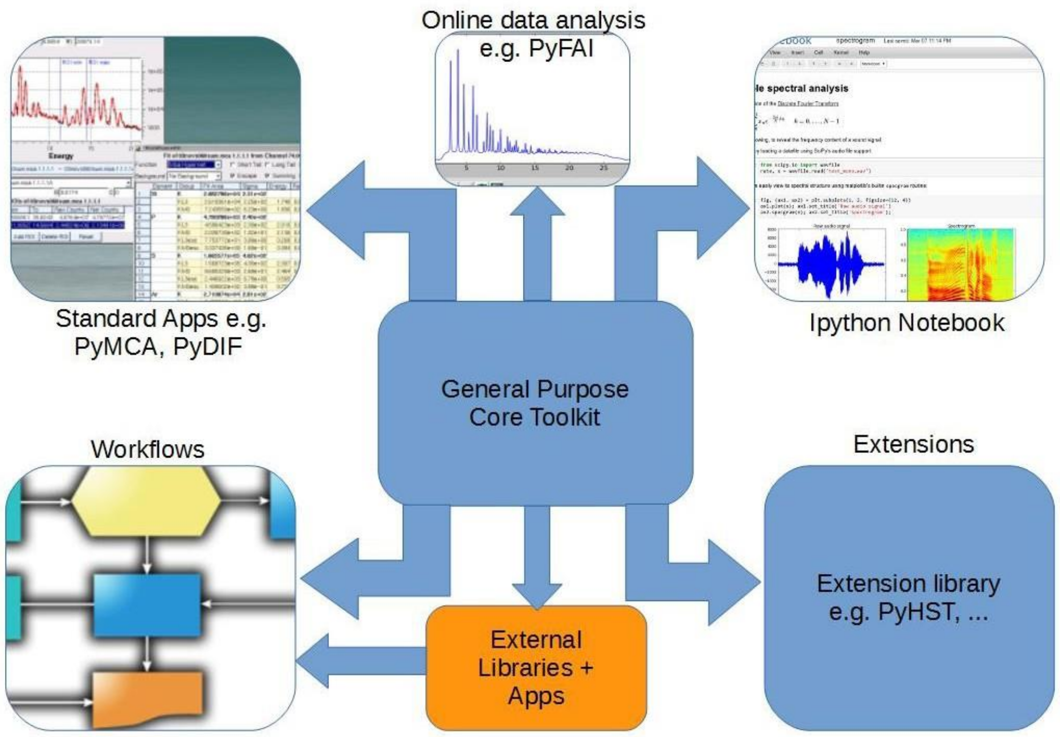
Data Analysis and Management Project
The silx library: IO



- Provide, document and maintain a set of tools for software development
- Train staff and users on their use
- Simple and accessible to scientists

Summary

- Introduction to ESRF data formats
  - SPEC file format (aka specfile)
  - ESRF Data Format (aka EDF)
  - HDF5
  - NeXus Convention
- Proposed Approaches for silex library
- Your feedback

SPEC file format

Until recently SPEC was the only acquisition sequencer used at the ESRF

- Its default data format is ASCII based
- SPEC provides functions to write other formats like EDF and HDF5

SPEC file layout

A complete SPEC file contains one file header and one or several scan headers

File header is introduced by the characters #F

Scan header starts by the characters #S and is preceded by a newline character : \n

Every line in the file ends by \n (newline character)

WARNING: No effort is made nor it is intended to be made to support \r\n ending

The SPEC file header

```
#F /mntdirect/_data_id03_inhouse/2006/Jul06/balmes/CuZnO/CuZnO_2.spec
#E 1145450191
#D Sat Jul 22 13:05:06 2006
#C sixcvertical  User = opid03
#O0  Delta  Theta  Chi  Phi  mu  Gamma  dummy  Mutrans
#O1  bver  bhor  xt  yt  zax  a2theta  ath  atran
#O2  delsl  gamsl  fs3  fs4  pssvo  pssvg  pssho  psshg
#O3  eh1_ao1  eh1_ao2
```

SPEC supports a space in names -> **Separation between names is two spaces.**

#F -> Original name of the file when generated

#E -> EPOCH

#D -> Date

#C -> Comment line, they can be anywhere in the file

#O… -> Names of all the motors in the SPEC session (in this case named sixcvertical

