How to access HDF5 data from Python

Objective

Analysis & Applications group works on PSANA project - generic framework for analysis of any experimental data. Though this framework is going to be universal, most likely it will not be simple. In this page we discuss a simple but flexible approach to analysis of data stored in HDF5 files. It is based on Python code with extensive exploitation of standard libraries. A few code examples of how to access and process data are presented at the end of this page.

There are obvious advantages in this approach:

- Flexibility; HDF5 file has indexed structure, that means direct access to any data of any file from your code.
- Python is a high-level scripting language allows to write transparent and compact code based on well-elaborated standard libraries.
- In general, code in Python works slow comparing to C++, but there are libraries like NumPy written on C++, which solve this problem for manipulation with large arrays.

There is a couple of drawbacks in this approach,

- you have to know or learn Python
- current version of the `h5py` library works quite slow with long HDF5 files

The first issue about Python is not really a drawback. Basic concept of this high-level language can be learned from scratches for about a couple of days. In a week you will feel yourself as an expert and will enjoy programming on this powerful language. Second issue about slow `h5py` library is really annoying, but we hope that authors will account for our comments and its performance can be improved soon.

Below we assume that everything is setup to work on LCLS analysis farm, otherwise see Computing (including Analysis) and Account Setup.

Libraries

Here is a list of Python libraries which we use in examples below:

- h5py
- NumPy
- matplotlib

These libraries can be imported in the top of the `Python`-code file, for example

```
#!/usr/bin/env python
import h5py
import numpy as np
import matplotlib.pyplot as plt
```

HDF5 file structure

Detailed description of the HDF5 file structure can be found in HDF5 or h5py web sites. Briefly speaking, its structure resembles the file system directory tree. The top level of the HDF5 tree is a file; file may contain groups and datasets; each group may contain other groups and datasets; each dataset contains the data objects, which in most cases can be associated with `NumPy` types. Group and file may also have additional parameters, which are called as attributes. So, there are three basic type of items in HDF5 file: File, Group, and Dataset. Their names are used as access keys.

Basic operations

Basic operations allows to access the dataset records from HDF5. Here we assume that user knows explicitly the names of file and dataset and event number, for example

```
hdf5_file_name = '/reg/d/psdm/XPP/xppcom10/hdf5/xppcom10-r0546.h5'
dataset_name   = '/Configure:0000/Run:0000/CalibCycle:0000/Camera::FrameV1/XppSb4Pim.1:Tm6740.1/image'
event_number   = 5
```

- Open file, get dataset, get array for current event, and close file:

```
file    = h5py.File(hdf5_file_name, 'r')   # 'r' means that hdf5 file is open in read-only mode
dataset = file[dataset_name]
arr1ev  = dataset[event_number]
file.close()
```

The `arr1ev` is a `NumPy` object. There are many methods which allow to manipulate with this object. For example, one can

- print array shape and content:

```
print 'arr1ev.shape =', arr1ev.shape
print 'arr1ev =\n',     arr1ev
```

Advanced operations

As in previous case we assume that all necessary parameters are defined,

```
file = h5py.File(hdf5_file_name, 'r')
item = file[item_name]
```

where `item` stands for file, group of dataset.

Check if the HDF5 item is "File", "Group", or "Dataset"

```
isFile    = isinstance(item, h5py.File)
isGroup   = isinstance(item, h5py.Group)
isDataset = isinstance(item, h5py.Dataset)
```

In this example the standard `Python` method `isinstance(...)` returns `True` or `False` in each case, respectively.

Get information about HDF5 item

- For all HDF5 items:
  these parameters are available:

  ```
  item.id      # for example: <GroupID [1] (U) 33554473>
  item.ref     # for example: <HDF5 object reference>
  item.parent  # for example: <HDF5 group "/Configure:0000/Run:0000/CalibCycle:0000" (5 members)>
  item.file    # for example: <HDF5 file "cxi80410-r0587.h5" (mode r, 3.5G)>
  item.name    # for example: /Configure:0000/Run:0000/CalibCycle:0000/Camera::FrameV1
  ```

- For Dataset

```
ds.dtype     # for example: ('seconds', '<u4'), ('nanoseconds', '<u4')]
ds.shape     # for example: (1186,)
ds.value     # for example: (1297610252L, 482193420L)
```

- Get item attributes for File or Group (if attributes available)
  In this example the `item` might be a group or file

```
item.attrs          # for example: <Attributes of HDF5 object at 230141696>
item.attrs.keys()   # for example: ['start.seconds', 'start.nanoseconds']
item.attrs.values() # for example: [1297608424L, 627075857L]
len(item.attrs)
```

For example, one of the file attributes is a run number,

```
run_number = file.attrs['runNumber']
```

