Recent Developments in the Scanning X-ray Fluorescence Microscopy Capabilities at CHESS

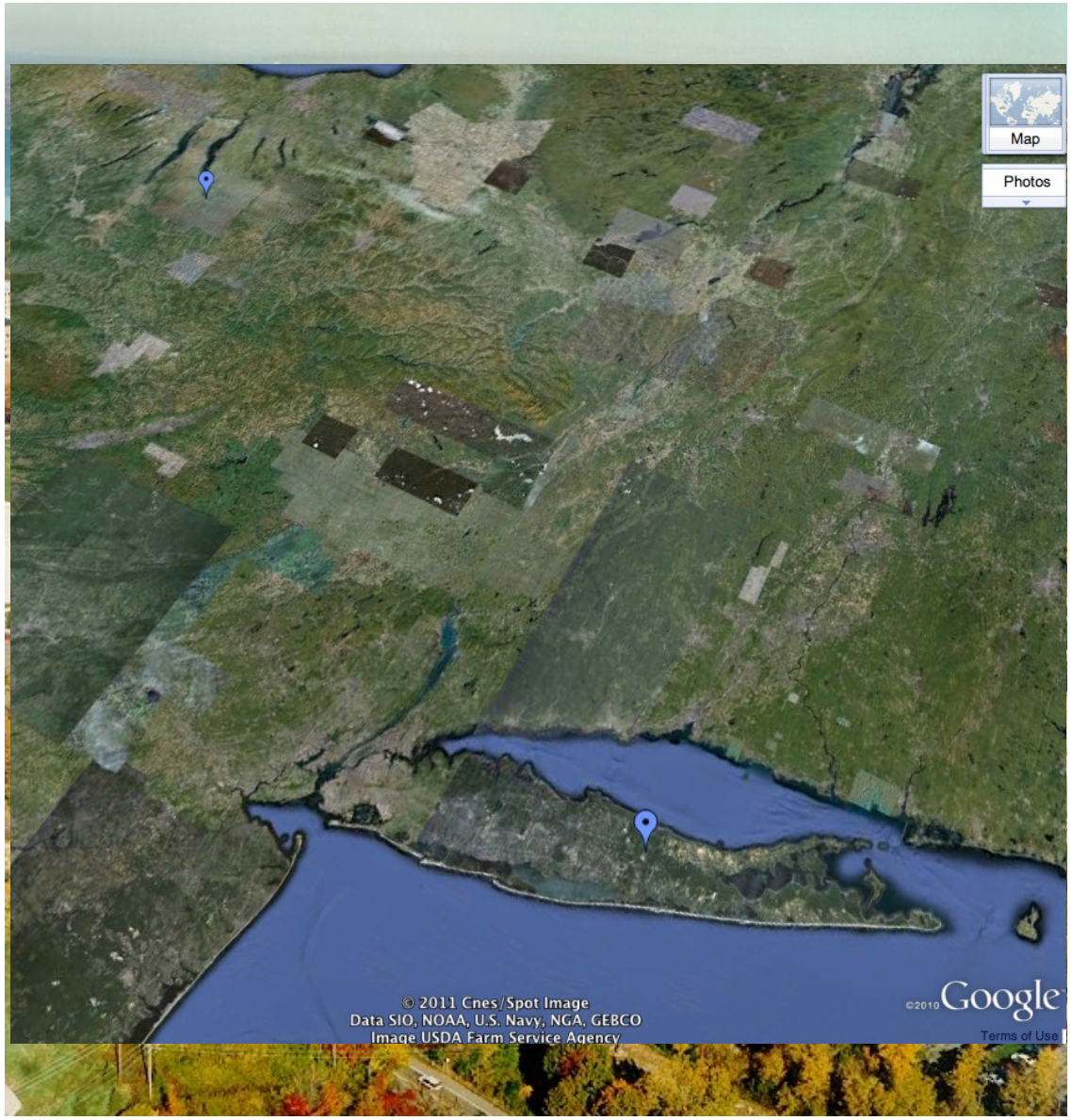
Darren Dale

Arthur Woll, Don Bilderback

Cornell High Energy Synchrotron Source (CHESS)

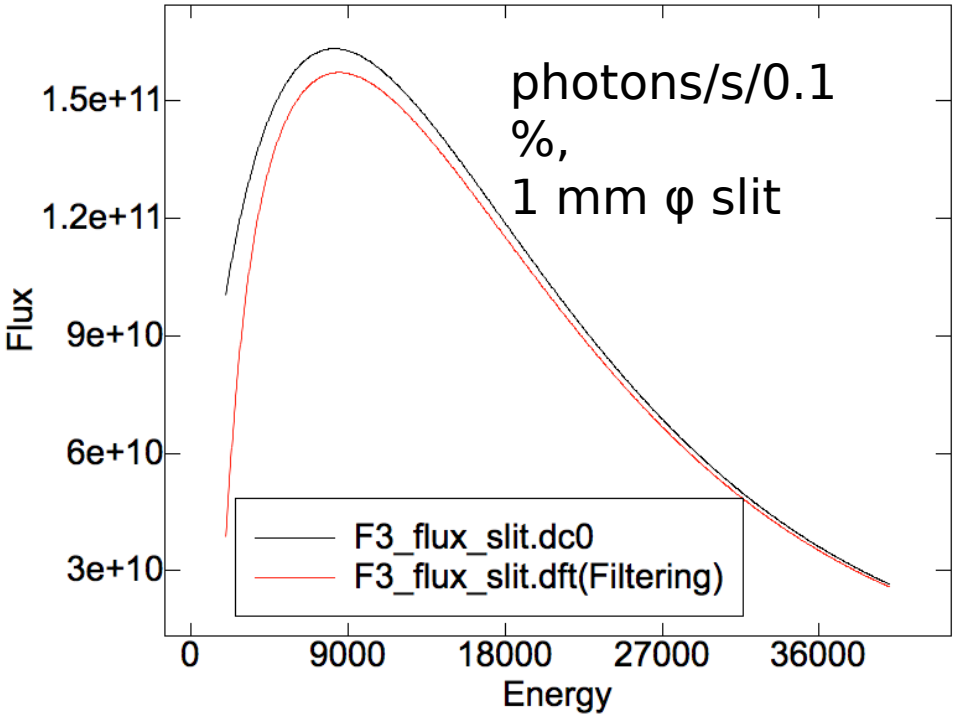
CHESS

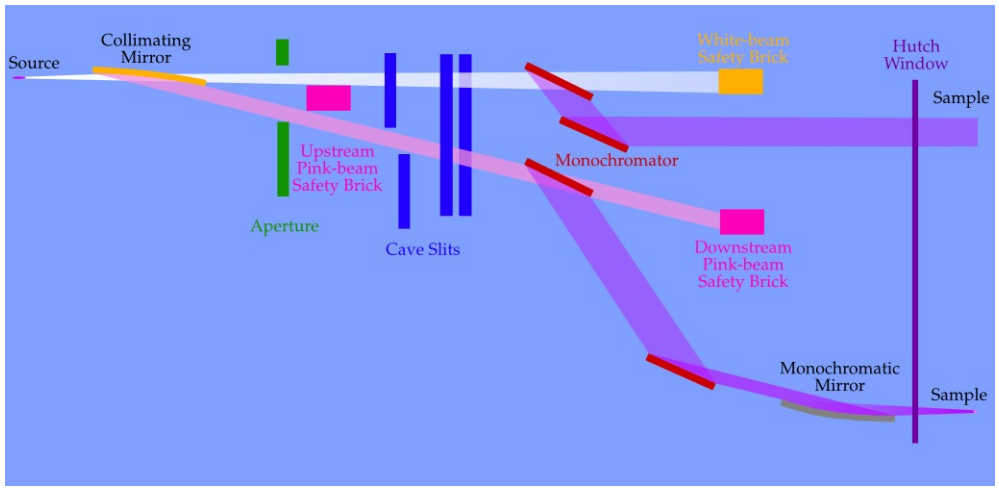
- Ithaca, NY
- Small facility funded by the National Science Foundation
- 5.3 GeV storage ring
- 200 mA electrons, positrons
- 6 beamlines
  - 2 wiggler IDs feeding 3 beamlines, 7 endstations
  - 4 bending magnets feed 4 endstations
- 10 keV critical photon energy
- Technique development
  - Frequently assemble experiments in an “empty hutch” at five endstations



XRF Microscopy at CHESS: F3

- F3 hardbend beamline
  - 10.4 keV critical energy
  - Source: 3.6 × 0.8 mm
  - 21.6 m
- Recently commissioned new, flexible monochromator
  - With or without mirrors
  - Variety of symmetric or miscut crystals
    - XAS
    - Talbot AC/DFC/PC imaging
  - Multilayers for higher flux
    - 0.6 % (SXFM)
    - 0.22 % (low-E PX)

