Hdf5Widget

November 14, 2016

```
In [ ]: %gui qt
```

1 Required knowledges

- `numpy` (basic knowledge)
- `Qt` (basic knowledge)
- `h5py` (optionally)

2 Useful silx resources

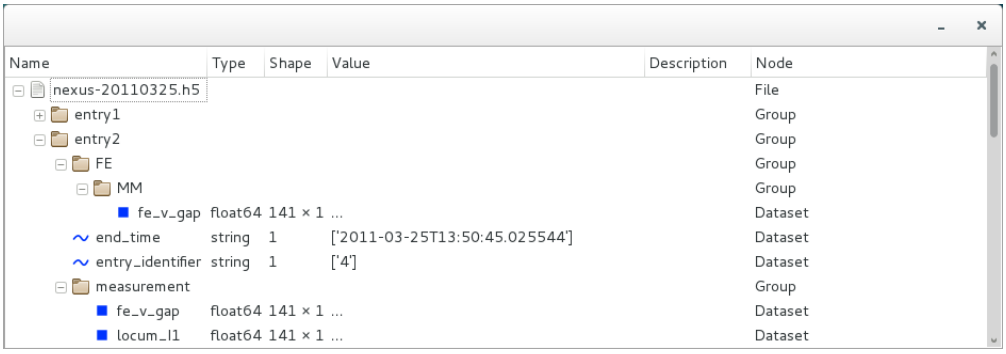
- Getting start with the Hdf5 (http://pythonhosted.org/silx/modules/gui/hdf5/getting_started.html)

3 Exercises summary

1. Features provided by Hdf5TreeView
   - Learn how to create an `Hdf5TreeView`
2. Create a HDF5 viewer
   - Learn how to use a dataset displayed by the tree
3. Create a tool to aggregate dataset and to create a diffraction mask
   - Use multi-selection node of the tree
   - Use features of the mask
4. Create a phase contrast viewer
   - Learn how to add context menu to the tree nodes

4 Features provided by Hdf5TreeView

```
from silx.gui import qt
from silx.gui import hdf5
app = qt.QApplication([])
tree = hdf5.Hdf5TreeView()
```



HDF5 Tree

```
tree.setVisible(True)
app.exec_()
```

4.1 Exercise 0

1. Execute this script
2. Drag and drop an HDF5 file and play with it

4.2 Exercise 1

You can use `exercices/ex1_display.py` as skeleton

1. Create an application to load HDF5 file provided on the command line
   - Use getting started with HDF5 widgets

```
for filename in filenames:
    #
    # TODO: Load each filename into the model tree
    #
    print("Load %s" % filename)
```

4.3 Solution

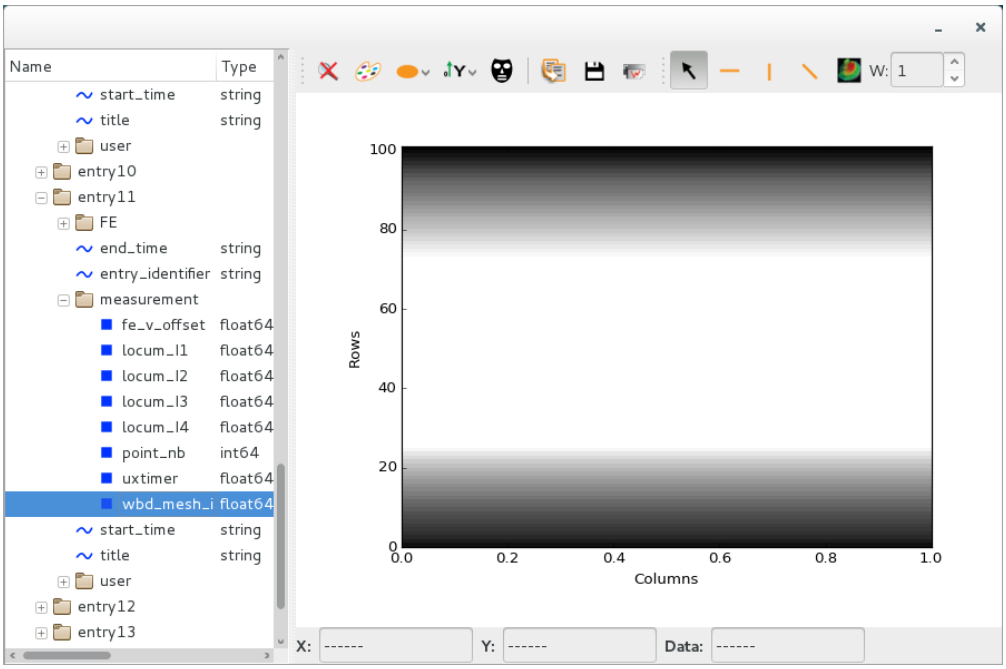
```
In [ ]: !./working_examples/display.py nexus-20110325.h5
```

5 Create an HDF5 viewer

This exercise you how to use the `Hdf5TreeView` to browse and display datasets. We provide a `DataViewer` widget to help you to display the data.