- Get the list of daughters in the `group`

```
list_of_item_names = group.items()
print list_of_item_names
```

  or convert the group in dictionary and iterate over their key and values,

```
for key,val in dict(group).iteritems():
    print key, val
```

Extract time

Time variable is stored in HDF5 as a tuple of two long integer numbers representing the **seconds** since 01/01/1970 and **nanoseconds** as a fraction of the second. Time is usually stored in the group attributes and/or in the data record with name "time", which can be extracted as shown below

- from the group attributes

```
group = file["/Configure:0000"]
time_sec  = group.attrs.values()[0]
time_nsec = group.attrs.values()[1]
```

- from the `time` data record

```
time_dataset = file['/Configure:0000/Run:0000/CalibCycle:0002/Acqiris::DataDescV1/XppLas.0:Acqiris.0
/time']

index = 0                   # this is an index in the dataset
time_arr = time_dataset[index]  # get the time tuple consisting of seconds and nanoseconds
time_sec  = time_arr[0]
time_nsec = time_arr[1]
```

Operations with CSPad pedestals

Most generic way to subtract the CSPad pedestals is to use Translator, as described in CsPad calibration in translator. If calibration is requested in the Translator the output HDF5 file has the CSPad image data with already subtracted pedestals. Otherwise, Translator saves raw CSPad data in HDF5 file. If the job execution time is not an issue, the pedestals can be subtracted from raw data directly in code, as explained in this section.

How to find the files with CSPad pedestals

CSPad pedestals are usually calibrated using the "dark" runs. If they were calibrated, the files for appropriate run range, `<run-range>.dat`, can be found in the directory
`/reg/d/psdm/<INSTRUMENT>/<experiment>/calib/<calib-version>/<source>/pedestals/`
If the pedestal file was available at translation time, the dataset
`/Configure:0000/CsPad::CalibV1/XppGon.0:Cspad.0/pedestals`
is saved in the HDF5 file and can be accessed directly.
One may prefer to calibrate and keep pedestal files in the local directory, as explained below.

How to calibrate CSPad pedestals

If the CSPad pedestals were not calibrated, they can be calibrated, as explained in
the description of the CsPadPedestals psana - Original Documentation module. Essentially, one need to run the `psana` for `cspad_mod.CsPadPedestals` module, using command
`psana -m cspad_mod.CsPadPedestals input-files.xtc`
which by default produce two files:

- `cspad-pedestals.dat` – for average values, and
- `cspad-noise.dat` – for standard deviation values.
  These files can be loaded in code as explained below.

Get CSPad pedestal array

The file with pedestal values can be read in code as a numpy array:

```
import numpy as np
ped_fname = '/reg/d/psdm/<INS>/<experiment>/calib/<calib-version>/<source>/pedestals/<run-range>.dat'
ped_arr = np.loadtxt(ped_fname, dtype=np.float32)
ped_arr.shape = (32, 185, 388) # raw shape is (5920, 388)
```

In this example the pedestal file is loaded from the standard `calib` directory. For your own pedestal file the path name should be changed.

Subtract CSPad pedestals

Assuming that the CSPad event array `ds1ev` and the pedestal array `ped_arr` are available,
the pedestals can be subtracted by the single operation for numpy arrays:

```
if ds1ev.shape == ped_arr.shape : ds1ev -= ped_arr
```

> ⚠ This operation will only be valid if the CSPad data array is completely filled (all sensors are available) and its shape is equal to (32, 185, 388). Otherwise, the pedestal subtraction can be done in a loop over available sensors, taking into account the CSPad configuration.