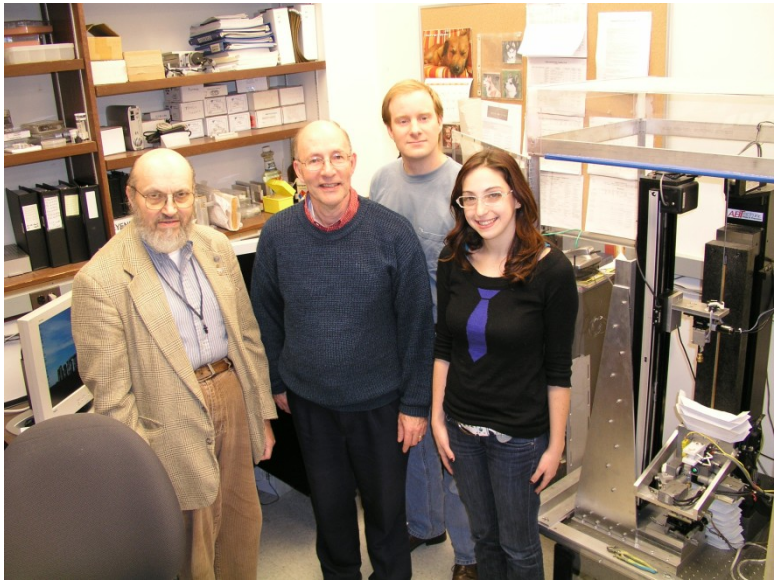


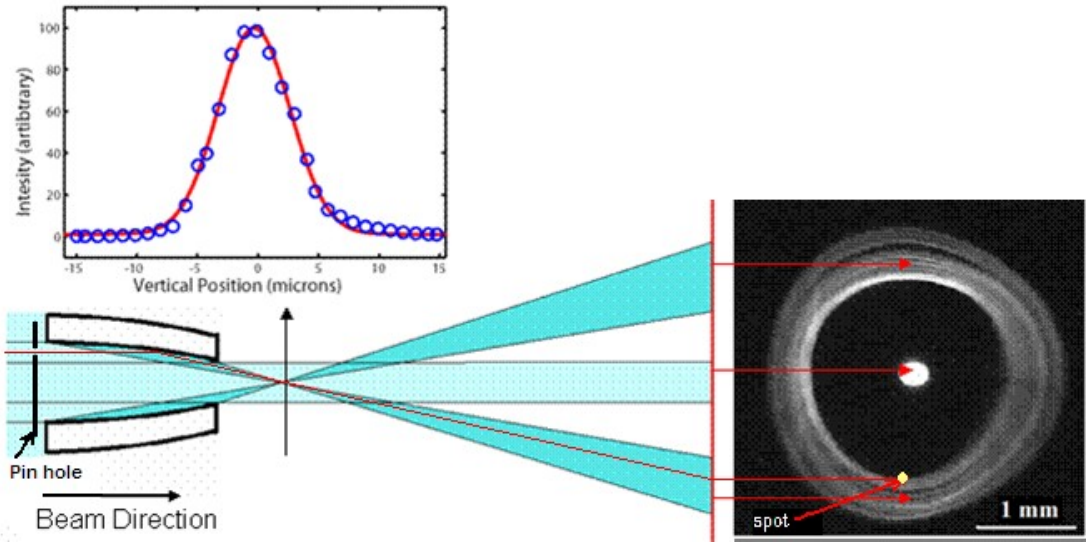


Developing XRF Microscopy at CHESS

- In-house capillary optics development
- Custom drawing tower
- Figure errors ~1-2 µm
- Slope errors ~20-30 µRad
- Tailored optics
  - 2-30 µm focal spot
  - E·θ< 32 keV·mrad
    - Total divergence = 4θ
    - Optimize for energy or divergence
- On-line capillary design program

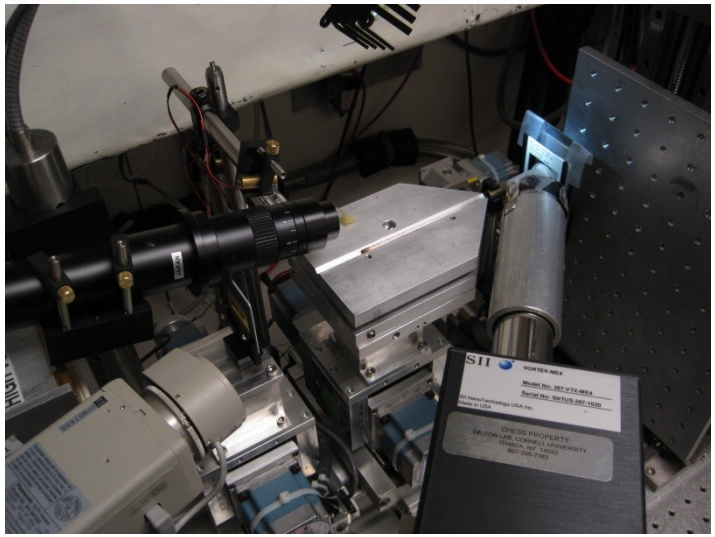
http://glasscalc.chess.cornell.edu/imageprof.shtml

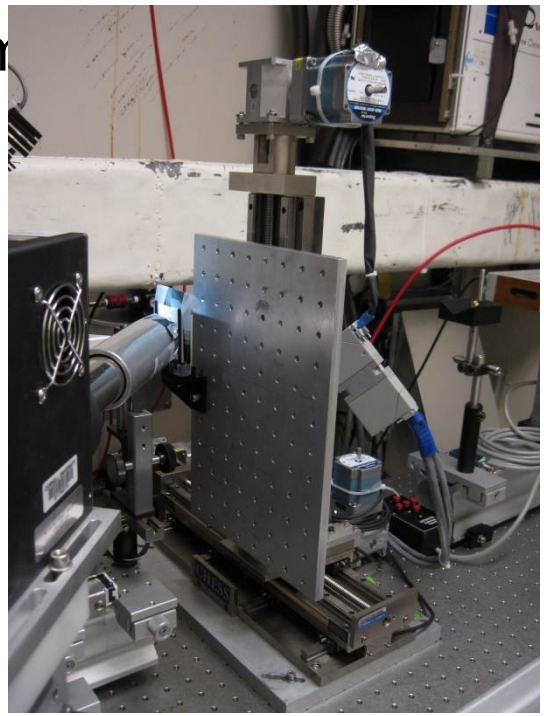


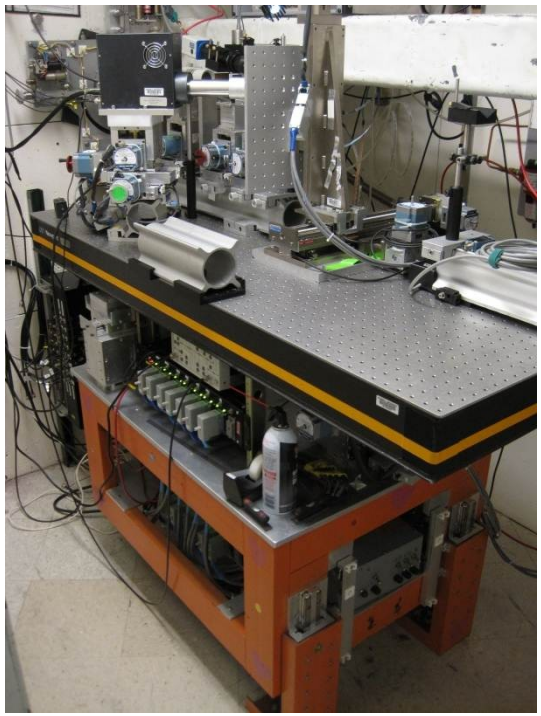


XRF Microscopy Instrumentation

- Intense monochromatic beam
  - ~10^{11} ph/s/mm^2 (0.6% ML)
  - 6-40 keV range
- Small focal spot
  - ~3×10^{10} ph/s 15 µm spot
- Sample manipulator
  - 300 mm x 300 mm x 50 mm range
  - 625 nm step size
  - 1 mm/s scanning speed
- 4-element SII Vortex SDD detector
- Integration time ≥ 0.25 s
  - Planned improvements, emulating GSECARS setup

