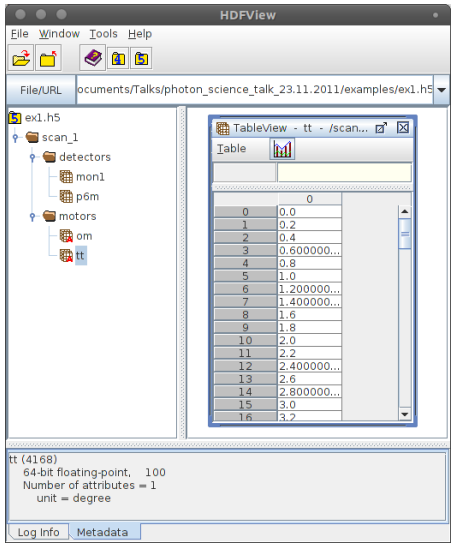
The HDRI Nexus C++API

Eugen Wintersberger
The HDRI Nexus C++ API
DESY, 27.02.2012

The future of data storage at PNI facilities ….

HDF5

- Binary data format (performance)
- File system like data organization
- Metadata as attributes to data objects
- Platform independent



NEXUS

- Adds semantics to HDF5 objects
- Standardized keywords making data searchable and indexable

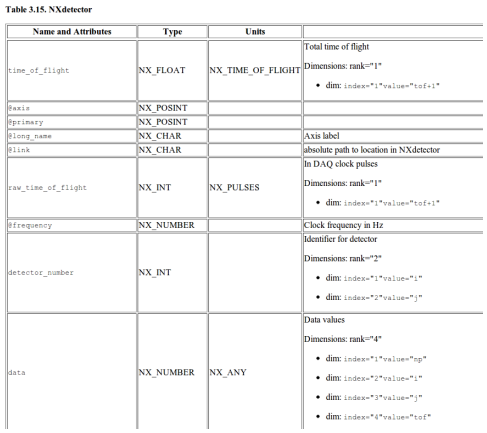


Table 3.15. NXdetector

Name and Attributes	Type	Units	
time_of_flight	NX_FLOAT	NX_TIME_OF_FLIGHT	Total time of flight Dimensions: rank="1" • dim: index="1"value="tof+1"
@axis	NX_POSINT		
@primary	NX_POSINT		
@long_name	NX_CHAR		Axis label
@link	NX_CHAR		absolute path to location in NXdetector
raw_time_of_flight	NX_INT	NX_PULSES	In DAQ clock pulses Dimensions: rank="1" • dim: index="1"value="tof+1"
@frequency	NX_NUMBER		Clock frequency in Hz
detector_number	NX_INT		Identifier for detector Dimensions: rank="2" • dim: index="1"value="i" • dim: index="2"value="j"
data	NX_NUMBER	NX_ANY	Data values Dimensions: rank="4" • dim: index="1"value="np" • dim: index="2"value="i" • dim: index="3"value="j" • dim: index="4"value="tof"

DESY

How do we get Nexus files ….

Nexus C/C++ API (NAPI) developed by the Nexus group itself:

➔Old style C/C++ interface (a lot of pointers and void)

➔Not very comfortable to use

➔Hard to get permanent handlers to individual objects

➔The concept is that of a file system!

Use HDF5 directly:

➔The API is pretty complex

➔If the C++ interface shall be used there is no thread safety

There must be a better solution!

The HDRI Nexus C++ API ...

Due to the inconveniences of the NAPI we decided to implement our own C++ API!

Design goals:

➔Thread safety

➔Exception safety **=> use only first class objects!**

➔High performance

➔Type safety (no need for void)

➔Should be easy to maintain

➔Stick more to C++ idioms

Item 18: Make interfaces easy to use correctly and hard to use incorrectly.

Scott Meyers, Effective C++

In order to satisfy this requirements we made some restrictions:

➔Support only HDF5

➔HDF5 must be compiled with thread-safety

➔Needed to make some additions to Nexus binary standard

➔Actually no support for non-object oriented languages

DESY

Supported datatypes ...

