A generic frame-reading API for a runtime-loadable library

K. Diederichs\textsuperscript{2}, V. Boccone\textsuperscript{1}, A. Förster\textsuperscript{1}, M. Mathes\textsuperscript{1}, V. Pilipp\textsuperscript{1},

High Data Rate MX, 26 May 2016

\textsuperscript{1} Dectris Ltd. (Baden, Switzerland) \textsuperscript{2} Universität Konstanz (Germany)
Current Situation

- **Efficient processing of data sets requires fast parallel (threaded) reading of a dataset.**
- **Developers of processing software have little resources to keep up with the changes, and little incentive to implement yet another format. Especially the more complex it is.**
- **Users are 😞 about workarounds.**
- **Any new format or change will require adaptations in each software packages consuming this kind of data.**

![Processing SW (XDS, DIALS, ...) to Data (HDF5/Nexus, CBF, ...)](image)
Frame reading in XDS

- The steps performed by XDS during processing can be cut down to four operations.
  - Open a resource providing the frames of a dataset
  - Gather information about the contained data
  - Get a frame for analysis
  - Close the resource

- This generalization is not limited to XDS but could also be applied to other programs.

- An interoperable (C, Fortran) API with a defined set of functions allows the separation of code for reading and processing.
Another approach

- A separation between code for reading data and processing data would make everyone's job easier.
- The code for reading data can be physically separated from the code processing data by a plugin mechanism.
- This allows every user to feed the processing software with his data in an optimal way.
- Ideally the community agree on a common set of plugins to ensure reliable usage.
XDS Plugin Mechanism

- XDS loads the library dynamically at runtime using dlopen/dlsym/dlclose.
- The user sets LD_LIBRARY_PATH to it. XDS opens the library specified with a “DETECTOR=libdectrish5toxds” line in XDS.INP, and uses the API-defined routine names.
- This works for (C, C++ and) Fortran, using Fortran 2003 C interoperability: see example at http://cims.nyu.edu/~donev/Fortran/DLL/DLL.Forum.txt
- A first version has been integrated into XDS.

- It is easy to implement a library to read any data.
- There is a clear separation in code/license ownership.
This may offer possibilities not hitherto available

- The library itself can be “signed” (see info array of generic_open_file) by the manufacturer (programmer), and the consumer (data processing program) can check it. The reverse is also true. This may be used to ensure reliable usage.
- A way to prevent the faking of data:
  - Frames can have their MD5SUM (cryptographically) “signed” by the library (see info array of generic_get_data)
  - The consumer program (e.g. XDS) can check the MD5SUM’s integrity
  - (Of course this only makes sense if \{INTEGRATE,XDS_ASCII\}.HKL are also signed, so requires more work to be fully implemented. But it’s a start.)
Reading HDF5

- Some notes about HDF5/NeXus
  - HDF5/NeXus has been questioned as being a data format suitable for the processing of MX data due to speed reasons
  - The hdf5lib has (or at least had) its deficiencies regarding fast parallel reading of data in a threaded environment
  - The hdf5lib is the reference implementation of the HDF5 standard. Thus it has to cover the full functionality and backward compatibility.

- HDF5/NeXus is for DECTRIS the data format of choice for archiving the data of an experiment and we will make our products compliant with the standard.
DECTRIS HDF5 Plug-In

- Two Goals
  - Using the plug-in mechanism to speed up processing of HDF5 data with XDS
  - Show HDF5 is not bound to the library provided by the HDF5 group as the format is documented

- HDF5 plugin
  - Does not use any code of the HDF5 group
  - Only reads a limited subset of HDF5 data
  - DISCLAIMER:
    Proof of principle, far from being production ready code
  - https://github.com/dectris/dectris-xds-plugin
## DECTRIS HDF5 Plug-In

<table>
<thead>
<tr>
<th>images</th>
<th>CBFs</th>
<th>HDF5</th>
<th>HDF5</th>
</tr>
</thead>
<tbody>
<tr>
<td>DETECTOR=</td>
<td>PILATUS</td>
<td>EIGER</td>
<td>libdectrish5toxds</td>
</tr>
<tr>
<td>8 JOBS</td>
<td>152 = total</td>
<td>190</td>
<td>160</td>
</tr>
<tr>
<td>8 PROCESSORS</td>
<td>46 = INIT.LP</td>
<td>64</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>79 = INTEGRATE.LP</td>
<td>99</td>
<td>83</td>
</tr>
<tr>
<td>12 JOBS</td>
<td>141</td>
<td>179</td>
<td>152</td>
</tr>
<tr>
<td>8 PROCESSORS</td>
<td>47</td>
<td>65</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>69</td>
<td>88</td>
<td>75</td>
</tr>
<tr>
<td>8 JOBS</td>
<td>142</td>
<td>178</td>
<td>157</td>
</tr>
<tr>
<td>12 PROCESSORS</td>
<td>46</td>
<td>64</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>73</td>
<td>89</td>
<td>80</td>
</tr>
<tr>
<td>12 JOBS</td>
<td>140</td>
<td>177</td>
<td>151</td>
</tr>
<tr>
<td>12 PROCESSORS</td>
<td>46</td>
<td>63</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>71</td>
<td>89</td>
<td>75</td>
</tr>
<tr>
<td>16 JOBS</td>
<td>140</td>
<td>188</td>
<td>153</td>
</tr>
<tr>
<td>8 PROCESSORS</td>
<td>46</td>
<td>64</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>69</td>
<td>96</td>
<td>76</td>
</tr>
</tbody>
</table>