The SPEC scan header

```
#S 1  ascan  zax -0.73 -0.23  20 1
#D Sat Jul 22 13:11:07 2006
#T 1  (Seconds)
#G0 13 0 1 0.0007850334972 0.0006321396896 0.9999994921 0 0 0 0 0 0 600 0 0 1 143.6
#G1 3.25 3.25 5.207 90 90 120 2.232368448 2.232368448 1.206680489 90 90 60 1 1 2 -1 2 2 26.132 7.41 -
88.96 1.11 1.000012861 15.19 26.06 67.355 -88.96 1.11 1.000012861 15.11 0.723353 0.723353
#G3 0.06337923671 0.027529133 1.206191273 -1.43467075 0.7633438883 0.02401568018 -1.709143587 -
2.097621783 0.02456954971
#G4 0 0 0 0.723353 0 0 0 0 180 88.96 -1.11 1 0 0 0 0 0 0 0 0 0 -180 -180 -180 -180 -180 -180 0
#Q 0 0 0
#P0 0 0 -88.96 1.11 0 0 0 167
#P1 0 0 -1.5 -2 -0.73 5.25 7 0
#P2 2 2 -30 -60 0.22 0.2 5.12 0.2
#P3 0 0
#N 23
#L zax  H  K  L  Epoch  Seconds  det  attn  dtime  detc  ndetc  delcnt  thcnt  mucnt  gamcnt  tthcnt  inbeam
outbeam  eh1_ai1  eh1_ai1  SRcurr  phd3  mon
```

#S -> Command used

#D -> Date

#T -> Preset acquisition time

#G -> Diffractometer geometry parameters (if any)

#Q -> HKL (if enabled)

#P -> Position of all the motors at the beginning of the scan

#N and #L -> Number and names of all the labels associated to the following data

The SPEC data block

```
-0.73 0 0 0 8116476 1 246553 15 0.25528 6.1599787e+11 8962185.1 0 0 0 0 0 7.4479203e-08 0 0 0 86.223145 68733 625307
-0.705 0 0 0 8116478 1 246496 15 0.25625 6.1585546e+11 8955683.1 0 0 0 0 0 7.44975e-08 0 0 0 86.219574 68767 624960
-0.68 0 0 0 8116479 1 245717 15 0.253826 6.1390918e+11 8944680.1 0 0 0 0 0 7.4302001e-08 0 0 0 86.216827 68634 623641
-0.655 0 0 0 8116481 1 245946 15 0.25388 6.1448132e+11 8931804 0 0 0 0 0 7.4408902e-08 0 0 0 86.211464 68797 624420
-0.63 0 0 0 8116483 1 245795 15 0.254015 6.1410405e+11 8936842.3 0 0 0 0 0 7.4502097e-08 0 0 0 86.209305 68716 624237
-0.605 0 0 0 8116484 1 243979 15 0.251457 6.0956689e+11 8864880.1 0 0 0 0 0 7.4416498e-08 0 0 0 86.205849 68762 624042
-0.58 0 0 0 8116486 1 241508 15 0.249518 6.0339324e+11 8760573.2 0 0 0 0 0 7.4274503e-08 0 0 0 86.201073 68876 624377
-0.555 0 0 0 8116488 1 235118 15 0.240175 5.8742821e+11 8528655.6 0 0 0 0 0 7.4428797e-08 0 0 0 86.197807 68877 624321
-0.53 0 0 0 8116490 1 211161 15 0.211069 5.2757308e+11 7672228 0 0 0 0 0 7.4130902e-08 0 0 0 86.191887 68764 623696
```

Immediately follows the line containing the labels (line introduced by #L

Each line corresponds to one acquisition point

Only numbers are allowed (no NaN …)

Number of values must match number of labels

SPEC MCA Support

Special entries were added to specfile to handle saving of Multichannel Analyzer data

They start with the symbol @

Lines starting by @A in the data block denote the beginning of a set of MCA values

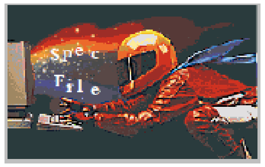
First channel is 0 unless specified by the #@CHANN key in the header

