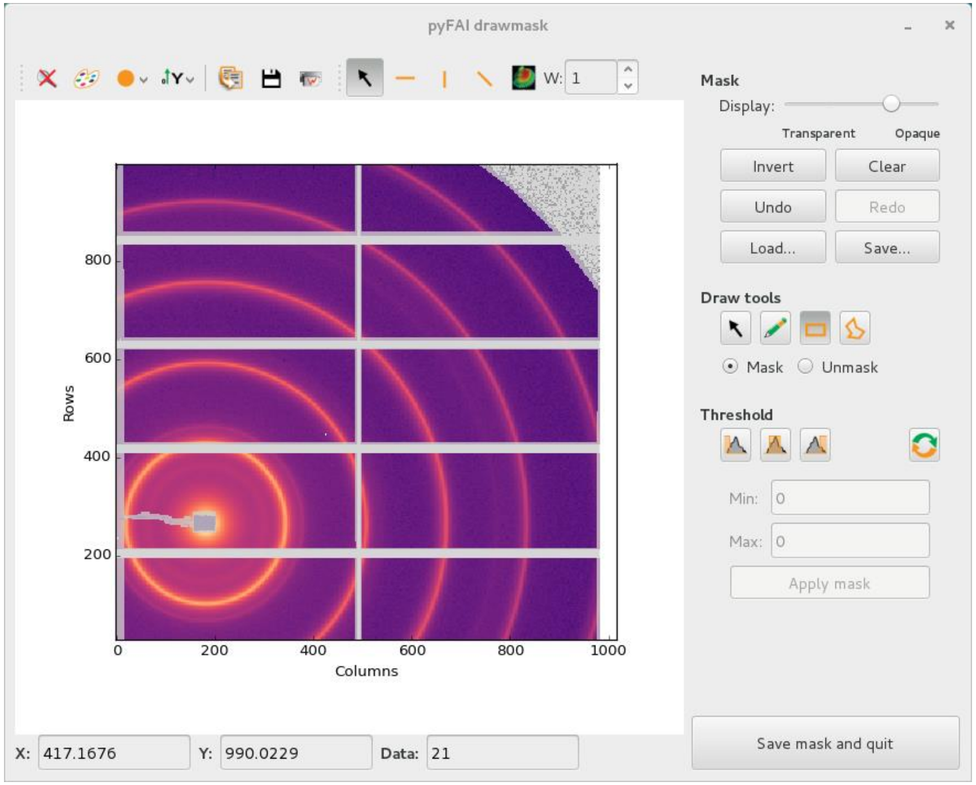
- `silx.opencl.codec.byte_offset`
  - OpenCL-based CBF compression
- 10x speed-up for compression/decompression of CBF streams
  - Compatible with the new Image processing framework
  - Compatible with pyFAI azimuthal integration
- Accepted in J. Synchrotron Radiation https://doi.org/10.1107/S1600577518000607





silx.image: image processing tools

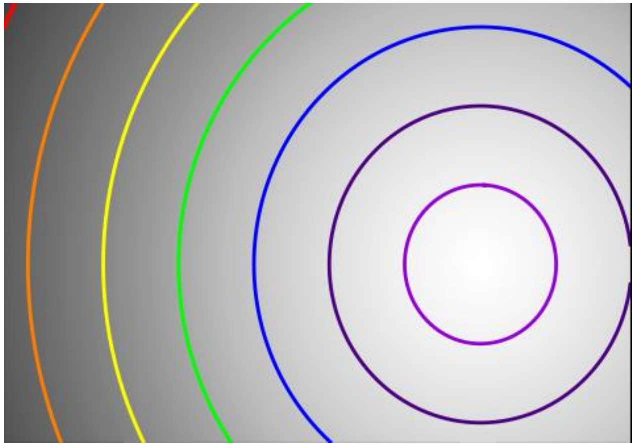
- Basic shapes for masks
  - Line profiles
  - Polygons
  - Circle
- Bilinear interpolation
  - Used to scale up/down images to display
- Gaussian blurring of images
  - GPU accelerated via OpenCL
- Image registration and alignment (SIFT)
  - GPU accelerated via OpenCL



- Marching Squares
- Median Filter
  - GPU accelerated via OpenCL

Marching Squares Algorithm

- **Designed to speed up PyFAI calibration GUI**
  - Cython + OpenMP
  - Support masking
  - Optimization to reach many contours from the same gradient image
  - Reach contours or pixels
- **Example:** `examples/findContours.py`



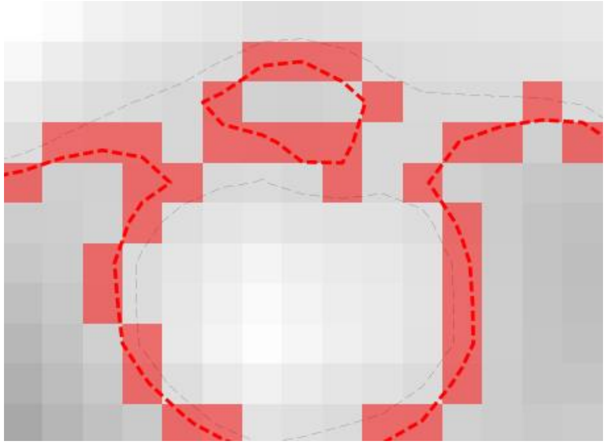


Image processing on opencl devices (GPU)

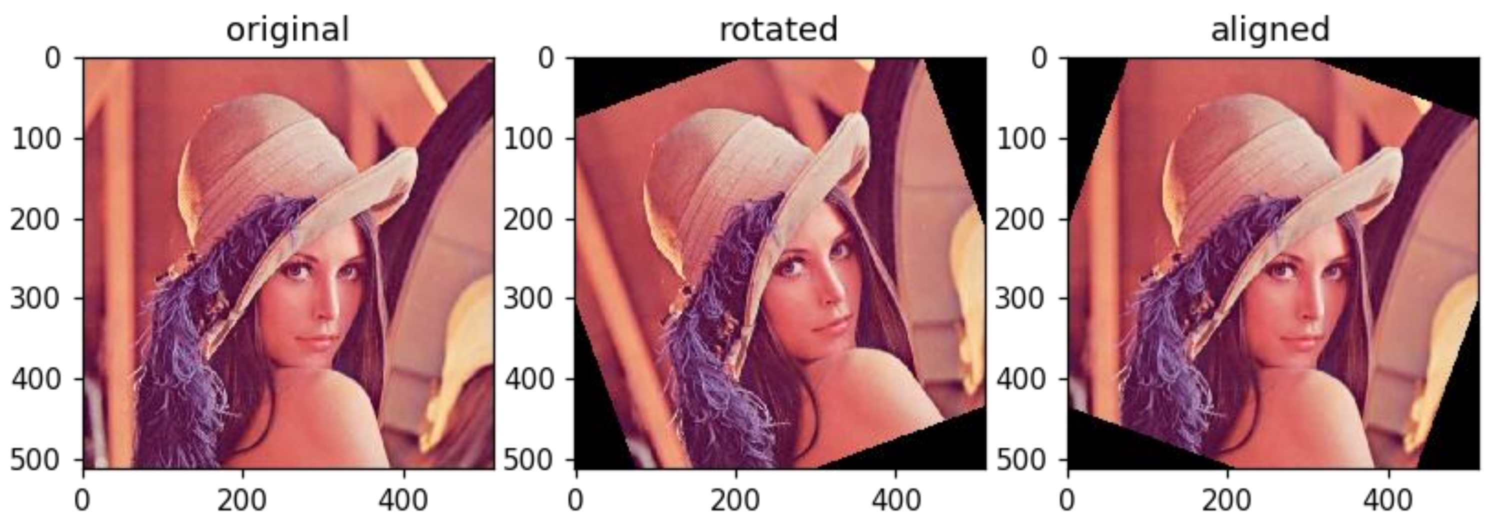
- New image processing framework:
  - Allows to exchange buffers on the device
  - Allows the creation of work-flow without copying data back & forth
  - Better performances