Datensatz: lysoHG4_t0p01_0p1d_0p1s_d150_360deg_master.h5 (lz4), PSI, 3 Feb 2015
Dell PowerEdge 930, elapsed wall-clock times for individual steps, all data on RAM disk
Summary

- One can implement arbitrary data structures (if one has to) AND fast procedures to read them.
- This takes away the implementation and maintenance burden from software developers - they can concentrate on improving the precision and accuracy of the data.
- Logical separation between data and program; enhances modularity; adapts easily to e.g. HDF5-SWMR.
- No external program like H5ToXds - no intermediate files needed - no conversion penalties (speed).
- Open: API source code is at https://github.com/dectris/dummy_xds_hook.
Thank you for your attention!

www.dectris.com
subroutine generic_open(library, template_name, info_array, error_flag)

! Requirements:
! 'LIBRARY' input (including path, otherwise using LD_LIBRARY_PATH)
! 'TEMPLATE_NAME' input (the resource in image data masterfile)
! 'INFO' (integer array) input Array of (1024) integers:
! INFO(1) = Consumer ID (1:XDS)
! INFO(2) = Version Number of the Consumer software
! INFO(3:8) = Unused
! INFO(9:40) = 1024bit signature of the consumer software
! INFO(>41) = Unused
! 'INFO' (integer array) output Array of (1024) integers:
! INFO(1) = Vendor ID (1:Dectris)
! INFO(2) = Major Version number of the library
! INFO(3) = Minor Version number of the library
! INFO(4) = Parch Version number of the library
! INFO(5) = Linux timestamp of library creation
! INFO(6:8) = Unused
! INFO(9:40) = 1024bit signature of the library
! INFO(>41) = Unused
! 'ERROR_FLAG' output Return values
! 0 Success
! -1 Handle already exists
! -2 Cannot open Library
! -3 Function not found in library
! -4 Master file cannot be opened (coming from C function)
! -10 Consumer identity not supported (coming from C function)
! -11 Consumer identity could not be verified (coming from C function)
! -12 Consumer software version not supported (coming from C function)
subroutine generic_get_header(nx, ny, nbyte, qx, qy, number_of_frames, info_array, error_flag)

! Requirements:
! 'NX' (integer)                  output  Number of pixels along X
! 'NY' (integer)                  output  Number of pixels along Y
! 'NBYTE' (integer)               output  Number of bytes in the image... X*Y*DEPTH
! 'QX' (4*REAL)                   output  Pixel size
! 'QY' (4*REAL)                   output  Pixel size
! 'NUMBER_OF_FRAMES' (integer)    output  Number of frames for the full dataset. So far unused
! 'INFO' (integer array)          input  Array of (1024) integers:
!                                           INFO(>1)     = Unused
!                                           INFO(1)       = Vendor ID (1:Dectris)
!                                           INFO(2)       = Major Version number of the library
!                                           INFO(3)       = Minor Version number of the library
!                                           INFO(4)       = Patch Version number of the library
!                                           INFO(5)       = Linux timestamp of library creation
!                                           INFO(6:64)    = Reserved
!                                           INFO(65:1024) = Dataset parameters
!
! 'ERROR_FLAG'                    output  Return values
! 0      Success
! -1     Cannot open library
! -2     Cannot read header (will come from C function)
! -4     Cannot read dataset informations
!        (will come from plug-in function)
! -10    Error in the determination of the Dataset parameters
!        (will come from plug-in function)
!
Plug-In API

subroutine generic_get_data(frame_number, nx, ny, data_array, info_array, error_flag)

! Requirements:
! 'FRAME_NUMBER' (integer) input Number of frames for the full dataset. So far unused
! 'NX' (integer) input Number of pixels along X
! 'NY' (integer) input Number of pixels along Y
! 'DATA_ARRAY' (integer array) output 1D array containing pixel data with length = NX*NY
! 'INFO' (integer array) output Array of (1024) integers:
! INFO(1) = Vendor ID (1:Dectris)
! INFO(2) = Major Version number of the library
! INFO(3) = Minor Version number of the library
! INFO(4) = Parch Version number of the library
! INFO(5) = Linux timestamp of library creation
! INFO(6:8) = Unused
! INFO(9:40) = 1024bit verification key
! INFO(41:44) = Image MD5 Checksum
! INFO() = Unused
! 'ERROR_FLAG' (integer) output Provides error state condition
! 0 Success
! -1 Cannot open library
! -2 Cannot open frame (will come from C function)
! -3 Datatype not supported (will come from C function)
! -4 Cannot read dataset informations (will come from C function)
! -10 MD5 Checksum Error
! -11 Verification key error