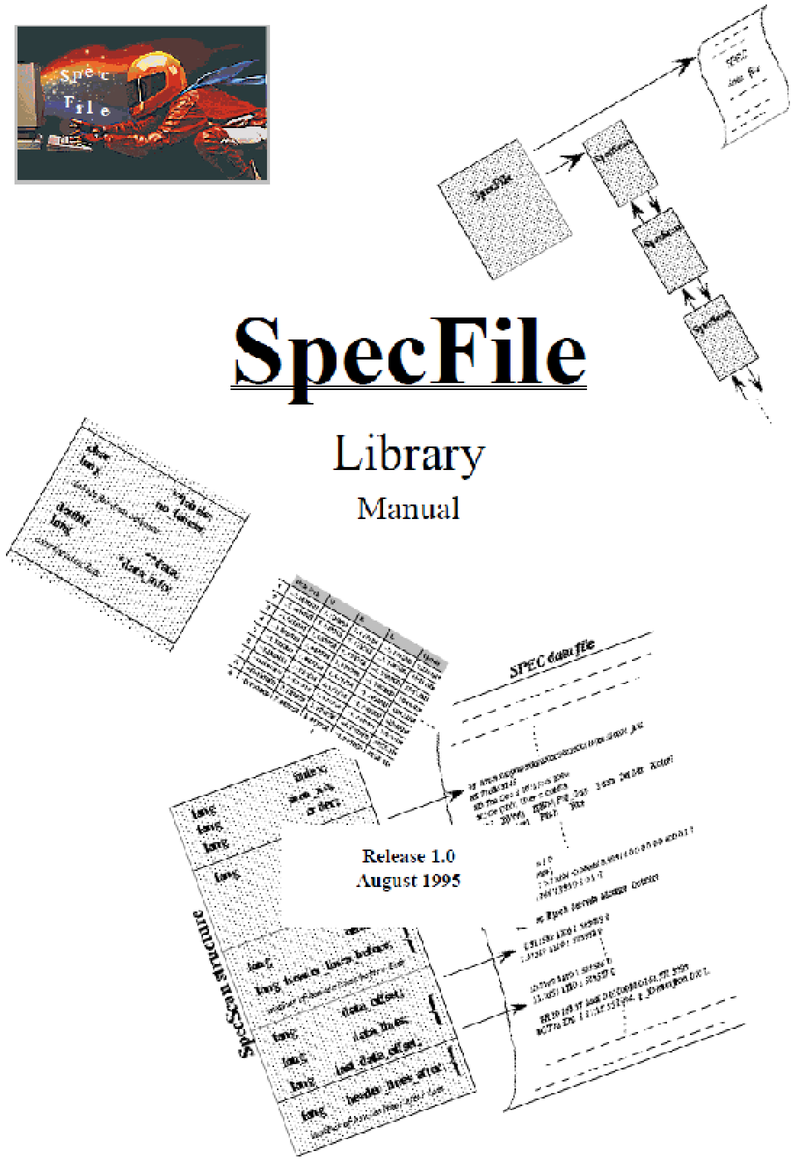
Calibration (in the form a + bx + c x^2) is found, if present, under #@CALIB

```
#F C:/titat/titat-2009/Xaloc-2013/Fluorescencia\ctrf6.mca
#D Mon Sep 01 12:57:10 2014

#S 1 Fluoscan_ctrf6-A.dat 2.1.1.2
#D Mon Sep 01 12:57:10 2014
#@MCA %16C
#@CHANN 1024 0 1023 1
#@CALIB -0.1025283 0.01976545 0
@A 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 3.0000 \
12.0000 37.0000 96.0000 265.0000 522.0000 870.0000 1268.0000 1569.0000 1574.0000 1335.0000
889.0000 549.0000 278.0000 107.0000 49.0000 12.0000 \
2.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 7.0000 23.0000 26.0000 26.0000
22.0000 30.0000 24.0000 \
29.0000 24.0000 26.0000 33.0000 24.0000 14.0000 17.0000 26.0000 21.0000 14.0000 17.0000 18.0000
21.0000 15.0000 19.0000 12.0000 \
….
1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000
1.0000 0.0000
```

