PaN-data ODI

Deliverable D8.1 - Update

Draft: Definition of a pHDF5 capable Nexus implementation (M9) - Software

Grant Agreement Number	RI-283556
Project Title	PaN-data Open Data Infrastructure
Title of Deliverable	Definition of pHDF5 capable Nexus implementation (M9) – Software additional paragraph to highlight definition component
Deliverable Number	D8.1 - update
Lead Beneficiary	STFC
Deliverable Dissemination Level	Public
Deliverable Nature	Report
Contractual Delivery Date	01 Apr 2014 (Month 30)
Actual Delivery Date	25 June 2014

The PaN-data ODI project is partly funded by the European Commission under the 7th Framework Programme, Information Society Technologies, Research Infrastructures.

Abstract

This document provides a draft definition of pHDF5 capable Nexus implementation (M9) - Software.

Keyword list

PaN-data ODI, Scalability

Document approval

Approved for submission to EC on 25.06.2014

Revision history

Issue	Author(s)	Date	Description
1.0	Bill Pulford	25 Jun 2012	Complete version for discussion
1.3	Bill Pulford with additional input from Mark Koennecke (SLS)	07 Mar 2014	SLS
1.4	Approved for submission	25 Jun 2014	

The screenshots contained in this document will be posted separately to the PANData web site together with the example data where possible.

This deliverable was approved during the first period review on condition that is was updated later converting it into a design document complete with implementation details. This version provides that update.

Table of contents

Page

1 UPDATE FROM EU TECHNICAL REVIEW 4

2 INTRODUCTION 7

3 DATA ACQUISITION 7

3.1 DATA ACQUISITION AND NEXUS 7
3.2 DERIVED REQUIREMENTS AND NOTES 9

4 PRACTICAL DATA ACQUISITION AND ANALYSIS USING NEXUS FILES. 11

4.1 CURRENT STATE OF DEPLOYMENT OF NEXUS AT DIAMOND. 12
4.2 CURRENT STATE OF DEPLOYMENT OF NEXUS AT DESY 14
4.2.1 The libpniutils, libpninx and libpninx-python libraries 14
4.2.2 Nexus Data Server (`NexDaTaS`) project 14
4.2.3 Example client code 15
4.2.4 Configuring the server 16

5 APPENDIX A - CURRENT NEXUS DATA MODELS AT DIAMOND 16

5.1 NON CRYSTALLINE DIFFRACTION 16
5.2 SPECTROSCOPY 20
5.3 DETECTOR 21
5.4 I24 – MACROMOLECULAR CRYSTALLOGRAPHY 21
5.5 TOMOGRAPHY 22
5.5.1 I13 - Tomography 22
5.5.2 I12 example – Energy Dispersive Exafs Detector (Mapping Mode) 23

1 Update - June 2014

This document acknowledges that data acquisition, evaluation, analysis and transfer at large user facilities is rendered increasingly difficult by a number of factors including:

1) The enhanced capabilities of the new research facilities enable users to perform more complex and ambitious experiments.

2) A high demand to optimize to components of the facilities such as beamlines; this has the effect of reducing the time for each experimental process and increasing the throughput.

3) The availability of new large area detectors with the potential of operating at high repetition rates enable the above increasingly advanced and efficient experiments.

1.1 Refined requirements

The key demands of the above include:

1) The availability of very high performance access to the underlying data storage

2) The development of or access to a widely accepted data format that that facilitates compatibility between applications written by collaborating partners.

3) An internationally accepted experimental metadata standard to optimize data workflows and to allow common views to collaborating scientists.

The starting hypothesis for demand 1) was that the file system performance provided by sequential links may be adequate in the short term but it is highly likely that the growth of the data volumes and speeds will soon outstrip this architecture. The most flexible solution is to implement a system with providing parallel data streams thereby multiplying the overall throughput available to the experimenter. The choice of Hierarchical Data Format (hdf5) was identified from the preceding PAN-Data Europe project (work package 5) and is increasingly being used across the collaborating facilities. In addition the same work package, recommended that the NeXuS metadata standard be superimposed on the hdf5 file for data tagging.

Thus the premise going forward is the definition of a parallel HDF5 capable NeXuS implementation i.e. pHDF5.

1.2 Overall design and implementation details

The design strategy for the scalability work package 8 was conceived as:

a) A re-evaluation of the recommendations from the work package 5 of PANData Europe.

b) An analysis of the issues and requirements initiated by moving from conventional single stream HDF5 to pHDF5.

c) A coordination of the progress made on high speed data acquisition both at collaborating facilities and from a wider field. DESY and SLS had strong contributions in this area.

d) An analysis of the practical implications of adding the NeXuS metadata into the parallel writing of data files.

The later documents in the work package concentrate on more detailed evaluation of the available solutions prior to organising their implementation or integration. The reports were focussed on two directions: a) the implementation of pHDF5 support b) the superposition of NeXuS metadata onto the HDF5 files concerned.

(Initial note: During the drafting of work package 8, the report numbering was inconsistent. Report 8.4 was omitted erroneously, this has been corrected below to make the originally submitted D8.5 - > D8.4 and D8.6 -> D8.5. D8.6 remains a deliverable and awaits the latest results)

1.2.1 D8.2: Evaluation of Parallel file systems and MPI I/O implementations

This report considered the technical challenges and potential solutions for providing efficient high performance pHDF5 file I/O supported by appropriate file systems. The challenges included:

- The parameters of high performance detectors to be supported; mainly speed and data volumes.
- An examination of the most promising technical solutions for file systems that indicated Lustre and GPFS as the most promising.
- A consideration of the practical details of implementing parallel HDF files on such systems including examples and recommendations for future developments; where feasible the latter have been planned or implemented.

1.2.2 D8.3 Implementation of pNeXus and MPI I/O on parallel file systems

The work reported on was complementary to that in report D5.2 but concentrated on:

- The practical implementations of software libraries to read and write HDF files.
- The definition of metadata to support particular scientific fields.
- A mechanism that enables the direct and transparent incorporation of additional widely used data formats into a NeXuS based data evaluation and analysis framework.
- This is very important to exploit NeXuS/HDF5 tools to operate on data of crystallographic origin and facilitate cross disciplinary data evaluation and analysis.

The continued elaboration of tools such as DAWN to use NeXuS access file sets; the key principle being that the metadata in one NeXus file should be sufficient to perform most common operations

1.2.3 D8.4 Evaluation of Parallel file systems and MPI I/O implementations

The report was a review of the solutions implemented at that point and included details of how these could be applied to the example of a new high speed detector being developed. The results were presented together with the following observations and recommendations:

- Both Lustre and GPFS require significant tuning to optimize performance for a given workload.

- Subsequent experience has shown that good results can be achieved without a thread safe HDF library but the implementation of the latter would be valuable for future projects.
- It was apparent from the Excalibur development that large HDF5 files provide a challenge to the underlying file system. Of particular concern, semi random file access and B-tree constraints can lead extended write delays and unexpected data stream behavior.
- In the case of very demanding data acquisitions using high data rate detectors an appropriate strategy may be to delegate a NeXuS file as the analysis/evaluation application entry. This file would consist of internal data sections and metadata and use file links to the large data files being created. These files would be written in the most efficient format possible, a strategy also being employed by Dectris for the EIGER detector
- There are no definitive recommendations for this report. The results gained from the commissioning of the Excalibur detector indicate that Lustre may be a more efficient underlying file system in this case. There are nevertheless arguments based on experience from other facilities that GPFS or derivatives would be preferable.

1.2.4 D8.5 Demonstrate capabilities reported in D8.3 and D8.4

- No bespoke software development was performed for this report. It was realized that the exploitation of the DAWNScience open source application development by collaboration (http://www.dawnsci.org/home) between Diamond, ESRF, EMBL (Grenoble) and the commercial company iscenia (www.iscencia.be), would provide a much better vehicle to demonstrate the capabilities of the combination of parallel file systems and NeXuS file formats. The resources initially foreseen for this part of the work package were used preferentially for the bridge between ImageCIF/CBF and NeXuS.