- Few image treatments implemented:
  - Buffer conversion to float arrays from any integer
  - Min/Max search (double-reduction)
  - Image normalization
  - Image histogram

- Tutorial available:
  - https://github.com/kif/silx/blob/1199_ocl_image/doc/source/Tutorials/Image.ipynb

Sift: Image alignment

- Use the “image” framework.
- Major re-work for compatibility with PyOpenCL > 2015
- Compatibility with “spectre” corrections
- Many memory-leak corrected
- New tutorial based on jupyter notebook.
  https://github.com/silx-kit/silx/blob/master/doc/source/Tutorials/Sift/sift.ipynb

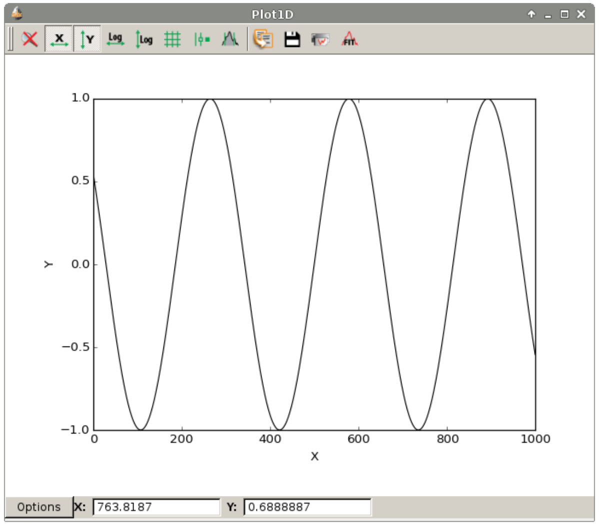


silx.sx: a module to simplify interactive use

pylab like module on steroids

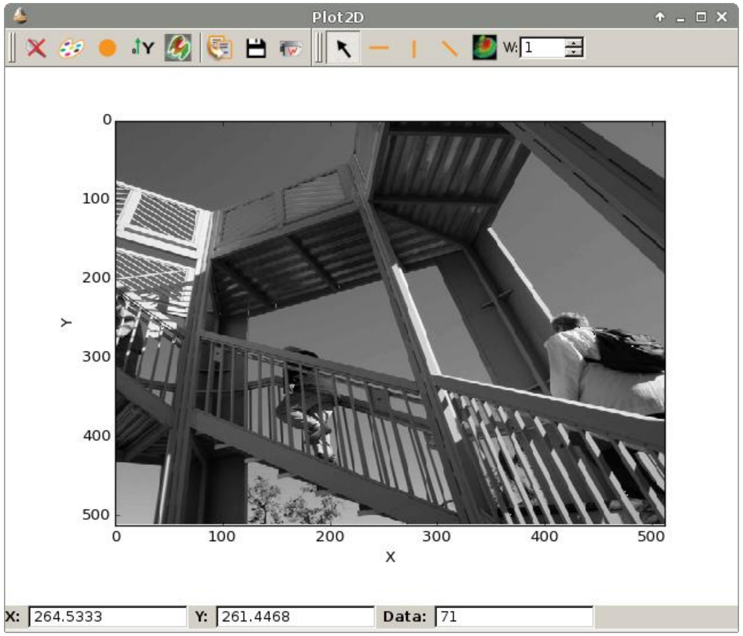
- 1D plotting: ROI, fitting & printing

```
>>> from silx import sx
>>> from numpy import sin, linspace
>>> sx.plot(sin(linspace(-10, 10, 1000)))
```



- 2D display: intensity, mask, profile

```
>>> from scipy.misc import ascent
>>> sx.imshow(ascent())
```



silx.io: input / output

- Built-in support of CSV, SPEC and TIFF
  - Images, SPEC files accessed in the same way as HDF5 files

    Unified widget dealing with ALL supported data formats!!!!!

  - Provide bridges SPEC ←→ HDF5 and octave ←→ HDF5
  - Utilities to save and restore configurations as HDF5, json or ini files
- HDF5 is supported via h5py
- Images (and many detector formats) are supported via FabIO

silx.io.commonh5

- This new module provides a common base for *silx.io.spech5* and *silx.io.fabioh5* to provide the h5py-like API for various data formats.

- If new formats are handled by silx in the future, and they inherit the commonh5 classes, they will benefit from the existing tools:
  - *silx.io.convert*
  - *silx.io.utils* (is_dataset, is_group, is_file,…)

silx.io.nxdata

- New functions
  - is_NXentry_with_default_NXdata(group)
  - is_NXroot_with_default_NXdata(group)
  - get_default(group)
    - Returns default silx.io.NXdata object or None. Group parameter can be NXdata, NXentry or NXroot.

  - save_NXdata(filename, signal, axes=None,
    signal_name="data", axes_names=None,
    signal_long_name=None, axes_long_names=None,
    signal_errors=None, axes_errors=None,
    title=None, interpretation=None,
    nxentry_name="entry", nxdata_name=None)

silx.io.utils: HDF5 External storage helpers

- Two helper functions to create an external dataset:
- From a .vol file (pyhst):

```
→ vol_to_h5_external_dataset(vol_file, output_url,
                              info_file=None,
                              vol_dtype=numpy.float32,
                              overwrite=False)
```

- From a binary file:

```
→ rawfile_to_h5_external_dataset(bin_file,
                                 output_url,
                                 shape,
                                 dtype, overwrite=False)
```

- Need h5py >= 2.9

silx.io.convert

- Module

- Before only SPEC files could be converted (*silx.io.spectoh5*)
- New *silx.io.convert* supports Fabio images (replaces *spectoh5)*

