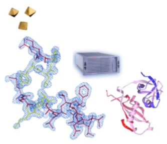
Second imgCIF Workshop. Joint NSLS and CFN Users' Meeting
Brookhaven National Laboratory
24 May 2007

Use of the CBF Library in the Context of Real-Time Image Analysis

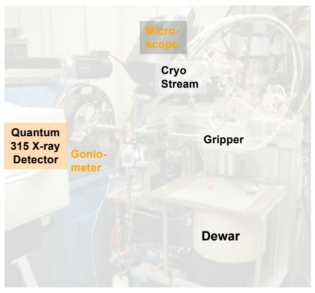
Nicholas Sauter | Lawrence Berkeley National Lab

Collaborator: *Ana González* | Stanford Synchrotron Radiation Lab



Standardization of the crystallographic image format has the potential for making data processing software much more portable. *LABELIT*, the LBNL autoindexing toolbox, currently supports data formats produced by most equipment vendors, but the many *ad hoc* rules required to implement this support make the code difficult to maintain. The present goal is to utilize the CBF library for image processing at the earliest possible date. CBF-formated images will be treated within a data processing pipeline, such as that offered by Stanford's *Web-Ice* package. This allows image characteristics to be immediately analyzed and reviewed, to optimize the experimental protocol.

Experimentation at Modern Automated Beamlines



Eventual Goals:
- Fully automated data collection with multi-wavelength protocol

Present Goals:
- Screen for best crystal growth conditions
- Select the highest-quality samples from a batch
- Discovery of drug leads and protein-ligand complexes
- Enable multi-crystal dataset acquisition
- Perform initial characterization with minimal radiation dose

- **>100 Crystal Samples; liquid nitrogen autofill**
- **All hutch equipment under motorized control**

Reliable* Software for High-Throughput Automation

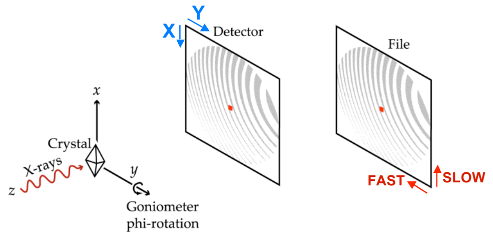
DISTL Zhang *et al.*(2006) **J Appl Cryst** 39:112
- Immediate identification of candidate Bragg spots (< 4 seconds)
- Pick out artifacts like ice rings
- Estimate resolution limits

LABELIT Sauter *et al.*(2004) **J Appl Cryst** 37:399
- Fast characterization of lattice without recourse to graphical interface (< 20 seconds)
- Estimate mosaicity
- Prepare input file for automatic spot integration with *MOSFLM*

Web-Ice González *et al.*(2005) **Acta Cryst** A61:C486
- Remote (Web) access to immediate results
- Tabular view permits comparison of multiple samples
- Calculation of optimal data acquisition settings

* That is, improved. Macromolecular diffraction patterns are very diverse. Basic well-known algorithms (*e.g.*, cell reduction & autoindexing) had to be rewritten to cover outlying cases. Legacy software (pre-2003) relied heavily on human input to recognize the challenging cases.

Inter-Related Coordinate Systems

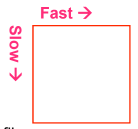


- At present the coordinate relationships must be worked out individually for each detector type.
- We must guard against writing CBF files that contain no information about either the detector type or the coordinate system transformations.

Support for Multiple Detector Vendors

- Detector Geometry
  - Flat surface, square pixels
    - ADSC Quantum 4, 210, 315
    - Mar CCD
    - Mar Image Plate
    - Rigaku Raxis IV and HTC
    - Rigaku Raxis II (after transformation from rectangular shape)
    - Rigaku Saturn 92 CCD
    - MacScience DIP 2030b
  - Only limited success with other geometries
    - Bruker Proteus CCD (1K x 1K)
    - APS SBC 19BM / 19ID requires calibration file

Thanks for providing information
Chris Nielson, ADSC
Michael Blum, Mar USA
Jim Pflugrath, Rigaku



Example Difficulties with Existing File Formats

- Local keyword dialects. The openness of the ADSC file format has allowed different facilities to utilize conflicting keywords.
  - Berkeley Center uses conflicting "DENZO_BEAM_CENTER" and "BEAM_CENTER" tags
- Coordinate system relationships are unspecified
  - There are 8 possible relationships between Detector and File coordinate systems. For ADSC detectors, two of them are in common use at different synchrotrons. LABELIT needs to maintain a list, keyed by DETECTOR serial number.
- Unit of measure is unspecified
  - ESRF writes MAR CCD beam center in mm instead of pixel units
- Redundant information
  - Start phi, end phi, and delta phi all defined.

