# SANDIA REPORT

SAND2003-8591
Unlimited Release
Printed November 2003

# One User's Report on Sandia Data Objects:

# Evaluation of the DOL and PMO for Use in Feature Characterization

W. S. Koegler, W. P. Kegelmeyer

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.



Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND-2003-8591
Unlimited Release
Printed November 2003

# One User's Report on Sandia Data Objects:

## Evaluation of the DOL and PMO for Use in Feature Characterization

W. S. Koegler and W. P. Kegelmeyer
High Performance Computing & Networking
Sandia National Laboratories
P.O. Box 969
Livermore, CA 94551-9915

## Abstract

The Feature Characterization project (FCDMF) has the goal of building tools that can extract and analyze coherent features in a terabyte dataset. We desire to extend our feature characterization library (FClib) to support a wider variety of complex ASCI data, and to support parallel algorithms. An attractive alternative to extending the library's internal data structures is to replace them with an externally provided data object. This report is the summary of a quick exploration of two candidate data objects in use at Sandia National Laboratories: the Data Object Library (DOL) and the Parallel Mesh Object (PMO). It is our hope that this report will provide information for potential users of the data objects, as well as feedback for the objects' developers.

The data objects were evaluated as to whether they 1) supported the same capabilities as the current version of FClib, 2) provided additional required capabilities, and 3) were relatively easy to use. Both data objects met the requirements of having the same capabilites as FClib and support for parallel algorithms. However, the DOL has a richer set of data structures that more closely align with the current data structures of FClib and our planned extensions. Specifically, the DOL can support time changing geometry, which is needed to represent features as datasets. Unfortunately, the DOL did not meet our ease of use requirement. The PMO was easier to learn and use, but did not support time-changing geometry.

Given the above results, we will extend the FClib API (Application Programming Interface) to handle time-changing geometry. Then we will replace the internal data structures with the DOL, but we will provide the FClib API in addition to the DOL API to support simplified usage.

This page intentionally left blank

# Contents

## Tables

# 1 Introduction

The Feature Characterization project (FCDMF) is part of the ASCI Data Discovery (DD) effort whose goal is to develop tools to extract useful information from large-scale datasets. One focus of DD is to build feature recognition and characterization tools that can extract the interesting coherent features in a terabyte dataset, characterize them in a way that is meaningful to a domain scientist, and generate succinct and useful reports. The initial product of the FCDMF project will be a library of feature characterizations (FClib).

As with any code operating on data, the FCDMF library requires that the data be accessible via some type of in-memory data structure. (For the purpose of this report, when we talk about data structures, our focus will be on the subset of data structures required to manage and represent the simulation data.) The bundle of these data structures and code to manage them can be labeled a *data object*. Object is used here in a generic sense; some of the data objects discussed in this report are neither an official object in the programming sense, nor restricted to being a single object. What is implied, however, is that the bundled data structures and codes form a cohesive whole focused on data access and manipulation. A data object is different from a *data format,* which defines the rules for how data is organized in files on disk. This line can be fuzzy however. Many data formats have APIs for accessing the data from files: some of these APIs approach the functionality of a data object; and some data 'formats' are really a coupled data object and data format. Also, data objects generally know how to read and write data in one or more data formats.

The current implementation of the data structures in FClib support simple finite element meshes and can read and write SAF files. In the near term, we need FClib to support a wider variety of complex ASCI data; and in the future, we will likely need FClib to support parallel algorithms. Extension of the current data structures to meet these requirements would take a large amount of work, which we would rather devote to developing novel characterizations. Another option is to replace the data structures with an available data object. The benefits we seek are to instantly gain the capabilities of the adopted data object and to automatically gain future extensions to the data object. In addition, we would like to partner with the developers of the data object to extend the data object as new requirements appear. Our rough criteria for such a data object include: 1) supports at least the current FClib capabilities; 2) has significant new capabilities; 3) needed future capabilities are present, planned, or negotiable; and, 4) learning to use the new data object takes significantly less time than extending the current one.

This report is the summary of a quick exploration of two candidate data objects, the Data Object Library (DOL) and the Parallel Mesh Object (PMO). The DOL is code currently under development by the ASCI Data Services project. The PMO is open source code hosted by TeraScale, and under development by researchers at Sandia and TeraScale. For each data object, a program was written to report the results of basic queries on a dataset, and two simple characterizations were implemented and used to analyze a subset of the dataset. The results are discussed in comparison to FClib's current implementation and usage. Because this is an implementation based evaluation, the focus will be on the usability and support of current capabilities requirements; the other criteria listed in the preceding paragraph will be discussed only briefly.

It is our hope that this report will provide information for potential users of the data objects, as well as feed-back for the object developers. Therefore, extra technical details are included at the end of some sections and all codes and their outputs are included in the appendices. The reader who does not need these details may safely skip these sections. A tarball of the source codes, data files and output files is available from the first author.

# 2 Experimental Methods: Test Programs & Use Cases

Test programs were written to exercise the data objects. The goals of testing were to explore the dataset and to perform a few simple characterizations on a subset of the dataset. Of particular interest was the effort involved in querying the dataset and implementing the characterizations. For comparison, a similar test program was written for FClib. The results of the testing are presented in Section 3 as six use cases:

1) installation
2) basic library usage
3) querying mesh geometry
4) bounding box implementation (a mesh based characterization)
5) querying fields
6) simple field statistics implementation (a field based characterization)

## *2.1 How to Read the Tables*

A dash entry (-) indicates that the value or concept does not exist for the specified data object. Values in parenthesis are either a comment or the data objects' innate/assumed value. A red, starred (*) entry indicates that the value is obtainable, but requires more coding work than a 'getX()' type function call. A double stared (**) entry was relatively more work to calculate than a single star entry in the same table. A star(s) by itself indicates that the test programs did not bother to calculate the value.

## *2.2 Nomenclature*

Comparing the output of the test programs will be difficult because each data object breaks down and organizes the data slightly differently. The way a data objects organizes the data is its *data model.* Table 1 is a comparison of the nomenclatures for the major data model members of the data objects. A few data formats are also included in Table 1 for comparison. The first column contains the nomenclature of the data model that will be used in this report when generally talking about the organization of the data. Not all of the data objects' members corresponded exactly to the categories in the table; more details will be illuminated in the discussion of the test program results. Also note that the general data model discussed in this paper cannot represent all types of datasets (e.g. time-varying mesh topology), and is meant to serve only as an aid in this particular discussion.

In general, a *dataset* is a container for the data generated by a single simulation run and often corresponds to a single file; a *mesh* is a description of the physical domain of the problem; and a *field* is data associated with a mesh.

A mesh is often broken down into *mesh blocks*. The sum of the mesh blocks on a mesh covers the same extent as the mesh. Mesh blocks are a way of breaking down a mesh into parts with similar properties, either for conceptual or computational purposes. There may be more than one type of mesh blocking defined on a mesh. For example in a parallel object, the mesh may have a set of blocks grouping elements by element type, and a set of blocks grouping elements by owning processor. The line separating mesh from mesh block may seem fuzzy because it is not always clear how a problem should be placed in a dataset. For example, a dataset modeling a car crashing into a wall can be broken down as a single mesh with two mesh blocks, one for the car and one for the wall. Or, it could be broken down as two separate meshes. It is advantageous if a data object has a way of distinguishing conceptual mesh blocks (e.g. separate physical entities) from mesh blocks created to aid computation (e.g. grouping by element type).