- Application

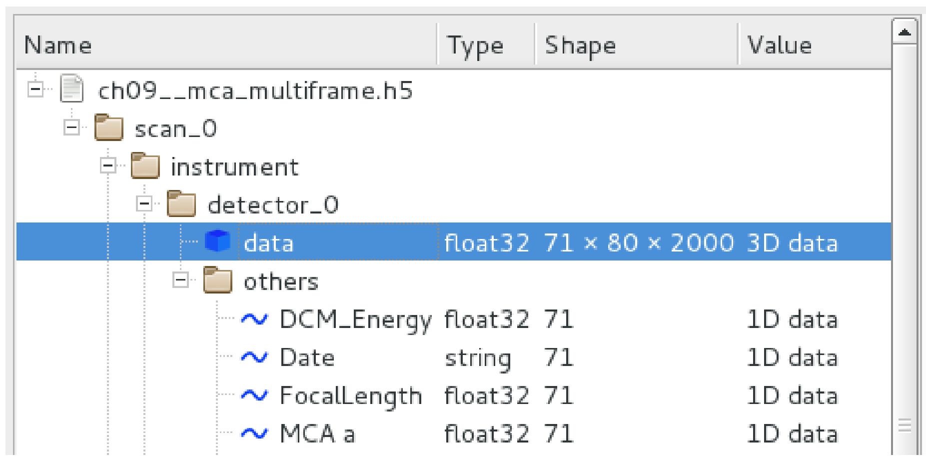
- New command line application to convert files to HDF5

silx convert –help
silx convert filename

silx convert

- Convert series of single frame images (EDF, TIFF...) into a HDF5 multiframe stack

```
silx convert --file-pattern ch09__mca_0005_0000_%d.edf -o ch09__mca_multiframe.h5
```




```
silx convert -h
```

silx convert

- Merging SPEC and EDF files.
  - Step 1. Convert the SPEC file to HDF5 file

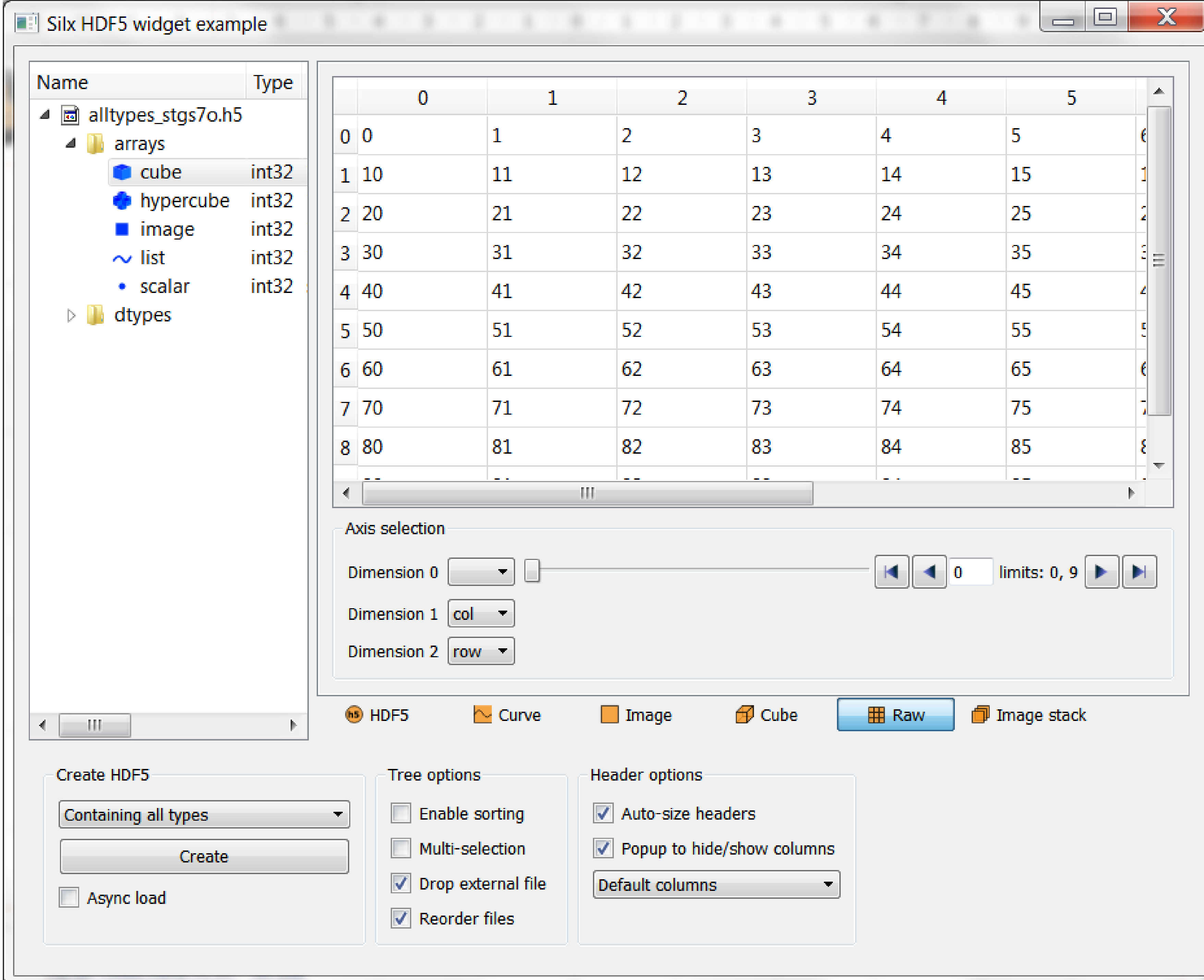
```
silx convert spec_file_name -o hdf5_file_name.h5
```

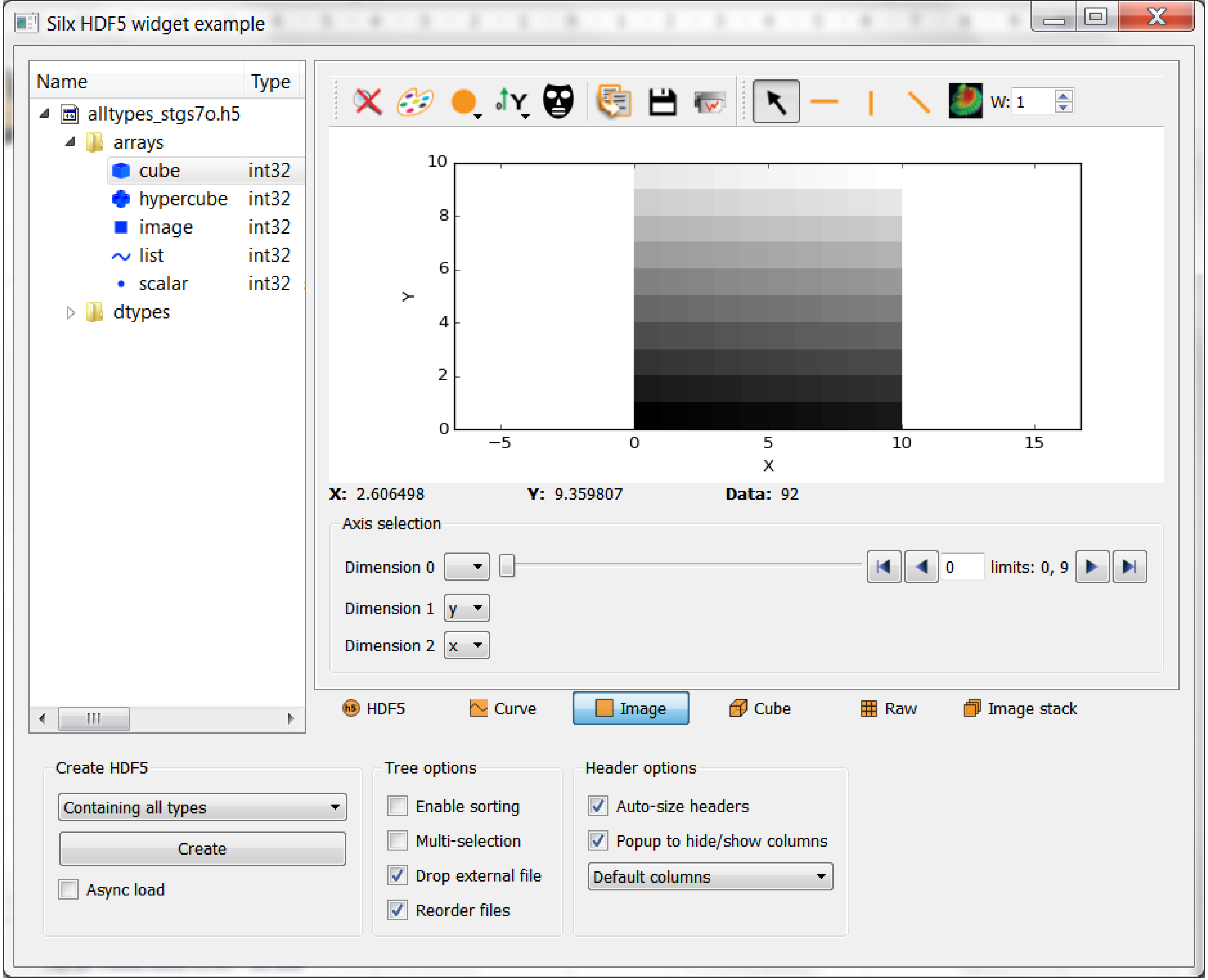
  - Step 2. Convert the EDF files selecting target path in generated HDF5 file

```
silx convert --file-pattern=root_%04d.edf --begin=100 --end=199 \
          --mode=r+ -o hdf5_file_name.h5::/1.1/instrument/detector_0
```