The SpecFile library




SpecFile

Library

Manual

Release 1.0

August 1995

It is a C library with a Python binding

Mainly developed by Vicente Rey

Maintained by V. Armando Solé

Reading SPEC files in Python

```
from PyMca5.PyMca import specfilewrapper as specfile
sf = specfile.Specfile(filename)
sf.scanno()                  #returns the number of scans
sf.allmotors()               #return the motor names
scan = sf.select(‘1.1’)      #select the scan 1.1 if exists!!!
scan = sf[0]                 #select the first scan
dir(scan)                    #list the available methods
scan.alllabels()             #list the labels
scan.data()                  #retrieve all data
scan.nbmca()                 #returns the number of mca in the scan
scan.mca(1)                  #retrieve first mca
scan.header(‘’)              #return the complete scan header
scan.header(‘S’)             #return the specified line (#S) of the scan header
```

specfilewrapper provides access to other text based 1D formats using the same API

ESRF Data Format (EDF)

We have seen how SPEC file format was “pushed” to support 1D data …

The ESRF data format was originally intended to stock images

It consists of an ASCII header followed by binary data

That structure can be repeated as many times as images are stored in the file

The EDF Header

EDF headers must have a size that is a multiple of 512 bytes

The header starts by the sequence **{\n** and ends by the sequence **}\n**

In between those delimiters, we find strings following the pattern **Key = value ;\n**

In between those delimiters, we find strings following the pattern **Key = value ;\n**

Mandatory keys:

```
HeaderID = EH:000001:000000:000000 ;
Image = 1 ;
ByteOrder = LowByteFirst ;
DataType = FloatValue ;
Dim_1 = 256 ;
Dim_2 = 256 ;
Size = 67108864
```

EDF Data types and numpy

SignedByte	int8
UnsignedByte	uint8
SignedShort	int16
UnsignedShort	uint16
SignedInteger	int32
UnsignedInteger	uint32
SignedLong	int32
UnsignedLong	uint32
Signed64	int64
Unsigned64	uint64
FloatValue	float32
DoubleValue	float64
QuadrupleValue	float128

Additional conventions

BCU macros foresee additional keys for storing positioners and counters in the header

counter_pos = counter_value1 counter_value2 counter_value3 counter_value4 … ;\n
counter_mne = counter_name1 counter_name2 counter_name3 counter_name4 … ;\n
motor_pos = motor_value1 motor_value2 motor_value3 motor_value4 … ;\n
motor_mne = motor_name1 motor_name2 motor_name3 motor_name4 … ;\n

Raster scans can reduce the number of files using multiframe EDFs following a pattern

USER_DEFINED_ROOT_NAME_%04d_%04d_%04d.edf

Where the first index is the SCAN number, the second the detector number and the third the order number in the sequence of files.

Two very similar APIs available

The EdfFile module developed by the BCU.

The FabIO package developed as part of the Fable project (originally ID11 development)

Both approaches go beyond just reading EDF files.

They read several other image formats using the same API used to read EDF files.

EdfFile

- Its most basic implementation consists on a single file reading and writing EDF
- It does not provide converters to other formats

FabIO

- Supports a larger amount of image formats and provide converters

- A complete description is available at:
  http://dx.doi.org/10.1107/S0021889813000150

ESRF current situation: SPEC File Format

Advantages

- Simplicity (multiple column ASCII)
- Widespread
- Counters, Motors and MCA in same file

Disadvantages

- Not suited to large datasets (images)

