The Measurement Context
a.k.a The Hybrid Approach

A simple, expressive HDF5 hierarchy and API for data acquisition and analysis

Darren Dale
Staff Scientist, Cornell High Energy Synchrotron Source (CHESS)

Opening Thought

- The general problem we are considering is how to develop **abstractions** to describe **data** in different **contexts**
  - We are here to discuss a format to express abstractions that describe data in context of hyperspectral analyses
  - NeXus seems to develop abstractions in the context of an instrument (originally conceived to describe neutron scattering instruments)
  - Armando and I worked on a “hybrid” approach to help us bridge the gap between the spec format and NeXus: a measurement context
    - Not a full instrument definition, only what is relevant to the current measurement
  - Measurement/Instrument and Analyses contexts together should provide a solution to the general problem
    - Let's keep this in the back of our mind as we consider formats for data exchange and analysis

CHESS

- Small facility funded by the National Science Foundation
- Located on the Cornell Campus
- 6 beamlines
  - 2 wiggler IDs feeding 3 beamlines, 7 endstations
  - 4 bending magnets feed 4 endstations
- 16 research associates (11 beamline scientists)
- ~23 support staff (1 dedicated software engineer)
- Technique development
  - Frequently assemble experiments in an "empty hutch" at four endstations
  - Require flexible solutions for data acquisition and data storage

Data Acquisition: Spec and EPICS

- Heavily reliant on the Spec data acquisition program
  - Sequencer with command line interface
  - Extensible via C-like macros
  - Primary interface to hardware
- Spec produces data in ASCII files
  - Multiple scans in a single file
  - File and scan headers contain motor names and positions, additional (arbitrary) metadata
  - “Table” containing scalar data (independent and dependent)
  - Vector data (e.g. MCA) interleaved with scalar data
- Use Spec to interface with EPICS-controlled devices (new to CHESS)
  - One “spec file” per device, per scan, formatted as above

Consider: Scanning X-ray Fluorescence Microscopy

- Users need real-time analysis and feedback
  - Fit fluorescence spectrum for each pixel during a scan, update a false-color element map during acquisition
  - Select one or more pixels from the map to see the spectrum and fit
  - Identify detector problems or artifacts early
  - Users leave CHESS with processed data
- Desire a more capable file format
  - Binary (Spec ASCII format is inefficient)
  - Hierarchical (convey context)
  - Accessible/cross-platform/open-source
  - Pervasive/authoritative
  - Began using HDF5 3 years ago

Thoughts on NeXus Format

- Merits:
  - Developing standard abstractions
  - Use hierarchy to organize abstractions and relationships (context)
- Inherent frustration between objectives: flexibility, standardization
  - Not specific to NeXus, but a problem we need to address
- Concerns:
  - Ambitious / not enough resources
  - Direction: disappointed with recent focus on XML
  - Difficulty finding example data files that pass NeXus validation tests
- CHESS is considering NeXus, but for now we need something a little closer to our existing approach