Integer types

Name	Size (bit)	Description
Int8/UInt8	8	Signed/unsigned
Int16/UInt16	16	Signed/unsigned
Int32/UInt32	32	Signed/unsigned
Int64/UInt64	64	Signed/unsigned

Floating point types

Name	Size (bit)
Float32	32
Float64	64
Float128	128
Complex32	64
Complex64	128
Complex128	256

String type

Use std::string<char> as string type.
Encoded as UTF8 in the HDF5 file.

The complex types are instances of the std::complex<T> template!

Binary type

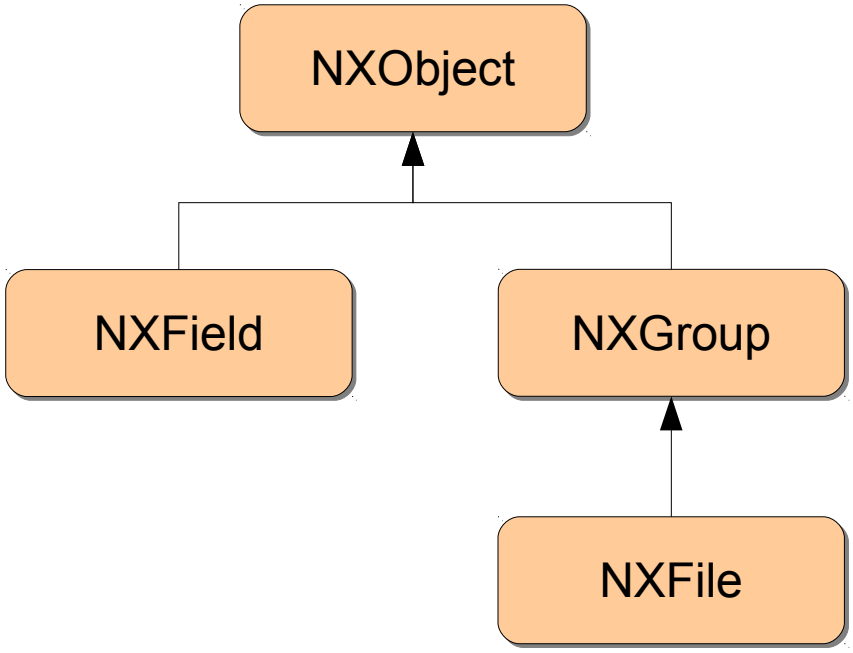
```
template<typename T>
class BinaryType
{
        ..........
};
```

Binary data is not handled as a simple `typdef` to `unsigned char` but is a compatible new type. In HDF5 this is stored as H5T_OPAQUE!

Class structure …. (only the front end)

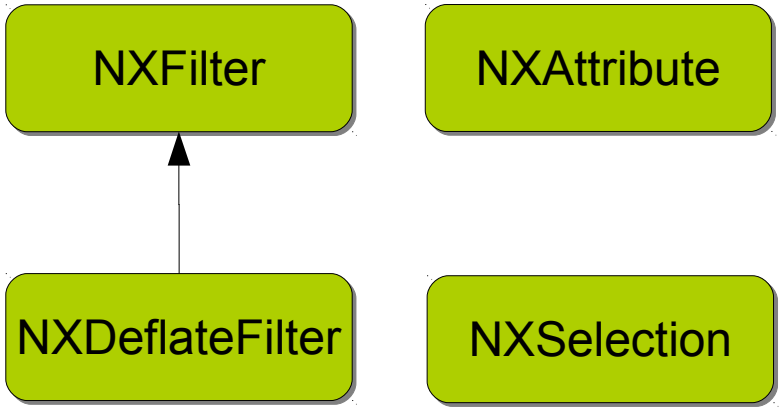
Keep things as simple as possible for the API user!

Fundamental classes



Classes assembling the Nexus tree.

Utility classes



Classes used to make data access more convenient and simple.

Classes: behind the scenes ...

Keep user interface independent of the concrete implementation:

Bridge Pattern

E. Gamma, R. Helm, R. Johnson, and J. Vlissides:
Design Patterns Elements of Reusable Object-Oriented Software

```
template<typename ImpT>
class NXObject
{
    private:
        ImpT _imp;
    public:
        …
};
```

Use a special implementation of the bridge pattern based on templates.