ESRF current situation: ESRF Data Format

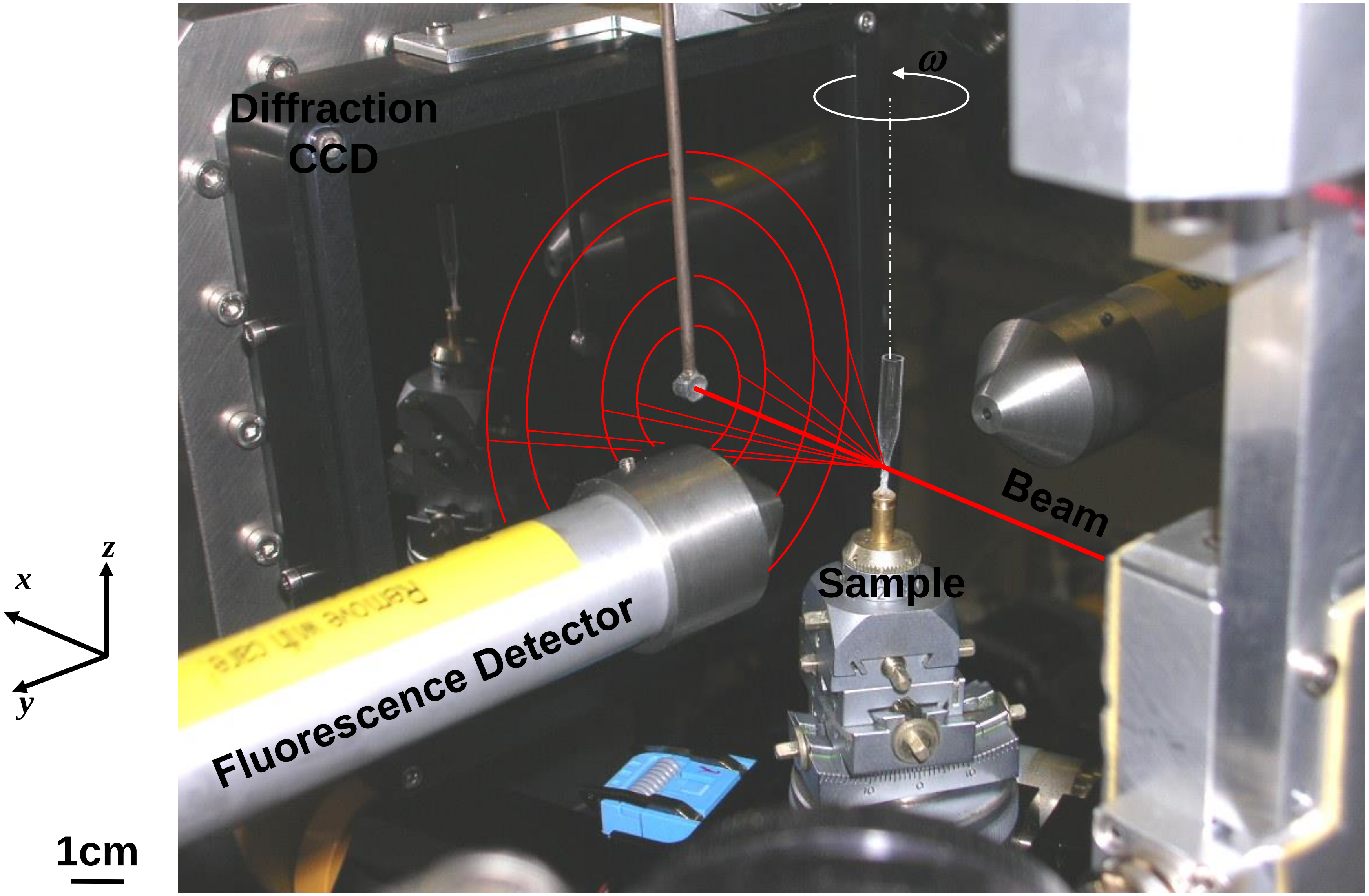
Advantages

- Suited to images

Disadvantages

- Not widespread (basically ESRF)
- Incomplete « official » metadata
- Not efficient for large datasets (parse N headers to read image N)

Fluorescence-Diffraction Tomography



Needs

Efficient format to store different data types

Keep together motors, counters, images, mca, …

Compression support

Editable

Widespread support

Efficient and easy access to the data for visualization and analysis

HDF5

HDF5

Hierarchical data format

Think about an HDF5 file as a portable file system (~ hard disk)

- You can write whatever you want to it
- It supports links
- It supports compression
- It is widely supported