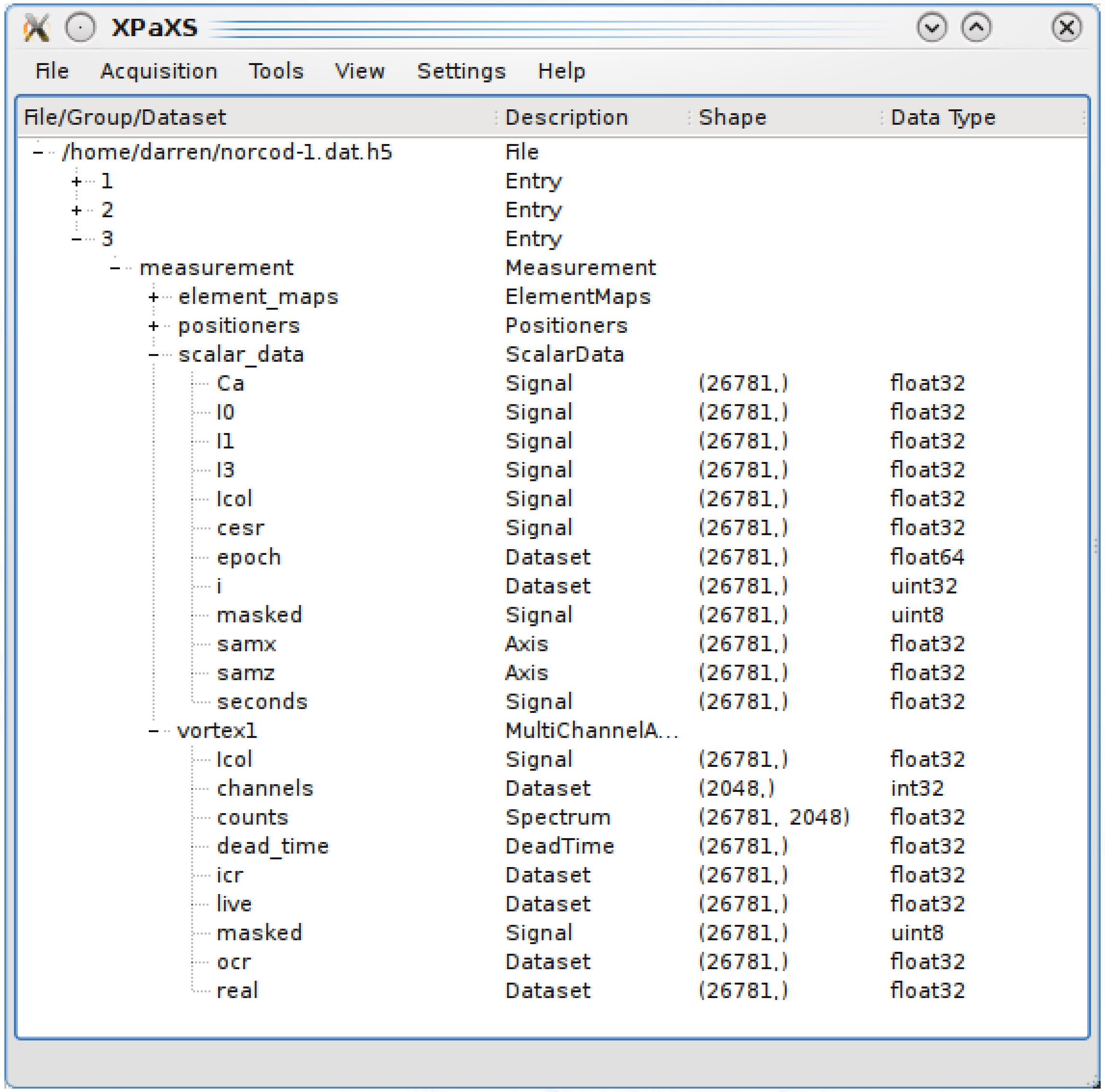
The Hybrid Approach (or Measurement Context)

- Take the Spec data file(s) organization and cast it into a "measurement" group located in an NXentry
  - Allows easy conversion of existing spec data to hdf5
  - Flexible to support rapidly changing instruments and data acquisition environments
  - Ability to build a standard nexus hierarchy alongside the measurement group – provides an upgrade path
- Provide metadata so user/analysis program can work with data in the context of the measurement (not the entire instrument):
  - Primary signals, monitor
  - Principle scan axes, ranges
  - Skipped scan points

Subgroups of Measurement Group

- ScalarData
  - 1-dimensional arrays (even for higher dimensional scans)
  - Axes (configurables)
  - Signals (responses)
- Positioners (probably should be called configuration)
  - Starting positions and configurations stored in spec scan header
- Devices (zero or more groups representing MCAs, CCDs, etc.)
  - Properties (channels)
  - Scalar data (dead time)
  - Vector (spectrum), array (image), or higher-dimensionality data
  - Calibration information and other metadata