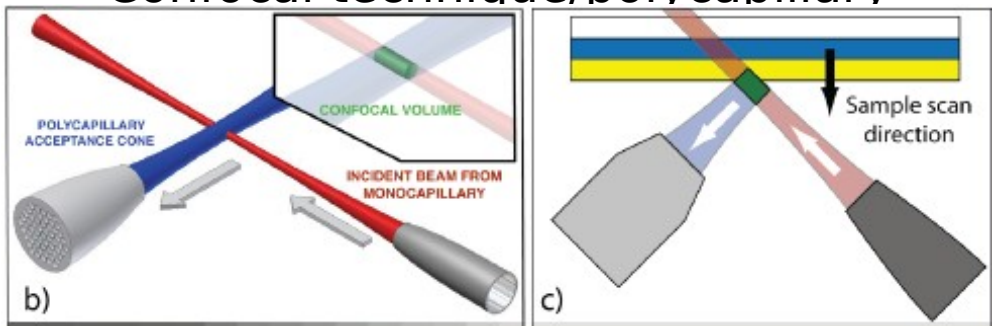




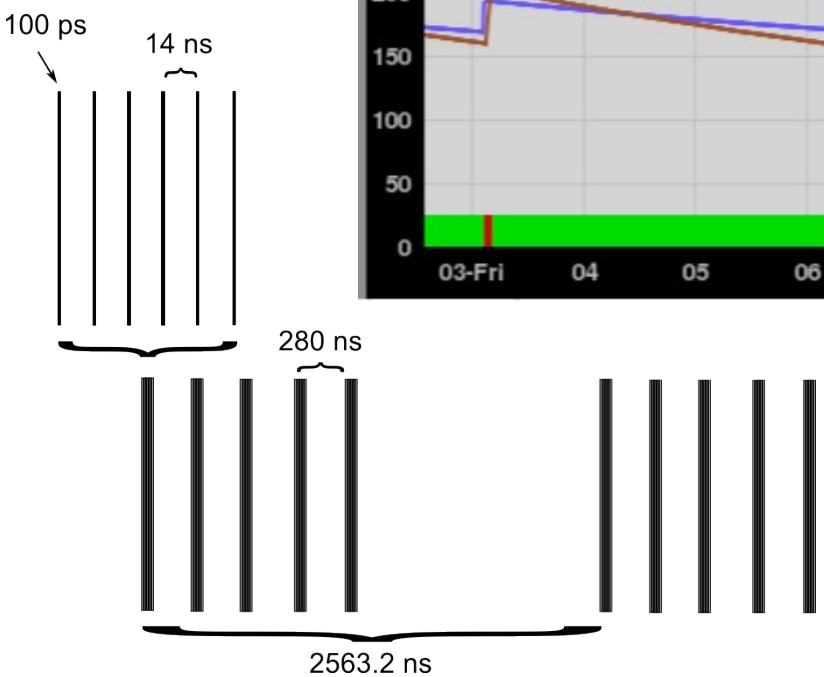


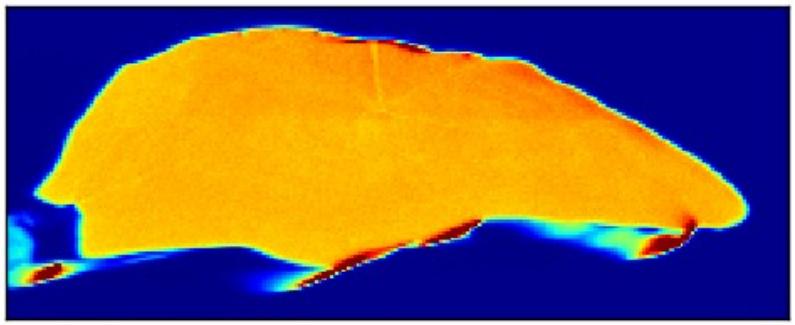
User Needs

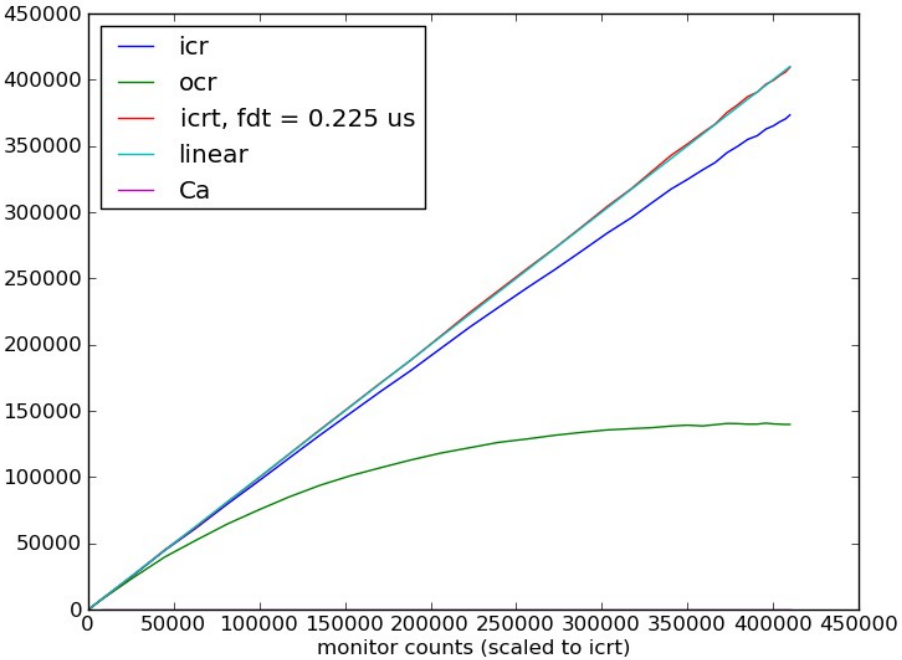
- Quantitative mass fraction determination
  - Major, minor, trace elements
  - Accomplished using PyMca library
    - Specify instrumental parameters
    - Use a standard reference to determine the monitor conversion efficiency
- Mapping trace elements on surfaces
- Fast scanning (just implemented)
  - Zig-zag mesh
  - integrate while moving (.04 s overhead/pt)
- Depth resolution
  - Art historians studying buried layers in paintings
  - Confocal technique/polycapillary



Dead time corrections

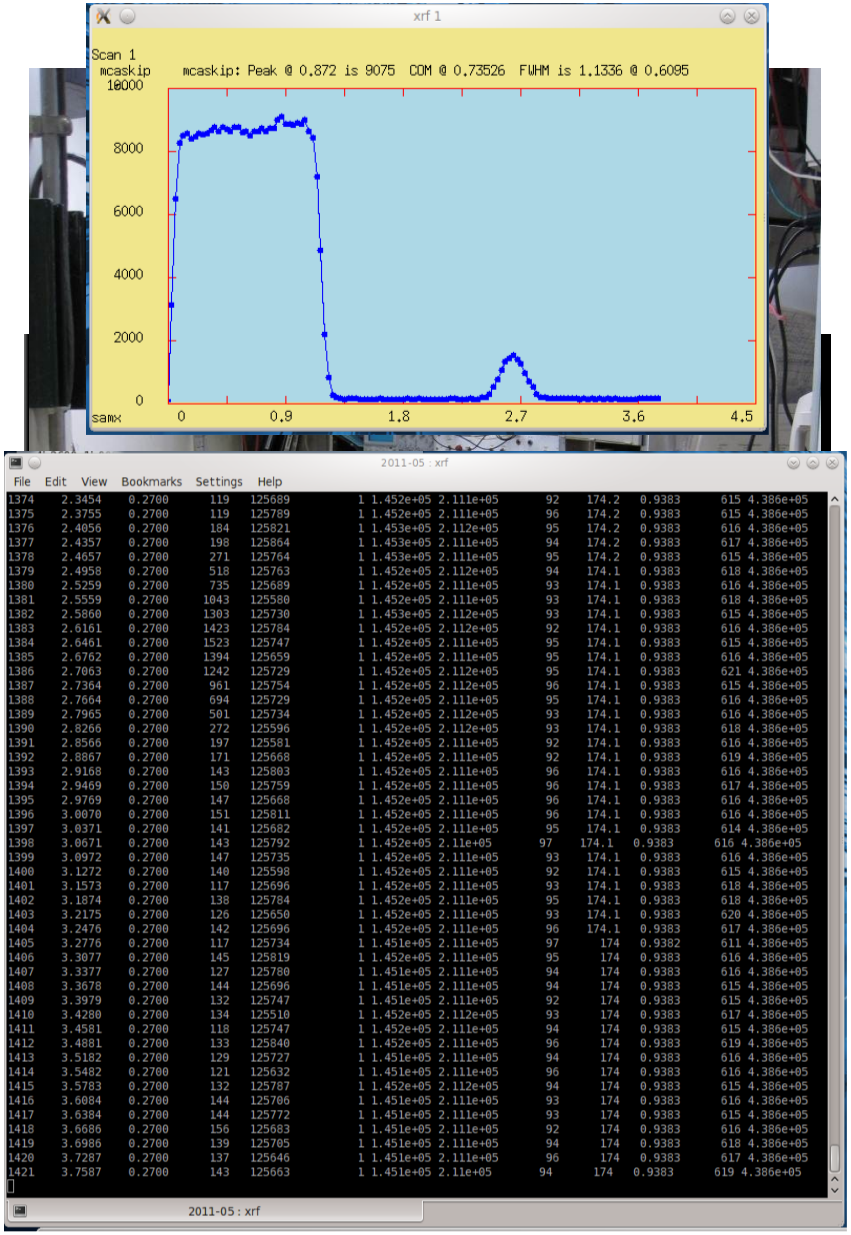






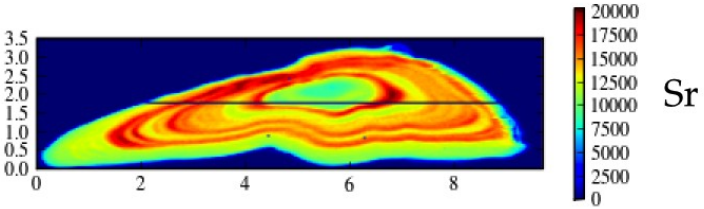
XRF Microscopy in practice

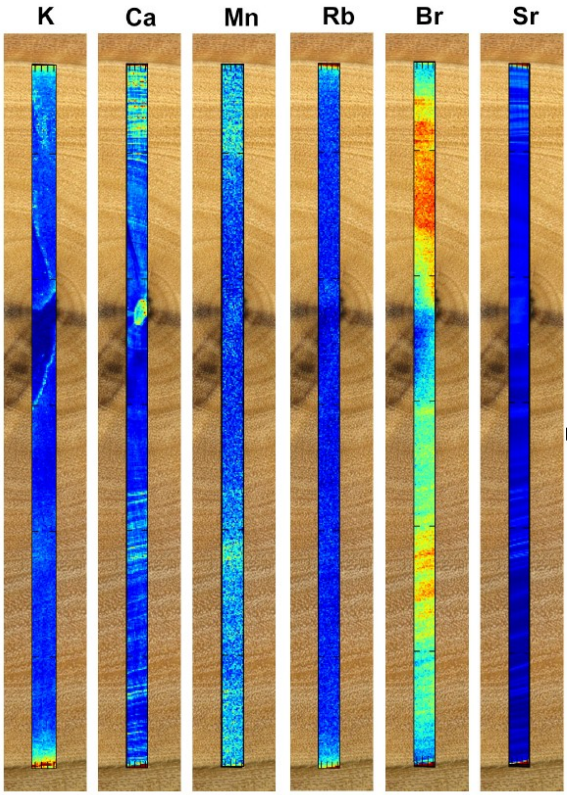
- Invested time in developing instrumentation and technique
- Using spec for data acquisition
  - Interfaces with EPICS devices
  - dxpApp EPICS software (M. Rivers)
- Now sit a user down at the station controls and see what happens
- Turns out, not all users love the command line as much as I do
- Rapid stream of dense information, what can we do to help the user?
- All this effort developing the technique, but need to improve data handling, analysis, feedback at the beamline during acquisition to capitalize on it



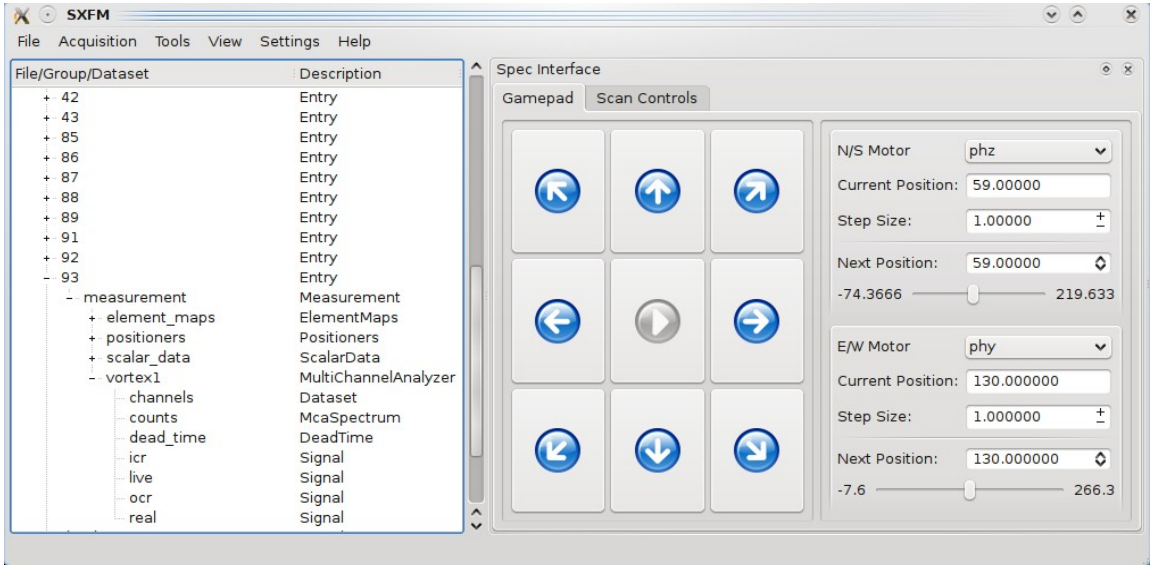
Improving the user experience

- Need to provide feedback in some useful form, in time to act on it
  - Is the detector functioning properly?
    - Artifacts may be difficult to identify in raw data
- Are the results meaningful/useful?
  - Want to provide analysis in real time, fitting spectra and generating element maps during acquisition
  - Interact with the data during acquisition
    - Select a region in the map, see the spectrum and fit that generated the result
    - Statistical analysis
- Send users home with processed data





Where we are going



Praxis

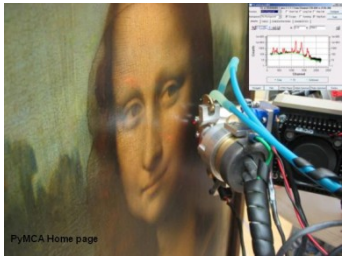
the process of putting theory into practice

the process of putting theory into practice

the process of putting theory into practice

How to get there?