D. Vandevoorde and N. M. Josuttis:
C++ Templates: The Complete Guide

This special implementation does not require a pointer (which is not a first class object) to the concrete implementation → we should keep exception safety into account.

As a user you usually do not need to care about such things!

Groups, Files, and Links ...

- ➔Files are created by factory methods (`open_file`,`create_file`) of NXFile
- ➔Groups can be created either by `NXFile` or `NXGroup` (`create_group`)
- ➔External as well as internal links can be created (`link(...)`)

Creating files and groups

```
NXFile file = NXFile::create_file("test1.h5",true,0);

NXGroup g = file.create_group("scan_1","NXentry");

g.create_group("instrument","NXinstrument");
```

Creating external and internal links

```
g = file["/scan_1/instrument"];

//create an internal link
g.link("/link_to_instrument");

//create an external link
g.link("pilatus_data.h5:/detector","detector");
```


DESY

The concepts behind NXField ...

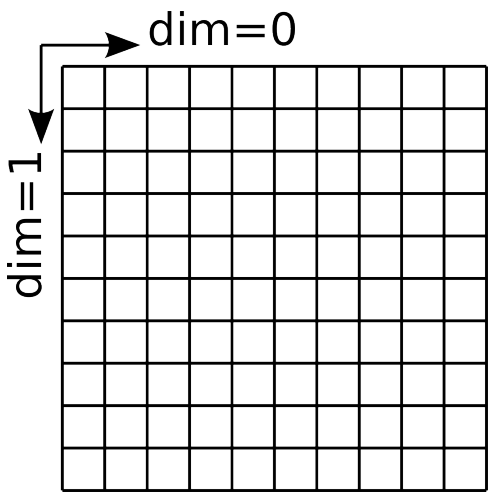
Consider NXField as a multidimensional array of a particular datatype!

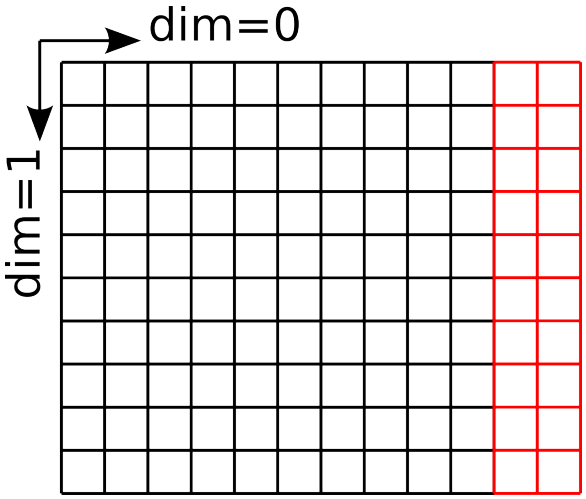
```
NXGroup g = …;

//datatype and number of dimensions fixed at creation
NXField f = g.create_field<Float32>("data",{512,1024});

Float64Array a({512,1024},...);
f.write(a); //write data
f.read(a);  //read data
```

The rank of a NXField is fixed at creation time, however the field can grow:






```
f.grow(0,2);
```

Grow a field by 2 elements along dimension 0.

DESY

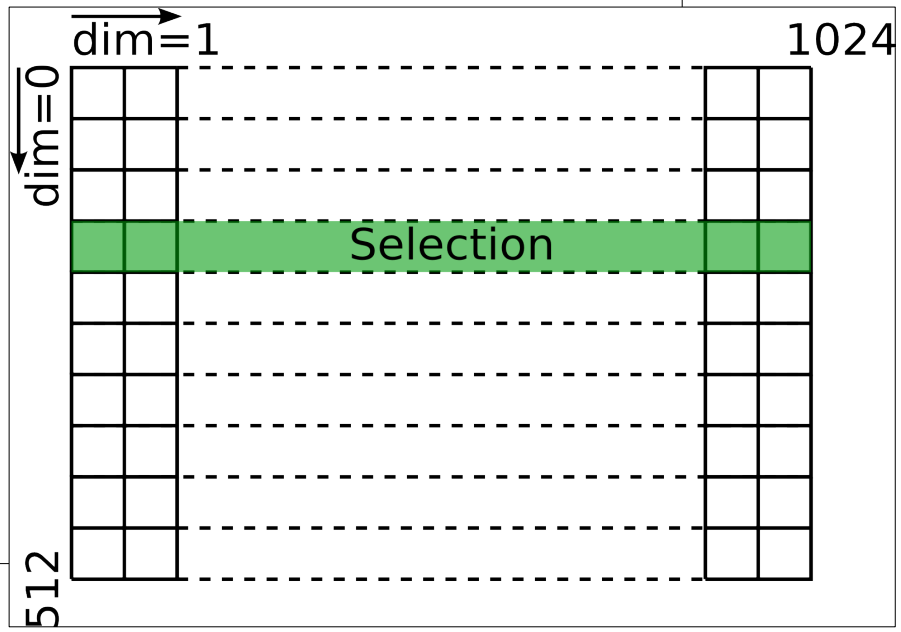
NXSelection – sometimes not everything is important