A slide from The HDF Group

HDF5
Basic File Structure

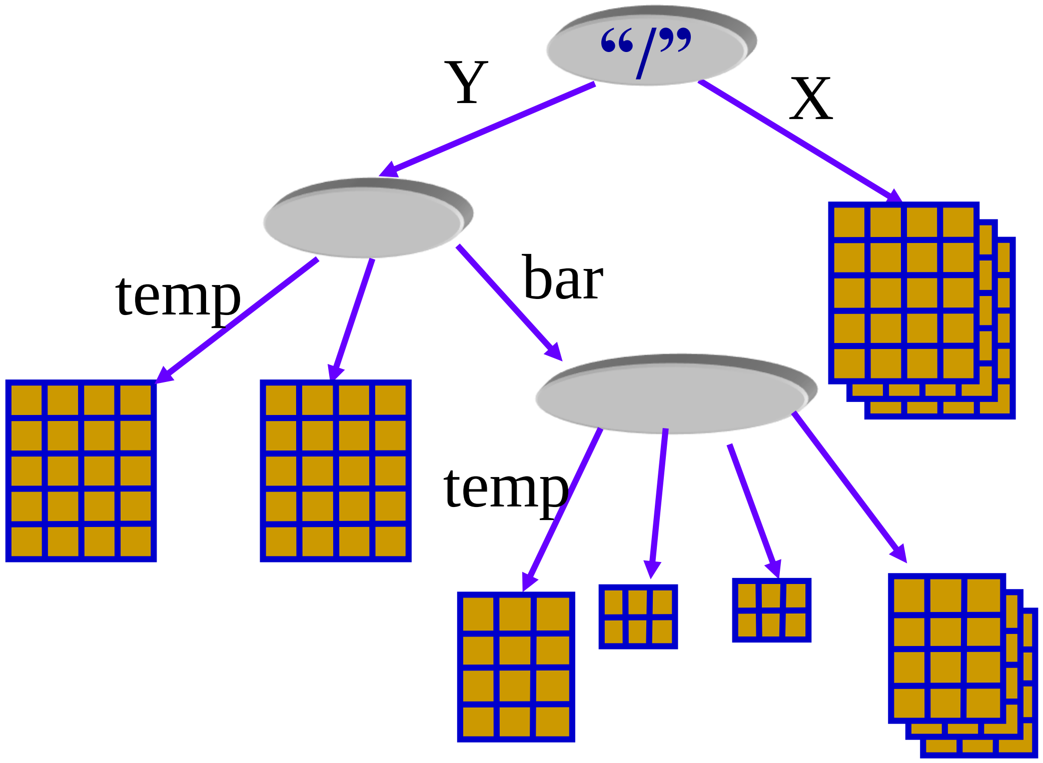
HDF5 model

- Groups – provide structure among objects
- Datasets – where the primary data goes
  - Rich set of datatype options
  - Flexible, efficient storage and I/O
- Attributes, for metadata annotations
- Links – point to other groups or datasets
  - Hard, soft and external flavors