Example data



File/Group/Dataset	Description	Shape	Data Type
/home/darren/norcod-1.dat.h5	File		
1	Entry		
2	Entry		
3	Entry		
measurement	Measurement		
element_maps	ElementMaps		
positioners	Positioners		
scalar_data	ScalarData		
Ca	Signal	(26781,)	float32
I0	Signal	(26781,)	float32
I1	Signal	(26781,)	float32
I3	Signal	(26781,)	float32
Icol	Signal	(26781,)	float32
cesr	Signal	(26781,)	float32
epoch	Dataset	(26781,)	float64
i	Dataset	(26781,)	uint32
masked	Signal	(26781,)	uint8
samx	Axis	(26781,)	float32
samz	Axis	(26781,)	float32
seconds	Signal	(26781,)	float32
vortex1	MultiChannelA...		
Icol	Signal	(26781,)	float32
channels	Dataset	(2048,)	int32
counts	Spectrum	(26781, 2048)	float32
dead_time	DeadTime	(26781,)	float32
icr	Dataset	(26781,)	float32
live	Dataset	(26781,)	float32
masked	Signal	(26781,)	uint8
ocr	Dataset	(26781,)	float32
real	Dataset	(26781,)	float32

Data acquisition

- Spec Client/Server mode
  - Spec still produces (large) ASCII files
  - Register variables to be broadcast to client programs
- SpecClient (Python Library and spec macros written by Mattias Guijarro at ESRF)
  - Contributed spec macros to broadcast arbitrarily structured data hierarchies to clients
  - Contributed patch for python clients to receive and assemble the data into standard python data structures
  - Clients receive data after each point in a scan, can analyze and display processed data
    - needs to be improved by developing analysis contexts

Example of Spec Macros for Broadcasting

```
# at scan start, specify hierarchy:
# python_repr produces a python string representation of spec data

# define the MCA group:
local i kwargs[]
kwargs["calibration"] = python_repr(python_repr(CAL))
kwargs["monitor"] = python_repr(cnt_name(MON))
client_set_scan_env("vortex1", "MultiChannelAnalyzer", kwargs)

# define the McaSpectrum  dataset:
for (i in kwargs) delete kwargs[i]
local array shape[2]
shape[0] = SC_NPOINTS; shape[1] = uch
kwargs["shape"] = python_repr(shape)
kwargs["dtype"] = python_repr("f")
client_set_scan_env("vortex1/counts", "Spectrum", kwargs)

# define deadtime dataset:
for (i in kwargs) delete kwargs[i]
local array shape[1]
shape[0] = SC_NPOINTS
kwargs["shape"] = python_repr(shape)
kwargs["dtype"] = python_repr("f")
kwargs["units"] = python_repr("percent")
kwargs["dead_time_format"] = python_repr("percent")
client_set_scan_env("vortex1/deadtime", "DeadTime", kwargs)
```

```
# at each scan point, broadcast new data:
client_set_data("vortex1/dead_time", dead)
client_set_data("vortex1/counts", MCA_DATA[1])
```

• SpecClient automatically sets up the entry and the measurement context (including scalar_data and positioners)
• SpecClient automatically broadcasts scalar data specified in spec config file
• Configurable for any additional information that needs to be broadcast

Example Python Code for Receiving and processing data

```
class QtSpecScanA(SpecScan.SpecScanA, QtCore.QObject):

    def newScan(self, scanParameters):
        tree = scanParameters['phynx']
        info = tree.pop('info')
        h5File = self.getFile(info)
        # create the entry
        entry = h5File.create_entry(name, **info) # info contains metadata
        measurement = entry.measurement
        # create all the groups under measurement, defined by clientutils:
        for key in sorted(tree.keys()):
            type, info = tree.pop(key)
            phynx.registry[type](measurement, key, create=True, **info)

    def newScanData(self, scanData):
        i = scanData['scalar_data/i']
        m = self._scanData.measurement
        for k, val in scanData.iteritems():
            m[k][i] = val
            m[k].acquired = i + 1
```

The Importance of the Interface

- The interface to data abstractions/context is critically important
  - Interface can help encourage conformance to a standard
  - Abstract base classes can define intended behavior of interface
- Required an intuitive, but flexible and powerful interface
  - Easily used by experimenters, interactively or in scripts
  - Powerful enough to meet the needs of application developers
- Considered:
  - Pytables: Used for a year, intuitive interface, not thread safe, no support (at the time) for hdf5 links, stores python-specific data
  - NeXus: some good ideas (establish standard hierarchy, metadata), but required a more intuitive interface than what is provided by API
  - Decided to build upon h5py: really elegant interface and implementation, threadsafe, support for hdf5 links