All of these problems can be corrected with CBF

```
                    # Image CIF Dictionary (imgCIF) #
            # and Crystallographic Binary File Dictionary (CBF) #
           # Extending the Macromolecular CIF Dictionary (mmCIF) #
                         # Version 1.4_DRAFT #
                          # of 2006-07-04 #

This example show the axis specification of the axes of a detector, source and gravity.
The order has been changed as a reminder that the ordering of presentation of tokens is not significant.
The centre of rotation of the detector has been taken to be 68 millimetres in the direction away from the source.

 source . source . 0 0 1 . . .
 gravity . gravity . 0 -1 0 . . .
 tranz translation detector rotz 0 0 1 0 0 -68
 twotheta rotation detector . 1 0 0 . . .
 roty rotation detector twotheta 0 1 0 0 0 -68
 rotz rotation detector roty 0 0 1 0 0 -68
```

First validation step

Legacy detector format (ADSC) ✹→ CBF format ✹→ Legacy format

↓▲ Autoindex　　↓▲ Autoindex

Second validation step

Legacy detector format (ADSC) ✹→ CBF format minus legacy detector info

↓▲ Autoindex　　↓▲ Autoindex

Distribute ✹ validation programs and ▲ test cases in parallel with libCBF code

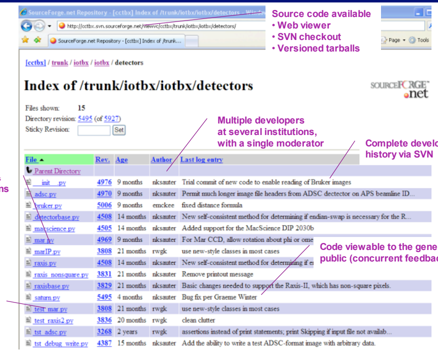
Intended Use of CBF within LABELIT

- Link the C-language CBF library into an existing core library of crystallography algorithms:
  cctbx (Computational Crystallography Toolbox) Grosse-Kunstleve et al. 2002, **J Appl Cryst** 35: 126. (http://cctbx.sourceforge.net)

  As work progresses, information for downloading source code and test cases for this sub-project will be posted at http://cci.lbl.gov/labelit

- The experience gained developing *cctbx* provides valuable ideas for development of the CBF library.

Concurrent *cctbx* Development at Sourceforge.net


SourceForge.net Repository - [cctbx] Index of /trunk/iotbx/iotbx/detectors
http://cctbx.svn.sourceforge.net/viewvc/cctbx/trunk/iotbx/iotbx/detectors/

[cctbx] / trunk / iotbx / iotbx / detectors

Index of /trunk/iotbx/iotbx/detectors

Files shown: 15
Directory revision: 5495 (of 5927)
Sticky Revision: Set

File ▲	Rev.	Age	Author	Last log entry
Parent Directory				
__init__.py	4976	9 months	nksauter	Trial commit of new code to enable reading of Bruker images
adsc.py	4970	9 months	nksauter	Permit much longer image file headers from ADSC dectector on APS beamline ID...
bruker.py	5006	9 months	emckee	fixed distance formula
detectorbase.py	4508	14 months	nksauter	New self-consistent method for determining if endian-swap is necessary for the R...
macscience.py	4505	14 months	nksauter	Added support for the MacScience DIP 2030b
mar.py	4969	9 months	nksauter	For Mar CCD, allow rotation about phi or ome
marIP.py	3808	21 months	rwgk	use new-style classes in most cases
raxis.py	4508	14 months	nksauter	New self-consistent method for determining if e
raxis_nonsquare.py	3831	21 months	nksauter	Remove printout message
raxisbase.py	3829	21 months	nksauter	Basic changes needed to support the Raxis-II, which has non-square pixels.
saturn.py	5495	4 months	nksauter	Bug fix per Graeme Winter
test_mar.py	3808	21 months	rwgk	use new-style classes in most cases
test_raxis2.py	3836	20 months	rwgk	clean clutter
tst_adsc.py	3268	2 years	rwgk	assertions instead of print statements; print Skipping if input file not availab...
tst_debug_write.py	4387	15 months	nksauter	Add the ability to write a test ADSC-format image with arbitrary data.