2 Introduction

PaNData Europe selected HDF5 as the underlying format and considered NeXus to be the starting point for a specification of file structures and application definitions. Implementing NeXus/HDF5 introduces the following topics:

a) Implementation details of the underlying file format Hierarchical Data Format or HDF5; the more advanced issues concerning the provision of high performance including parallel access is discussed on the companion report PANData 8.2.
b) Specification of library as an extension of the HDF5 routines to implement a native interface to superimposing the NEXUS metadata model onto files complying with HDF5.
c) Defining the NEXUS metadata model describing the contents of the NEXUS format file particularly with respect to the performing the continuing scientific data analysis.
d) This report is complementary to report D5.1 (particularly section 3) – Virtual Laboratories – in that it reviews experience implementing and using the NEXUS metadata model in current user operation.
e) Many of the examples is the report are based on the Dawn Science collaboration between the ESRF, EMBL Grenoble and Diamond Light Source. http://www.dawnsci.org/. This continues to be a very valuable tool not only for data analysis but also for its ability to explore the contents of NEXUS files and ICAT repositories in one application.

3 Data Acquisition

3.1 Data Acquisition and NEXUS

Normally the data acquisition process at a large facility involves the manipulation and scan of one or more experimental samples while subjecting them to a number of different external conditions. The resulting data are measured with one or more detectors specialized for X-Ray or Neutron depending on facility. The virtue of the NEXUS is that it provides a standard metadata model for storing the experimental variables and the data from the detectors in an annotated file structure that provides at least an audit of the experimental process and should enable the following data analysis without needing the input of further parameters.

Report D5.1 section 3.1 provides a clear list of requirements for a NEXUS library capable of providing basic support for the above. These requirements do however need to be reviewed in the light of experience of current data acquisition particularly at synchrotron facilities; these have been described in more detail in PANdata report D8.2 - Scalability.

Modern large area detectors with shutterless operation and connected by 10Gb/s networks that may even be used in multiple on any one experiment are capable of creating huge data rates in parallel streams. An example of this observation is that the largest scans at an X-ray facility for certain experiments may easily exceed 100Gb and the process may include many such scans. It is evident that the community needs to move toward highly optimized data writing and reading including the exploitation of parallel technology at the lowest level. HDF5 is a mature but advancing technology with wide support in the global scientific community and should be the major focus of development to fulfill this type of requirement. NEXUS is designed to provide a key component of scientific data acquisition and analysis by defining the standard for metadata that could allow a researcher to progress with data analysis without prior knowledge of the acquisition process. It is not clear that the planned or implemented advanced capabilities of HDF5 should be replicated directly in the NEXUS library or by allowing the NEXUS metadata to be written directly into the HDF5 file.

The NEXUS developers have recognized this issue and provide documented examples in the reference manual of NEXUS file writing using both the standard and the HDF APIs.
See http://download.nexusformat.org/doc/html/Examples.html

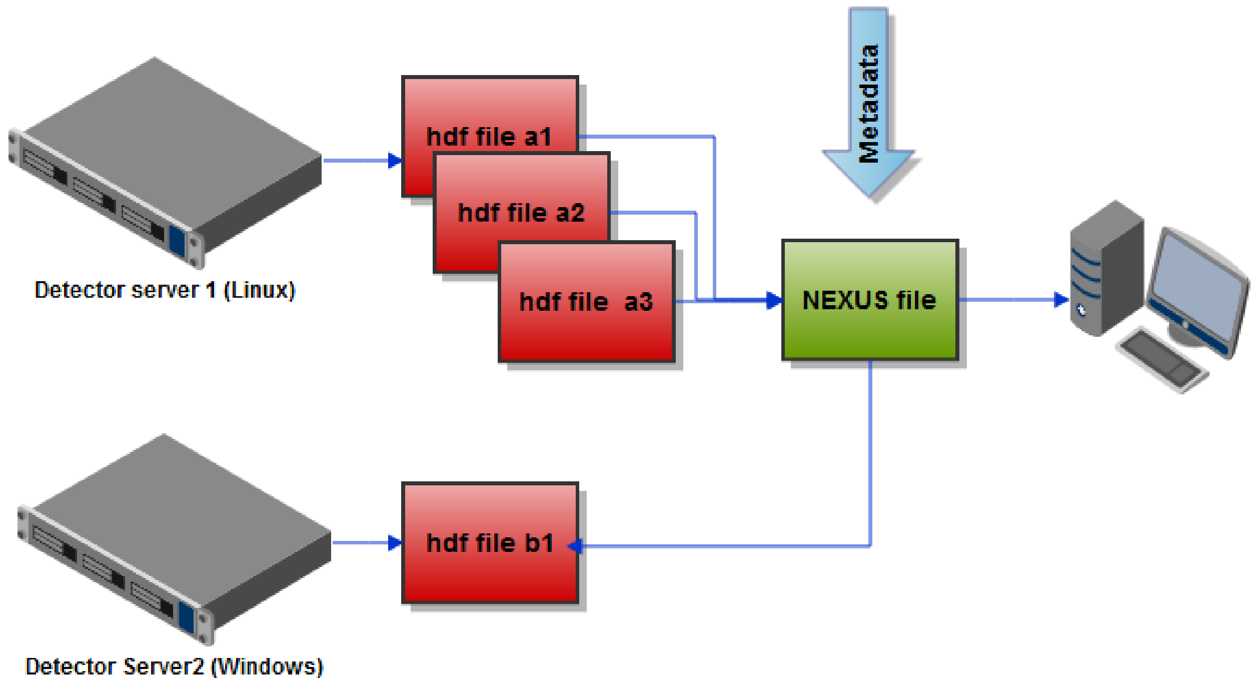


Figure 1 gives a schematic representation of an experimental setup found increasing frequently where multiple detectors are configured for one experimental procedure. In the case of high data rate or complex experiments data may originate from a number of detectors and practically is stored in separate files written directly by the detector server. The consistency of the NEXUS file is implemented using the NXDataFile constructor together with the associated metadata.

The NEXUS file will store:

a) All metadata appropriate for the experimental process.
b) NXdata segments for real data from small devices and detectors.
c) NXdatafile links to the relevant separate files.

It should be possible to write the complete NEXUS file using only the HDF5 libraries.

3.2 Derived requirements and notes

The list of requirements for the NEXUS library as specified in report D5.1 is repeated below for convenience and augmented with notes where necessary.

1) *The NEXUS library or NAPI must contain C++ API and Python bindings because these languages are widely used in the neutron and photon community.*
2) *The interface should be strictly object oriented and type-safe. The latter means that data have to be fully qualified when they are transferred between an application and the library. The aim of this restriction is to avoid misinterpretations of the data structures and thereby make the usage of the NEXUS library more reliable.*
3) *The NeXus library should not cause substantial performance losses compared to HDF5 as far as I/O speed is concerned.*
   Notes
   - **The direct NEXUS library (NAPI) may provide a simpler interface to creating single file acquisitions but where performance and multiple detectors and associated files are needed then HDF5 may need to be used directly.**
   - **The NEXUS International Advisory Committee after feedback from the community also advise writing NeXus files directly using HDF-5 calls.**
   - **DESY developers have nevertheless recent applied significant effort to the direct NAPI library as outlined in 3.2 below which may lead to the review of the above recommendations.**
   - **The current most important role of the NAPI library is to ensure that files created with NEXUS compatible file layout (HDF5) and metadata conform to the agreed standard.**
4) *The library code has to be maintained in a public repository to allow for a cooperative code development.*
5) **Detector vendors should be encouraged to write at least HDF5 files for increased performance and compatibility.**
6) **Full descriptions and data file APIs should be provided by the detector vendor to enable a library of filters to be constructed for use within the NEXUS framework to present data from the subordinate data files when these are not HDF5.**
   - **It is very important particularly for Crystallography to coordinate imgCIF format and metadata.**
   - **There are now efforts to implement appropriate filters.**
7) *Data compression is of increasing importance. The library should make the default HDF5 filters available and allow for additional, discipline-specific filters which are provided by application programmers.* **See Report PANData R8.2.**
8) *Detector vendors may install FPGA- or ASIC-based data compression in the near future. Future releases of NeXus/HDF5 have to implement a scheme for treating external filters in a way that they are transparent from the users point of view.*
9) *A complete documentation of the NeXus library, including a user guide, is mandatory.*
10) *All NeXus developments have to be coordinated with the NeXus International Advisory Committee (NIAC).*
11) *Future NeXus/HDF5 versions should implement the single writer/multiple reader functionality.*
    Note
    - **This is underway at the HDF5 level - See Report PANData R8.2.**

4 Practical Data Acquisition and Analysis using NEXUS files.

The PANData report 5 sections 3.2 and 3.3 highlight some of the issues concerning the standardization of the metadata in acquired files. This is clearly important for sharing analysis data and applications across facilities.