*Mesh groups* are a way of grouping arbitrary sets of mesh subelements. Unlike mesh blocks, groups do not need to cover the extent of the mesh. Like mesh blocks, these groups may be either for conceptual or

**Table 1: Nomenclature for Various Data Objects & Data Formats.**

| | Data Objects | | | Data Formats | | |
|---|---|---|---|---|---|---|
| | FClib | PMO | DOL* | SAF | Exodus II† | Ensight |
| Dataset | Dataset | - | ModelDataObject/ MESH_MODEL | Database | Exodus File | Case |
| Mesh | Mesh | Mesh | MeshDataObject/ GEN_MESH | Top Set | - | Part |
| Mesh Block | - | MeshBlock | MESH_BLOCK | Set | Element Block | - |
| Mesh Group | - | NodeSet, EdgeSet, FaceSet, ElementSet | SW_SET | Set | Node Set, Side Set | - |
| Suite | Sequence | FieldManager | Sequence Set | Suite | - | Time Set |
| Step | - | Issue | - | State | - | - |
| Sequence | SeqVariable | - | SW_FIELD | - | - | Variable |
| Field | Variable | Field | SW_FIELD_INST | Field | Results Data | - |
| Relation | - | - | SW_REL | Relation | - | - |

*Mixed case names are C++ objects, uppercase names are corresponding C structs.
†These are guesses at the nomenclature after a quick perusal of the documentation.

computational purposes. One example is the skin of a mesh, which is the set of all faces on the outside of a mesh. Another example is a node set, a set of the nodes on the side of a mesh used to apply boundary conditions. It is advantageous if a data object has a way of classifying common types of mesh groups.

Suites, steps and sequences are all containers for fields. A *suite* organizes fields along a parametric axis, and the values that fields exist at are the suite's *coordinates*. The most common type of suite is a time series. For example, if pressure, velocity and temperature fields exist for a number of time points, the suite contains the fields at each time point as well as the time values at each point, and these time values are the coordinates of the parameter axis. Steps and sequences are orthogonal cross-sections of a suite. A *step* is the set of fields existing for at a single parameter value, e.g. the pressure, velocity and temperature fields at time 0. A *sequence* is the set of fields representing the same variable over all the parameter values, e.g. the pressure fields at time 0 through time t. For simulation codes, steps are the natural way to think about field data. For post-processing codes, sequences will be a popular way of accessing the field data.

Fields are generally associated with some mesh subelement type, such as nodes or elements, and can be distinguished as *nodal* or *element fields*. There can also be *global fields* associated with an entire mesh or dataset.

*Relations* relate assorted entities within a dataset with other assorted entities. Common relations, such as element to node connectivity, are often incorporated into the basic data organization. Relations were not used during testing, but are included in the Table 1 for completeness.

### 2.3 Data Object Codes

It should be noted that both the PMO and the DOL are currently under development and that their current capabilities likely differ from the versions I have chosen to evaluate. These versions, while slightly dated, should still provide adequate indication of general capabilities and usability.

### 2.4 Test Dataset: Can Crush

All test cases will be using the dataset 'can crush'. It is a finite element dataset where a half cylinder, the can, is crushed by a falling brick. The topology of the mesh does not change with time, but there is a displacement field on the vertices. In addition to the can and the brick, two node sets and one side set are defined. There are a number of vector fields defined on the mesh vertices, and a single scalar field defined on the elements.

The can crush dataset was obtained as an ExodusII file (serial) and used directly for testing the PMO and the DOL. The file was converted to SAF for the testing of FClib and additional testing of the DOL. The SAF file was created by first opening the exodus file in Ensight and saving the geometric entities, and then converting the ensight file to SAF using a utility in FClib (ensight2saf). Because of limitations of the Ensight file format, not all of the geometry and field information was translated completely. Most notable was that global variables and node and side sets were dropped. However, this conversion was adequate for evaluation of the simpler parts of the data structure. A separate issue was that neither the DOL nor FClib could read the same SAF file; FClib expects meshes to be stored at the top level and the DOL expected them to be stored as subsets of the top set. Therefore two different SAF files were created for testing; they contained the same data but were organized as appropriate for either FClib or the DOL.

## 3 Results

### 3.1 Data Object Installation

The DOL and the PMO were installed on the two platforms that I currently use for development work. The first was a Sun Workstation (Sunblade 1000) running Solaris 2.8. The second was a Macintosh laptop (PowerBook G4) running OS X 10.2. The installation process took much longer than expected.

The PMO was built using *configure* and was easier to build than the DOL, although not problem free. In retrospect, the PMO build should have been very easy, but it took me a long time to figure out all the environment variables, especially on the Sun. The errors reported by configure and the README file were not very helpful for the problems I had. The best advice I got was from another user of the PMO who suggested I look at the configure log file. To build, the PMO requires a compiler than can build shared libraries and the presence of MPI libraries. To read ExodusII files, the PMO requires only the NetCDF libraries.

The DOL requires the user to create a *make include* file for each computer architecture. This file provides paths to libraries and flags for the compiler. These variables were determined by trial and error while looking at other architecture's include files and poking through the library's make files to see how the variables are used. Better documentation on the minimum set of variables and what they are used for is needed. ExodusII and SAF libraries are needed to read ExodusII and SAF files.

### 3.1.1 PMO Installation Details

The PMO 1.1.1 source code was obtained as a tarball download from TeraScale's website (http://www.terascale.net/pmo) . Documentation was also available from the website in the form of a user manual[1] and html pages generated using Doxygen.

The PMO required MPI libraries, so MPICH was installed on both platforms (this ended up being surprisingly time consuming with gcc issues on the Sun and ssh issues on the PowerMac). To read ExodusII/Nemesis files, netCDF was installed on both platforms. The PMO does not require ExodusII or Nemesis libraries. (ExodusII is based on netCDF, so Exodus files can be accessed via netCDF.) netCDF is open source and easily obtained.

PMO can be built using configure. It does require a compiler that can generate dynamic libraries. By default it tries to use gcc/g++. The basic commands for compiling were `"./configure -with-exodusii; make; make check"` . On both platforms, the following environment variables needed to be set to pick up include and library directories: MPI_INCLUDES, MPI_LDFLAGS, NETCDF_INCLUDES, and NETCDF_LDFLAGS. The MPI libraries also had to be specified as MPI_LIBS = `"lmpich -lsocket -lnsl -lrt"` on the SUN and MPI_LIBS = `"-lmpich -lpmich"` on the PowerBook. In addition, on the Sun the gnu linker had to be explicitly specified using LD (for some reason this was not the default linker for gcc) and `"make check"` failed because the test script called `"test -e"` which is not supported.

### 3.1.2 DOL Installation Details

The DOL 2002/09/12 source code was obtained as a tarball directly from the lead developer[2]. Documentation in the form of text files and html pages were included with the source code.