  - Hint Multiple indices supported (indexed files, indexed directories, …)

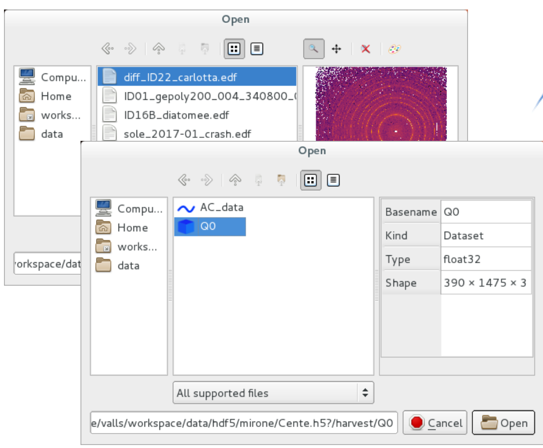
```
root_ssss_dddd_nnnn.edf

--file-pattern=root_%04d_%04d_$04d.edf -begin=1,0,0 -end=1,0,99
```


Silx HDF5 widget example
Name
Type
alltypes_stgs7o.h5
arrays
cube
int32
hypercube
int32
image
int32
list
int32
scalar
int32
dtypes
0 1 2 3 4 5
0 0 1 2 3 4 5
1 10 11 12 13 14 15
2 20 21 22 23 24 25
3 30 31 32 33 34 35
4 40 41 42 43 44 45
5 50 51 52 53 54 55
6 60 61 62 63 64 65
7 70 71 72 73 74 75
8 80 81 82 83 84 85
Axis selection
Dimension 0
0
limits: 0, 9
Dimension 1
col
Dimension 2
row
HDF5
Curve
Image
Cube
Raw
Image stack
Create HDF5
Containing all types
Create
Async load
Tree options
Enable sorting
Multi-selection
Drop external file
Reorder files
Header options
Auto-size headers
Popup to hide/show columns
Default columns


Silx HDF5 widget example
Name
Type
alltypes_stgs7o.h5
arrays
cube
int32
hypercube
int32
image
int32
list
int32
scalar
int32
dtypes
W: 1
Y
X
X: 2.606498
Y: 9.359807
Data: 92
Axis selection
Dimension 0
0
limits: 0, 9
Dimension 1
y
Dimension 2
x
HDF5
Curve
Image
Cube
Raw
Image stack
Create HDF5
Containing all types
Create
Async load
Tree options
Enable sorting
Multi-selection
Drop external file
Reorder files
Header options
Auto-size headers
Popup to hide/show columns
Default columns

Dialog to reach data

File system



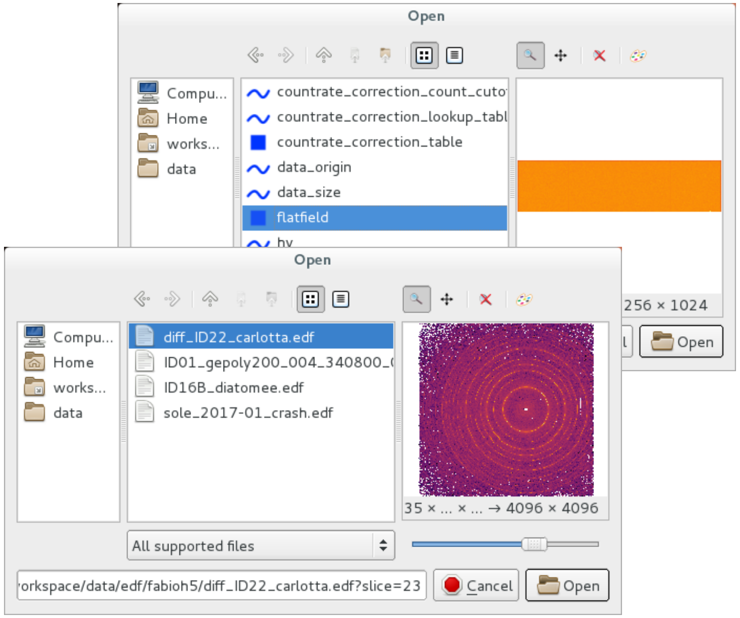
silx.gui.dialog

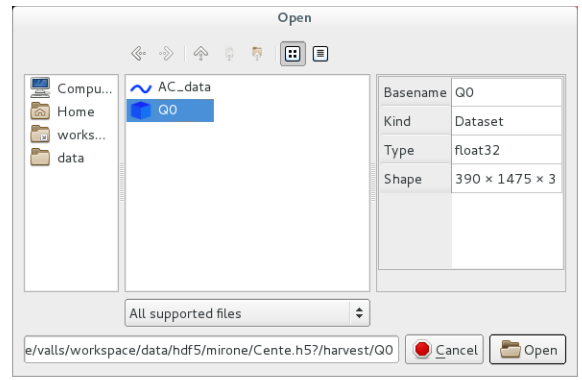
URL

silx.io.open
(h5py-like context)

silx.io.get_data
(numpy data)

Data dialogs





`ImageFileDialog`

- **Specialised to select an image**
- **Support slicing of hypercubes**
- **Support h5-like**
- **Support raw image files (edf, tiff, cbf)**