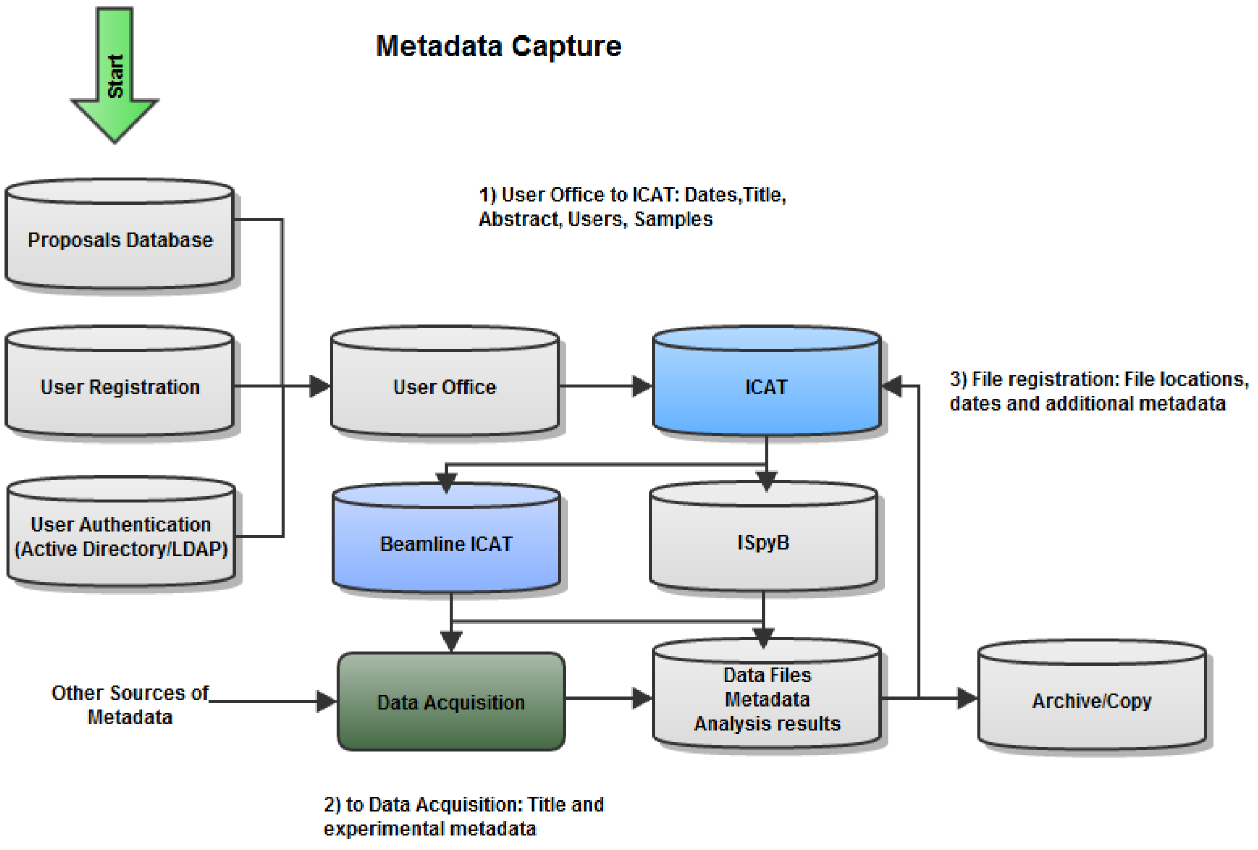
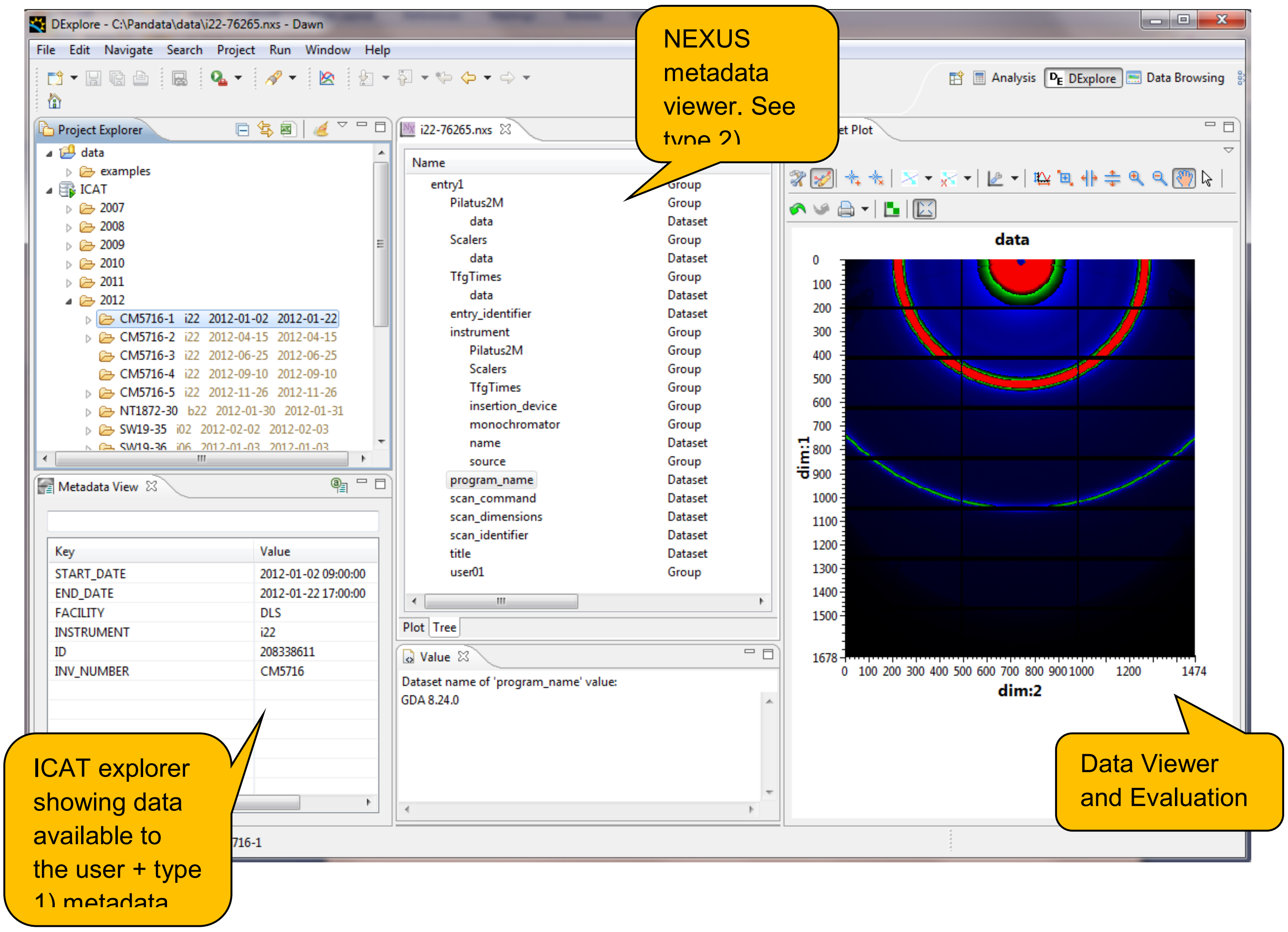

Start
Metadata Capture
Proposals Database
1) User Office to ICAT: Dates,Title, Abstract, Users, Samples
User Registration
User Office
ICAT
3) File registration: File locations, dates and additional metadata
User Authentication (Active Directory/LDAP)
Beamline ICAT
ISpyB
Other Sources of Metadata
Data Acquisition
Data Files Metadata Analysis results
Archive/Copy
2) to Data Acquisition: Title and experimental metadata

Figure 2 – A schematic view of the association of metadata to acquired data at the Diamond Light Source. This is analogous to the systems currently available at other facilities. The overall aims of PANData should provide the necessary software and infrastructure to implement this elsewhere.