Phynx: An intuitive high-level interface

- Originally intended to be a higher-level, more accessible interface to hdf5 files containing a NeXus hierarchy
  - Intentions discussed on the NeXus mailing list. NeXus requests:
    - Do not advertise as official python bindings to NeXus
    - Do not introduce new NX_classes
- Determine high-level abstractions (implementations) from context and metadata
- Provide a high-level interface via subclasses and data proxies
  - May need more specificity than NX_class can provide (new NX_class definitions require approval from the NeXus committee)
  - Pipe dream: Would be nice if we all used the same interpreted language (Python), then the high-level implementation for particular data abstractions could be provided along with the data

Phynx: Implementation

- Wrap h5py File, Group and Dataset classes
  - Take these containers and build abstractions to give them some personality, functionality, based on context
    - ED-MCA: could be subclass of Detector
    - MCA counts: subclass of Dataset, with data proxy that corrects for monitor counts and deadtime
    - Deadtime: contains data proxies so deadtime can be expressed in numerous ways (percent, correction, normalization, etc.)
- Users interact with instances of Phynx classes, using associated GUI views or interactive interpreter
  - Started with Qt4 GUI tree widget to navigate HDF5 file, extended in PyMca

Phynx: Examples of high-level capabilities in the measurement context

- Automatically identify scan type and default plot
  - Sorted axes (fast, slow axes)
  - Axes ranges (establish image bounds)
  - Signals and priority (monitor, skipped points, primary detector)
- Automatically identify MCAs or other devices
- Iterate over (non-skipped) data points
  - Know when to pause iteration (during acquisition, awaiting data)
- Compare raw and corrected data
  - Alternative views of the same data, using proxies
- Reshape arrays to the dimensions of the scan (e.g. false color maps)

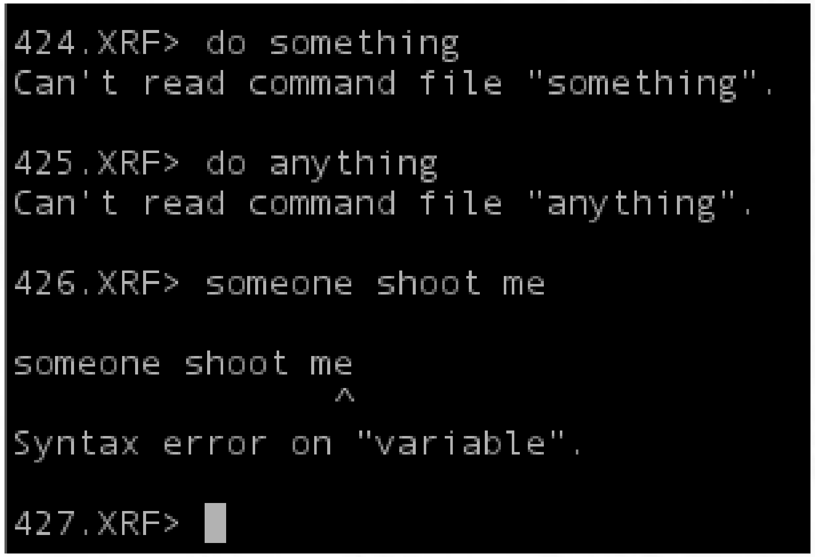
Extending Phynx: defining new classes

- "NX_class" attribute is reserved for NeXus-approved abstractions (data containers)
- Phynx uses "class" to specify an implementation class
  - provides a way to identify more specialized interfaces to data abstractions without abusing "NX_class"
  - When opening a group or dataset, the "class" attribute is checked to identify what constructor to use, defaults to Group or Dataset
    - The "class" attribute contains a string, like "MultiChannelAnalyzer"
    - String is a key associated with the appropriate class in a registry
    - Subclassing phynx.Group or phynx.Dataset automatically registers the key (the class name) and the class itself in the registry
- Phynx can be extended by third parties (contributions also welcome)