`DataFileDialog`

- **Select anything from h5-like structure**
- **Filter to select only datasets or groups**

Data URLs

- **Custom schemes**
  - `silx:///home/user/foo.edf?path=/group/&slice=5`
  - `fabio:///home/user/foo.edf?slice=5`
  - Also available for relative paths
- **Reach data from datasets and fabio URLs**

```
data = silx.io.get_data(url)
```

- **Reach data from other URLs**

```
with silx.io.open(url) as node:
      print(node)
```

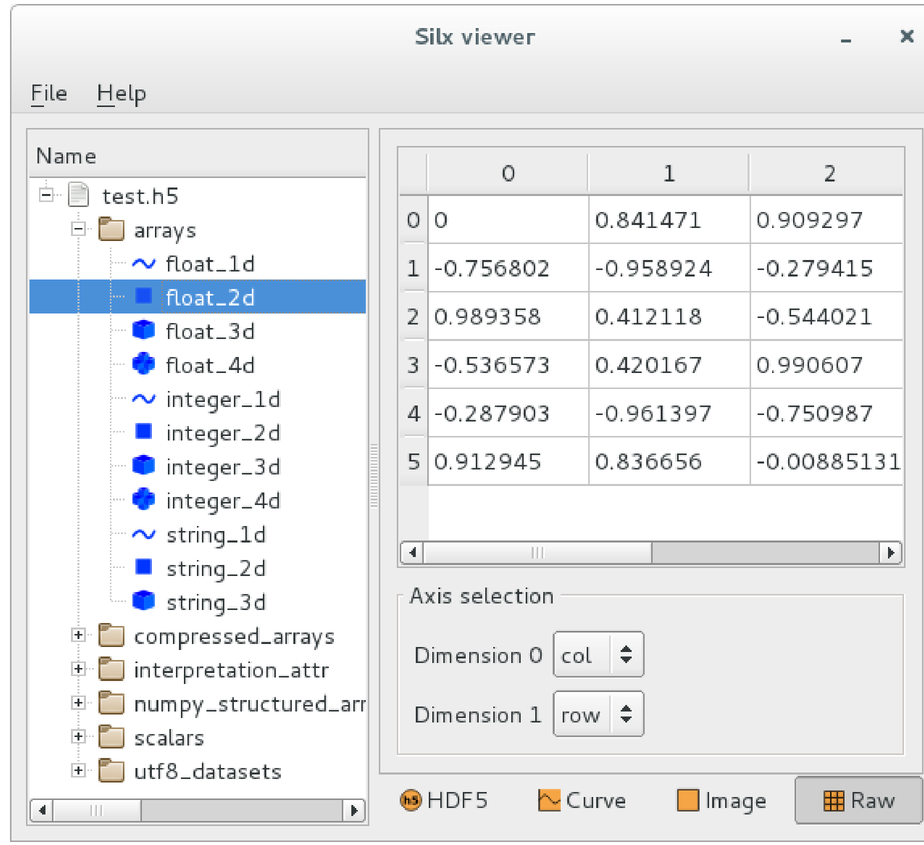
- **An object is provided to parse our URLs**
  - `silx.io.url.DataUrl`
- **We also support h5pyd URLs**
  - `http://127.0.0.1:5000/tall.public.hdfgroup.org`

Viewer Application

- Browse and display HDF5 files
  (plus any supported file as HDF5)

- File from:
  - *command line / open dialog / drag and drop*

- Commands
  - *silx view <filename>*
  - *python -m silx view*
  - *python3 -m silx view*
  - *./bootstrap.py silx view*