In more detail:

1) The principle input available from the user office systems is fed into the ICAT with an option to be added to the metadata in the NEXUS files. This is normally “the who, what, where, when, abstract and samples” that is used for data searches.
2) These are the data from the acquisition process that would be placed directly into each NEXUS file and would be important to the further data evaluation and analysis.
3) These are the logistical metadata derived during the acquisition concerning relative file locations and additional resources that are important for both the ICAT and the actual data files.

The DAWN project currently provides a software application convenient for exploring the architecture and data conforming to the structure shown in figure 2. The screenshot illustrates a small part of functionality available that is enabled by the architecture and is useful in mapping the experimental investigation to the data files.


DExplore - C:\Pandata\data\i22-76265.nxs - Dawn
File Edit Navigate Search Project Run Window Help
Analysis DExplore Data Browsing
Project Explorer
i22-76265.nxs
Plot
Metadata View
Value
NEXUS metadata viewer. See type 2)
ICAT explorer showing data available to the user + type 1) metadata
Data Viewer and Evaluation
data
dim:1
dim:2

```
data
  examples
ICAT
  2007
  2008
  2009
  2010
  2011
  2012
    CM5716-1   i22   2012-01-02   2012-01-22
    CM5716-2   i22   2012-04-15   2012-04-15
    CM5716-3   i22   2012-06-25   2012-06-25
    CM5716-4   i22   2012-09-10   2012-09-10
    CM5716-5   i22   2012-11-26   2012-11-26
    NT1872-30   b22   2012-01-30   2012-01-31
    SW19-35   i02   2012-02-02   2012-02-03
```

Key	Value
START_DATE	2012-01-02 09:00:00
END_DATE	2012-01-22 17:00:00
FACILITY	DLS
INSTRUMENT	i22
ID	208338611
INV_NUMBER	CM5716

Name	
entry1	Group
Pilatus2M	Group
data	Dataset
Scalers	Group
data	Dataset
TfgTimes	Group
data	Dataset
entry_identifier	Dataset
instrument	Group
Pilatus2M	Group
Scalers	Group
TfgTimes	Group
insertion_device	Group
monochromator	Group
name	Dataset
source	Group
program_name	Dataset
scan_command	Dataset
scan_dimensions	Dataset
scan_identifier	Dataset
title	Dataset
user01	Group

```
Dataset name of 'program_name' value:
GDA 8.24.0
```

4.1 Current state of deployment of NEXUS at Diamond.

(Please note that it would be very useful for collaborators to add their current status at this point)

- The Data Acquisition software at Diamond now writes data files subject to the NEXUS standard as the default file format on a significant number of beamlines.
    - The main analysis software at Diamond is the Generic Data Acquisition (GDA)
    - NEXUS data files are nevertheless being written by supplementary applications on those beamlines where GDA is not appropriate where possible.
    - Logistically there are considerable advantages to using the NEXUS files:
        - Certain experiment types or detectors may produces very large numbers of small files that are inefficient for both storage and for including in data analysis. Using NEXUS avoid this by providing the possibility to aggregate these into one file.

        - Data acquisition such as for tomography and imaging can create sequences of files that are more efficiently stored as one or a small number of appropriately tagged NEXUS files.

- High priority has been given to implementing analysis software able to read and write NEXUS files.
    - The standard analysis package employed is that from the Open Source project DAWN Science. See 1e) above.
    - Third party applications such as Matlab and Igor Pro can also read validated HDF5 files and are used with potentially with the need for additional input required from the user.

- The scientific disciplines with metadata models that have proven sufficient for supporting data analysis are:
    - Non Crystalline Diffraction. - NCD
    - Spectroscopy
    - Tomography and Imaging
    - Limited Macromolecular Crystallography (MX)

Appendix A provides the XML extracts of the NEXUS metadata for the above disciplines.

4.2 Current State of Deployment of Nexus at DESY

NeXus is currently been implemented at the PETRA III beamlines. This format has been chosen for several reasons:

- The internal structure of the NeXus files can be adopted to specific experimental techniques.
- Metadata can be stored for a complete description of the measurement.
- NeXus can store many image frames that are created by a measuring sequence in a single file. This way data can be managed efficiently.

The next paragraph gives some details about the NeXus related libraries which have been developed at DESY. This section ends with an introduction of the Nexus Data Server project. The result will be a Tango server that facilitates the file I/O for the online control programs.

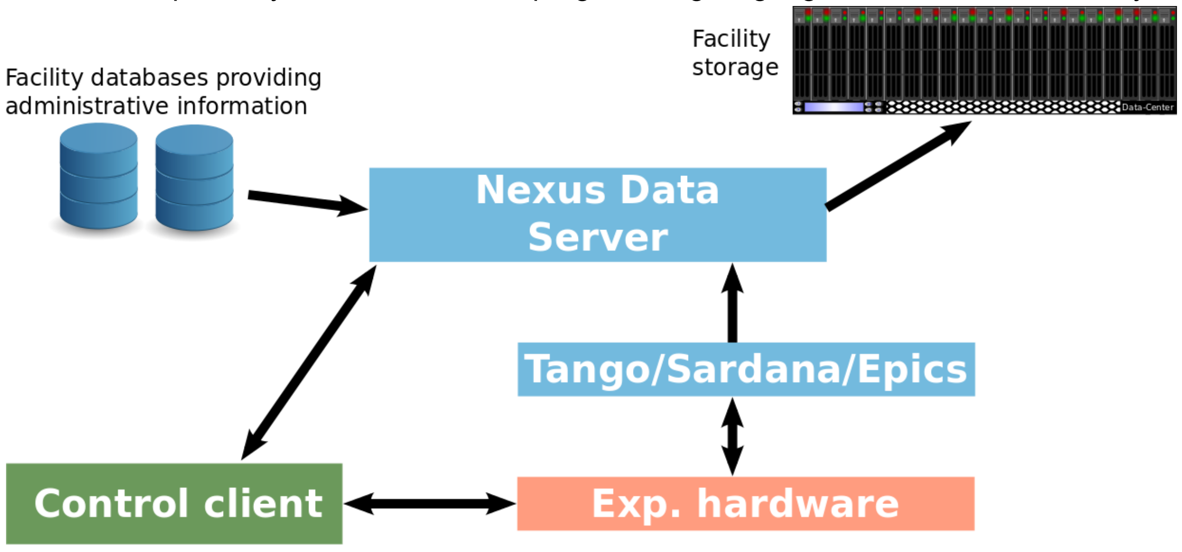
4.2.1 The libpniutils, libpninx and libpninx-python libraries

A couple of C++ libraries have been developed at DESY within the framework of the HDRI project with the intention to simplify the creation of Nexus files. Among others, `libpniutils` provides simple types of well defined size, templates for buffers and arrays, as well as reader code to import data from proprietary formats. `libpninx` provides classes for writing Nexus files using HDF5 as its storage backend. It uses a Bridge pattern in order to make the development of the Nexus API independent of the HDF5 development. Finally Python bindings are provided to `libpninx` via the `libpninx-python` Python package. All libraries are actually in use to develop the software required to establish Nexus as a data format at DESY.
The libraries source code is published and distributed via sourceforge

- `libpniutils` - http://sourceforge.net/projects/libpniutils/
- `libpninx` - http://sourceforge.net/projects/libpninx/
- `libpninx-python` - http://sourceforge.net/projects/libpninxpython

4.2.2 Nexus Data Server (`NexDaTaS`) project

Until now, at most facilities (including DESY) the control client software (SPEC, ONLINE, …) is responsible for writing data to disk. As the complexity of Nexus files increases with more advanced experiments so does the code in charge for data IO. Consequently one would have to maintain a lot of code, most probably written in different programming languages, for different control systems.



