# io

March 13, 2017

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

## 1 Required knowledges

- `numpy` (basic knowledges)
- `Qt` (basic knowledges)
- `h5py` (basic knowlegdes)

## 2 Silx IO API

```
In [ ]: import silx.io
```

### 2.1 Open a file

```
In [ ]: # open
        obj = silx.io.open("data/test.h5")
        # do your stuff here
        obj.close()
```

### 2.2 Open a file with context manager

```
In [ ]: # or using context manager
        with silx.io.open("data/test.h5") as obj:
            # do your stuff here
            # the close is called for you at the end of the with
            pass
```

### 2.3 Common properties

```
In [ ]: obj = silx.io.open("data/test.h5")

        # get the node path
        obj.name

        # test object type
        if silx.io.is_file(obj):
            # this is a root file
```

```
        # path of the file from the file system
        obj.filename

    if silx.io.is_group(obj):
        # this is a group
        # BTW a file is a group
        pass

    if silx.io.is_dataset(obj):
        # this is a dataset\
        pass
```

### 2.4 Node traversal

```
In [ ]: obj = silx.io.open("data/test.h5")

    if silx.io.is_group(obj):
        # it can contains child

        # number of child
        len(obj)

        # iterator on child names
        obj.keys()

        # access to a child
        child = obj["arrays"]

        # access to a child using a path
        child = obj["arrays/float_3d"]

        # the path can be absolute
        child = obj["/arrays/float_3d"]
```

### 2.5 Data access

```
In [ ]: h5 = silx.io.open("data/test.h5")
        obj = h5["arrays/float_3d"]

    if silx.io.is_dataset(obj):
        # it contains data

        # a dataset provides information to the data
        obj.shape    # multidimensional shape
        obj.size     # amount of items
        obj.dtype    # type of the array

        # copy the full data as numpy array
```

```
        data = obj[...]

        # or a part of it (using numpy selector)
        data = obj[1:2, ::3, 2]

    scalar = h5["scalars/int64"]
    if silx.io.is_dataset(scalar):
        # scalar dataset have an empty shape
        assert scalar.shape == ()

        # special case to access to the value of a scalar
        data = scalar[()]
```

### 2.6 Spec file as HDF5

```
In [ ]: import silx.io

    h5like = silx.io.open('data/oleg.dat')

    # print available scans
    print(h5like['/'].keys())

    # print available measurements from the scan 94.1
    print(h5like['/94.1/measurement'].keys())

    # get data from measurement
    time = h5like['/94.1/measurement/Epoch']
    bpm = h5like['/94.1/measurement/bpmi']
    mca = h5like['/94.1/measurement/mca_0/data']
```

### 2.7 EDF file as HDF5

```
In [ ]: import silx.io.utils

    h5like = silx.io.open("data/ID16B_diatomee.edf")

    # here is the data as a cube using numpy array
    # it's a cube of images * number of frames
    data = h5like["/scan_0/instrument/detector_0/data"]
    # here is the first image
    data[0]
    print(data[0].shape)

    # groups containing datasets of motors, counters
    # and others metadata from the EDF header
    motors   = h5like["/scan_0/instrument/positioners"]
    counters = h5like["/scan_0/measurement"]
    others   = h5like["/scan_0/instrument/detector_0/others"]
```

```
print("motor names", list(motors.keys()))

# reach a motor named 'srot'
# it's a vector of values * number of frames
srot = motors["srot"]
# here is the monitor value at the first frame
srot[0]
print(srot[...])
```

# 3 Silx HDF5 tree


HDF5 tree screenshot

- Getting start with the HDF5 tree (http://pythonhosted.org/silx/modules/gui/hdf5/getting_started.html)

## 3.1 Create the widget

```
In [ ]: import silx.gui.hdf5
        tree = silx.gui.hdf5.Hdf5TreeView()
        tree.setVisible(True)
        model = tree.findHdf5TreeModel()
```