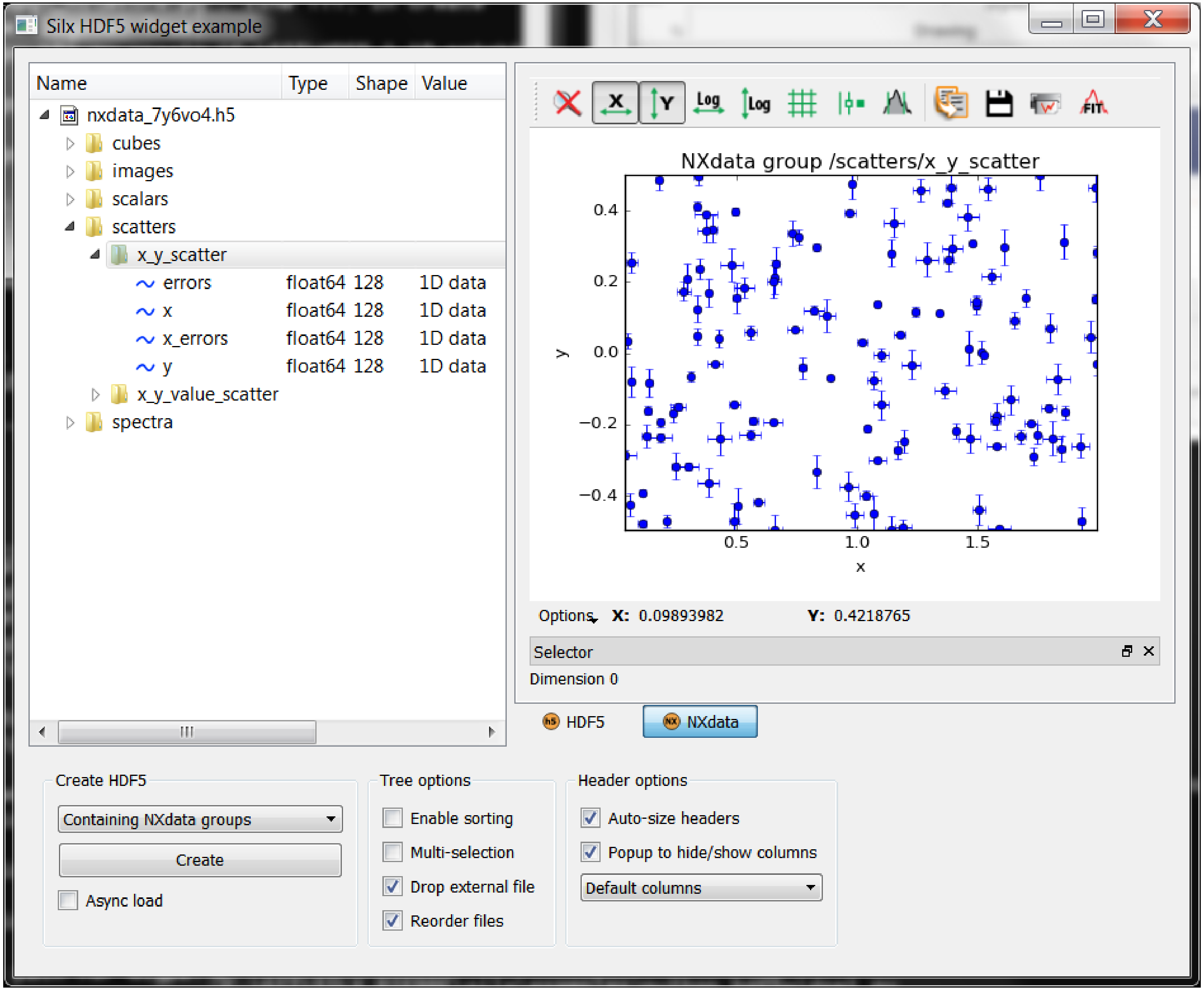
NXdataViewer

- Data viewer for viewing data in a Nexus NXdata group

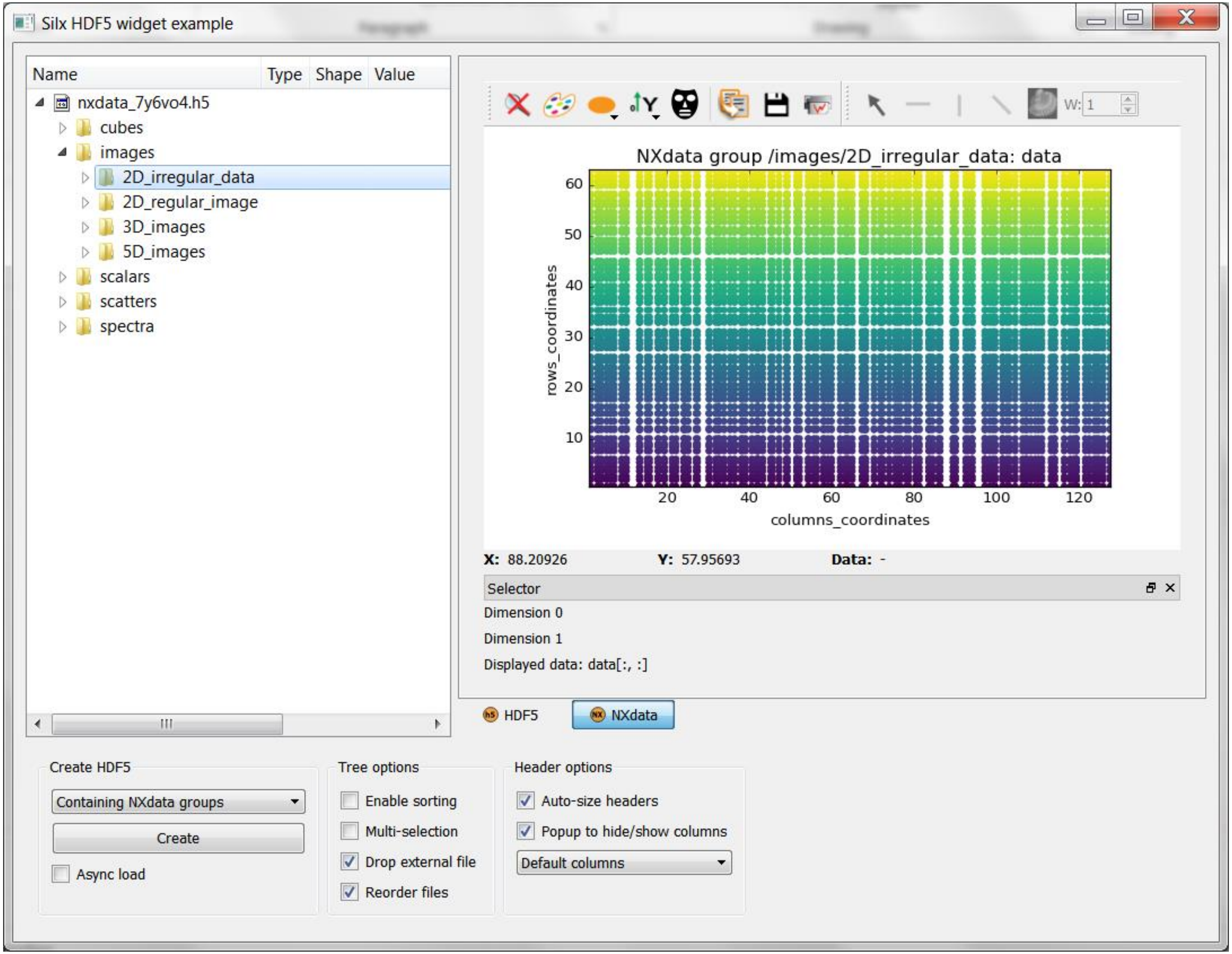
- Supports:
  - Scalars, curves, images, scatters, image stack for 3D data
  - Uncertainties, displayed as error bars for 1D data
  - Axes scaling (via @axes)
  - Axes labels (via @long_name)
  - Forcing of predefined views for high dimensionality data (via @interpretation=scalar/spectrum/image)

- See examples/hdf5widget.py for a demo
  (Create HDF5 > Containing NXdata groups)