We thus have decided to move the responsibility for data IO out of the control client (CC) into a separate Tango server: the Nexus Data Server (NexDaTaS). To communicate with this server the

control system must only be aware of TANGO for which bindings to many common programming languages exist (C++, Java, Python, Matlab, IGOR Pro, LabView).
The server is capable of creating Nexus files and filling its field from various data sources including

- databases
- other TANGO and SARDANA servers
- and directly from the CC using JSON strings

As configuration data the server needs to know the structure of the Nexus tree for a particular entry and the data sources for each field in the tree. The configuration is passed from the CC to the Nexus server as NXDL code including some extra tags for data sources and storage strategy.

```
<field name=”tth” type=”NX_FLOAT” unit=”degree”>
  <strategy mode="STEP" trigger="trigger1"/>

  <datasource type="TANGO">
    <device hostname="haso.desy.de"
            member="attribute"
            name="p09/motor/exp.01"
            port="10000"/>
    <record name="Position"/>
  </datasource>
</field>
```

Once this configuration information is submitted to the server it can run more or less without any client interaction. For the time being the data server is intended to be used only for low data rate sources. The first implementation of NexDaTaS is written in Python and available from http://code.google.com/p/nexdatas/ . High data rate servers might be implemented in future if required and technically reasonable.

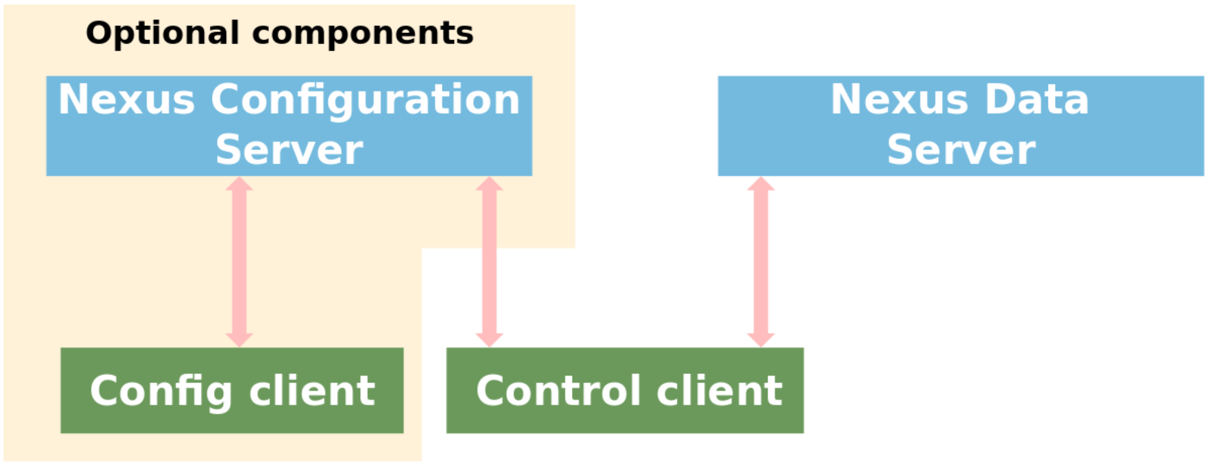
4.2.3 Example client code

A short example should show how easy it is to use the server from a client application:

```
import PyTango
device = "p09/tdw/r228"
dpx = PyTango.DeviceProxy(device)
dpx.Init()

# open a new file to store data in
dpx.FileName = "test.h5"
dpx.OpenFile()

# send configuration for a new entry and write initial data (strategy type INIT)
xml = open("configuration.xml", 'r').read()
dpx.TheXMLSettings = xml
dpx.TheJSONRecord = '{"data": {"parameterA":0.2}}'
dpx.OpenEntry()

# experiment main loop (write data with strategy STEP)
for i in range(100):
    dpx.Record('{"data": {"counter.exp01":0.1,
                          "counter.exp02":1.1}}')

# write final data (strategy FINAL) - close the entry - close the file
dpx.TheJSONRecord = '{"data": {"parameterB":0.3}}'
dpx.CloseEntry()
dpx.CloseFile()
```

4.2.4 Configuring the server

As long as the experiment is quite static and not too complex creating the configuration NXDL stream for the Nexus Data Server should be easy and could be done by the control client. However, in the case of complex experiments we may just have shifted the complexity of the client code from writing the Nexus file directly towards creating an advanced NXDL stream. We have thus decided to add a configuration server to manage different beamline configurations and provided the control client with the required configuration stream.



The configuration server not only simplifies the creation of server configuration streams it also acts as a central storage facility for experiment configurations. Thus several control clients on a beamline can rely on the same configuration data. The configuration server provides the control client with a list of components available at the beamline. From this list the control client selects those items it wants to be stored in the file for a particular experiment and submits its selection back to the configuration server. The server finally assembles the configuration NXDL stream from the control client's selection and returns it to the control client. With this procedure the control client does not have to take care about creating the configuration at all. It only has to make a proper choice from the components available from the configuration server.

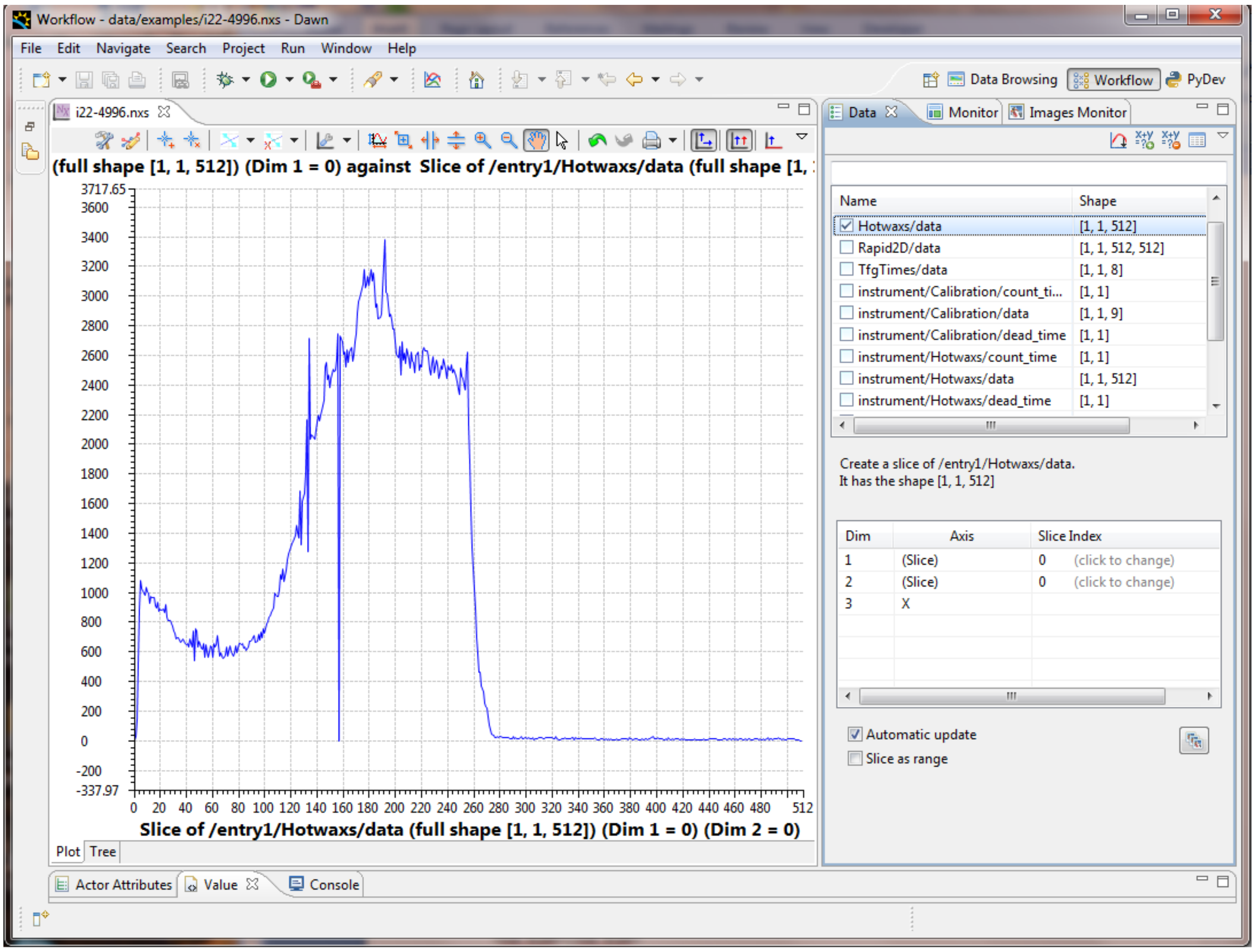
5 Appendix A - Current NEXUS data models at Diamond