Putting it all together: Extensible Packages for X-ray Science

- GUI based on Qt4
- Phynx interface to hdf5, including HDF5 file navigation
- Use SpecClient to interact with equipment, run scans
- Interact with data while acquisition and/or analysis is in progress
  - Real-time or offline fluorescence analysis using PyMca
  - Threaded task manager spawns additional processes (using SMP or computer clusters) to process data in parallel
  - h5py/phynx are threadsafe, but...
  - Cpython's multithreading capabilities are limited by the existence of the Global Interpreter Lock
- In use at CHESS/F3 for 2 years
- Considering significant refactor using Enthought Tool Suite

XPaXS



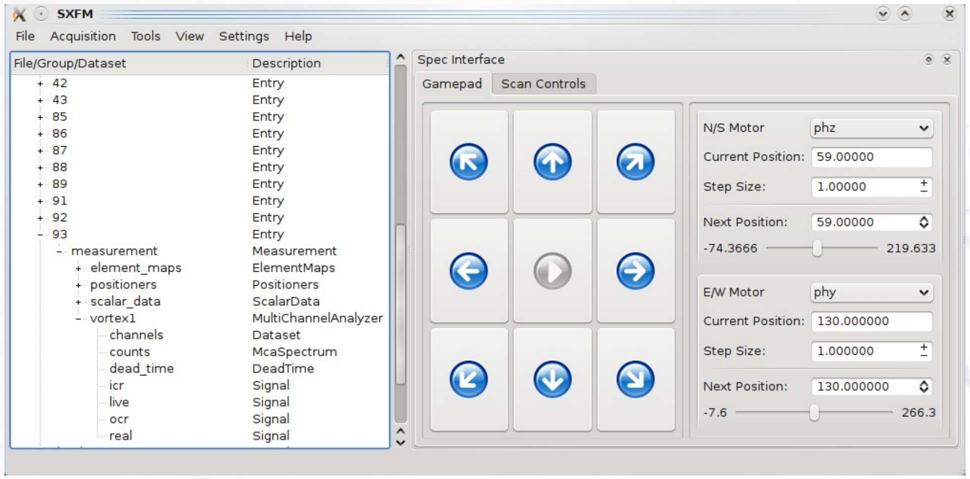
```
424.XRF> do something
Can't read command file "something".

425.XRF> do anything
Can't read command file "anything".

426.XRF> someone shoot me

someone shoot me
               ^
Syntax error on "variable".

427.XRF> █
```



SXFM

File Acquisition Tools View Settings Help

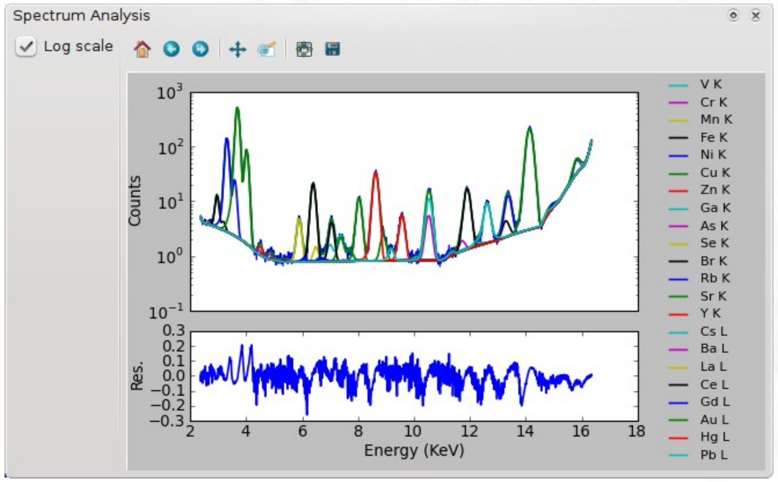
File/Group/Dataset	Description
+ 42	Entry
+ 43	Entry
+ 85	Entry
+ 86	Entry
+ 87	Entry
+ 88	Entry
+ 89	Entry
+ 91	Entry
+ 92	Entry
- 93	Entry
- measurement	Measurement
+ element_maps	ElementMaps
+ positioners	Positioners
+ scalar_data	ScalarData
- vortex1	MultiChannelAnalyzer
channels	Dataset
counts	McaSpectrum
dead_time	DeadTime
icr	Signal
live	Signal
ocr	Signal
real	Signal