To read ExodusII/Nemesis files, netCDF, ExodusII and Nemesis libraries are required. NetCDF was already installed on both platforms. ExodusII and Nemesis are part of the SEACAS suite which was difficult to obtain and proved too difficult to build. On the Sun, ExodusII and Nemesis were linked to via /net/troi. They are not available on the PowerBook. To read SAF files, the SAF library is required. SAF is another Sandia code which can be obtained directly from its developers. SAF 1.2.1-pre3 already existed on both platforms.

To build the DOL, first the file 'makeincl' in the DOL root directory needed to be edited to include one of the files in dol/architectures. Then that file needed to be tuned for each platform (see files in Appendix A). The file included the locations of the needed libraries and compiler flags. Next, in the main directory, the command `"make"` makes everything and `"make testrun"` tests the DOL library. To test exodus and saf reading, within the exoio_test or safio_test directories, type `"make testrun."` In addition, a few more hoops had to be jumped through on the PowerBook. First, the PowerBook needs to ranlib libraries and this command had to be added to the 'makeincl' file but only in one place. Second, the building mysteriously stopped between if blocks in some shell commands issued by make. This was countered by adding extra echo statements (?!). Third and finally, the file 'search.h' is missing from the PowerBook's C library. But the DOL does not make use of this file on this platform, so the include statement was removed. These issues were reported to the developers and will be addressed in the next release.

The GL tools did not successfully build on either platform. The exodus file IO library built, but did not pass all of its tests; it appears that it can read, but not write exodus files. I did not get the dynamic libraries to build; but since I only needed that static libraries for this evaluation, I did not bother to find the proper compiler flags.

## 3.2 Basic Usage

FClib is a C library, PMO is a C++ library, and DOL is a mixture of C and C++. With the DOL, the higher levels have been wrapped up as C++ objects, but when you get into the details, it's mostly C structs and functions.

Executables Compiling the programs that used the data objects was straightforward. The test codes were compiled and linked to the data object libraries, and to the libraries that the data objects required.

Headers Neither the DOL or PMO have a single 'master' header file which takes care of importing all header files in the proper order. Such a file is very useful to users who do not need to understand the structure of the underlying code, only how to use it.

Error Codes All of the data objects reported errors as returned int values or null pointers.

Documentation FClib has some automatically generated documentation with functions grouped by what data entities they are operating on. The PMO has a user manual and Doxygen generated html pages that usefully grouped functions into categories. Although there were a large number of member functions for the object and it was a little difficult to find functions. The DOL has some useful general information about sets and relations and some documentation of the C++ data object layers and how to get to the C structs. But there is no really useful documentation on using the functions that manipulate the C structs. To find functions I poked through and grepped the .h files, and then went to the .c files did have comprehensive documentation. For all three code bases, the documentation is not very clear to new users. The PMO User Manual was the best of the three, but didn't go into details beyond the basic manipulations.

### 3.2.1 PMO & DOL Basic Usage Details

There were a few minor things that the data object developers might want to think about changing, at least for release versions, that will make usage more pleasant. First, when compiling with warnings enabled (`"-Wall"`), the DOL reports a large number of warnings of strings that are defined in the include files, but not used. This made it difficult to pick out warnings and errors during development of the test program. Also, when reading data files, both the DOL and PMO generated diagnostic output. The DOL generates ASCII code that can be converted to a graph of the mesh structure; the PMO generates a summary of the exodus file. In addition, the PMO generated a lot of warnings about the exodus file because it was a serial file.

## 3.3 Mesh Queries

The goal of this use case was to open the can crush dataset and query the basic form of the geometric data. Results for the dataset and mesh level are grouped together in Table 2. The first major difference we see at the dataset level is that the PMO, designed as a single mesh, does not have the concept of database. The second major difference is that FClib represents the can and block as meshes instead of mesh blocks on a single mesh, as do the PMO and the DOL. FClib meshes fall between meshes and mesh blocks, and will be treated as mesh blocks for the remainder of the use cases.

The remainder of Table 2 is the information directly available by querying the mesh. The PMO mesh knows its spatial dimension while each mesh block in the DOL and FClib can have different dimensions, and must be queried individually. The PMO and the DOL both have the concept of mesh blocks while FClib does not (as mentioned above, an FClib mesh is really a combination of a mesh and a mesh block). The PMO mesh supports four specific types of mesh groups: node sets, edge sets, face sets and element

**Table 2: Geometry Queries on Datasets and Meshes.**

| | | FClib | PMO | DOL (ex2) | DOL (saf) |
|---|---|---|---|---|---|
| Dataset | Dataset Name | can_fcdmf.saf | - | can.ex2 | can_dol.saf |
| | Number of Meshes | 2 | - | 1 | 1 |
| Mesh | Mesh Name | (see Table 3) | (arbitrary) | General Mesh | TOP_CELL |
| | Spatial Dimension | (see Table 3) | 3 | - | - |
| | Number of Mesh Blocks | - | 2 | 4 | 3 |
| | Number of Coordinate Fields | (see Table 6) | (1) | 1 | 2 |
| | Number of Node Sets | - | 2 | - | - |
| | Number of Edge Sets | - | 0 | - | - |
| | Number of Face Sets | - | 1 | - | - |
| | Number of Element Sets | - | 0 | - | - |
| | Number of Sets | - | - | 7 | 15 |
| | Number of Relations | - | - | 6 | 14 |

sets. The DOL does not appear to have an interface for specific mesh groups, and also did not import the node and side sets present in the exodus file. But the DOL does have a generic interface for sets and relations through which mesh groups can be manipulated, although without code modifications discovery of these mesh groups amongst all sets in a would be very difficult. At the mesh level, more information is directly available from the PMO than FClib or the DOL.

The results of geometry queries on the mesh blocks (or meshes for FClib) are shown in Table 3. Roughly the same information is available for all three data objects: the mesh block name, spatial dimension and numbers of nodes and elements. Both the PMO and DOL also had support for edge and face mesh subelement, but this was not explored in the testing. All three data objects also provided ways to use the element type to get the number of vertices per element and other topological information.

One significant difference between the three data objects (and also between the Exodus and SAF versions of the DOL test code) was where they stored the mesh coordinates. The PMO and the DOL reading exodus stored the coordinates over all nodes present in the mesh, while the DOL reading SAF and FClib stored the coordinates on the separate mesh blocks. This is probably directly related to the exodus to saf translation via Ensight. For post-processing this form of coordinate access makes sense; users who wish only to examine the can block do not need to load or manipulate the entire dataset's coordinates. Another difference is that the DOL does not have a simple 'getX()' type interface for these queries.

### 3.3.1 FClib Geometry Query Details

To open the dataset (more directly related to the mesh level than dataset), first FClib is initialized and then open_dataset() is called. The name of the dataset is the name of the file. The number of meshes, and their names, can be directly queried from the dataset. This takes three function calls and involves two entities, the library and the dataset.