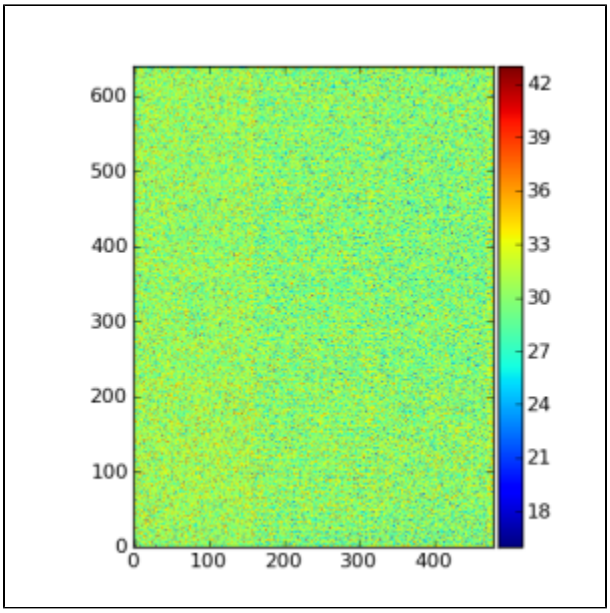
Code examples

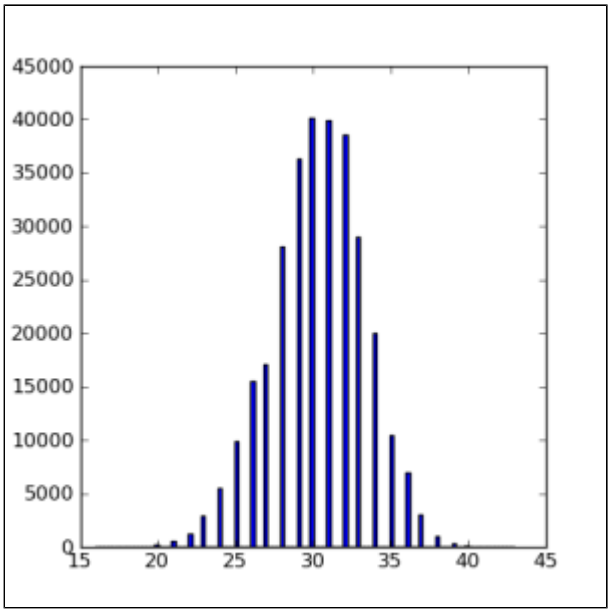
Example 1: Basic operations

```
#!/usr/bin/env python

import h5py
import numpy as np

eventNumber = 5

file    = h5py.File('/reg/d/psdm/XPP/xppcom10/hdf5/xppcom10-r0546.h5', 'r')
dataset = file['/Configure:0000/Run:0000/CalibCycle:0000/Camera::FrameV1/XppSb4Pim.1:Tm6740.1/image']
arr1ev  = dataset[eventNumber]
file.close()

print 'arr1ev.shape =', arr1ev.shape
print 'arr1ev =\n',     arr1ev
```

Similar code plots the dataset as image or histogram using the matplotlib library

```
#!/usr/bin/env python
import h5py
import numpy as np
import matplotlib.pyplot as plt

def plotImage(arr) :
    fig  = plt.figure(figsize=(5,5), dpi=80, facecolor='w',edgecolor='w',frameon=True)
    imAx = plt.imshow(arr, origin='lower', interpolation='nearest')
    fig.colorbar(imAx, pad=0.01, fraction=0.1, shrink=1.00, aspect=20)

def plotHistogram(arr) :
    fig  = plt.figure(figsize=(5,5), dpi=80, facecolor='w',edgecolor='w',frameon=True)
    plt.hist(arr.flatten(), bins=100)

eventNumber = 5
file    = h5py.File('/reg/d/psdm/XPP/xppcom10/hdf5/xppcom10-r0546.h5', 'r')
dataset = file['/Configure:0000/Run:0000/CalibCycle:0000/Camera::FrameV1/XppSb4Pim.1:Tm6740.1/image']
arr1ev  = dataset[eventNumber]

plotImage(arr1ev)
plotHistogram(arr1ev)
plt.show()
file.close()
```





Example 2: Extract and print the time variables

```
#!/usr/bin/env python

import h5py
import time

#-----------------------------------------------------

def print_time(t_sec, t_nsec):
    """Converts seconds in human-readable time and prints formatted time"""

    tloc = time.localtime(t_sec) # converts sec to the tuple struct_time in local
    print 'Input time :',t_sec,'sec,',  t_nsec,'nsec, '
    print 'Local time :', time.strftime('%Y-%m-%d %H:%M:%S',tloc)

#-----------------------------------------------------
file_name = '/reg/d/psdm/xpp/xpp22510/hdf5/xpp22510-r0100.h5'
file = h5py.File(file_name, 'r') # open read-only

print "EXAMPLE: Get time from the group attributes:"

group = file["/Configure:0000"]
t_sec  = group.attrs.values()[0]
t_nsec = group.attrs.values()[1]
print_time(t_sec, t_nsec)

print "EXAMPLE: Get time from the data record 'time':"

dataset = file['/Configure:0000/Run:0000/CalibCycle:0002/Acqiris::DataDescV1/XppLas.0:Acqiris.0/time']
index = 0
time_arr = dataset[ind]
t_sec  = time_arr[0]
t_nsec = time_arr[1]
print_time(t_sec, t_nsec)

file.close()
#-----------------------------------------------------
```