## 3.2 Feed it with an HDF5

```
In [ ]: h5 = silx.io.open("data/test.h5")
        model.insertH5pyObject(h5)
```

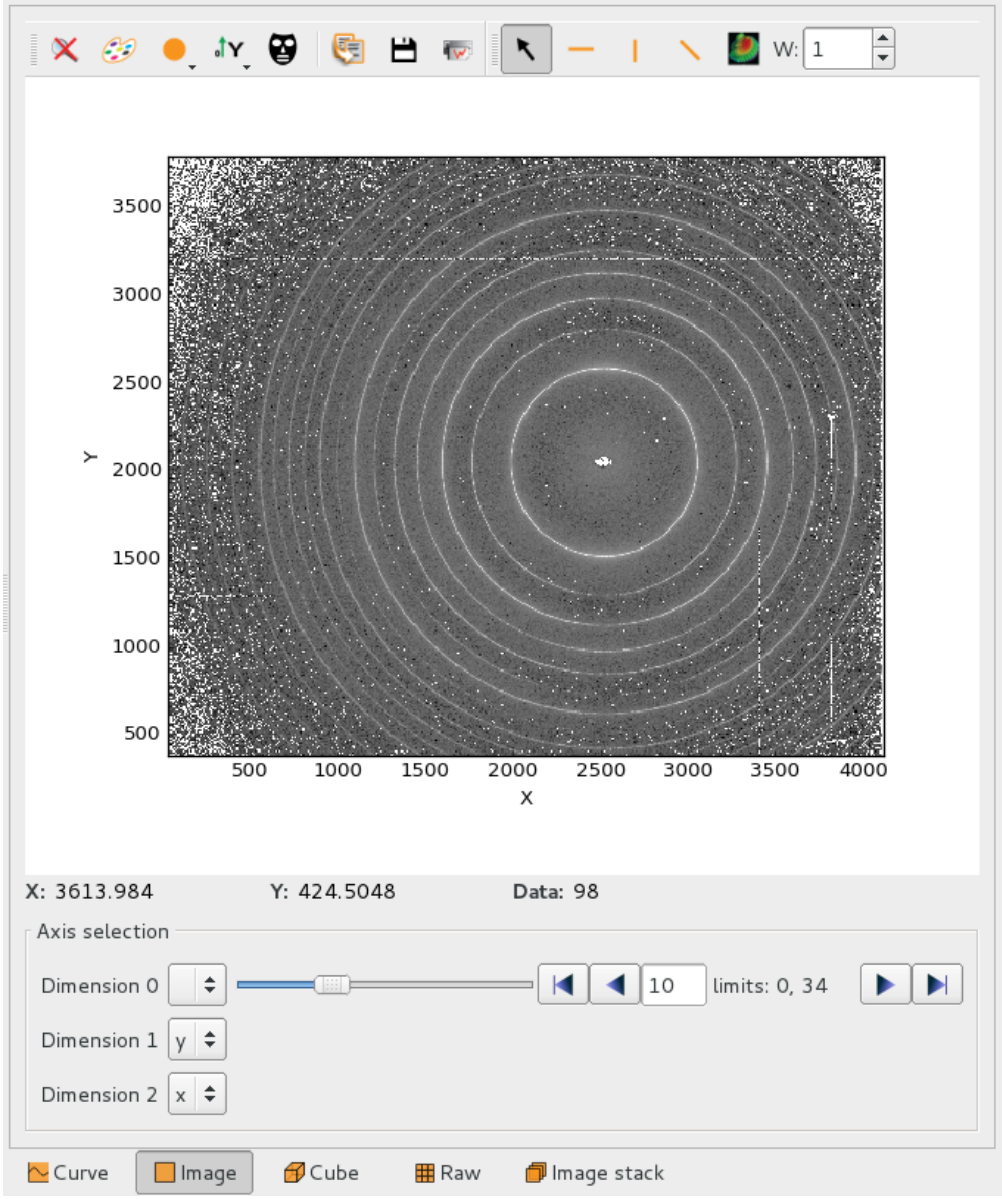
## 3.3 Feed it with a Spec file

```
In [ ]: h5 = silx.io.open("data/oleg.dat")
        model.insertH5pyObject(h5)
```

## 3.4 Feed it with an EDF file

```
In [ ]: h5 = silx.io.open("data/ID16B_diatomee.edf")
        model.insertH5pyObject(h5)
```

# 4 Silx DataViewer




DataViewer screenshot

## 4.1 Create the widget

```
In [ ]: import silx.gui.data.DataViewerFrame
        dataviewer = silx.gui.data.DataViewerFrame.DataViewerFrame()
        dataviewer.setVisible(True)
        dataviewer.resize(500, 500)
```

### 4.2 Feed it with a numpy array

```
In [ ]: import numpy
        data = numpy.random.rand(100, 100, 100)
        dataviewer.setData(data)
```

### 4.3 Feed it with a HDF5 dataset

```
In [ ]: import silx.io
        h5like = silx.io.open("data/ID16B_diatomee.h5")
        dataset = h5like["/data/0299"]
        dataviewer.setData(dataset)
```

## 5 Exercises

The exercise is based on a phase contast data. It will help you to create a custom application to browse data.