**Table 3: Geometry Queries on Mesh Blocks.**

| | FClib | PMO | DOL (ex2) | DOL (saf) |
|---|---|---|---|---|
| Mesh Block Name | | | Top_Set | TOP_CELL |
| Spatial Dimension | | | **(no coordinates) | **(no coordinates) |
| Number of Nodes | | | *10088 | *10088 |
| Number of Elements | | | *7152 | *7152 |
| Element Type | | | *OtherPolyhedron | *Hexahedron |
| Mesh Block Name | | | Domain 0 | |
| Spatial Dimension | | | **3 | |
| Number of Nodes | | | *10088 | |
| Number of Elements | | | *7152 | |
| Element Type | | | *OtherPolyhedron | |
| Mesh Block Name | Block ID 1 – HEX | 1 | Exodus_Domain_0 _Block_1 | Block ID 1 – HEX |
| Spatial Dimension | 3 | (same as mesh) | **(no coordinates) | **3 |
| Number of Nodes | 6724 | *6724 | *0 | *6724 |
| Number of Elements | 4800 | 4800 | *4800 | *4800 |
| Element Type | hexahedron | HEX8 | *Hexahedron | *Hexahedron |
| Mesh Block Name | Block ID 2 – HEX | 2 | Exodus_Domain_0 _Block_2 | Block ID 2 – HEX |
| Spatial Dimension | 3 | (same as mesh) | **(no coordinates) | **3 |
| Number of Nodes | 3364 | *3365 | *0 | *3364 |
| Number of Elements | 2352 | 2352 | *2352 | *2352 |
| Element Type | HEX | HEX8 | *Hexahedron | *Hexahedron |

*Information not directly available using getX() type queries.
**Like single starred (*) entries, but takes significantly more work to obtain values.

Each mesh (these are more directly related to the mesh block level) is opened via its name and queried for number of vertices, number and type of elements, and coordinate field dimensionality. Information about number of vertices, edges and faces per element is available via a function call given the element type.

### 3.3.2 PMO Geometry Query Details

To open the mesh (no dataset level), first MPI is initialized, next a mesh object is created, and finally, a mesh reader object is created and handed to the mesh object. The name of the mesh is set by the user during mesh instantiation. This takes six function calls and involves 3 entities, the MPI library, the PMO mesh, and the PMO reader/writer. The reader object reports a large number of errors because it was expecting a parallel Exodus file, and then spits out a bunch of summary information. Direct queries on the mesh return the dimensionality of the mesh as well as the number of mesh blocks and side/edge/face/element sets.

Each mesh block can be queried by name and it is easy to get element information. The number of elements comes directly from the mesh block, and the element information comes from an ElemTopology object. The number of nodes was not directly available. The helper objects NodeList, EdgeList, and FaceList are provided to calculate node, edge, and face information on mesh blocks. However, creating a NodeList failed (I believe that NodeList implementation was not yet complete). So, I assembled my own list of node IDs by getting the block's elements, then getting the elements' nodes' IDs, and making a unique list to find the number of nodes.

### 3.3.3 DOL Geometry Query Details

To open the dataset, first a DO_FileObject was created, then a MESH_MODEL was created, then the mesh description was read into a MESH_MODEL, then finally, a ModelDataObject was created from the MESH_MODEL and the DO_FileObject. There's also a DataObjectList involved, which is an object for managing model objects. This involved 4 function calls and involved 4 different entities of which the MESH_MODEL appears only as an intermediate and DataObjectList is not used by the testing code. The name of the dataset is the name of the file.

The ModelDataObject (the dataset) is queried for its MeshDataObjects, which are the meshes. Direct queries on the mesh reported the number of mesh blocks and returned an array of MESH_BLOCK pointers can be obtained by function call from the MeshDataObject.

To find the number of nodes and elements in a mesh block, first you check to see if the appropriate MESH_CELLS pointer exists in the MESH_BLOCK struct (there are four: nodes, edges, faces and zones), and then if it is, get the MESH_CELL's SW_SET pointer, and then query that set for its member count. The element MESH_CELLS can also be queried for element type, and basic information about numbers of nodes and edges for each cell type can be obtained with function calls. To get the dimensionality of the coordinates, first you must determine if there is a coordinate field on the node's set and then get the information from the SW_FIELD struct. This involved a macro to scan the SW_FIELD_REFs on the set and checking the flag 'is_coord' on each SW_FIELD_REF's field.

## *3.4 Bounding Box Implementation*

The goal of this use case was to implement a simple characterization that only involved the geometric entities of the mesh. The bounding box characterization currently implemented in FClib returns the coordinates of the lower and upper corners of an axis-aligned hexahedron, which just encompasses all nodes in the specified mesh. A bounding box routine was implemented for the PMO and the DOL and applied to the mesh blocks. The results are shown in Table 4. The basic algorithm iterates over all nodes in the mesh blocks and all coordinate axis, and saves the minimum and maximum values that define the extents of the bounding box.

Implementation of the bounding box routine was relatively straightforward for the PMO. Implementation for the DOL was more difficult for three reasons. First, mesh blocks are not guaranteed to have a coordinate field. For this dataset, the coordinates were available only on the domain block in the exodus case (similar to the PMO) or on the can and brick blocks for the saf case (similar to FClib). A second difficulty was that necessary information, including the coordinates, is not readily available using a 'getX()' type call. Lastly, in the DOL fields can be made up of other fields; this was the case for the coordinate field on the domain block in the exodus case. The coordinate field pointed to individual fields for x, y, and z. Because this is only a test implementation, the extra effort to code this was not made.

**Table 4: Bounding Box Characterization.**

| | Bounding Box | | Bounding Box Returned Correct Result | | | |
|---|---|---|---|---|---|---|
| Mesh Block | $X_{min}$ | $X_{max}$ | FCDMF | PMO | DOL (ex2) | DOL (saf) |
| TOP_CELL | | | | | No coordinates | No coordinates |
| Domain 0 | (-7.88, 0, -15)† | (8.31, 8, 4.78)† | | | Implementation not completed | |
| Block ID 1 | (-5.2, 0, -15)† | (5.2, 5.2, 0)† | Yes | Yes | No coordinates | Yes |
| Block ID 2 | (-7.88, 0, -0.46)† | (8.31, 8, 4.78)† | Yes | Yes | No coordinates | Yes |

†Values are rounded to two decimal places for clarity

### 3.4.1 FClib Bounding Box Details

The bounding box characterization has already been implemented in FCDMF and takes the mesh as input and returns the bounding box coordinates. Within the characterization, the mesh is queried for the number of nodes and the coordinates' data type (e.g. int, float). The raw coordinates values are obtained from the mesh and were cast to the proper data type before use.

### 3.4.2 PMO Bounding Box Details

The characterization took as input the mesh and mesh block's name. The number of nodes on the mesh block, and their IDs, were calculated (see Section 3.3.2). The node coordinates for the entire mesh were retrieved (and locked) by a member function on the mesh, and then iterated over using the list of the mesh block's node IDs. The PMO's coordinates are always doubles.

### 3.4.3 DOL Bounding Box Details

The characterization took as input the mesh and the mesh block. The mesh was needed for its sophisticated field retrieval abilities (the C++ objects are more savvy about file issues). To get the coordinate field, first the node set is retrieved from the mesh block's C struct dereferencing. Next a macro is used to scan the node set's field to find a coordinate field. Then the field is queried for the number of values, dimensionality and data type (e.g. float, int, etc.) and the field type. A simple field's field instance is retrieved by a call on the mesh. Finally, the coordinate field is cast to the appropriate type and the bounding box calculated. The dimensionality is returned along with the bounding box coordinates.