Everything else is built essentially from these parts

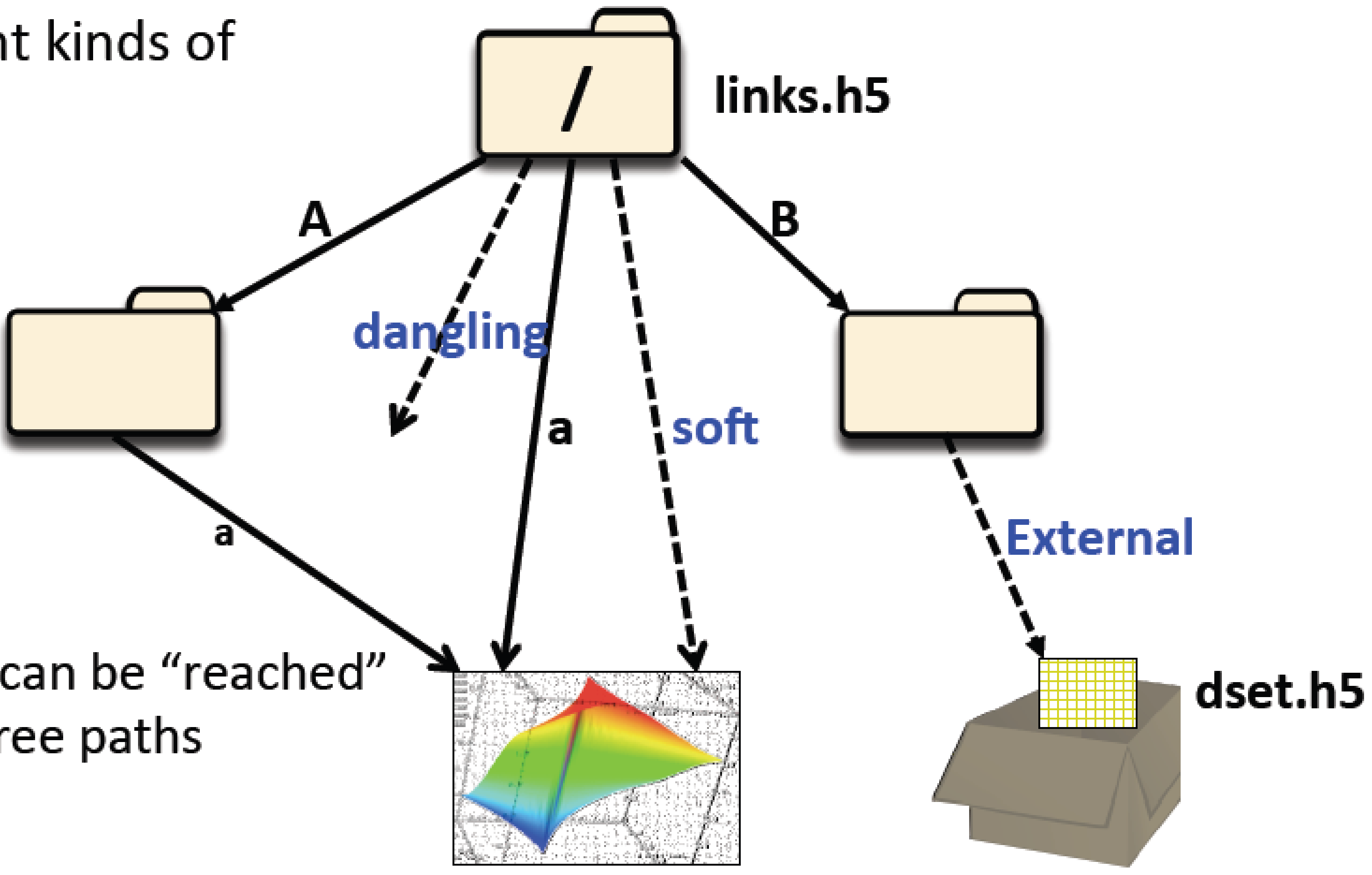
Path to HDF5 object in a file

/ (root)
/X
/Y
/Y/temp
/Y/bar/temp



HDF5 links

Different kinds of links



Dataset can be “reached” using three paths

/A/a

/a

/soft

Dataset is in a **different** file

Accessing HDF5 files

There are two main packages to access HDF5 files from Python: PyTables and h5py

At the ESRF we advocate the use of h5py:

- It is more intuitive for Python users (dictionary like access)
- It provides a direct binding to HDF5

The documentation is available at:

http://docs.h5py.org/en/latest/

HDF5

Hierarchical data format

Think about an HDF5 file as a hard disk

- You can write whatever you want to it
- It supports links
- It supports compression
- It is widely supported (Fortran, C, Java, Python, Matlab, IDL, …)
- It can be messy

What is NeXus?

According to the NeXus web pages, NeXus is

- A set of design principles
  To help people understand what is in the files
- A set of data storage objects
  To allow the design of portable analysis software
- A set of subroutines
  To make easy to write and to read NeXus files
- A scientific community
  To provide a forum to discuss ideas about data storage

What is NeXus today?

The NeXus API was able to read and to write files using HDF4, XML, HDF5, …

Most (all?) synchrotrons have made the choice of using HDF5

The evolution of HDF5 is faster than that of the NeXus API

The HDF5 API is more robust than that of NeXus

NeXus API (NAPI) has been frozen

This leads us to consider NeXus **a set of design principles**

What are those principles?

Structure you data in a set of Groups, Fields (=datasets), Attributes and Links

The expected content of a Group is defined by the attribute NX_class

The name of the Groups is not imposed

There are more than 50 “classes” defined:

http://download.nexusformat.org/doc/html/classes/base_classes/index.html

Most relevant classes

Nexus

NXroot

Top level. One per file.