- Integrate existing open-source libraries
  - Python: Rapid development
    - Cython: overcome occasional performance bottlenecks
  - PyQt4: GUI development
  - SpecClient: communicate data between data acquisition “server” and analysis “client” (M. Guijarro, ESRF)
  - IO: lots of data; strategy for structuring, storing and interpreting
  - PyMca: Fit MCA data (V. Sole, ESRF)
  - Multiprocessing/ParallelPython: leverage every available processor
  - Matplotlib: plotting library



IO: Spec and EPICS

- Heavily reliant on the Spec data acquisition program
  - Use Spec to interface with EPICS-controlled devices
- Spec produces raw data in ASCII files
  - Multiple scans in a single file
  - File and scan headers contain motor names and positions, additional (arbitrary) metadata
  - "Table" containing list of observations
    - Independent and dependent variables
  - Vector data (e.g. MCA) interleaved with scalar data
- Arbitrary-named columns and devices
  - Flexible
  - Enough context to identify:
    - Axes
    - Monitor, Detectors, MCAs
  - Up to user to provide remaining context
    - Dead time correction
    - Monitor efficiency
    - Associated data files (images from area detectors)
- Encapsulate this scheme in a more sophisticated file format for real-time analysis

Encapsulate The Scheme

File Format vs Ontology

- Recent community interest in developing common exchange formats for XRF/XAS
- Creating a common exchange format implies a shared and standardized conceptualization…
  - Establishing an ontology…
    - The shared concepts and vocabulary used to model the problem domain
    - Complicated: subjective, no “right answer”. Consider Newton/Leibniz formalisms for Calculus
    - Necessarily an iterative process
- …and then establishing a common file format in which the ontology will be used to express the information

Ontologies

- Ontology (wikipedia): a formal representation of knowledge as a set of concepts within a domain, and the relationship between those concepts
- Among the reasons to develop an ontology:
  - Shared understanding of the structure of information among people or software
- Ontology components
  - Classes (types of things)
  - Individuals (instances of things)
  - Relations
    - Subclasses (taxonomy: is-a relationships) (Caution! Read “Design Patterns” by Gamma et.al. first!)
    - Composition (meronomy: part-of relationships)
  - Restrictions
  - Rules
  - Axioms
  - Attributes (properties, parameters)
  - Events (changes in attributes or relations)

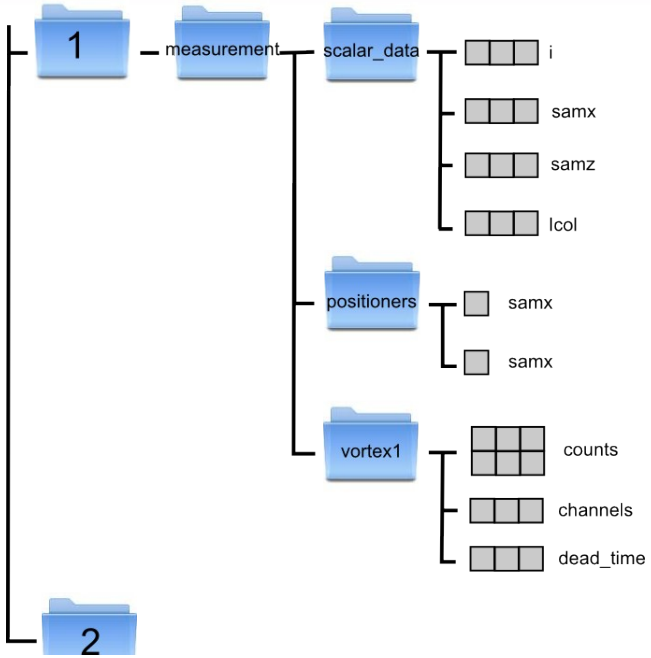
Abstractions

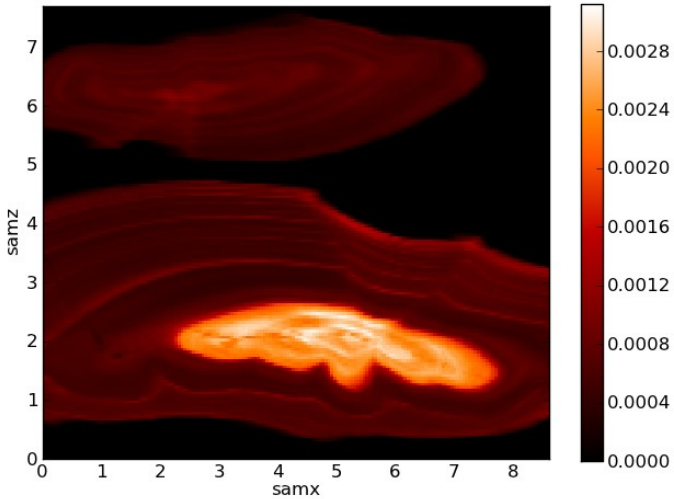
Logic/ Conventions/ Assumptions

Observations

Measurement ontology

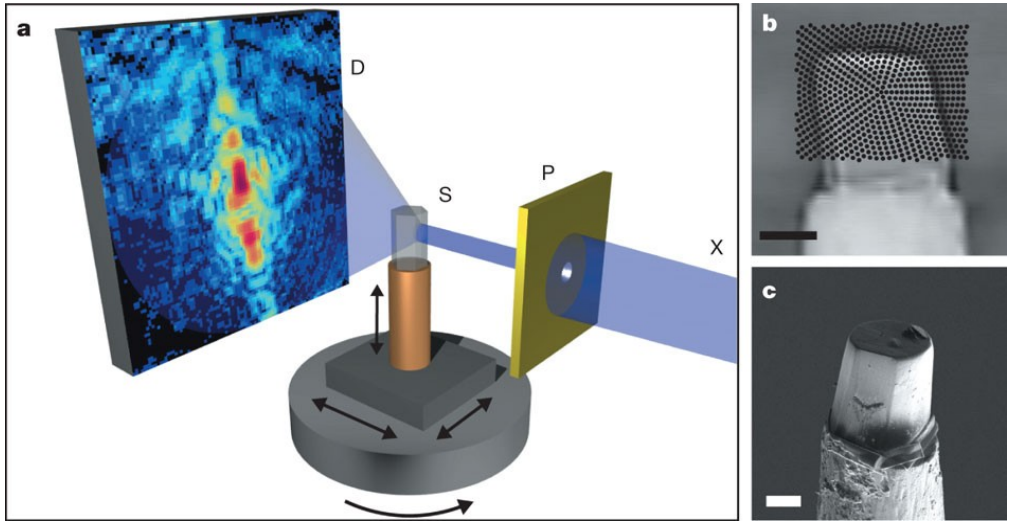
- Emulated spec's instrument- and application-independent "measurement" ontology
- Classes
  - Axes/Positioners (independent variables)
  - Signals (dependent variables)
  - Detectors
    - Signals: Counts, deadtime, etc.
    - Attributes: Calibration, efficiency, dark signal
- Each dataset a list of observation events for a given attribute during the measurement (scan)
- Basic types of data: scalar, vector (spectrum), image
- Instances arbitrarily named, grouped into a simple directed acyclic graph
- A few attributes provide context needed for real-time analysis and visualization
  - Scan dimensionality, trajectory (regular or zig-zag mesh)
  - Axes identification (fast/slow, compound motions)
  - Preserving information communicated by spec's conventions or metadata