The current implementation exits with an error if the field is not simple. For a decomposed field (i.e. it represents a number of related component arrays), it should be possible to adjust the coordinate iterations to deal with three component arrays instead of all of the components interleaved. However, retrieving the decomposed field members was not obvious since they were stored in a linked list and I was not sure what order they would come back in. Checking for names (e.g. x, y, z) is too brittle. There was a convenience function (swFieldInstGetArray) that was supposed to return values of the component fields, but its use caused strange behavior in the program (probably memory related).

## *3.5 Field Queries*

The goal of this use case was to query field information on the dataset. Table 5 shows the results of field queries at the data set and suite levels. The first difference we see is that information about suites exists at the dataset level for FClib and at the mesh level for the PMO and DOL. The biggest difference between

**Table 5: Sequence and Field Queries on Dataset and Mesh.**

| | | FClib | PMO | DOL (ex2) | DOL (saf) |
|---|---|---|---|---|---|
| Dataset | Number of Suites | 1 | - | - | - |
| Mesh | Number of Suites | - | (0 or 1) | 1 | 9 |
| | Number of Time Fields | - | - | 1 | 1 |
| | Number of Coordinate Fields | - | - | 1 | 2 |
| | Number of Fields | - | - | 16 | 11 |
| Suite | Number of Steps | 44 | 44 | 44 | 44 |
| | Suite Coords. Data Type | float | (std::str) | *FLOAT | *FLOAT |
| | Suite Coords. Data Organization | - | - | *VE[§] | *VE[§] |
| | Number of Global Fields | - | 6[†] | - | - |
| | Number of Nodal Fields | - | 9[†] | - | - |
| | Number of Element Fields | - | *2[†] | - | - |

*Information not directly available using getX() type queries.
[†]Values for first step (not necessarily the same for all steps).
[§]Indexed by value, then by entry (e.g. xxxyyyzzz as opposed to xyzxyzxyz).

the data objects is that the DOL and FClib can have multiple suites, while a PMO mesh has only one (embodied as the FieldManager object). The DOL mesh can report the total number of fields as well as the number of time and coordinate fields that it contains.

All three data objects' suites provide the number of steps and the values of the steps (i.e. time values of time steps). The PMO stores these values as strings while the DOL and FClib support a variety of data types. The FClib and DOL suites do not provide information about the fields associated with them, while the PMO suite knows the number of global, nodal and, per block, element associated fields. The DOL treats the time coordinates as a field, and so has extra flexibility for storing this information.

The results of field queries on the mesh blocks is given in Table 6. Roughly the same information about fields can be determined on all three data objects, however the access patterns are quite different. For FClib, the total number of sequences can be queried directly, while for both the PMO and the DOL, only the number of fields for each type of mesh subelement association (global, nodal, element) are available directly. In fact, for the PMO, you need to know the time step, the subelement association, and the extent (the specific set of nodes, elements, etc., that the field is associated with) before you can get the fields. It should also be noted that the DOL field is really a sequence that stores the metadata for related DOL field instances.

The results of querying the first time step of (FClib and PMO) or the entire (DOL) field named 'EQPS' are shown in Figure 7. All data objects reported the number of data points–essentially the number subelements of the mesh that the field is defined on–and provided additional necessary information to calculate the total number of data values. The most straight forward of which was the DOL, which provides the number of data values per point. FClib and the PMO provided a little more information about the mathematical shape of the data. The PMO specifies a math type (e.g. "SCALAR", "VECTOR3D", "SYMTEN21", etc.), which defines the number of values to represent a data point (e.g. 1 for scalar, 3 for vector3d, etc). Somewhat similarly, FClib specifies a data dimensionality and the lengths

**Table 6: Field Queries on Mesh Blocks.**

| | FClib | PMO | DOL (ex2) | DOL (saf) |
|---|---|---|---|---|
| Mesh Block Name | | | Top_Set | TOP_CELL |
| Num. of Global Fields | | | - | - |
| Num. of Nodal Fields | | | *0 | *0 |
| Num. of Zonal Fields | | | *0 | *0 |
| Num. of Coord. Fields | | | *0 | *0 |
| Number of Sequences | | | *0 | *0 |
| Total Number of Fields | | | *0 | *0 |
| Mesh Block Name | | | Domain 0 | |
| Num. Global Fields | | | - | |
| Num. of Nodal Fields | | | *13 | |
| Num. of Zonal Fields | | | *0 | |
| Num. of Coord. Fields | | | **1 | |
| Number of Sequences | | | **9 | |
| Total Number of Fields | | | *13 | |
| Mesh Block Name | Block ID 1 - HEX | 1 | Exodus_Domain_0_Block_1 | Block ID 1 - HEX |
| Num. of Global Fields | - | 6† | - | - |
| Num. of Nodal Fields | * | 9† | *0 | *4 |
| Num. of Zonal Fields | * | 1† | *1 | *1 |
| Num. of Coord. Fields | (1) | - | **0 | **1 |
| Number of Sequences | 4 | - | **1 | **4 |
| Total Number of Fields | 177 | *16† | *1 | *5 |
| Mesh Block Name | Block ID 2 - HEX | 2 | Exodus_Domain_0_Block_2 | Block ID 2 - HEX |
| Num. of Global Fields | - | 6† | - | - |
| Num. of Nodal Fields | * | 9† | *0 | *4 |
| Num. of Zonal Fields | * | 1† | *1 | *1 |
| Num. of Coord. Fields | 1 | - | **0 | **1 |
| Number of Sequences | 4 | - | **1 | **4 |
| Total Number of Fields | 177 | *16† | *1 | *5 |

*Information not directly available by getX() type queries.
**Like single starred (*) entries, but takes significantly more work to obtain values.
†Values for first step (not necessarily the same for all steps).

**Table 7: Queries on Field Named 'EQPS'.**

| | FClib | PMO | DOL (ex2 & saf) |
|---|---|---|---|
| Field Name | EQPS_0 | EQPS | EQPS |
| Data Association | elements | - | - |
| Number of Data Points | 4800 | 4800 | 4800 |
| Number of Data Dimensions | 0 | - | - |
| Data Dimension Lengths | N/A | - | - |
| Math Type | - | SCALAR | - |
| Cardinality | - | 1 | - |
| Number of Values per Point | - | - | 1 |
| Total Number of Values | *4800 | *4800 | *4800 |
| Persistence Type | - | STATE | - |
| Data Type | float | REAL | FLOAT |
| Field Type | - | - | FT_SIMPLE |
| Data Organization | - | - | VE[§] |

*Information not directly available using getX() type queries.
[§]Indexed by value, then by entry (e.g. xxxyyyzzz as opposed to xyzxyzxyz).

of those dimension (see details section below). In addition, the PMO also has a value for cardinality, or the number of points per mesh subelement (e.g. if the field is defined on the center and sides of triangle elements, the cardinality is four).