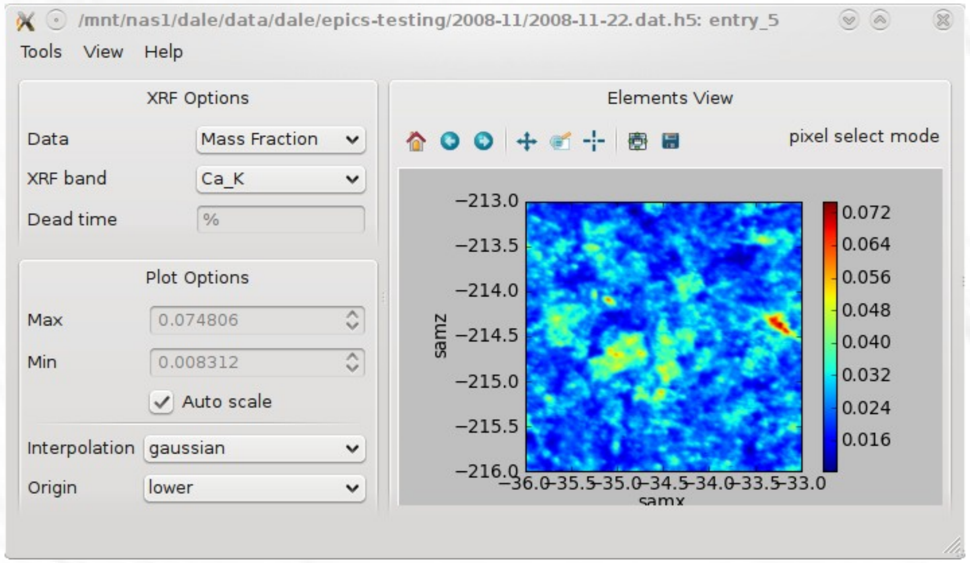
Spec Interface

Gamepad | Scan Controls

N/S Motor: phz
Current Position: 59.00000
Step Size: 1.00000
Next Position: 59.00000
-74.3666 — 219.633

E/W Motor: phy
Current Position: 130.000000
Step Size: 1.000000
Next Position: 130.000000
-7.6 — 266.3





/mnt/nas1/dale/data/dale/epics-testing/2008-11/2008-11-22.dat.h5: entry_5

Tools View Help

XRF Options

Data: Mass Fraction
XRF band: Ca_K
Dead time: %

Plot Options

Max: 0.074806
Min: 0.008312
✓ Auto scale
Interpolation: gaussian
Origin: lower

Where to go from here?

- Possibility of supporting measurement context in NeXus?
- Possibility of specifying metadata in NeXus so the high-level interfaces can identify appropriate subclass implementations?
- Analysis contexts in NeXus?
- Phynx: Return unit-aware Quantity values (see quantities at PyPI)
- Your input is crucial
  - Phynx is hosted in a DVCS at launchpad (will probably move to git/github in the near future)
  - What are your concerns?
  - What do you consider the strengths and weaknesses of this approach?
  - What would be required to suit your needs?

Summary

- The measurement context fits with existing data acquisition schemes
  - Easily convert spec data files
  - Contains sufficient metadata for applications to determine data abstractions and relationships in the context of the measurement
    - Compatible with simple real-time and offline analysis
    - Provides an upgrade path from spec to NeXus hierarchy
    - Couple with instrument/analysis contexts for more general solution
    - Can help bridge the acquisition format/exchange format divide
- The interface is crucial to the success of any format
  - Abstract base classes can specify intended behavior of an interface
  - Help encourage conformance to some format standard
- Phynx provides an intuitive interface to HDF5 (NeXus) data
  - Simple implementation, should be portable to other languages