silx view – Generic Viewer Interpreting NXdata Groups



NXdataViewer



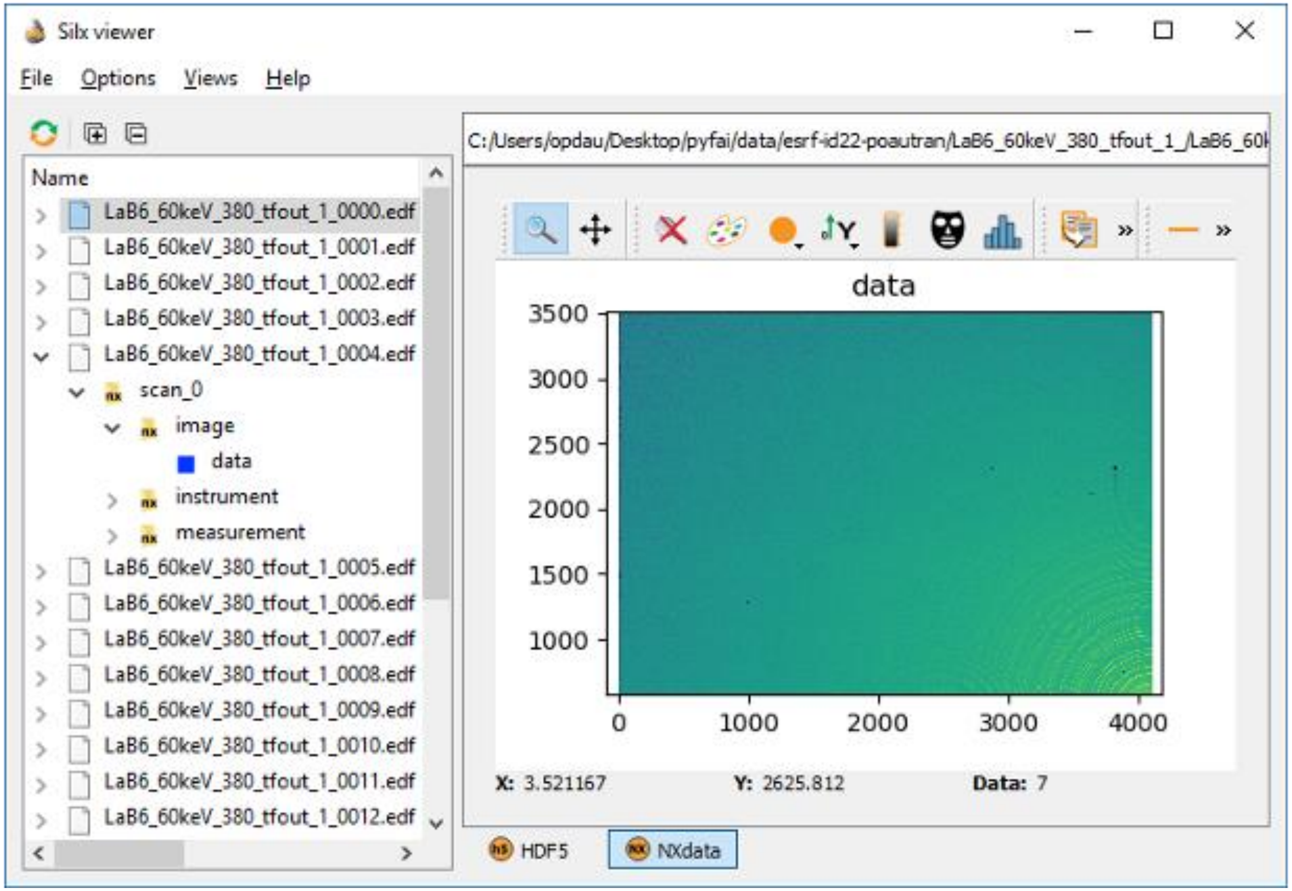
silx view

- Display *NXdata* view when viewing a *NXentry* or a *NXroot* group defining a @default attribute pointing to a valid *NXdata* group.

```
root:NXroot
  @default = "main_entry"

  main_entry:NXentry
    @default = "data"

    data:NXdata
      @signal = "counts"
      @axes = "mr"
      counts: float[100]
      mr: float[100]

  secondary_entry:Nxentry
    ...
```