All three data objects also provide a way to store different data types (e.g. ints, floats, etc). However, the DOL also provides different ways of specifying the storage shape of the data, i.e. whether the field is decomposed and how the data components are interleaved (xxxyyyzz vs. xyzxyzxyz). In the PMO and FClib, the field values have a set storage shape. The PMO also provides a flag for persistence type, which indicates the conditions under which the field is to be written during simulations.

### 3.5.1 FClib Field Query Details

The dataset is queried for the number and the names of its suites. A suite is opened by name, and information about the coordinates of the suite (e.g. the time step values) can be obtained directly from the suite. Information about the fields in the suite is not available and must be obtained via the meshes.

FClib meshes can report total number of fields and the number of field sequences. One of the fields will be the coordinate field. For this dataset, the field sequences have 44 steps and we can see that the total number of fields is 44 * 4 + 1 = 177.

When querying a field, the number of data points is directly available, but the number of values per data point has to be calculated from the data dimensionality and the data length array parameters. For example, for 3D vector, the data dimensionality = 1, the data length = { 3 }, and therefore the number of values per data point = 3; for a tensor, the data dimensionality = 2, the data length = { 3, 3 }, and the number of values per data point = 3*3 = 9.

### 3.5.2 PMO Field Query Details

In the PMO, the FieldManager object serves as a mesh object's suite. The field manager is created and handed the same mesh reader object that was given to the mesh when it was created. The field manager was then handed to the mesh (I'm not sure if this step is necessary). The first step in accessing the fields was to ask the field manager for the number and names of its issues (e.g. time steps). Next you can ask the field manager for the number and names of extents for a particular issue and a particular scope. The combination of a scope and an extent is meant to be used as a name space so that fields on different mesh entities can have the same names and still be resolved. The scope defines a field's relationship to the subelements of the mesh and there are a restricted set of permission values (only "GLOBAL", "NODAL", and "ELEMBLOCK" were applicable to this dataset). The extent can be any string the user wishes, but is usually "ALL" or the name of the element block, node set, etc., that the field is over. Next, the field manager is asked for the number and names of the fields for a particular issue, scope and extent. The first thing to note is that the number of extents (and also fields) may vary from time step to time step. A second thing to note is that to find the total number of fields on the mesh or a mesh block, or to find a particular field by name, you will have to iterate over a number of loops. And, you have to keep the scope and extent information along with the field name in order to access the field values.

After the scope and extent were determined for the field 'EQPS', information about a specific time step can be accessed with a single call on the field manager, 'readFieldRegistration', which reports the data type, math type, persistence, number of values in the field, and cardinality.

### 3.5.3 DOL Field Query Details

The number of time fields, coordinate fields, and the total number of fields on a mesh can be obtained directly from the MeshDataObject. A sequence set can also be obtained from the mesh and queried with a function call for the number of its members (number of time steps). I assume it is also possible to obtain the time field from this set, and perhaps the fields that reference it, but I could not figure out how to get this information from the structures' members. Fortunately a single time field was available directly from the mesh and I was able to use that for the field characterization in Section 3.6.

To query the fields on a mesh block, first you check to see if the appropriate MESH_CELLS pointer exists in the MESH_BLOCK struct (there are four: nodes, edges, faces and zones), and then if it is, get the MESH_CELL's SW_SET pointer, and walk the field reference linked list. Fields can be classified as time fields, coordinate fields, or sequence fields by checking flag values in the SW_FIELD structure. Fields are accessed most directly by knowing their mesh subelement association.

## *3.6 Simple Field Statistics Implementation*

The purpose of this use case was to implement a simple characterization which involved fields. The chosen characterization was to determine the minimum and maximum values of a field. It also reports the data point IDs of those minimum and maximum values.

The algorithm used is very simple. The current minimum and maximum values are initialized with the first data point. Each of the remaining data points is examined and if the data point is less than or greater than the current minimum or maximum values, respectively, that data point replaces the current value (and the current minimum or maximum ID is also updated). The results for all time steps of the field named 'EQPS' were essentially the same for all three data objects, except for round off error in that last few digits for the objects reading the SAF files (because conversion via Ensight converted the double values to floats).

Implementation was easier than that for the bounding box characterization; but only because it was working on simple scalar data. The only complicating factor was that for all three data objects, a field can be of a variety of different data types (int, float, etc). All three data objects test for the data type and then use a version of the algorithm for that data type.

### 3.6.1 FClib Field Statistics Details

The min/max field characterization has already been implemented in FClib and takes a field as input and returns the min and max values as floats. Within the characterization, the field is queried for the number of data points, the data type, data dimensionality and a void pointer to the large data. For scalar fields, the algorithm is implemented as above. For vector fields, the algorithm is performed on the magnitude of the vectors and the values reported will be the minimum and maximum magnitude values. For all other math types (tensor, etc), the characterization exits with an error. The algorithms are actually implemented multiple times in a switch block, which tests the data type. Within the block, the void data is cast to the appropriate data type before use.

### 3.6.2 PMO Field Statistics Details

To specify a single field, the PMO field statistics characterization takes as input the mesh, the issue name (time step), the scope name, the extent name, and finally the field name. The field manager is obtained from the mesh and a single query, 'readFieldRegistration', reports all of the field's information and the size of the field (total number of values) is obtained from 'readFieldSize'. Because data type was specified by a string instead of an enumerated type, an if/else if block was used instead of a switch block. Inside the block, the storage for the large data array is allocated by the caller and then filled in by a 'readXXXField' on the field manager where XXX is Int or Real. The algorithm is performed, then the large data is freed.

### 3.6.3 DOL Field Statistics Details

To specify a single field, the DOL field statistics characterization takes as input the DOL field and an instance index. In addition, the mesh is passed in because it will be the entity asked for the large data. The number of data points and the data type are found as members of the SW_FIELD structure. An SW_FIELD_INST field instance is obtained from the mesh by specifying the field and the instance index. For a scalar field, the raw data is available via a void pointer on the field instance structure. Like FClib, a switch block tests the data type, casts the data, and then performs the algorithm.

# 4 Discussion

From the results of the testing in the previous section, it is clear that both the DOL and the PMO can provide the same basic capabilities currently available in FClib. Here we will discuss additional requirements for usability and extended capabilities. These requirements will be addressed in terms of two related issues. The first issue, the complexity of the data models, has bearing on both capabilities and usability. The second issue is that usage requirements for post-processing will differ from usage requirements for data generation. These two issues will be briefly discussed in general terms before addressing the requirements.

## *4.1 The Data Models*

While all three data objects had essentially the same view of mesh datasets (see Table 1), the DOL's data model differs in a fundamental way from FClib and the PMO. The DOL uses sets, relations, and fields as

the basic building blocks for the data structures. (SAF uses a similar model.) The advantage of this type of model is that a wide variety of types of datasets (some might argue all types) can be represented as a group of sets, relations, and fields. Unfortunately, proper interpretation of a group of sets, relations, and fields can be very difficult, if not impossible, if nothing is known a priori about the dataset.