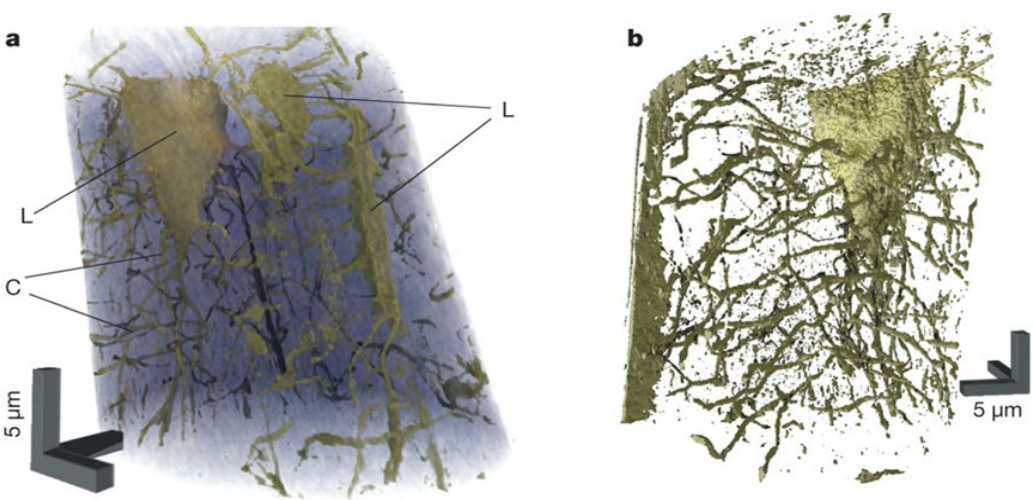




Visualization Context

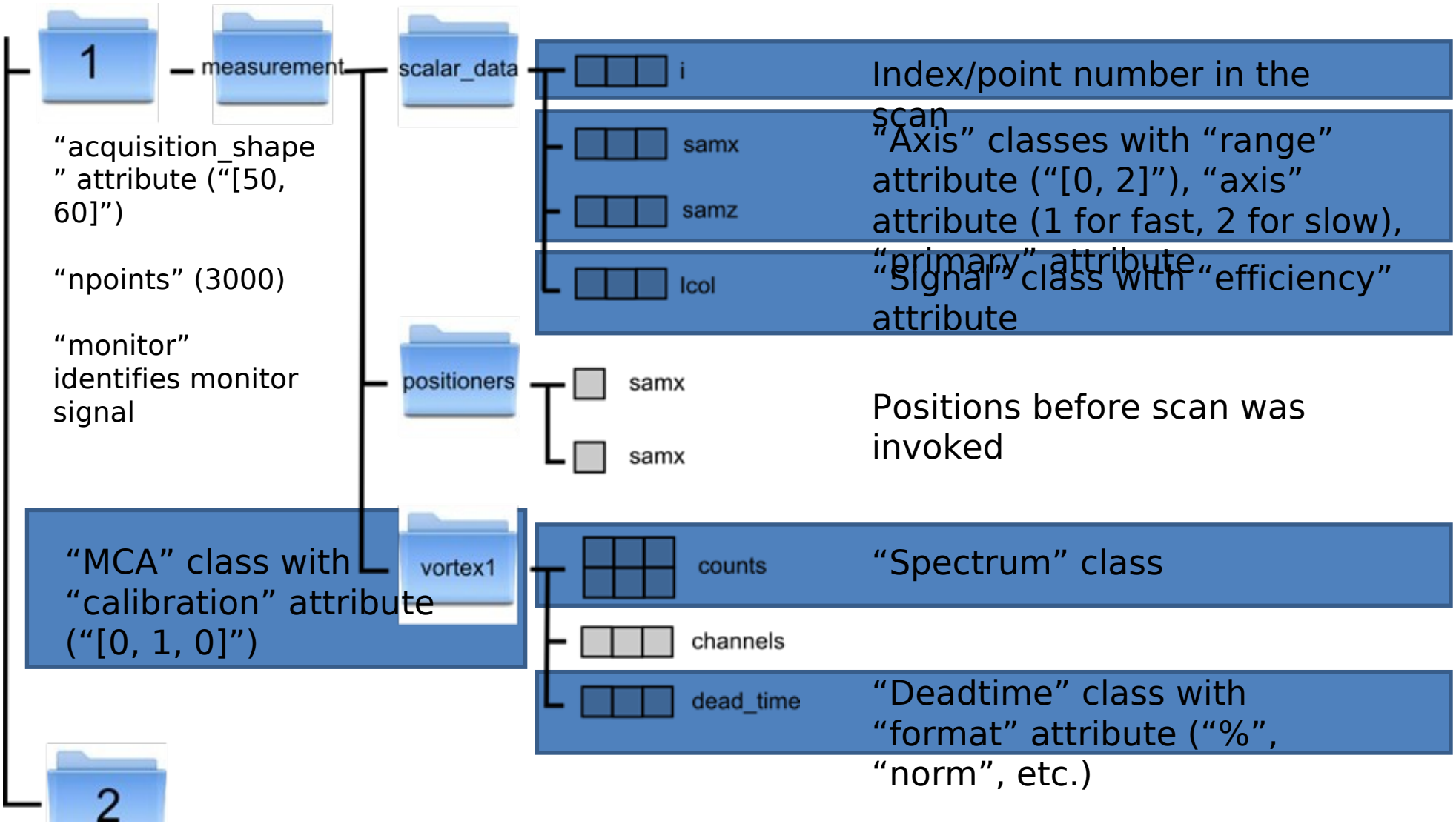
- Event lists are fundamentally the natural storage mechanism: all data collection is parameterized as a function of time
  - Area scans on a regular array grid: easy
  - Area scans on a regular array grid following a zig-zag trajectory: easy
  - Non-linear trajectories
  - Spiral tomograms
  - Irregular grids (avoid “raster grid pathologies” in ptychography)
- Need to provide context so event lists can be visualized
  - Type of trajectory, shape if a regular grid, etc.





M Dierolf *et al. Nature* **467**, 436-439 (2010) doi:10.1038/nature09419

Providing context for visualization and analysis



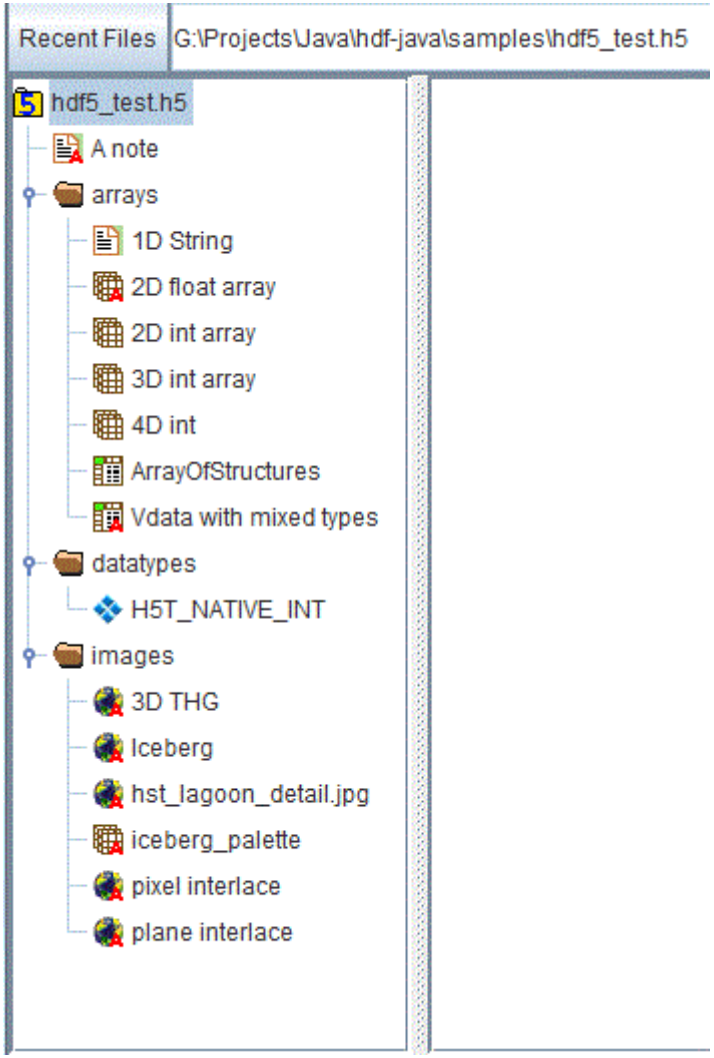
Context for visualization and analysis

- Identify scan axes and ranges, based on inspection of class and attributes
  - Decoupled from type of application/instrument/measurement
  - Independent of how signals/axes named
- Identify the signal used to monitor the incident flux
  - Attribute indicates conversion efficiency
- Identify different kinds of detectors, like MCAs
  - Independent of name
  - Dead time, how dead time is formatted
- Observations grouped into event lists
  - Decouples details of scan type from details of storage
  - Easily iterate over points in the scan during analysis
  - Use attributes to process lists for visualization
- Import context specific to particular analysis program
  - the fit model encapsulated in PyMca configuration file
    - Included elements, transitions
    - Attenuators
    - Beam/sample/detector geometry
    - Detector calibration, peak fitting parameters

A Capable File Format

File format

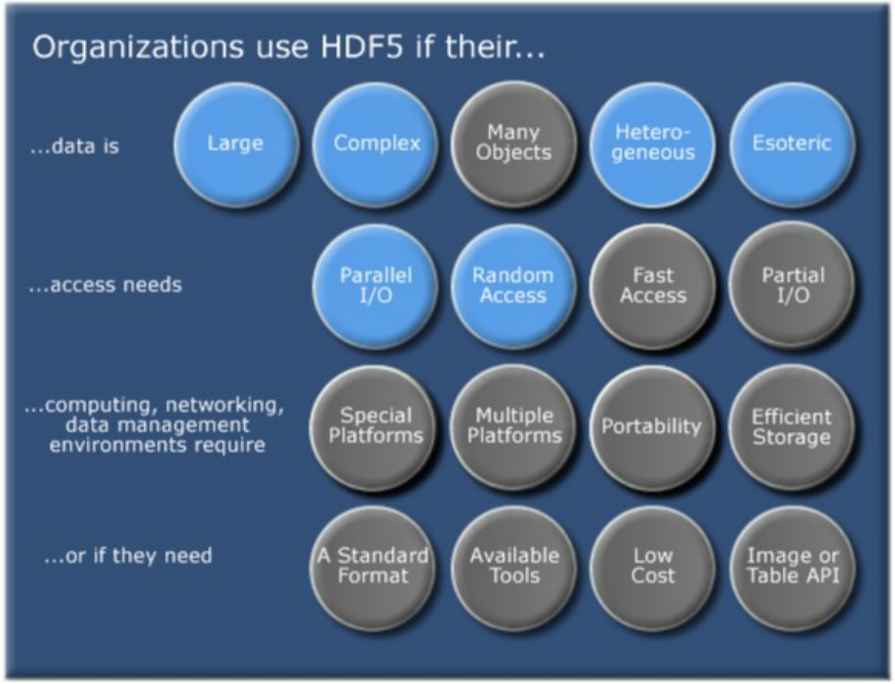
- Desire a more capable file format
  - Binary, able to handle huge data payload
  - Express hierarchy, classes, attributes
  - Accessible/cross-platform/open-source
  - Pervasive/authoritative
- Options:
  - XML: not binary, inefficient for manipulating huge datasets
  - SQLite: drawback: hierarchy/arrays; atomic transactions a huge benefit
  - NetCDF (now based on HDF5): frequently used in geosciences
  - HDF5
  - NeXus: aims to be a common exchange format for neutron, x-ray facilities

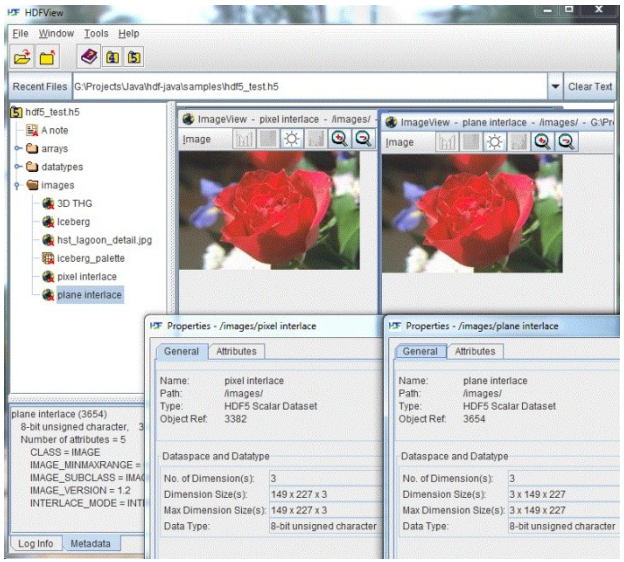


http://www.hdfgroup.org/hdf-java-html/hdfview/UsersGuide/ug02start.html#ug02tree

HDF5

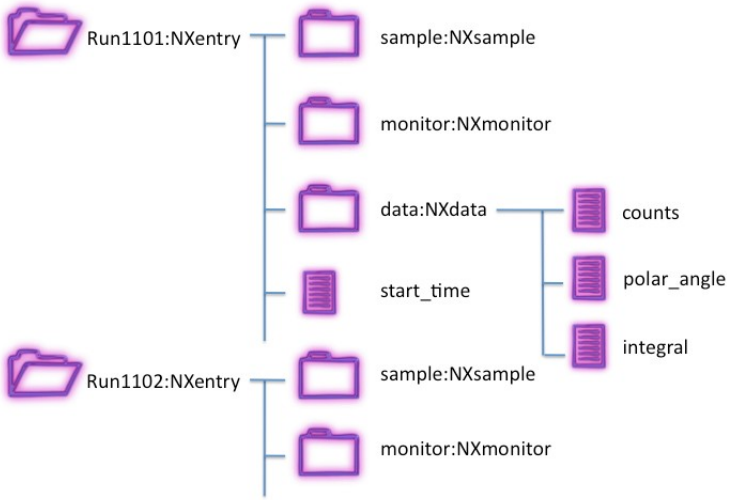
- Advantages:
  - www.hdfgroup.org/why_hdf
  - Access portions of file without parsing entirety
  - High-level interfaces: h5py, PyTables
  - Visualization programs (HDFView)
  - Basically provides the framework for the binary format, we provide the schema
- Disadvantages:
  - Unlike DMS, no transaction support
  - No support/protection for writing to a file from multiple processes
  - Caching mechanisms: if your program crashes, possible to corrupt file
  - If you add a new feature to an old group, the group itself will be unreadable from older versions of the library
  - Tendency to include features/behavior changes in bug fix releases