### 5.1 Exercise 1

- Browse an HDF5 file
- Use getting started with HDF5 widgets
- Identify path of the data
- Access to the data

```
In [ ]: import silx.io

        #
        # EXERCISE: Open the file 'data/ID16B_diatomee.h5'
        #

        h5 = ...

        #
        # EXERCISE: Display the file into the HDF5 tree
        #

        from silx.gui import hdf5
        tree = hdf5.Hdf5TreeView()
        model = tree.findHdf5TreeModel()
        ...
        tree.setVisible(True)
```

```
In [ ]: #
        # EXERCISE: Access to one frame of the image
        #

        print(...)
```

```
#
# EXERCISE: Display it with the data viwer
#
import silx.gui.data.DataViewerFrame
viewer = silx.gui.data.DataViewerFrame.DataViewerFrame()
...
viewer.setVisible(True)
```

### 5.2 Exercise 2

1. From the HDF58 tree, identify path name for
   - one data frame
   - one background
   - one flatfield
2. Compute flatfield correction

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

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

```
In [ ]: def correctedImage(data, background, flatfield):
            data = numpy.array(data, dtype=numpy.float32)
            flatfield = numpy.array(flatfield, dtype=numpy.float32)
            return (data - background) / (flatfield - background)

        #
        # EXERCISE: Reach one data frame, a background and a flatfield from 'data/
        #

        ...

        #
        # EXERCISE: Compute the corrected image
        #

        ...

        #
        # EXERCISE: Display it with the data viewer
        #

        ...
```

### 5.3 Exercise 3

1. Connect together an HDF5 tree view and a data viewer

- Use getting started with HDF5 widgets

In [ ]:
```
from silx.gui import qt
from silx.gui import hdf5
import silx.gui.data.DataViewerFrame


class ViewerEx3(qt.QMainWindow):

    def __init__(self, parent=None):
        qt.QMainWindow.__init__(self, parent)
        widget = self.createCentralWidget()
        self.setCentralWidget(widget)

    def createCentralWidget(self):
        splitter = qt.QSplitter(self)

        # the tree
        self.tree = silx.gui.hdf5.Hdf5TreeView(self)
        # the data viewer
        self.viewer = silx.gui.data.DataViewerFrame.DataViewerFrame(self)

        splitter.addWidget(self.tree)
        splitter.addWidget(self.viewer)
        splitter.setStretchFactor(1, 1)

        #
        # EXERCISE: Connect the callback onTreeActivated (bellow)
        #           to a mouse event from the tree
        #

        return splitter

    def onTreeActivated(self):

        #
        # EXERCISE: Reach selected objects from the tree
        #

        #
        # EXERCISE: Provide it to the data viewer
        #
        pass

    def appendFile(self, filename):
        model = self.tree.findHdf5TreeModel()
        model.insertFile(filename)
        print("Load %s" % filename)
```

```
In [ ]: viewer = ViewerEx3()
        viewer.appendFile('data/ID16B_diatomee.h5')
        viewer.setVisible(True)
```

### 5.4 Exercise 4

1. Use the previous application to display corrected data

```
In [ ]: class ViewerEx4(ViewerEx3):

            def onTreeActivated(self):
                selectedObjects = list(self.tree.selectedH5Nodes())
                if len(selectedObjects) == 0:
                    self.viewer.setData("Nothing selected")

                elif len(selectedObjects) > 1:
                    self.viewer.setData("Too much things selected")

                else:
                    obj = selectedObjects[0]
                    node = obj.h5py_object

                    if "/data/" in node.name:
                        # That's a data from the /data group
                        data = self.computeCorrectedImage(node)
                        self.viewer.setData(data)
                    else:
                        # Other data is displayed in a normal way
                        self.viewer.setData(obj)

            def computeCorrectedImage(self, h5data):
                """
                :param h5data: H5py dataset selected from the group /data/
                """
                background = self.getBackground(h5data)
                flatfield = self.getFlatField(h5data)

                raw = numpy.array(h5data, dtype=numpy.float32)
                flatfield = numpy.array(flatfield, dtype=numpy.float32)
                background = background[...]
                return (raw - background) / (flatfield - background)

            def getBackground(self, h5data):
                """
                :param h5data: H5py dataset selected from the group /data/
                """

                #
```

```
            # EXERCISE: Return the background image from the dataset
            #

            return None

        def getFlatField(self, h5data):
            """
            :param h5data: H5py dataset selected from the group /data/
            """

            #
            # EXERCISE: Return the flatfield image from the dataset
            #
            #           1) you can return a flatfield by default
            #           2) you can return the closest flat field according to t
            #           3) you can return an interpolation of the 2 flatfields

            return None
```

```
In [ ]: viewer = ViewerEx4()
        viewer.appendFile('data/ID16B_diatomee.h5')
        viewer.setVisible(True)
```