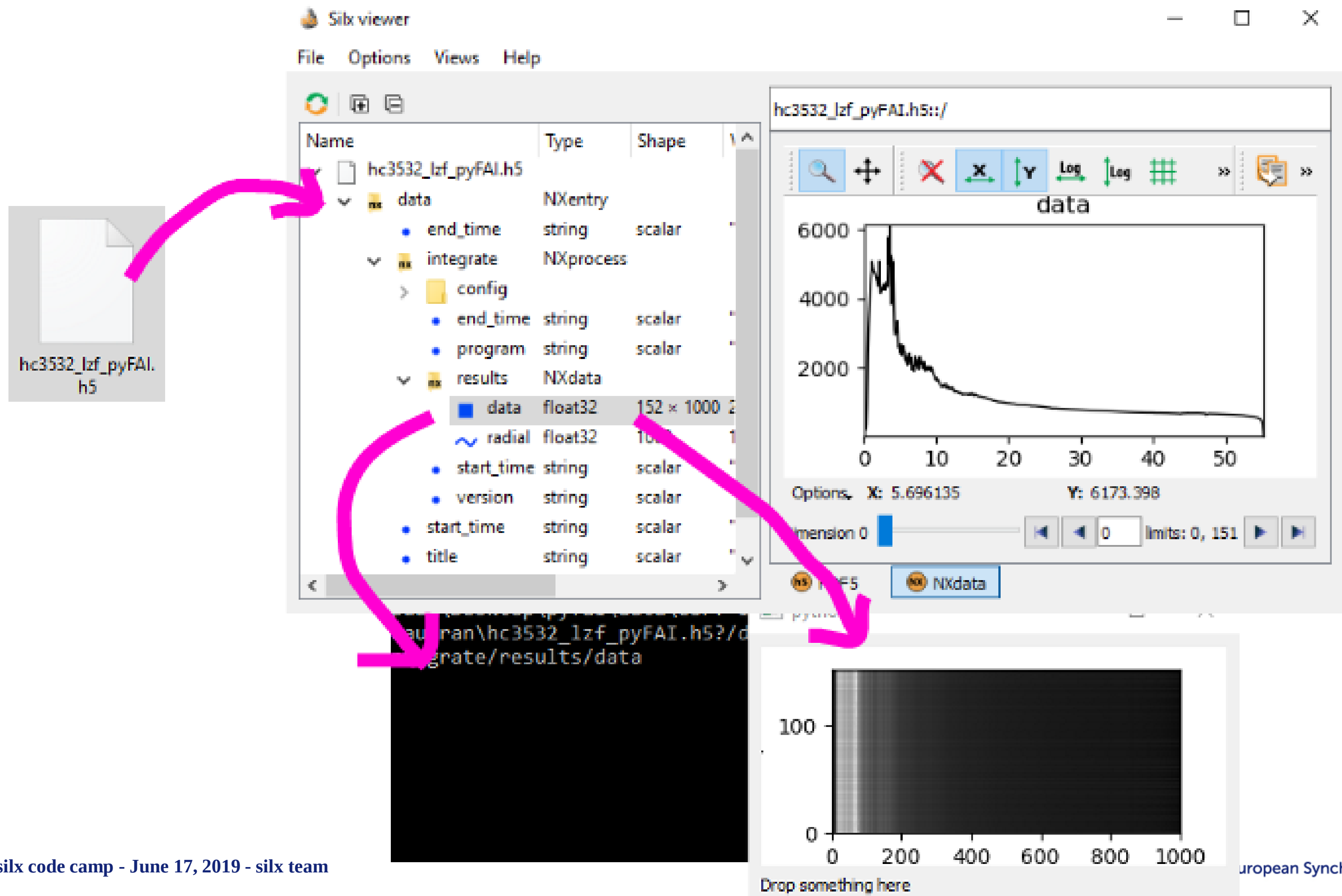
Silx view

- Image displayed without browsing it
  - A default NXdata is generated

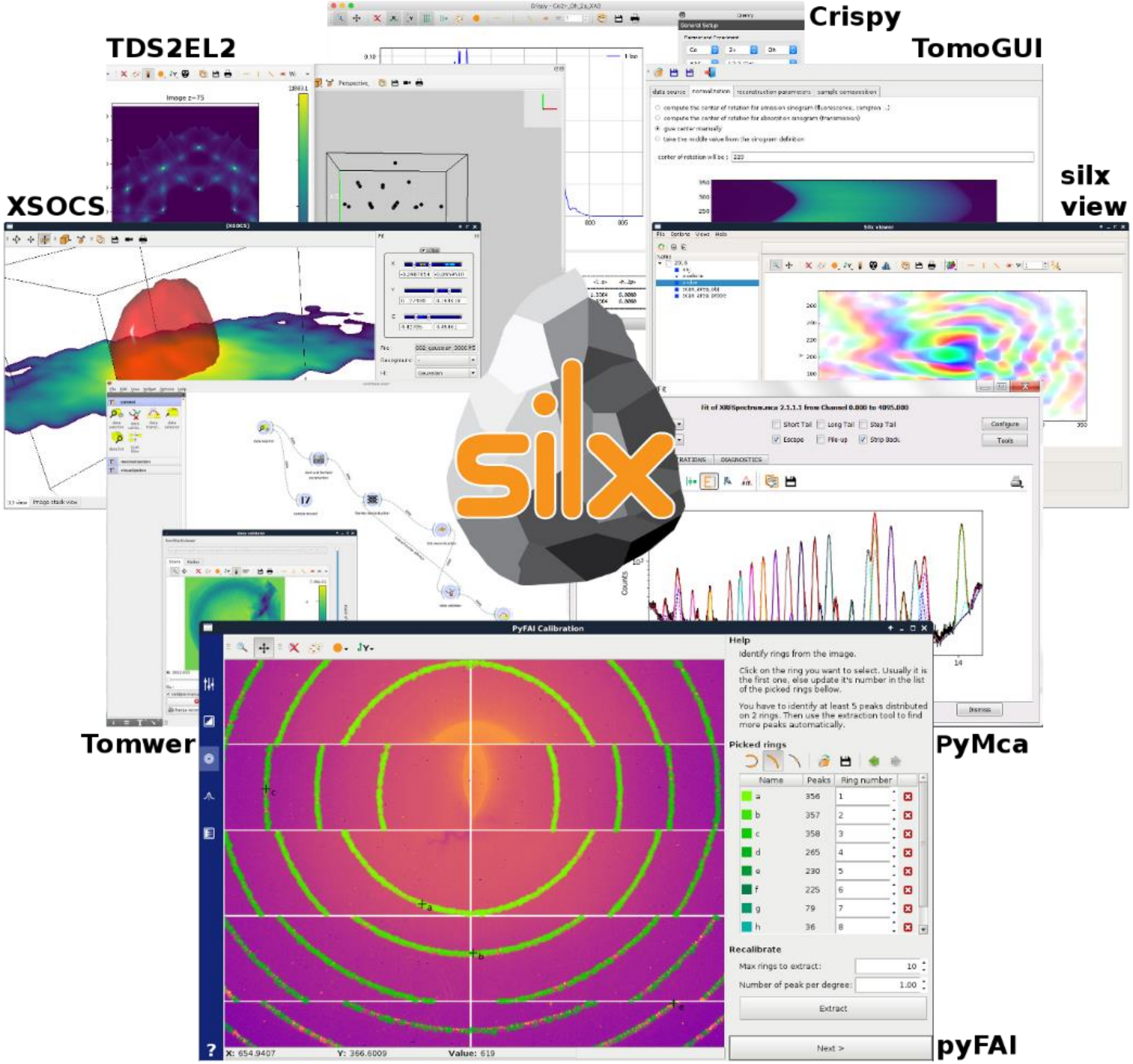


Silx view

- Drag & drop data as silx URL
- As `text/plain` and `application/x-silx-uri`

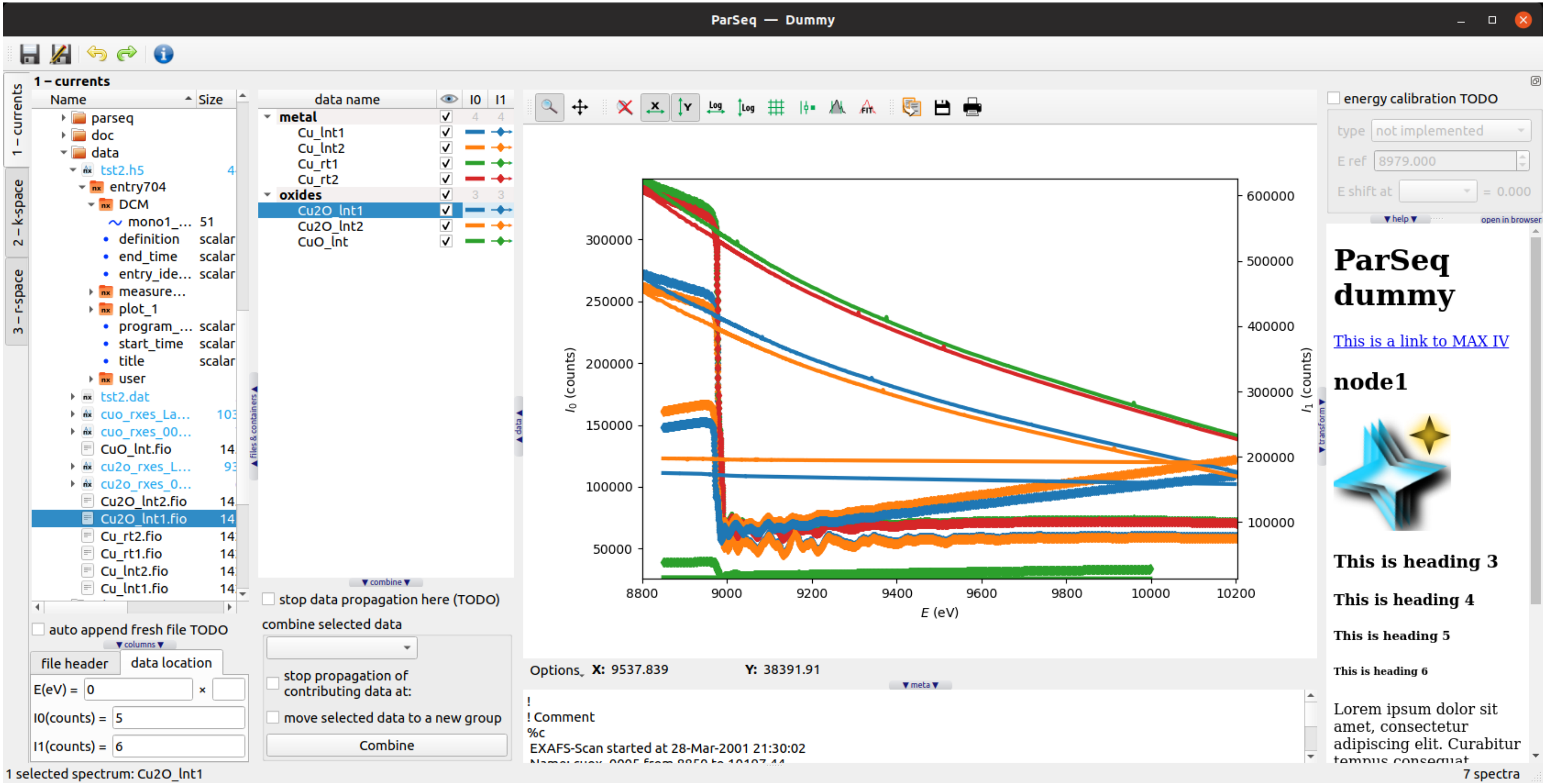


Applications – Growing family of applications using silx



Applications - ParSeq

https://github.com/kklmn/ParSeq



Role of Non-core developers

- Identify something you are interested on
- Try to achieve it
- Wow! I can do what I want, what next?
  - Start again
  - Make suggestions
  - Contribute with a demo/recipe
- I cannot do it
  - Ask help

Role of core developers

- Help non-core developers
- Create issues
  - Bugs
  - Documentation
  - Desired features
- Fix issues
  - Bugs
  - Documentation
  - Unlikely for new features
- Review pull requests

Hands on!

- Try to start with a single entry point www.silx.org
  - You should be able to install 0.10.1 version
- For this code camp we'll use 0.11.0a, you can either:
  - clone the repository (and use your compilation chain)
  - install a nightly built package (debian)
  - use a pre-built binary wheel:
    - https://silx.gitlab-pages.esrf.fr/bob/silx/