`NXSelection` allows you to read only a small portion of the data stored in a field!

```
NXField field = g.create_field<Float32>("data",{512,1024});
NXSelection sel = field.selection();

//setup the selection object
sel.offset({0,0});
sel.stride({1,1});
sel.count({1,1024});
Float64Array data({1024},...);

for(size_t i=0;i<512;i++)
{
    data = …;
    sel.write(data);
}
```



Selections can be created only via the `selection()` factory method of an NXField object. The reason is that they hold an internal reference to this field and are thus closely connected to the dataset from which they stem.

DESY

A very simple example – writing data

That kind of code could appear in a TANGO server for an area detector:

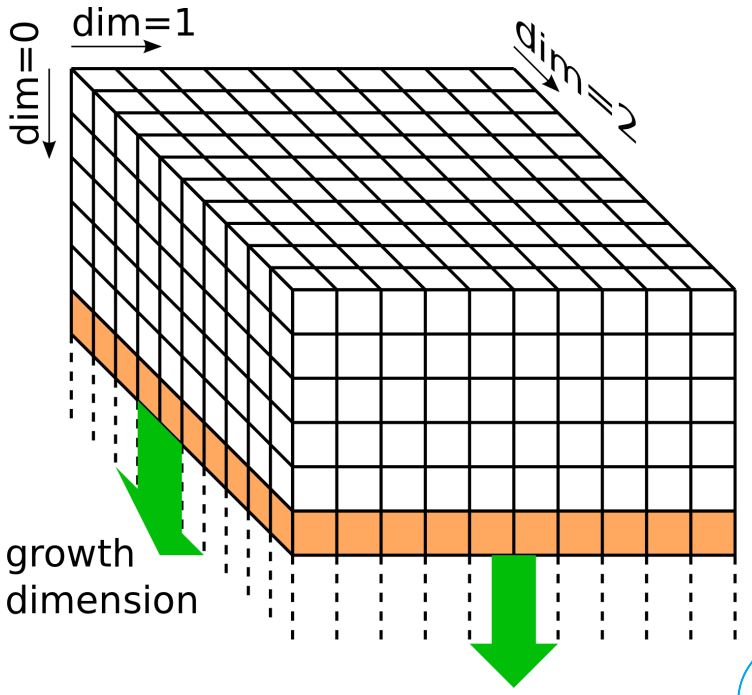
```
NXFile f = NXFile::create_file("detector.h5",true,0);
UInt32Array frame({1024,1024},...);

//create detector group
NXGroup g = f.create_group("/scan_1/instrument/detector","NXdetector");

//create field
NXField data = g.create_field<UInt32>("data",{0,1024,1024});

//create selection
NXSelection fsel = data.selection();
fsel.count({1,1024,1024});
fsel.stride({1,1,1});

//write data to file
for(size_t i=0;i<NP;i++){
    frame = … ; //get frame data
    data.grow(0);
    fsel.offset({i,0,0});
    fsel.write(frame);
}
```