However, the DOL also has additional data structures specific to mesh datasets to help with the interpretation of the sets, relations, and fields. For example, the MESH_BLOCK structure has pointers to the sets that represent the nodes and elements of the mesh block. The DOL also stores some mesh specific information on the sets, relations and fields (e.g. an "is_coord" flag on a field). In addition, the DOL provides functions for retrieving the mesh specific data like element connectivities. (It is interesting that knowledge of relations was not required to perform the testing.) The DOL has structures that allow a dataset to be accessed as a mesh, but the real data is stored on the sets, relations, and fields. In the PMO and FClib, data is stored in the specific mesh data structures. (FClib has a slightly more general data model in that it stores the coordinates as fields instead of in a dedicated coordinates data structure.)

There are at least two major advantages of a sets, relations, and fields data model. The first advantage is that the data model is more easily extensible to new dataset types. This is due to the fact that most of the work is done at the building block level; higher-level routines can be added but the low-level stuff has already been implemented. Anything that can be represented as sets, relations, and fields can immediately be represented. However, data structures and routines that ease dataset access may need to be extended. Another advantage of the sets, relations, and fields data model, is the possibility of performing relational math on the sets and relationships. For example, an inverse routine can calculate the node to element connectivity relation from the element to node connectivity relation.

The major disadvantage of a sets, relations, and fields data model is that it tends to be more complicated than a data model with more specific data structures. Although a pure sets, relations, and fields interface would be simple, the data stored would be too difficult to interpret. So a user must learn about how the data model stores and accesses mesh data in addition to learning about sets, relations, and fields. However, data models that represent complex datasets (e.g. time-varying mesh topology), or that attempt to represent more than one type of dataset, will have more complicated usage. Since sets, relations, and fields type data models are expected to represent a variety of dataset types, some complication of usage is probably unavoidable.

## *4.2 Data Access: Data Generation Vs. Post-Processing*

Although data objects usually support both data generation and data consumption, their design may be geared more towards one or the other. There are often fundamental differences in data access patterns between processes that generate data, and processes that consume data. Data objects designed for one of these processes may suffer in usability and performance when data access is of the other type.

Differences in access pattern are most exemplified by the access of time-varying field data. One of the major differences in these access patterns is due to how the processes see time. The most common type of simulation is to evolve a system over time. New data is generated for all fields at each time step, and so fields are by necessity written as states (a group of fields at a specific time slice, see Section 2.2). Typically a data generation oriented data object will only know how to read fields as states. On the other hand, a typical post-processing application will want to access a field as a sequence (the field at all time points).

Another difference in the data access pattern may be due to the expected coverage of the data. This difference can involve both mesh and field access. Each calculation of a simulation generally touches all mesh structures and all fields. Because the same algorithm is typically applied to the same types of mesh

structure and fields, data access typically groups mesh structures and fields by type. (Loop over global fields and do this, loop over nodal fields and do that, etc.) On the other hand, post-processing applications may only want a small portion of the data at a time. Typically a mesh or field is retrieved by name, not type, although the type might be asked for after retrieval so that the proper algorithms can be used.

FClib is unabashedly a post-processing data object. It was designed to allow users to easily access only the meshes and fields they are interested. Data access is usually by name and fields are accessed as sequences. The PMO's emphasis is on data generation. Data access in the PMO is structured by time slice and data type and it has the additional machinery for keeping a minimum number of states in memory during a simulation (e.g. like the current and previous time steps). The DOL falls somewhere in between but more in the post-processing camp; its concept of fields as sequences fits the post-processing view.

### *4.3 Extended Capabilities*

Currently FClib can read, write and manipulate time-varying fields on finite element meshes and can handle a wide assortment of element and field types. The meshes are limited to those with static topology. The single most important new capability that FClib needs is the ability to handle time-varying topology to support the creation of feature datasets. That is, once features have been identified and tracked, we desire to subset them into a new dataset; and because the feature's members will vary with time, these datasets will have time-varying topology. Another important capability is that the data object will support all the types of meshes that FClib plans to support. However, we do not yet know what these will be and this requirement will change over time. One final requirement, which is not planned in the near term, is parallel capabilities. When the datasets get large enough, or because they exist on multiple machines, performance may necessitate parallel characterizations.

Both the DOL and the PMO had significant capabilities beyond those currently existing in FClib. Most notably, the DOL can support time-varying topology and relational math, and the PMO has a complete parallel implementation (although this wasn't evaluated). The DOL has parallel support, but does not yet have a complete parallel implementation. The PMO would face major revision to support time-varying topology. The DOL's data model (sets, relations, and fields) also suggests that it will be able to handle a wider variety of dataset types and be more easily extensible to a variety of purposes.

Future plans for DOL probably align more closely with those for FClib as their focus is also on ASCI data.

### *4.4 Usability*

We are expecting two levels of users for the FClib. The first set of users should see a very simple data interface which allows them to choose the meshes and fields which are of interest to them, and then to choose from a menu, characterizations to apply to these meshes and fields. The second set of users would be characterization developers and would need to manipulate the data contained in the meshes and fields, but are still unlikely to need to truly understand the fine details of the data structures. We are hoping to use a data object that is simple enough to expose directly to the first level of users, but rich enough to satisfy the needs of the second level of users.

Because installation and linking of both the PMO and the DOL was relatively easy, the discussion of usability will focus on their APIs (Application Programming Interface). There are two issues here: 1) how easy the interface is to learn and use, in general; and 2) if the interface aligns with our post-processing orientation.

### 4.4.1 Learning the API

I was surprised at how long it took to learn to use the PMO and the DOL considering that I have experience with using SAF and Ensight, and writing the current implementation of FClib. I had originally thought it would only take a few weeks to write the testing code, but it has easily taken five times that long. The major factor was probably the lack of a standard data model to represent meshes and I had to learn two new data models and somehow combine them with my pre-existing ideas. In addition, there were deficiencies in the documentation and there were no example usage codes. And the DOL is difficult to use.

In retrospect, the PMO has a relatively straightforward API; The PMO has a limited interface (although still large) and some decent documentation. However, the manual is not as good (and it is not mean to be) as the Doxygen generated documentation.

On the other hand, the DOL was rather difficult to use. It does not yet have a consistent API or external documentation. (However, the helper functions do follow a consistent naming convention and there is good documentation within the code.) The interface is complicated and very fat-data object entities can be accessed as C++ objects, or by C struct dereferencing, or by using helper functions. And the only documentation, while complete, exists in the code and the user has to delve for it.

### 4.4.2 Using the API

Some requirements for a simple interface include: 1) a minimal number of entity types, 2) separately manipulatable entities, and 3) a minimal number of calls for data access. Table 8 shows some numbers gleaned from the test cases to help evaluate these requirements. In general, low values are desirable as they indicate a simple interface; however, an interface with limited capabilities could also have low values. Overall, FClib has the lowest values of the three data objects as would be expected because its interface has been designed to satisfy these requirements.

The first section of Table 8 lists the minimum number of data object entities, and 'accesses' of those entities that would needed in order to apply the implemented characterizations. The modes of accessing the data include calls on the data objects and dereferencing of structure member values. For the known data, it is assumed that the user knows the names of the mesh blocks and fields they are interested in; for the unknown data, mesh block and field names must be discovered. Overall, the DOL has more data entities, and therefore more access calls are needed. What is interesting are the relative difference in the number of data accesses for unknown data compared to known data. For discovery of fields in the PMO, this takes many more access calls.