The following are xml extracts from NEXUS files. They cover examples of the necessary information to be stored in the file to enable at least initial data evaluation without demanding more from the user. In some examples the DAWN Science application has been used to illustrate the contents of particular NXdata segments

5.1 Non Crystalline Diffraction

```
<?xml version="1.0"?>
<NXroot NeXus_version="4.2.0" file_name="/dls/i22/data/2010/sm2891-1/i22-4996.nxs"
 HDF5_Version="1.6.10" file_time="2010-07-07T12:02:16+00:00">
  <NXentry name="entry1">
    <NXdata name="Calibration">
      <data type="NX_FLOAT32[1,1,9]" signal="1" units="counts"
        target="/entry1/instrument/Calibration/data"></data>
    </NXdata>
```

```
<NXdata name="Hotwaxs">
    <data type="NX_FLOAT32[1,1,512]" signal="1" units="counts"
     target="/entry1/instrument/Hotwaxs/data"></data>
  </NXdata>
```



Name	Shape
☑ Hotwaxs/data	[1, 1, 512]
☐ Rapid2D/data	[1, 1, 512, 512]
☐ TfgTimes/data	[1, 1, 8]
☐ instrument/Calibration/count_ti...	[1, 1]
☐ instrument/Calibration/data	[1, 1, 9]
☐ instrument/Calibration/dead_time	[1, 1]
☐ instrument/Hotwaxs/count_time	[1, 1]
☐ instrument/Hotwaxs/data	[1, 1, 512]
☐ instrument/Hotwaxs/dead_time	[1, 1]

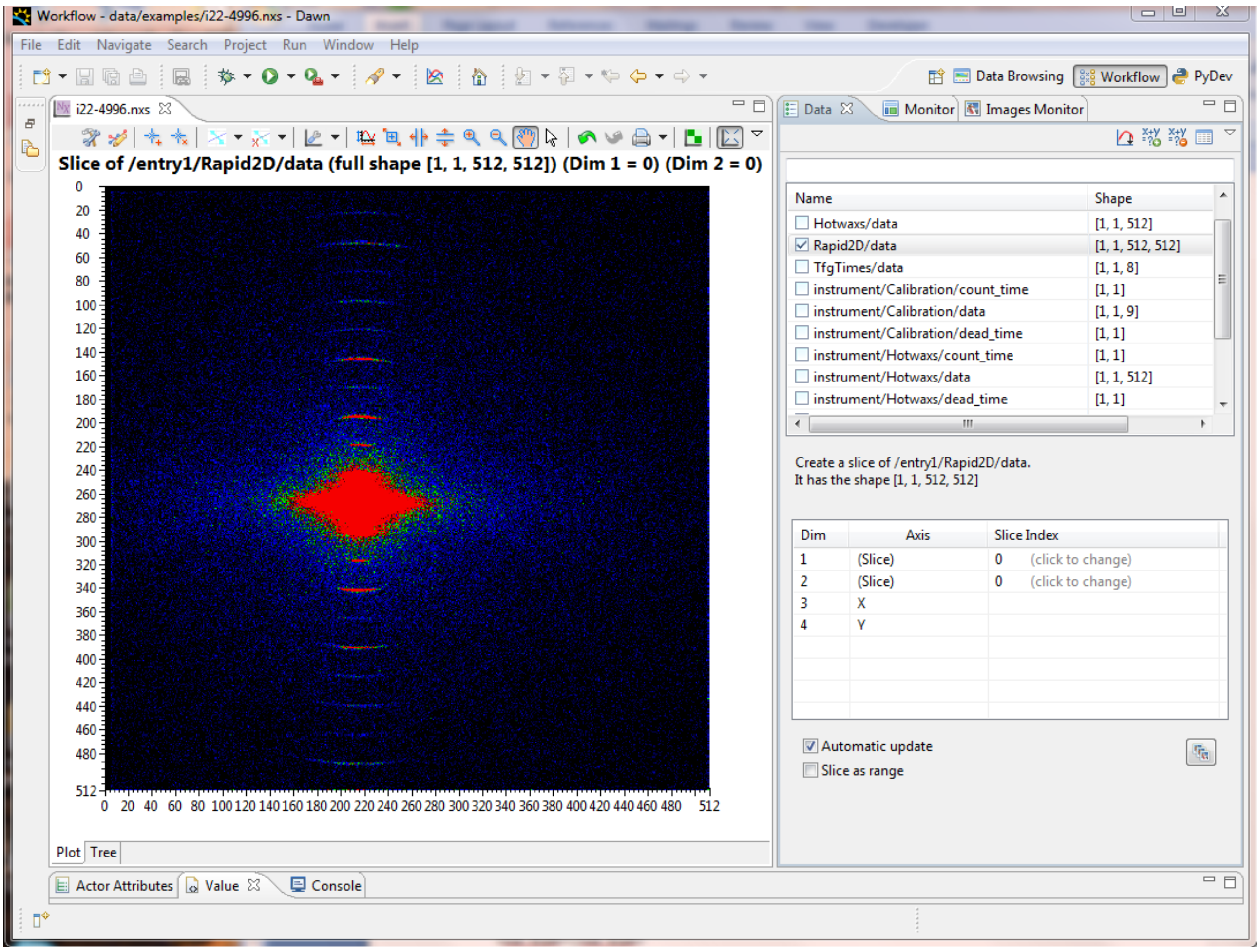
Create a slice of /entry1/Hotwaxs/data.
It has the shape [1, 1, 512]

Dim	Axis	Slice Index	
1	(Slice)	0	(click to change)
2	(Slice)	0	(click to change)
3	X		

☑ Automatic update
☐ Slice as range

The HOTWaxs data histogram.

```
  <NXdata name="Rapid2D">
    <data type="NX_FLOAT32[1,1,512,512]" signal="1" units="counts"
     target="/entry1/instrument/Rapid2D/data"></data>
  </NXdata>
```


Workflow - data/examples/i22-4996.nxs - Dawn
File Edit Navigate Search Project Run Window Help
Data Browsing
Workflow
PyDev
i22-4996.nxs
Slice of /entry1/Rapid2D/data (full shape [1, 1, 512, 512]) (Dim 1 = 0) (Dim 2 = 0)
Plot
Tree
Data
Monitor
Images Monitor
Actor Attributes
Value
Console

Name	Shape
☐ Hotwaxs/data	[1, 1, 512]
☑ Rapid2D/data	[1, 1, 512, 512]
☐ TfgTimes/data	[1, 1, 8]
☐ instrument/Calibration/count_time	[1, 1]
☐ instrument/Calibration/data	[1, 1, 9]
☐ instrument/Calibration/dead_time	[1, 1]
☐ instrument/Hotwaxs/count_time	[1, 1]
☐ instrument/Hotwaxs/data	[1, 1, 512]
☐ instrument/Hotwaxs/dead_time	[1, 1]

Create a slice of /entry1/Rapid2D/data.
It has the shape [1, 1, 512, 512]

Dim	Axis	Slice Index
1	(Slice)	0 (click to change)
2	(Slice)	0 (click to change)
3	X	
4	Y	