NeXus Ontology and Common Exchange File Formats

- NeXus Ontology
  - Upper ontology: standard abstractions for observations at Synchrotron and Neutron Facilities
    - Instrument/equipment/sample definitions
    - Visualization context
  - Use classes, hierarchy to organize abstractions and relationships
  - Application definitions establish more domain-specific ontologies
- Self-describing, cross-platform file formats
  - NeXus API (don't have to use it)
  - HDF5: Binary, supports compression, Requires HDF5 API
  - XML: ASCII, Many APIs available for accessing data



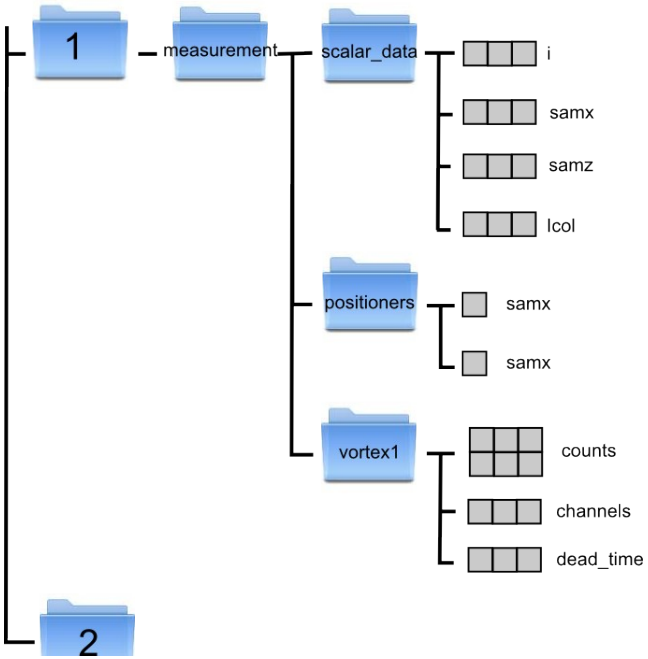
http://download.nexusformat.org/doc/html/Introduction.html

What role should NeXus play at CHESS?

- How well does NeXus serve as a common exchange format in practice?
  - Sufficient utilities to validate against a domain-specific ontology?
  - Are the NeXus ontologies robust, or do facilities experience problems exchanging NeXus data?
  - Are the available file formats acceptable to the community?
- How well does NeXus ontology accommodate changing experimental needs?
  - This is a common requirement at CHESS, applications and instruments always evolving
- Existing NeXus formats may not be most appropriate choice for data acquisition, archiving, real-time analysis
  - Database management system probably best
  - I wouldn't rely on HDF5 alone for acquisition, especially collecting many scans in a single file
    - Caveat: I have not explored HDF5 support for MPI, might address my concerns
    - HDF5 caches data for performance reasons
    - Application crash results in corrupted data file (concurrent programming is hard, this does happen)
    - At CHESS, spec ASCII files can be used to regenerate HDF5

Finding common ground

- Use our weaker measurement ontology to store data in HDF5 “measurement group” located in an empty NXentry
  - Easy conversion of existing spec data to HDF5
  - Flexible to support rapidly changing instruments and data acquisition environments
  - Retain ability to use measurement group to construct stronger NeXus ontology
    - Provides upgrade path
    - Ability to cast measurement data into a common exchange format
- Recent NeXus accommodations
  - Addition of NXcollection class to accommodate such use-cases
  - Addition of NXsubentry to support multiple domain-specific ontologies in a single scan
  - Encouraging structuring datasets as lists of observations



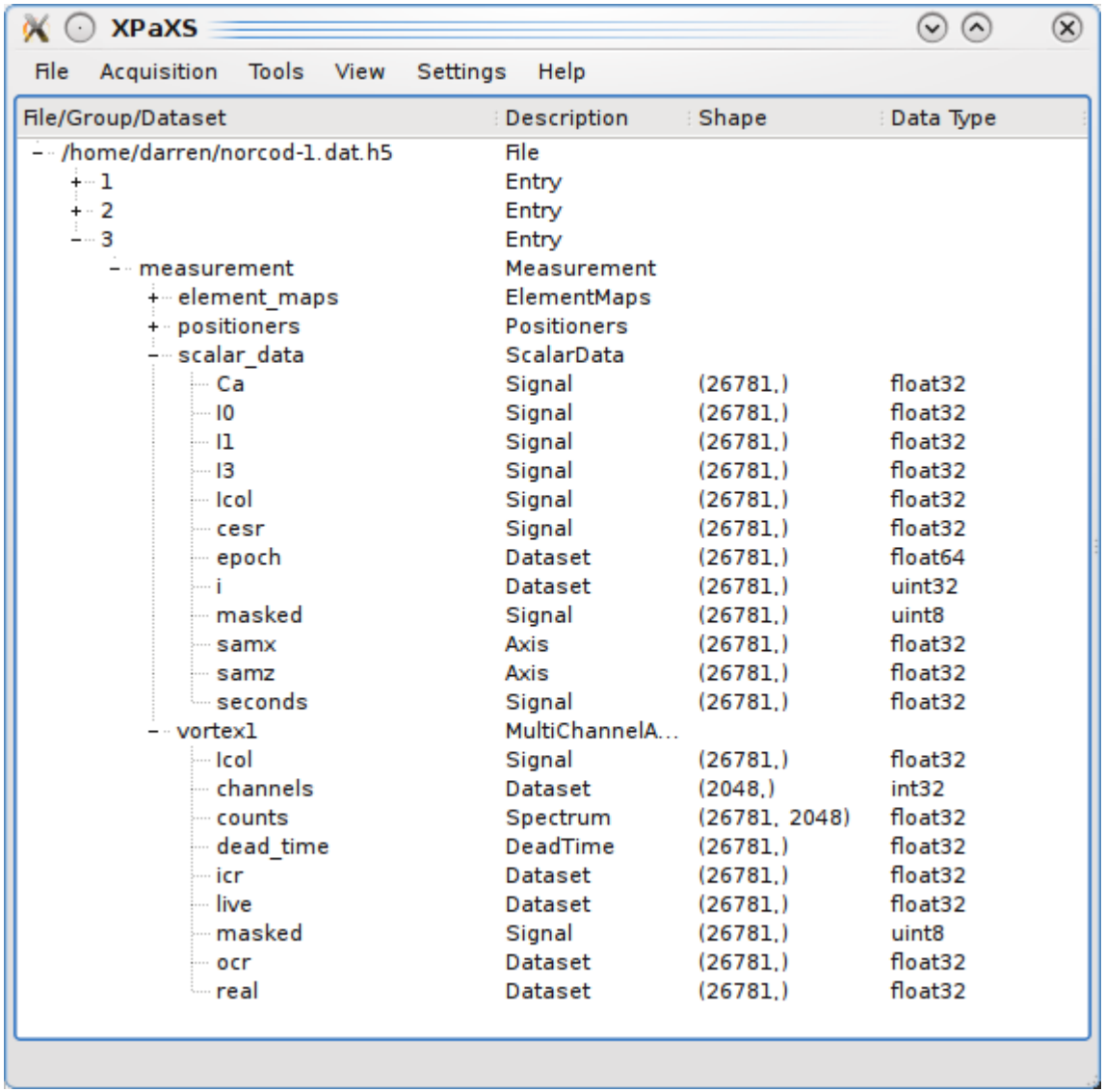
Phynx: An intuitive interface for HDF5 data

- Required a compelling, intuitive interface to interact with data
  - Object-oriented
  - Easily used by experimenters, interactively or in scripts
  - Powerful enough to meet the needs of application developers
- Inspect HDF5 objects/attributes, produce an instance of an appropriate class from the ontology
  - Specialized methods/properties
  - Proxies
    - Corrections based on conversion efficiency, dead time, etc.
- Considered building upon:
  - Pytables: Used for a year, intuitive interface, not thread safe, no support (at the time) for hdf5 links, stores python-specific data
  - NeXus python bindings: preferred other APIs
  - Decided to build upon h5py: elegant interface and implementation, threadsafe
    - Custom completer for Ipython (based on object introspection and readline capabilities)
    - Support for python-3

Phynx: Implementation

- Phynx wrap h5py File, Group and Dataset classes
  - Take these objects and use composition to breath some life into ontological abstractions
    - MCA: subclass of Detector, subclass of Group
    - Spectrum: subclass of Dataset, with data proxy that corrects for dead time
    - Deadtime: contains data proxies so deadtime can be stored and expressed in numerous ways (percent, correction, normalization, etc.)
  - Not tied to hdf5 format (although that is the only one currently supported)
- Supports elements from the NeXus ontology
- 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

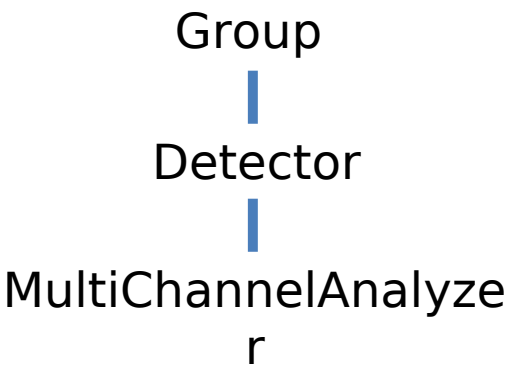
Example data

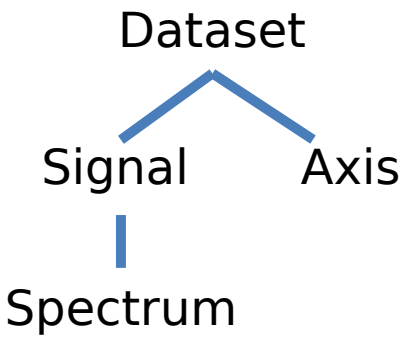


Select item from tree menu, go to tools, and the current item is passed to a validation method of each tool which returns “true” if the tool knows how to handle that item. Then the tools menu is populated with tools that are compatible with the currently selected item.