A very simple example – reading data

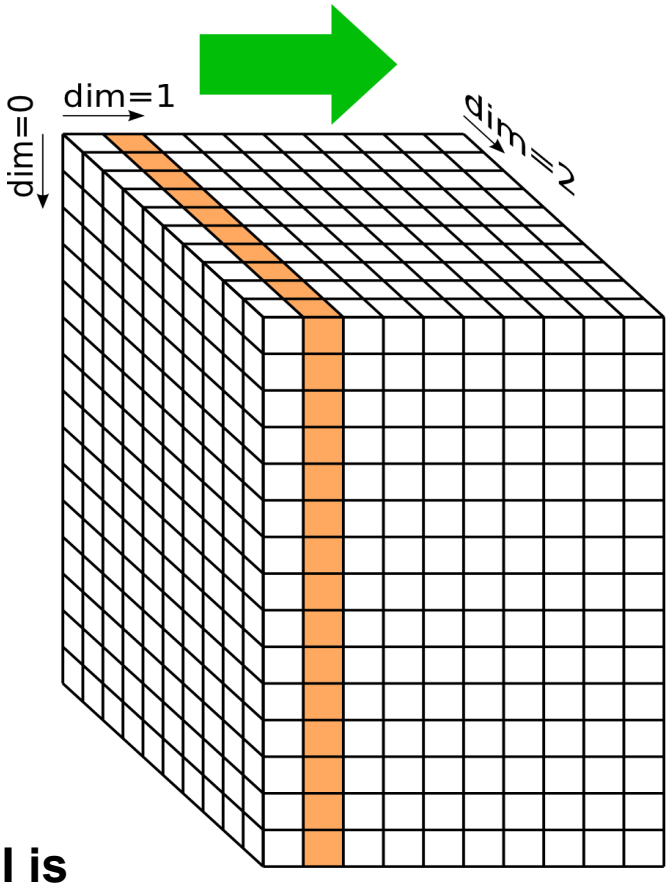
Something like this you may have in an analysis program

```
NXFile f = NXFile::open_file("detector.h5");
Float64Array slice({6000,1024},...);
size_t NP = 6000;

//open data field
NXField data = f["/scan_1/instrument/"
                 "detector/data"];

//create selection
NXSelection fsel = data.selection();
fsel.count({NP,1,1024});
fsel.stride({1,1,1});

//write data to file
for(size_t i=0;i<1024;i++){
    fsel.offset({0,i,0});
    fsel.read(slice);
    //do some processing here
}
```



The last two examples have shown that the API is pretty simple to use – even from C++!

DESY

Performance of the API ...

Remember: High performance was a design goal!

Benchmark scenario:

➔Write 4800 frames of size 1k x 1k of datatype UInt16 to a file

➔Harddrive: 1TByte SATA3

➔RAM-Disk: 10GByte tmpfs partition

	Frames per second	Transfer rate (Mbyte/s)
Ram-Disk		
libpninx	3200	6500
Native HDF5	3300	6600
SATA - Drive		
libpninx	53	107
Native HDF5	57	115

Even without optimization of program flow the overhead produced by the API is negligible!

Status of the API ...

- ➔ APIs interface is stable
- ➔ Doxygen documentation is at 90%
- ➔ Users-Guide and Developers-Guide are on their way
- ➔ Unit tests for all available data-types show no serious problems
- ➔ Build tested on Debian/Ubuntu 64Bit with g++ 4.4, 4.5,4.6
- ➔ Currently HDF5 1.8.4 (SO_VERSION=6) is supported

The API is ready to use – best time for early adaptors!

Get the code
For libpniutils: https://sourceforge.net/projects/libpniutils/
For libpninx: https://sourceforge.net/projects/libpninx/

DESY

Outlook and Todo ...

➔**Implement python interface!**

➔Implement locking mechanism for thread safety

➔Windows support

➔Implement a better method for group creation

```
NXGroup g = f.create_group("scan_1:NXentry/instrument:NXinstrument")
```

➔STL container decorators for NXField

```
NXListDecorator decorator(0,field);
Float32Array data(...);
Decorator.append(data);
…
for(Float32Array &a: decorator){
    …
}
```

Implement configuration utilities and classes. However it is questionable if these things should go into the API – they are maybe too specific to a particular facility.