Source code available
- Web viewer
- SVN checkout
- Versioned tarballs

Multiple developers at several institutions, with a single moderator

Complete development history via SVN

Base class & specializations for detector types. Add CBF here for use within LABELIT.

Code viewable to the general public (concurrent feedback)

Unit tests reside in same directory

General Ettiquette for Checking in Revised Code

- SVN update
- Resolve conflicts
- Run all the unit tests
- Clean up trailing whitespaces
- Double check SVN diffs
- Atomic commit with useful comment

…*cctbx* has suceeded without chaos.

…developers guarantee that committed code works.

- For use within *cctbx & LABELIT*, encapsulate the complexities of CBF function within wrapper C++ classes, exposing only the limited set of features that will actually be used for file reading and data processing (although this could be extended at any time).

```
class CBFAdaptor {
 private:
  cbf_handle cbf_h;
  bool read_header_already;
  void read_header();

 public:
  CBFAdaptor(const std::string& filename);
  ~CBFAdaptor();
  double wavelength();
  double pixel_size();
  double osc_range();
  double osc_start();
  int size1();
  int size2();
  double twotheta();
  double distance(){
    read_header();
    cbf_detector detector1;
    cbf_construct_detector(cbf_h,&detector1,0);
    cbf_get_detector_distance(detector1,
                                    &d_detector_distance);
    cbf_free_detector(detector1);
    return d_detector_distance;
  }
  data_array_t read_data();
};
```

- Abstract CBF data structures into C++ classes so that memory management can be handled by constructors and destructors.

```
//Constructor allocates memory

CBFAdaptor(const std::string& filename):
  filename(filename),read_header_already(false),id(0){
    /* Create the cbf */
    cbf_failnez (cbf_make_handle (&cbf_h))
}


// Destructor frees memory

~CBFAdaptor(){
    /* Free the cbf */
    cbf_failnez (cbf_free_handle (cbf_h))
}
```

• Re-define the error-handling macros so that C++ exceptions will be thrown and handled by the user code.

```
#undef cbf_failnez
#define cbf_failnez(x) {                                              \
    int err;                                                          \
    err = (x);                                                        \
    if (err) {                                                        \
      std::cout<<"error code "<<err<<std::endl;                       \
      throw iotbx::detectors::Error ("CBFlib error in " #x " ");\
    }                                                                 \
}
```

• Expose the C++ wrapper classes at the Python scripting level with Boost.Python bindings

```
#include <boost/python.hpp>
BOOST_PYTHON_MODULE(cbflib_ext)
{
class_<CBFAdaptor >("CBFAdaptor",init<std::string>())
     .def("read_header",          &CBFAdaptor::read_header)
     .def("read_data",            &CBFAdaptor::read_data)
     .def("rawdata",              &CBFAdaptor::read_data)
     .def("pixel_size",           &CBFAdaptor::pixel_size)
     .def("wavelength",           &CBFAdaptor::wavelength)
     .def("distance",             &CBFAdaptor::distance)
     .def("size1",                &CBFAdaptor::size1)
     .def("size2",                &CBFAdaptor::size2)
     .def("osc_range",            &CBFAdaptor::osc_range)
     .def("osc_start",            &CBFAdaptor::osc_start)
     .def("twotheta",             &CBFAdaptor::twotheta)
     .def_readonly("beam_index_slow",&CBFAdaptor::beam_index1)
     .def_readonly("beam_index_fast",&CBFAdaptor::beam_index2)
;
}
```

• Use Python scripts to rapidly prototype new approaches for data processing.

```
#!/usr/bin/python
from iotbx.detectors import ADSCImage,CBFImage

A = ADSCImage("/home/sauter/MB_LP_1_001.img")

C = CBFImage("/home/sauter/MB_LP_1_001.CBF")

A.read(); C.read()

adsc_data = A.linearintdata
cbf_data = C.linearintdata

assert A.parameters == C.parameters

for i in xrange(adsc_data.size()):
  assert adsc_data[i]==cbf_data[i]
```