5.1 DataViewer class

We provide a `DataViewer` widget, to display data using `Silx` plots.



HDF5 viewer

```
In [ ]: from silx.gui import plot

        class DataViewer(qt.QStackedWidget):
            """Widget to display any kind of data"""

            def __init__(self, parent=None):
                """Constructor"""
                super(DataViewer, self).__init__(parent)

                self.__plot1d = plot.Plot1D()
                self.__plot2d = plot.Plot2D()
                self.__text = qt.QLabel()
                self.__text.setAlignment(qt.Qt.AlignCenter)

                self.__index1d = self.addWidget(self.__plot1d)
                self.__index2d = self.addWidget(self.__plot2d)
                self.__indexText = self.addWidget(self.__text)
                self.setCurrentIndex(self.__indexText)

            def showAsString(self, data):
                """Display a data using text"""
                self.__text.setText(str(data))
                self.setCurrentIndex(self.__indexText)

            def show1d(self, data):
                """Display a data using silx Plot1D"""
```

```
            self.__plot1d.clear()
            self.__plot1d.addCurve(legend="data", x=range(len(data)), y=data)
            self.setCurrentIndex(self.__index1d)

        def show2d(self, data):
            """Display a data using silx Plot2D"""
            self.__plot2d.clear()
            self.__plot2d.addImage(legend="data", data=data)
            self.setCurrentIndex(self.__index2d)

        def show(self, data):
            """Display a data using the widget which fit the best"""
            isAtomic = len(data.shape) == 0
            isCurve = len(data.shape) == 1 \
                            and numpy.issubdtype(data.dtype, numpy.number)
            isImage = len(data.shape) == 2 \
                            and numpy.issubdtype(data.dtype, numpy.number)
            if isAtomic:
                self.showAsString(data)
            elif isCurve:
                self.show1d(data)
            elif isImage:
                self.show2d(data)
            else:
                self.showAsString(data)
```

Here is an example of use.

```
In [ ]: import numpy
        viewer = DataViewer()
        viewer.setVisible(True)

In [ ]: # To display an image
        viewer.show(numpy.random.rand(100, 100))

In [ ]: # or a curve
        viewer.show(numpy.random.rand(100))

In [ ]: # or a value
        viewer.show(numpy.random.rand(1)[0])
```

5.2 Viewer class

We also provide a `Viewer` class. This class display together an `Hdf5TreeView` and a `DataViewer`.

```
In [ ]: window = qt.QSplitter()
        tree = hdf5.Hdf5TreeView(window)
        viewer = DataViewer(window)
```

```
window.addWidget(tree)
window.addWidget(viewer)
window.setStretchFactor(1, 1)
window.setVisible(True)
```

5.3 Exercise 2

You can use `exercises/ex2_viewer.py` as skeleton

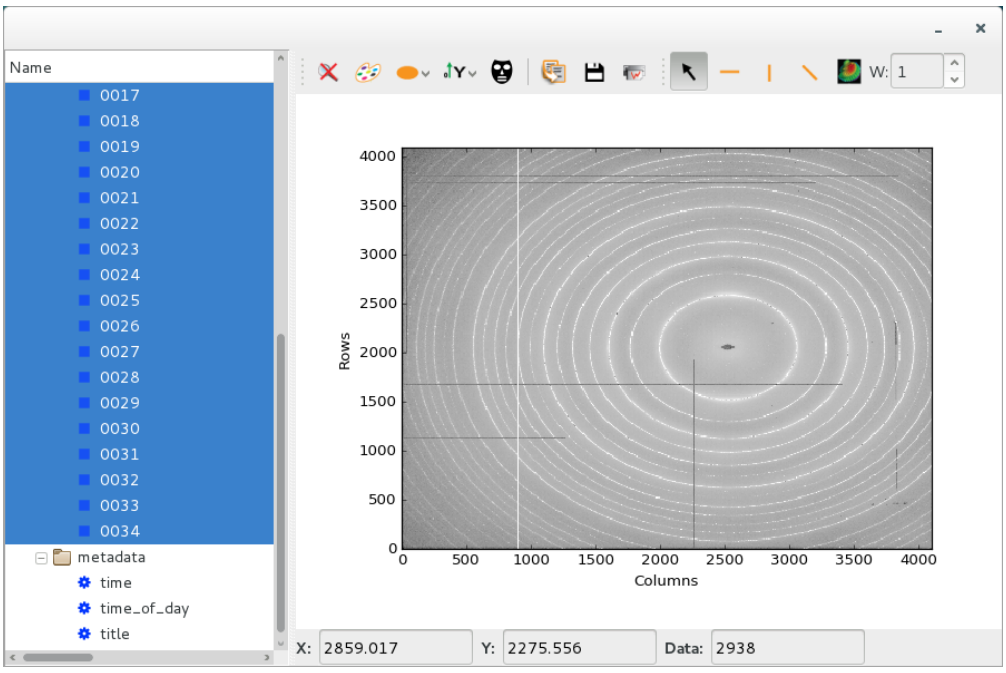
1. Connect the tree to the viewer together
   - Use getting started with HDF5 widgets

5.4 Solution

```
In [ ]: !./working_examples/viewer.py nexus-20110325.h5
```

6 Create an aggregation from diffraction acquisition

This exercise show how to configure and use the `Hdf5TreeView` with multi-selection. It will be used to compute an aggregation on images. The use case is an aggregation of diffraction acquisitions in order to create a better mask.