☑ Automatic update
☐ Slice as range

The Rapid2D data view

```
<NXdata name="TfgTimes">
    <data type="NX_FLOAT32[1,1,8]" units="s" target="/entry1/instrument/TfgTimes/data"></data>
  </NXdata>
  <entry_identifier></entry_identifier>
  <NXinstrument name="instrument">
    <NXdetector name="Calibration">
      <count_time type="NX_FLOAT32[1,1]" units="s"></count_time>
      <data type="NX_FLOAT32[1,1,9]" signal="1" units="counts"
        target="/entry1/instrument/Calibration/data"></data>
      <dead_time type="NX_FLOAT32[1,1]" units="s"></dead_time>
      <description></description>
      <sas_type></sas_type>
    </NXdetector>
    <NXdetector name="Hotwaxs">
      <count_time type="NX_FLOAT32[1,1]" units="s"></count_time>
      <data type="NX_FLOAT32[1,1,512]" signal="1" units="counts"
        target="/entry1/instrument/Hotwaxs/data"></data>
      <dead_time type="NX_FLOAT32[1,1]" units="s"></dead_time>
      <sas_type></sas_type>
    </NXdetector>
    <NXdetector name="Rapid2D">
      <count_time type="NX_FLOAT32[1,1]" units="s"></count_time>
```

```
        <data type="NX_FLOAT32[1,1,512,512]" signal="1" units="counts"
          target="/entry1/instrument/Rapid2D/data"></data>
        <dead_time type="NX_FLOAT32[1,1]" units="s"></dead_time>
        <sas_type></sas_type>
      </NXdetector>
      <NXdetector name="TfgTimes">
        <data type="NX_FLOAT32[1,1,8]" units="s" target="/entry1/instrument/TfgTimes/data"></data>
        <sas_type></sas_type>
      </NXdetector>
      <name></name>
      <NXsource name="source">
        <current type="NX_FLOAT64"></current>
        <frequency type="NX_FLOAT64"></frequency>
        <name></name>
        <notes></notes>
        <power type="NX_FLOAT64"></power>
        <probe></probe>
        <type></type>
        <voltage type="NX_FLOAT64"></voltage>
      </NXsource>
    </NXinstrument>
    <program_name></program_name>
    <scan_command></scan_command>
    <scan_dimensions type="NX_INT32"></scan_dimensions>
    <scan_identifier></scan_identifier>
    <title></title>
    <NXuser name="user01">
      <username></username>
    </NXuser>
  </NXentry>
</NXroot>
```

5.2 Spectroscopy

```
<?xml version="1.0"?>
<NXroot NeXus_version="4.2.0" file_name="/dls/b18/data/2010/cm1901-
3/Experiment_1/nexus/Ptfoil3_3_568.nxs"
  HDF5_Version="1.6.10" file_time="2010-04-22T23:20:51+00:00">
   <NXentry name="entry1">
     <NXdata name="counterTimer01">
       <Energy type="NX_FLOAT64[846]" target="/entry1/instrument/xas_scannable/Energy" ax-
is="1"></Energy>
       <I0 type="NX_FLOAT64[846]" target="/entry1/instrument/counterTimer01/I0"></I0>
       <Iref type="NX_FLOAT64[846]" target="/entry1/instrument/counterTimer01/Iref"></Iref>
       <It type="NX_FLOAT64[846]" target="/entry1/instrument/counterTimer01/It"></It>
       <Time type="NX_FLOAT64[846]" target="/entry1/instrument/xas_scannable/Time" axis="1"></Time>
       <lnI0Iref type="NX_FLOAT64[846]" target="/entry1/instrument/counterTimer01/lnI0Iref"></lnI0Iref>
       <lnI0It type="NX_FLOAT64[846]" target="/entry1/instrument/counterTimer01/lnI0It"></lnI0It>
     </NXdata>
     <entry_identifier></entry_identifier>
     <NXinstrument name="instrument">
       <NXdetector name="counterTimer01">
         <I0 type="NX_FLOAT64[846]" target="/entry1/instrument/counterTimer01/I0"></I0>
         <Iref type="NX_FLOAT64[846]" target="/entry1/instrument/counterTimer01/Iref"></Iref>
         <It type="NX_FLOAT64[846]" target="/entry1/instrument/counterTimer01/It"></It>
         <description></description>
         <id></id>
         <lnI0Iref type="NX_FLOAT64[846]" target="/entry1/instrument/counterTimer01/lnI0Iref"></lnI0Iref>
         <lnI0It type="NX_FLOAT64[846]" target="/entry1/instrument/counterTimer01/lnI0It"></lnI0It>
         <type></type>
       </NXdetector>
       <name></name>
       <NXsource name="source">
         <current type="NX_FLOAT64"></current>
         <frequency type="NX_FLOAT64"></frequency>
         <name></name>
         <notes></notes>
         <power type="NX_FLOAT64"></power>
         <probe></probe>
         <type></type>
         <voltage type="NX_FLOAT64"></voltage>
       </NXsource>
       <NXpositioner name="xas_scannable">
         <Energy type="NX_FLOAT64[846]" target="/entry1/instrument/xas_scannable/Energy" ax-
is="1"></Energy>
         <Time type="NX_FLOAT64[846]" target="/entry1/instrument/xas_scannable/Time" axis="1"></Time>
       </NXpositioner>
     </NXinstrument>
     <program_name></program_name>
     <scan_command></scan_command>
     <scan_dimensions type="NX_INT32"></scan_dimensions>
```

```
    <scan_identifier></scan_identifier>
    <NXuser name="user01">
      <username></username>
    </NXuser>
  </NXentry>
</NXroot>
```

5.3 Detector

```
<?xml version="1.0"?>
<NXroot HDF5_version="1.8.6" file_name="/home/voo82357/SAS_Data/Tim_lucas_data/Images/BAp4_p1-
00001_test.nxs"
  file_time="2012-02-22T21:40:14+0000">
  <NXentry name="entry1">
    <NXdata name="PerkinElmer">
      <data type="NX_FLOAT32[1,1,2048,2048]" signal="1" units="counts"></data>
    </NXdata>
    <NXinstrument name="instrument">
      <NXdetector name="PerkinElmer">
        <data type="NX_FLOAT32[1,1,2048,2048]" signal="1" units="counts"></data>
        <sas_type></sas_type>
      </NXdetector>
    </NXinstrument>
  </NXentry>
</NXroot>
```

5.4 I24 – Macromolecular Crystallography

```
<?xml version="1.0"?>
<NXroot NeXus_version="4.2.1" file_name="/scratch/rbv51579/gda-8.14/i24-config/users/data/2.nxs"
  HDF5_Version="1.8.7" file_time="2011-11-01T12:07:25+00:00">
  <NXentry name="entry1">
    <NXdata name="default">
      <BPM1IN type="NX_FLOAT64[61]" signal="1" target="/entry1/instrument/BPM1IN/BPM1IN"></BPM1IN>
      <DCMFPitch type="NX_FLOAT64[61]" label="1" primary="1"
        target="/entry1/instrument/DCMFPitch/DCMFPitch" axis="1"></DCMFPitch>
      <Time type="NX_FLOAT64[61]" axis="1" target="/entry1/instrument/counter/Time"></Time>
    </NXdata>
    <entry_identifier></entry_identifier>
    <NXinstrument name="instrument">
      <NXpositioner name="BPM1IN">
        <BPM1IN type="NX_FLOAT64[61]" signal="1" tar-
get="/entry1/instrument/BPM1IN/BPM1IN"></BPM1IN>
      </NXpositioner>
      <NXpositioner name="DCMFPitch">
        <DCMFPitch type="NX_FLOAT64[61]" label="1" primary="1"
          target="/entry1/instrument/DCMFPitch/DCMFPitch" axis="1"></DCMFPitch>
      </NXpositioner>
      <NXpositioner name="counter">
```

```
        <Time type="NX_FLOAT64[61]" axis="1" target="/entry1/instrument/counter/Time"></Time>
      </NXpositioner>
      <name></name>
      <NXsource name="source">
        <current type="NX_FLOAT64"></current>
        <frequency type="NX_FLOAT64"></frequency>
        <name></name>
        <notes></notes>
        <power type="NX_FLOAT64"></power>
        <probe></probe>
        <type></type>
        <voltage type="NX_FLOAT64"></voltage>
      </NXsource>
    </NXinstrument>
    <program_name></program_name>
    <scan_command></scan_command>
    <scan_dimensions type="NX_INT32"></scan_dimensions>
    <scan_identifier></scan_identifier>
    <NXuser name="user01">
      <username></username>
    </NXuser>
  </NXentry>
</NXroot>
```