Extending Phynx: defining new classes

- "NX_class" attribute is reserved for NeXus-approved abstractions
- Phynx uses "class" attribute to specify its own abstractions
  - provides a way to identify more specialized interfaces without abusing "NX_class"
  - When opening a group or dataset, the "class" attribute is checked to identify what class to instantiate, 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
  - Phynx designed to be extended, even by third parties
  - Subclassing phynx.Group or phynx.Dataset automatically registers the key (the class name) and the class itself in the registry
    - Accomplished using a simple metaclass pattern

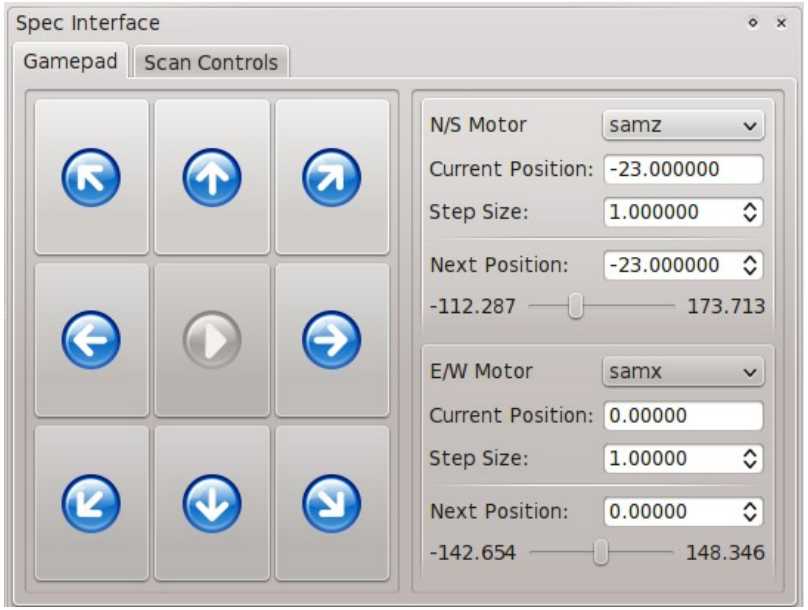


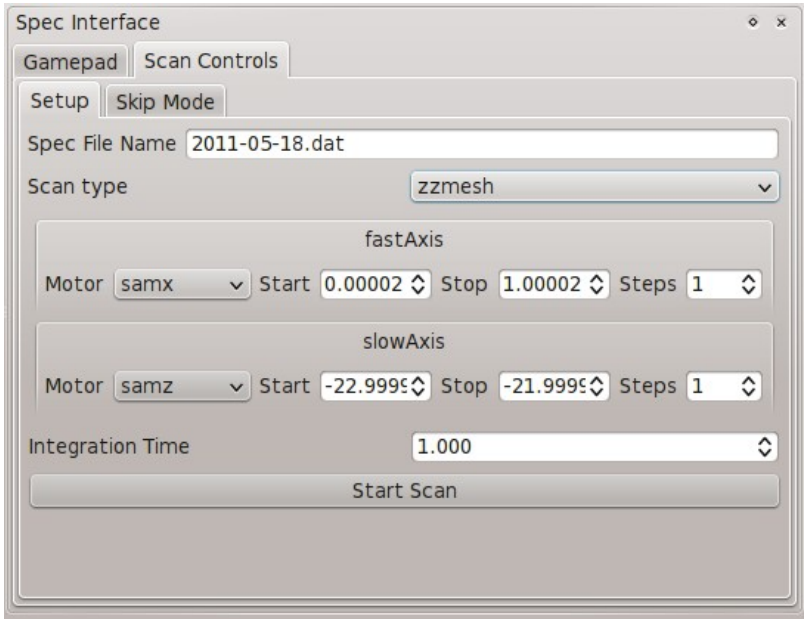


Real-time Analysis

Data acquisition

- Spec Client/Server mode
  - Spec still produces (large) ASCII files
  - Register spec variables to be broadcast to client programs
- SpecClient (Python Library and spec macros written by Mattias Guijarro at ESRF)
  - Manipulate instrument from client program
  - Define and start scans
  - Update GUI based on equipment state



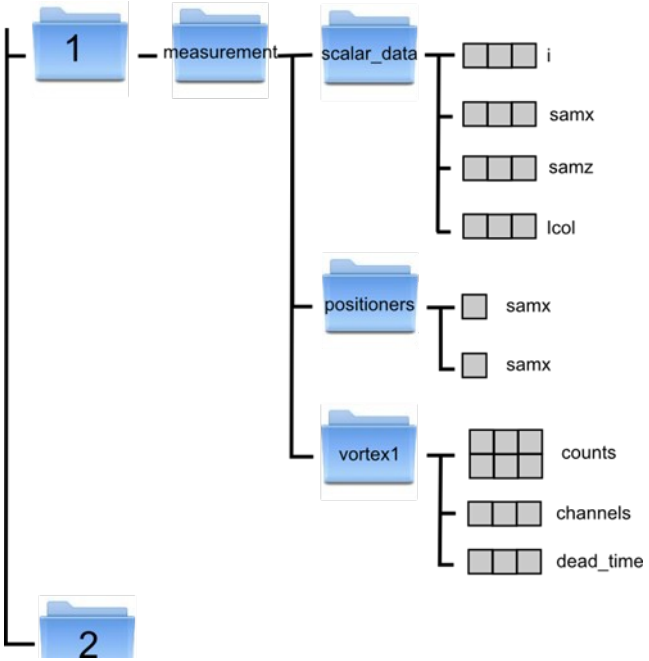


Spec Client/Server mode

- Contributed spec macros to broadcast arbitrarily structured data hierarchies to clients (JSON formatting)
- SpecClient: clientutils.mac automatically sets up the entry and the measurement hierarchy (including scalar_data and positioners)
- SpecClient automatically broadcasts scalar data
- Configurable for any additional information that needs to be broadcast (EPICS detectors)
  - Extensible using spec's hooks
  - Serialize spec data in JSON format (mapping, lists, strings, numbers)
    - Currently very inefficient at macro level, G. Swislow looking into adding a fast built-in serializer function
  - Configure environment at start of scan
    - client_set_scan_env(identity, classname, attribute_mapping)
  - Report data for each point in scan
    - client_set_data(identity, value)
  - Heavily documented in SpecClient's clientutils.mac

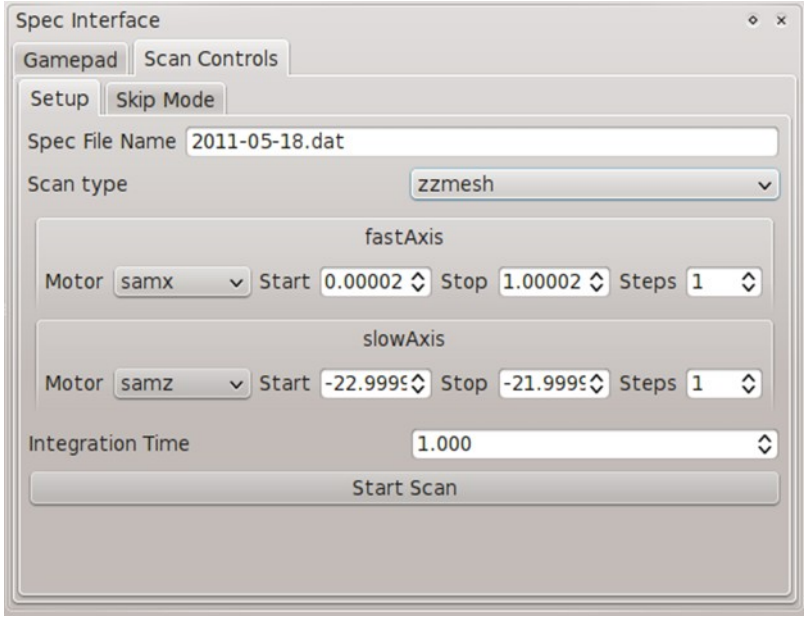
SpecClient: Client program

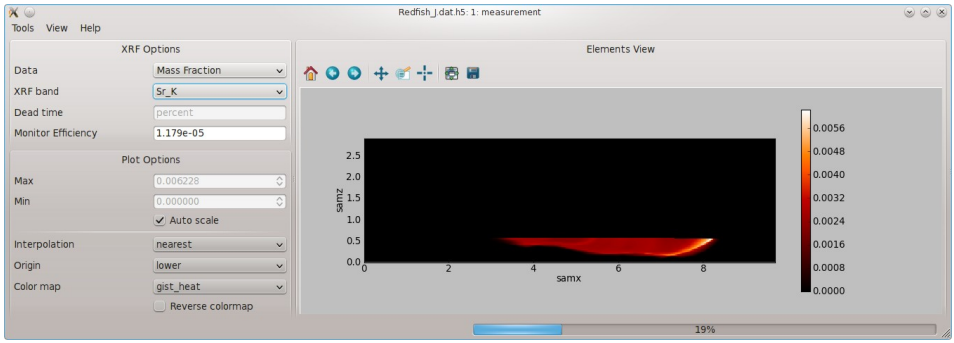
- Client program needs to respond to spec's broadcasts
  - Contributed patch for python clients to receive and assemble the data into standard python data structures
    - Dynamically assemble HDF hierarchy based on data from spec server
  - Clients receive data after each point in a scan
    - Update event lists in hdf5 file.
    - Update progress reports in client GUI
- Asynchronous spec scan class
  - newScan
    - Receive scan ID
    - Create new entry in file
    - Populate measurement hierarchy from JSON dictionary:
      - {'path/to/obj': [type, {'attr_key1': 'attr_value1', ...}], ...}
  - newScanData
    - Receive index "i" of point in scan
    - Update event lists from JSON dictionary
      - {'path/to/obj': val, ...}