HDF5 diffraction mask

6.1 Creating an aggregation

A sum of many images can be done like that with `numpy`. It is not the better way to have the best contrast for a diffraction mask, but is is enough for this exercice.

```
In [2]: import numpy
        a = numpy.random.rand(5, 5)
        b = numpy.random.rand(5, 5)
        c = numpy.random.rand(5, 5)
        aggregate = numpy.sum([a, b, c], axis=0)
```

6.2 Exercise 3

You can use `exercises/ex3_diffraction_mask.py` as skeleton

1. Configure the tree as multi-selectable
   - Use `QAbstractItemView` documentation
2. Aggregate selected datasets on `onTreeActivated`
3. Show the result in the viewer
4. With the GUI, use the mask tool to create a mask from aggregated images

6.3 Solution

```
In [ ]: !./working_examples/diffraction_mask.py ID22_ma2909_Ti37Nb_450_72h_1.h5
```

7 Create an phase contrast viewer

This exercice show how to use the `Hdf5TreeView` context menu to a custom use. The use case is the phase contrast acquisition, in order to display better images from the raw data. To correct this images, we have to remove a background and apply a flat field. We can use the context menu to identify this dataset from an HDF5 file. The exercice provides few functions to help the computation.

7.1 Provided functions

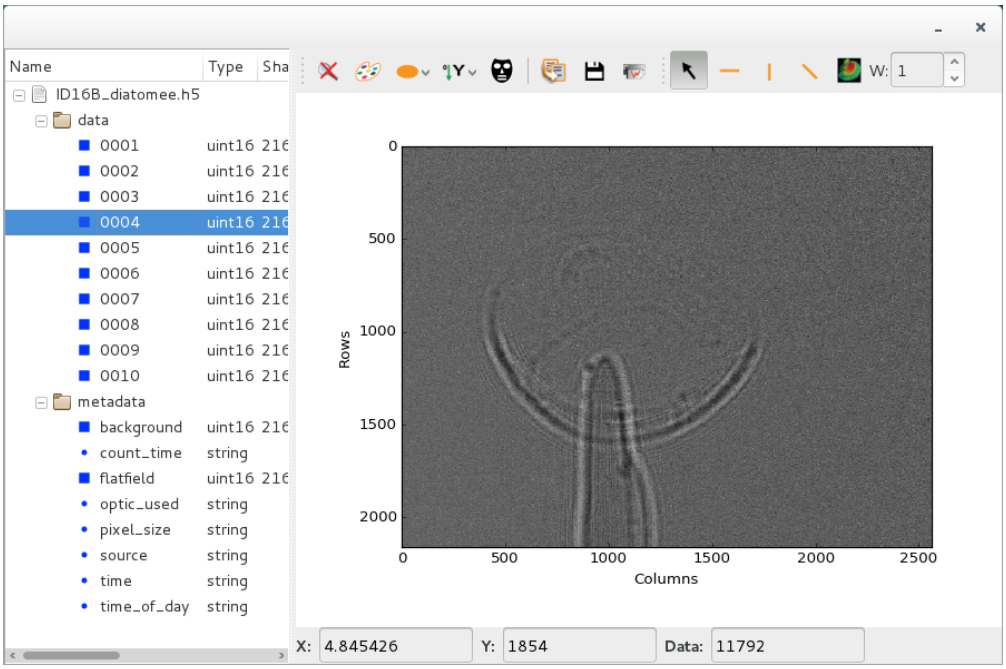
The computation of corrected images is done using this equation using `raw`, `flatfield`, and `background` information.

$$corrected = \frac{raw - background}{flatfield - background}$$

```
In [ ]:     def computeCorrectedImage(self, raw):
                if self.flatfield is None:
                    raise RuntimeError("Flatfield is not defined")
                if self.background is None:
                    raise RuntimeError("Background is not defined")

                raw = numpy.array(raw, dtype=numpy.float32)
                flatfield = numpy.array(self.flatfield.value, dtype=numpy.float32)
                background = self.background.value
                return (raw - background) / (flatfield - background)

            def setBackground(self, dataset):
```



HDF5 phase contrast viewer

```
        self.background = dataset

    def setFlatField(self, dataset):
        self.flatfield = dataset
```

7.2 Exercise 4

You can use `exercises/ex4_phase_contrast.py` as skeleton

1. Register a callback function for the context menu of the tree
   - Use getting started with HDF5 widgets
2. Create action to the menu to use the hovered dataset as backround of flatfield
   - Use getting started with HDF5 widgets
3. Try to compute the corrected image when an image is selected in the tree and show it in the viewer

7.3 Solution

```
In [ ]: !./working_examples/phase_contrast.py ID16B_diatomee.h5
```