NXentry

One group per measurement

NXinstrument

Describe the instrument.
Only one per NXentry

NXsample

Define the physical state of the sample
during the scan

NXdata

The data to be plotted.
One NXdata group per plot

NXuser

Details of a user, i.e., name, affiliation, email
address, *etc*

NXsubentry

Data or links to data for particular analysis

The measurement group

Reasons about why NeXus has found a lot of rejection by beamline scientists

-Attribute based instead of name based:

- Not easy ”interactive” use

- Cumbersome to look for other information than the default plot

- Not easy diagnostics

The goal of the “measurement” group

- Solve the common misuse of the NXdata group
- To provide a “quick index” to commonly used items
- Not to reinvent NeXus: for further information, NXinstrument

What do we have in mind?

Access all the measured items in a “measurement” group

```
measurement
    positioners  #group containing the equivalent of SPEC motor positions
         motor0
         motor1
         ….
    scalar_data  #group containing the equivalent of SPEC counter values
         counter0
         counter1
         …
     mca_device0
          data #shape (npoints, nchannels) @interpretation=“spectrum”
          info  # group containing information related to mca_data0, typically a link to the
device description in the NXinstrument group)
     camera_device0
          data #shape (npoints, nrows, ncolumns) @interpretation=“image”
          info  #group containing information related to image_data0, typically a link to
the device description in the NXinstrument group)
```

The measurement group

Access all the measured items in a “measurement” group

GOALS:

- Provide quick access to everything measured
- Have as little structure as possible (it can be totally flatten)
- The ESRF Metadata Working Group expected to provide the guidelines

ESRF NeXus files

NXroot

Top level. One per file.

NXentry

One group per measurement

NXinstrument

Describe the instrument.
Only one per NXentry

measurement (@NXcollection)

Flattened view of everything measured

Only one per NXentry

NXsample

Define the physical state of the sample during the scan

NXdata

The data to be plotted.
One NXdata group per plot

user (@NXuser)

Details of a user, i.e., name, affiliation, email address, *etc*

NXsubentry

Data or links to data for particular analysis

Exclusive **BCU** Domain

Almost exclusive **BCU** Domain

User/Scientist Domain

User/Scientist Domain

MIS Domain

Analysis Domain

A first proposal – SPEC file

- Conceptually, a SPEC file is quite similar to a NeXus file
    - It provides user information
    - It provides instrument information
        - Diffractometer geometry
        - Motor positions
        - MCA Detectors used and their calibration parameters
    - It provides sample information
        - Crystal Lattice
        - Crystal Orientation
    - Scan data block analogue of measurement group

A first proposal – SPEC file

- The h5py API is very appreciated and we use it to read NeXus files

- SPEC files are analogues of NeXus files

-Provide a new python binding to the SpecFile library using h5py API

    - Only one API to teach to scientists

    - Same family of widgets to access HDF5 and SPEC files

A first proposal – EDF file

- Can we make “Image” formats fit into “NeXus” formalism with h5py API?

- It risks to make current easy access more complicated

A second proposal - I

Use a simple formalism to look for particular data

– The access to image formats is very simple. Only access to data and info header.

– Modernize the access to files using iterators

– Straightforward interactive use

– That approach cannot appropriately deal with the information of HDF5 files or the case of multiple detectors into a SPEC file

A second proposal - II

```
class SourceReader(object or metaclass -> to be decided):
    """Implements a basic reader (info and data).

         GOAL: It opens the source to access its contents (by default in
                    readonly mode) looking for "auto" type of data.
    """


    def __init__(self, source_name, mode="r", target="auto")
         """
         target  for the time being it is foreseen "auto", "image", "spectrum"
         """
    def __iter__(self):
        iterator to retrieve objects sequentially
    def __len__(self):
        return the number of items according to the chosen target
    def __getitem__(self, XXXX):
        Return a DataObject with attributes data and info
```

Summary

– It looks “natural” to treat SPEC files as if they would have been converted to their NeXus equivalent

– It does not look “natural” to treat EDF files as if they would have been converted to their NeXus equivalent, but it can be done.

Ideas? Comments?