HDF5 issues

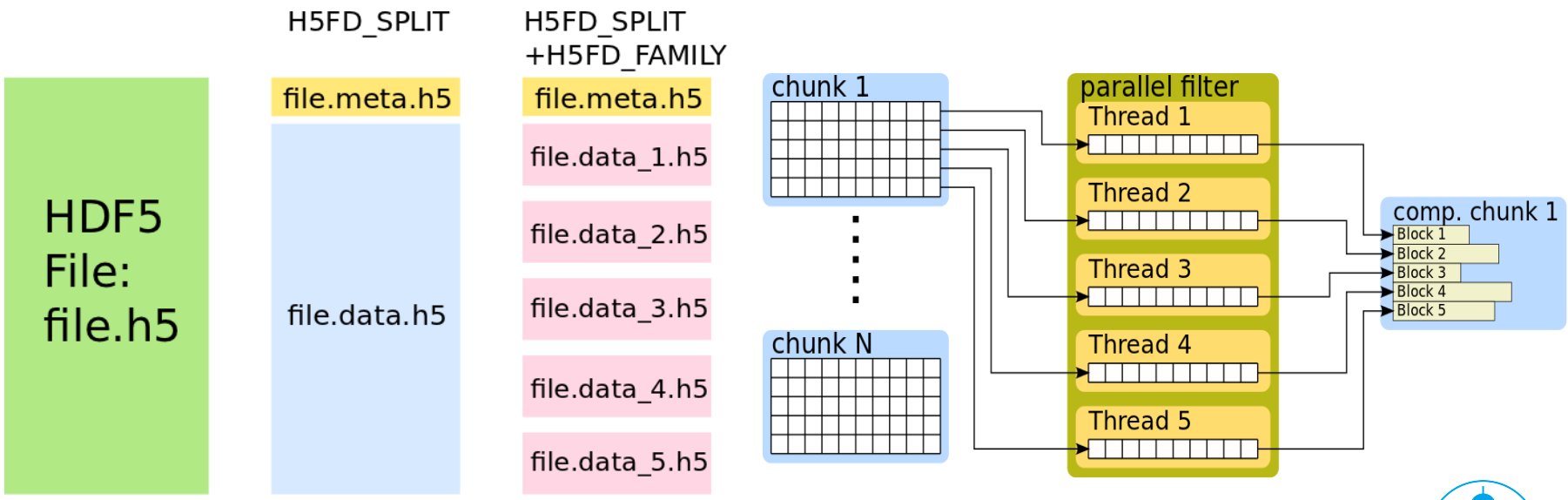
We established contact with the HDF5 group to fix two major issues with HDF5:

➔ Improve archiving features of HDF5

➔ General interface for external filters

Archiving:
Store meta-data and payload in separate files where the later one shall be split too.

External filters:
Parallel, 3rd party filter code shall improve compression performance.



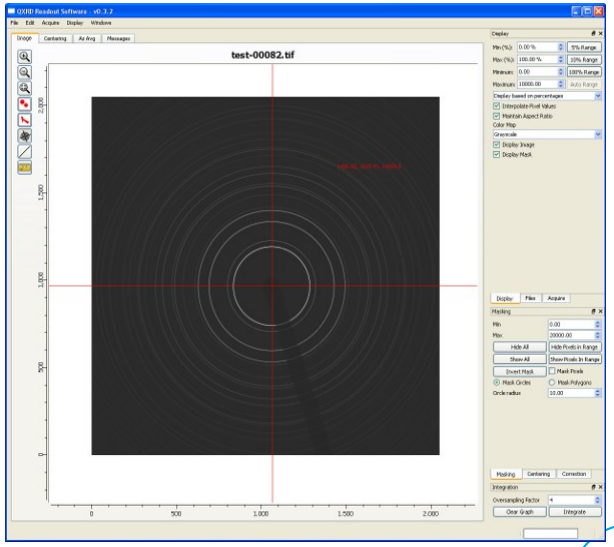
Native Nexus support by detectors and software ...

DECTRIS

DECTRIS has announced native support for Nexus for Pilatus and future Aiger detectors most probably as of 2013.

Nexus for QXRD

We use QXRD as a readout software for PerkinElmer area detectors. The data is actually written to TIF files. However, QXRDs main developer already agreed to help us with integrating Nexus to the software.
Planned for 2012/2013.



DESY

Thank you for your attention...