Praxes: real-time analysis

- Define and start scan in GUI client
- Data reported to client
  - Automatically open an MCA analysis window for current scan
    - Prompts for PyMca config file
    - Can be extended to choose from various types of analyses
- Launch task manager in new thread
  - Fit spectra without freezing the GUI
  - Python's global interpreter lock (GIL) limits multithreading to a single processor!
  - Task manager thread hosts a multiprocessing server
    - Multiprocessing (python standard library)
    - ParallelPython (third party)
    - Works around the GIL
      - more efficient use of multiple processor cores
      - computing clusters

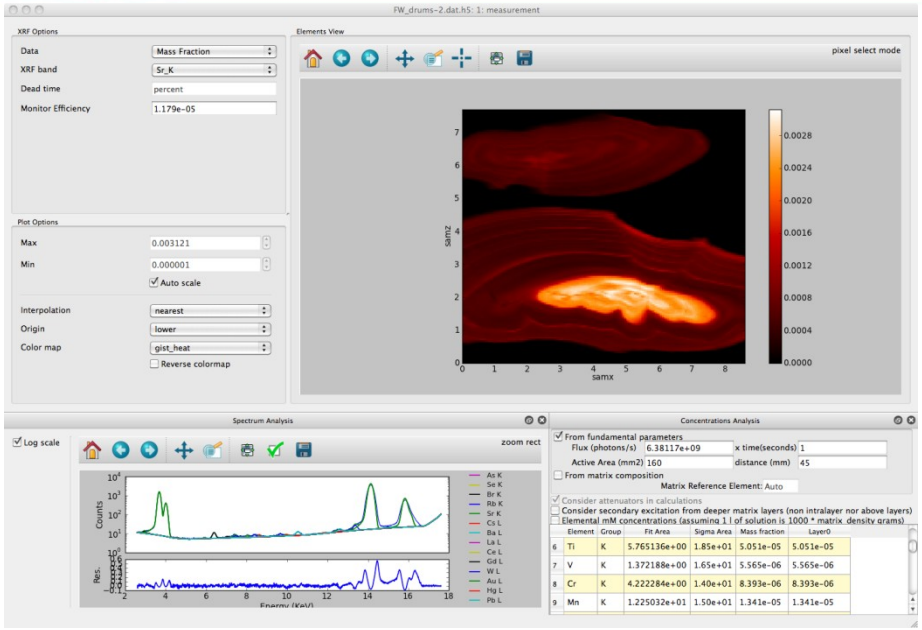




Real-time analysis with efficient multiprocessing

- Task manager iterates over all data points in scan
  - Check the scan index event list
    - If less than current iteration, the next data point has not been reported yet
      - Sleep, try again
  - If the scan point is masked, continue to next iteration
  - Assemble data for processing
    - Scan index
    - Deadtime-corrected spectrum/spectra
    - Incident flux
  - Pass data to multiprocessing library for fitting with PyMca
  - When fit is complete, results are passed to a callback function
    - Update element maps in hdf5 file
    - Add index to queue, GUI knows to update the map
- Implementation doesn't know or care if analysis occurs during acquisition, or offline
  - Interrupt real-time fitting, adjust parameters, restart analysis
  - Use Praxes after leaving CHESS

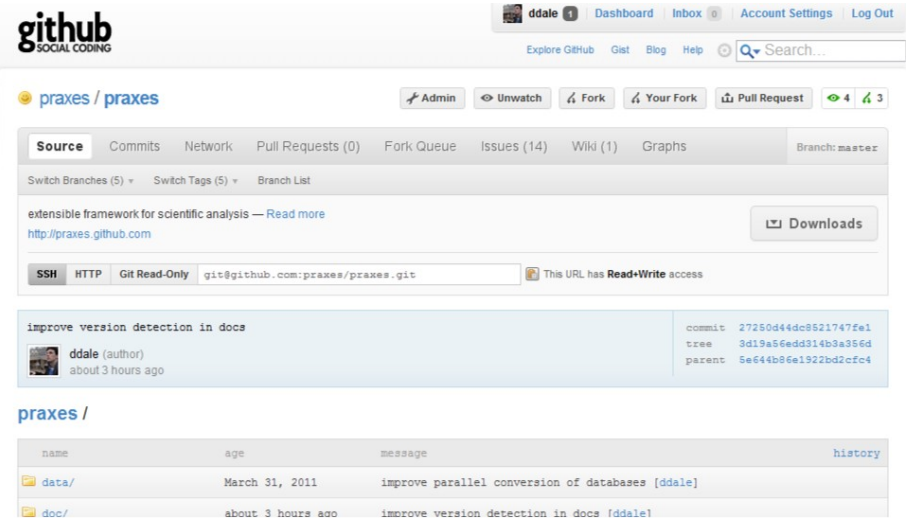
Putting it all together: Praxes

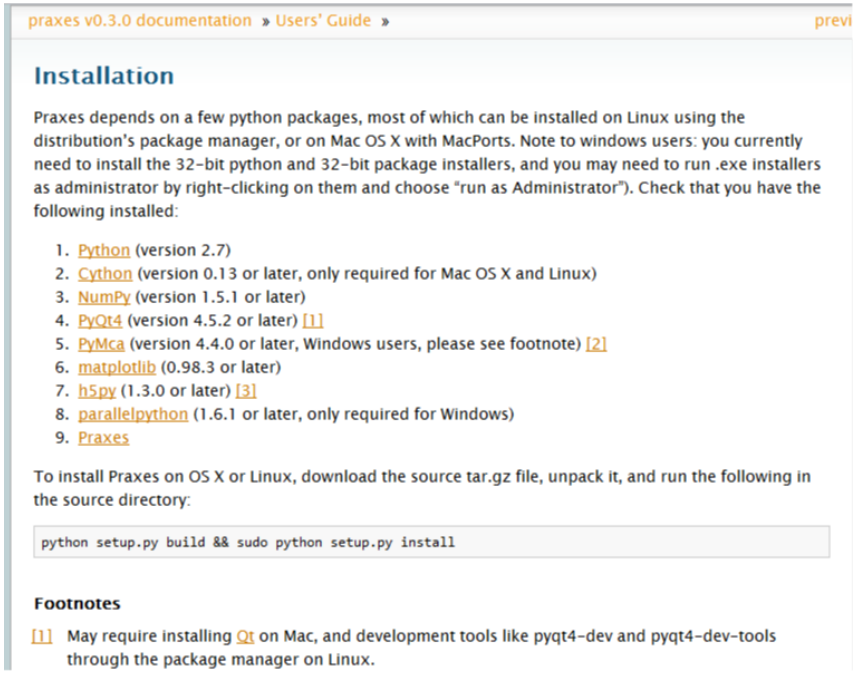


- Praxes is an open-source project (BSD)
- GUI based on Qt4/PyQt4
- Fast, intuitive interface to spec data files
- Phynx interface to hdf5
  - GUI, command line navigation
- SpecClient to interact with equipment from GUI
- Matplotlib to visualize data
- Interact with data while acquisition and/or analysis 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
  - threadsafe
    - Inspect/interact with data while both acquisition and analysis are ongoing

Praxes

- In use at CHESS/F3 for 3 years
  - Send users home with processed data
- Windows, GNU/Linux, OS X
  - BSD license
- Project and documentation hosted at github
  - Along with many other scientific programming projects
- Ongoing development project
- Emphasis on documentation and unit testing
  - If my own code is not documented and tested, it is potentially useless to me 6 months from now





praxes v0.3.0 documentation » Users' Guide » previ

Installation

Praxes depends on a few python packages, most of which can be installed on Linux using the distribution's package manager, or on Mac OS X with MacPorts. Note to windows users: you currently need to install the 32-bit python and 32-bit package installers, and you may need to run .exe installers as administrator by right-clicking on them and choose "run as Administrator"). Check that you have the following installed:

1. Python (version 2.7)
2. Cython (version 0.13 or later, only required for Mac OS X and Linux)
3. NumPy (version 1.5.1 or later)
4. PyQt4 (version 4.5.2 or later) [1]
5. PyMca (version 4.4.0 or later, Windows users, please see footnote) [2]
6. matplotlib (0.98.3 or later)
7. h5py (1.3.0 or later) [3]
8. parallelpython (1.6.1 or later, only required for Windows)
9. Praxes

To install Praxes on OS X or Linux, download the source tar.gz file, unpack it, and run the following in the source directory:

```
python setup.py build && sudo python setup.py install
```

Footnotes

[1] May require installing Qt on Mac, and development tools like pyqt4-dev and pyqt4-dev-tools through the package manager on Linux.

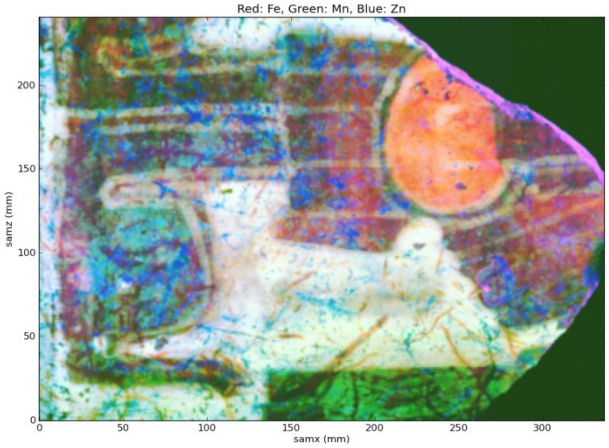
Where to go from here?

- Long-awaited migration to python-3
  - Waiting upcoming releases of Matplotlib/h5py
- Continue fleshing out documentation
- Improve unit test coverage for older code
- Integrating routines for combinatorial powder diffraction analysis
- Adding physical reference library
  - Based on quantities: numpy based library supporting units and physical constants (http://packages.python.org/quantities/user/tutorial.html)
  - Elam XRF database and API already available
  - Waasmaier isotropic form factor approximations database and API nearly complete
  - Henke anomalous dispersion corrections database and API nearly complete
- How can this work be of greater use to the community?
- Interested in collaborations
  - Currently loosely knit
  - Opportunity collaborative development of open collection of packages for synchrotron facilities
    - Distributed revision control / code reviews (github is amazing)
    - Well-documented (Sphinx actually makes this fun: http://sphinx.pocoo.org/)
    - Thoroughly unit-tested (eat your vegetables)

Acknowledgements

- CHESS
  - Arthur Woll
  - Don Bilderback
  - Jeff Lipton
  - F3 users
- ESRF
  - V. Armando Sole
  - Matias Guijarro
- Argonne
  - Mark Rivers
  - Matt Newville

- h5py/hdf5 developers
- Qt/PyQt developers
- Matplotlib developers
- Python Developers



CHESS