5.5 Tomography

5.5.1 I13 - Tomography

```
<?xml version="1.0"?>
<NXroot NeXus_version="4.3.0" file_name="/dls/i13-1/data/2011/0-0/998.nxs"
 HDF5_Version="1.8.7" file_time="2011-12-15T00:31:50+00:00">
  <NXentry name="entry1">
    <NXdata name="default">
      <d4_i type="NX_FLOAT64[1431,9]" signal="1" target="/entry1/instrument/d4_i/d4_i"></d4_i>
      <qcm_bragg1 type="NX_FLOAT64[1431,9]" label="2" primary="1"
       axis="1,2" target="/entry1/instrument/qcm_bragg1/qcm_bragg1"></qcm_bragg1>
      <qcm_bragg1_pitch type="NX_FLOAT64[1431,9]" label="1" primary="1"
       axis="1,2" target="/entry1/instrument/qcm_bragg1_pitch/qcm_bragg1_pitch"></qcm_bragg1_pitch>
    </NXdata>
    <entry_identifier></entry_identifier>
    <NXinstrument name="instrument">
      <NXpositioner name="d4_i">
        <d4_i type="NX_FLOAT64[1431,9]" signal="1" target="/entry1/instrument/d4_i/d4_i"></d4_i>
      </NXpositioner>
      <name></name>
      <NXpositioner name="qcm_bragg1">
        <qcm_bragg1 type="NX_FLOAT64[1431,9]" label="2" primary="1"
         axis="1,2" target="/entry1/instrument/qcm_bragg1/qcm_bragg1"></qcm_bragg1>
      </NXpositioner>
      <NXpositioner name="qcm_bragg1_pitch">
        <qcm_bragg1_pitch type="NX_FLOAT64[1431,9]" label="1" primary="1"
         axis="1,2" target="/entry1/instrument/qcm_bragg1_pitch/qcm_bragg1_pitch"></qcm_bragg1_pitch>
      </NXpositioner>
      <NXsource name="source">
        <current type="NX_FLOAT64"></current>
        <frequency type="NX_FLOAT64"></frequency>
        <name></name>
        <notes></notes>
        <power type="NX_FLOAT64"></power>
        <probe></probe>
        <type></type>
        <voltage type="NX_FLOAT64"></voltage>
      </NXsource>
    </NXinstrument>
    <program_name></program_name>
    <scan_command></scan_command>
    <scan_dimensions type="NX_INT32[2]"></scan_dimensions>
    <scan_identifier></scan_identifier>
    <NXuser name="user01">
      <username></username>
    </NXuser>
  </NXentry>
</NXroot>
```

5.5.2 I12 example – Energy Dispersive Exafs Detector (Mapping Mode)

```
<?xml version="1.0"?>
<NXroot NeXus_version="4.3.0" file_name="/dls/i12/data/2012/cm5706-1/default/5043.nxs"
  HDF5_Version="1.8.7" file_time="2012-03-08T12:32:05+00:00">
 <NXentry name="entry1">
  <NXdata name="EDXD_Element_01">
   <data type="NX_FLOAT64[11,4096]" signal="1" units="counts"
    target="/entry1/instrument/EDXD_Element_01/data"></data>
   <edxd_energy_approx type="NX_FLOAT64[4096]" axis="3" primary="2"
    target="/entry1/instrument/EDXD_Element_01/edxd_energy_approx"
unit="keV"></edxd_energy_approx>
   <edxd_q type="NX_FLOAT64[4096]" axis="3" primary="1"
    target="/entry1/instrument/EDXD_Element_01/edxd_q" unit="units"></edxd_q>
   <t3_x type="NX_FLOAT64[11]" label="1" primary="1"
    target="/entry1/instrument/t3_x/t3_x" axis="1"></t3_x>
  </NXdata>
  <NXdata name="EDXD_Element_0n"> - repeated for n=2 to 23
   <data type="NX_FLOAT64[11,4096]" signal="1" units="counts"
    target="/entry1/instrument/EDXD_Element_02/data"></data>
   <edxd_energy_approx type="NX_FLOAT64[4096]" axis="3" primary="2"
    target="/entry1/instrument/EDXD_Element_02/edxd_energy_approx"
unit="keV"></edxd_energy_approx>
   <edxd_q type="NX_FLOAT64[4096]" axis="3" primary="1"
    target="/entry1/instrument/EDXD_Element_02/edxd_q" unit="units"></edxd_q>
   <t3_x type="NX_FLOAT64[11]" label="1" primary="1"
    target="/entry1/instrument/t3_x/t3_x" axis="1"></t3_x>
  </NXdata>
```

......

```
    <entry_identifier></entry_identifier>
    <NXinstrument name="instrument">
      <NXdetector name="EDXD_Element_01">
        <data type="NX_FLOAT64[11,4096]" signal="1" units="counts"
          target="/entry1/instrument/EDXD_Element_01/data"></data>
        <edxd_dead_time type="NX_FLOAT64[11]" unit="seconds"></edxd_dead_time>
        <edxd_dead_time_percent type="NX_FLOAT64[11]" unit="percent"></edxd_dead_time_percent>
        <edxd_energy_approx type="NX_FLOAT64[4096]" axis="3" primary="2"
          target="/entry1/instrument/EDXD_Element_01/edxd_energy_approx"
unit="keV"></edxd_energy_approx>
        <edxd_energy_live_time type="NX_FLOAT64[11]" unit="seconds"></edxd_energy_live_time>
        <edxd_events type="NX_INT32[11]" unit="counts"></edxd_events>
        <edxd_input_count_rate type="NX_FLOAT64[11]" unit="counts/second"></edxd_input_count_rate>
        <edxd_output_count_rate type="NX_FLOAT64[11]" unit="counts/second"></edxd_output_count_rate>
        <edxd_q type="NX_FLOAT64[4096]" axis="3" primary="1"
          target="/entry1/instrument/EDXD_Element_01/edxd_q" unit="units"></edxd_q>
        <edxd_real_time type="NX_FLOAT64[11]" unit="seconds"></edxd_real_time>
        <edxd_trigger_live_time type="NX_FLOAT64[11]" unit="seconds"></edxd_trigger_live_time>
      </NXdetector>
      <NXdetector name="EDXD_Element_0n"> - repeated for n=2 to 23
        <data type="NX_FLOAT64[11,4096]" signal="1" units="counts"
          target="/entry1/instrument/EDXD_Element_02/data"></data>
        <edxd_dead_time type="NX_FLOAT64[11]" unit="seconds"></edxd_dead_time>
        <edxd_dead_time_percent type="NX_FLOAT64[11]" unit="percent"></edxd_dead_time_percent>
        <edxd_energy_approx type="NX_FLOAT64[4096]" axis="3" primary="2"
          target="/entry1/instrument/EDXD_Element_02/edxd_energy_approx"
unit="keV"></edxd_energy_approx>
        <edxd_energy_live_time type="NX_FLOAT64[11]" unit="seconds"></edxd_energy_live_time>
        <edxd_events type="NX_INT32[11]" unit="counts"></edxd_events>
        <edxd_input_count_rate type="NX_FLOAT64[11]" unit="counts/second"></edxd_input_count_rate>
        <edxd_output_count_rate type="NX_FLOAT64[11]" unit="counts/second"></edxd_output_count_rate>
        <edxd_q type="NX_FLOAT64[4096]" axis="3" primary="1"
          target="/entry1/instrument/EDXD_Element_02/edxd_q" unit="units"></edxd_q>
        <edxd_real_time type="NX_FLOAT64[11]" unit="seconds"></edxd_real_time>
        <edxd_trigger_live_time type="NX_FLOAT64[11]" unit="seconds"></edxd_trigger_live_time>
      </NXdetector>
```

....

```
      <name></name>
      <NXsource name="source">
        <current type="NX_FLOAT64"></current>
        <frequency type="NX_FLOAT64"></frequency>
        <name></name>
        <notes></notes>
        <power type="NX_FLOAT64"></power>
        <probe></probe>
        <type></type>
        <voltage type="NX_FLOAT64"></voltage>
      </NXsource>
      <NXpositioner name="t3_x">
        <t3_x type="NX_FLOAT64[11]" label="1" primary="1"
          target="/entry1/instrument/t3_x/t3_x" axis="1"></t3_x>
      </NXpositioner>
    </NXinstrument>
    <program_name></program_name>
    <scan_command></scan_command>
    <scan_dimensions type="NX_INT32"></scan_dimensions>
    <scan_identifier></scan_identifier>
    <title></title>
    <NXuser name="user01">
      <username></username>
    </NXuser>
  </NXentry>
</NXroot>
```