Example 3: Print entire file/group structure using recursive method

```
#!/usr/bin/env python
import h5py
import sys

def print_hdf5_file_structure(file_name) :
    """Prints the HDF5 file structure"""
    file = h5py.File(file_name, 'r') # open read-only
    item = file #["/Configure:0000/Run:0000"]
    print_hdf5_item_structure(item)
    file.close()

def print_hdf5_item_structure(g, offset='    ') :
    """Prints the input file/group/dataset (g) name and begin iterations on its content"""
    if   isinstance(g,h5py.File) :
        print g.file, '(File)', g.name

    elif isinstance(g,h5py.Dataset) :
        print '(Dataset)', g.name, '    len =', g.shape #, g.dtype

    elif isinstance(g,h5py.Group) :
        print '(Group)', g.name

    else :
        print 'WORNING: UNKNOWN ITEM IN HDF5 FILE', g.name
        sys.exit ( "EXECUTION IS TERMINATED" )

    if isinstance(g, h5py.File) or isinstance(g, h5py.Group) :
        for key,val in dict(g).iteritems() :
            subg = val
            print offset, key, #,"   ", subg.name #, val, subg.len(), type(subg),
            print_hdf5_item_structure(subg, offset + '    ')

if __name__ == "__main__" :
    print_hdf5_file_structure('/reg/d/psdm/XPP/xppcom10/hdf5/xppcom10-r0546.h5')
    sys.exit ( "End of test" )
```

Example 4: Time-based syncronization of two datasets

```
#!/usr/bin/env python
import os
import sys
import h5py
import numpy as np

class TwoDatasetSynchronization ( object ) :
    """Matching elements of two datasets using their time stamps"""
    def __init__ ( self, file, Xdsname, Ydsname ) :
        """Initialization"""

        self.dsX        = file[Xdsname]
        self.dsY        = file[Ydsname]
        XTimedsname     = get_item_path_to_last_name(Xdsname) + '/time'
        YTimedsname     = get_item_path_to_last_name(Ydsname) + '/time'
        self.dsXT       = file[XTimedsname]
        self.dsYT       = file[YTimedsname]
        self.XTarr      = 0.000000001 * self.dsXT['nanoseconds'] + self.dsXT['seconds']
        self.YTarr      = 0.000000001 * self.dsYT['nanoseconds'] + self.dsYT['seconds']
        self._nXpoints  = self.dsX.shape[0]
        self._nYpoints  = self.dsY.shape[0]
        self._indX      = 0
        self._indY      = 0
        self._tmapXlist = []
        self._tmapYlist = []
        print 'Xdsname     =',Xdsname
        print 'Ydsname     =',Ydsname
```

```
        print 'XTimedsname =',XTimedsname
        print 'YTimedsname =',YTimedsname
        print 'Initialization: datasets X and Y have length =', self._nXpoints, self._nYpoints

    def twoDatasetSynchronizationIterations( self ) :
        """Iteration over time indexes and appending of syncronized arrays."""

        while self._indX < self._nXpoints and self._indY < self._nYpoints :

            if self.XTarr[self._indX] == self.YTarr[self._indY] :   # Time is the same
                self._tmapXlist.append(self.dsX[self._indX])
                self._tmapYlist.append(self.dsY[self._indY])
                self._indX += 1
                self._indY += 1

            elif self.XTarr[self._indX] > self.YTarr[self._indY] :  # Time X > Time Y
                self._indY += 1
                self.printMissingSynchronization()

            else :                                                  # Time X < Time Y
                self._indX += 1
                self.printMissingSynchronization()

    def printMissingSynchronization( self ) :
        print 'Missing of syncronization for X,Y indexes ',self._indX,self._indY

    def runSynchronization( self ) :
        """Executes synchronization and makes the references for synchronized arrays."""
        self.twoDatasetSynchronizationIterations()
        self.Xarr = np.array(self._tmapXlist)
        self.Yarr = np.array(self._tmapYlist)
        print 'Number of synchronized in time X and Y array elements =', self.Xarr.shape, self.Yarr.shape

def get_item_path_to_last_name(dsname):
    """Returns the path to the last part of the item name"""
    path,name = os.path.split(str(dsname))
    return path

def main() :
    """EXAMPLE: Time synchronization of two datasets.

    In this example we open the file, which contains correct dataset "Y" and the dataset with lost records "X".
    We access these arrays and associated time arrays through the class TwoDatasetSynchronization.
    Then we iterate over indexes of these arrays and append the lists of syncronized arrays.
    Program prints the message in case of missing synchronization.
    """

    file     = h5py.File('/reg/d/psdm/CXI/cxi80410/hdf5/cxi80410-r0730.h5', 'r')
    Xdsname  = '/Configure:0000/Run:0000/CalibCycle:0000/Bld::BldDataFEEGasDetEnergy/NoDetector.0:NoDevice.2
/data'
    Ydsname  = '/Configure:0000/Run:0000/CalibCycle:0000/Ipimb::DataV1/CxiDg1.0:Ipimb.0/data'

    synchro  = TwoDatasetSynchronization (file, Xdsname, Ydsname)
    synchro.runSynchronization()

#--------------------------------
if __name__ == "__main__" :
    main()
    print('Exit')
    sys.exit ()
#--------------------------------
```