The second section of the table are the number of data object entities and accesses that exist within the characterizations. FClib has function calls that returns all necessary meta data about a data object entity at once, while the PMO and DOL tend to return meta data one value at a time (member functions for the PMO data entities and struct member dereferenceing for the DOL). One exception is that the PMO has a single function, readFieldRegistration, that returns all information needed to interpret the large data array.

The total number of entity types in the test program also includes entities that were used for the general querying of information about the data. The starred entries in the tables in the results section indicated that a value was not easily obtainable by a 'getX()' type function call. A large number data entities or a large number of starred entries indicates an interface that is not as simple to use. In addition, there are a number of data structures in the DOL that should not be exposed to our users; these structures include the reference The DOL also needs more helper functions to return often queried information like the mesh blocks queries in Table 3.

**Table 8: API Measures of Test Programs.**

| | FClib | PMO | DOL |
|---|---|---|---|
| Minimum number of entity types needed to apply bounding box | 2 | 2 | 6 |
| Minimum number of entity types needed to apply field statistics | 3 | 3 | 8 |
| Min. num. of data 'accesses' needed to apply bounding box to known data | 3 | 4 | 9 |
| Min. num. of data 'accesses' needed to apply field statistics to known data | 4 | 7 | 9 |
| Min. num. of data 'accesses' needed to apply bounding box to unknown data | 4 | 6 | 8 |
| Min. num. of data 'accesses' needed to apply field statistics to unknown data | 6 | 15 | 10 |
| Number of entity types needed in bounding box | 1 | 1 | 7 |
| Number of entity types needed in statistics | 1 | 2 | 3 |
| Number of data object calls/dereferences in bounding box | 2 | 7 | 13 |
| Number of data object calls/dereferences in field statistics | 2 | 4 | 6 |
| Total number of entity types in test program | 4 | 3 | 11 |
| Number of starred entries for geometry queries on mesh blocks (see Table 3) | 0 of 5 | 1 of 5 | 4 of 5 |
| Number of starred entries for field queries on mesh blocks (see Table 6) | 2 of 6 | 1 of 5 | 5 of 6 |
| Number of input arguments to bounding box characterization | 1 | 2 | 2 |
| Number of input arguments to field statistics characterization | 1 | 5 | 3 |
| Lines of code in bounding box characterization | 30 | 48 | 95 |
| Lines of code in field statistics characterization | 105 | 56 | 64 |
| Lines of code in test program 'main' | 157 | 168 | 248 |

The fourth section of the table are the number of arguments needed as inputs to the characterizations. This is an indicator of how separately manipulatable the data object entities are. For FClib, a mesh block is the only data required for the bounding box and field statistics characterizations. The DOL, like FClib, has separate data entities for mesh blocks and fields, but the owning mesh is also passed in because it is needed to properly access the large data arrays. For fields, a sequence index is also necessary to get the proper time step. The PMO does not have a separate mesh block or field objects, so the mesh and the appropriate identifying strings are passed to the characterizations. A field on the PMO requires three strings to specify its context (scope, extent and name) in addition to the name of the time step.

The last section of the table shows the number of lines of code used in different parts of the test programs. The lines of code for FClib's field statistics characterization is much greater than the other two data objects' because it has code for doing statistics on vector fields in addition to scalar fields. In general, the PMO tended to take a few more lines to code than FClib, and the DOL took many more lines to code than FClib.

# 5 Conclusions

In this report, the PMO and DOL were explored and evaluated for use as replacement data objects for a feature characterization library. Testing consisted of writing sample programs to query and perform simple characterizations on an example finite element data set. The results, and the experience of doing

the testing, were compared to the current FCDMF library (FClib). Both data objects met the minimum requirement of supporting at least the current capabilities of FClib, and have additional desirable capabilities. The data objects were evaluated on two criteria: required additional capabilities and usability.

The DOL has a richer set of data structures that more closely align with the current data structures of FClib and that extend in the direction we are planning to extend FClib. Most importantly, the DOL can support our requirement of time changing geometry. The DOL's interface is also more geared towards supporting post-processing data access. Unfortunately, the DOL's user interface got low marks in usability. The user has to be aware of both the C++ wrappings and the C underpinnings and their overlapping nomenclature. I would also require more convenience functions to do what I want (i.e. many of the starred entries in the query results). For example, I would like to have a function like 'getCoordField(MESH_BLOCK)' instead of having to walk three different C structs and then use a macro to walk a list. Our users should never have to touch *_REF or *_LIST structs, or traverse them with macros. That is too hard.

The PMO was easier to learn and use than the DOL, although it could be improved by making it easier to use for post-processing. Unfortunately, the PMO does not have the flexibility needed for our required data types; the current version has no ready path for implementing the requirements for time-changing geometry.

The PMO cannot meet our capabilities requirement for time varying geometry, and the DOL does not meet our easy use requirement for our level one users. Therefore, our current plan is to first complete FClib's current interface by adding time-changing geometry to the FClib, and then to reimplement the FClib API on top of the DOL while concurrently replacing the current data structures with the DOL. This will provide a simple interface (FClib) for characterization users, but the DOL's interface will also be available for more sophisticated users and characterization developers.

## References

[1] TeraScale, "The Parallel Mesh Object Annotated Reference Manual: Version 1.0," LLC Report TSC02-01, June 24, 2002.

[2] James R. Holten, III, Sandia National Laboratories, NM.

# Appendix A: FClib Makefile

```
# test of fcdmf data object
SRC = test_fcdmf.c
OBJ = ${SRC:.c=.o}
EXE = ${SRC:.c=}

# the data object (& characterizations)
FC_HOME         = /users/wkoegle/Work/fcdmf/fclib
FC_INCLUDE_PATH = -I${FC_HOME}
FC_LIB_PATH     = -L${FC_HOME}
FC_LIBS         = -lfc

# other needed libraries: SAF, HDF5
SUPPORT_HOME         = /users/wkoegle/util
SUPPORT_INCLUDE_PATH = -I${SUPPORT_HOME}/include
SUPPORT_LIB_PATH     = -L${SUPPORT}/lib
SUPPORT_LIBS         = -lsafapi -lvbt -ldsl -lhdf5 -lz -lm

# flags
CC       = gcc
CFLAGS   = -g -Wall
INCLUDES = -I. ${SUPPORT_INCLUDE_PATH} ${FC_INCLUDE_PATH}
LIBS     = ${SUPPORT_LIB_PATH} ${FC_LIB_PATH} ${FC_LIBS} ${SUPPORT_LIBS}

${EXE} : ${OBJ}
        ${CC} -o $@ ${OBJ} ${LIBS}

regtest : ${EXE}
        rm -f test.out
        ./${EXE} > test.out
        diff test.out test.out.worked

%.o : %.c
        ${CC} -c ${CFLAGS} ${INCLUDES} $<


clean :
        rm -f *.o *~ core *#
        rm -f ${EXE} test.out
```

# Appendix B: FCLib Test Program

```
/*
 * Program for quick testing of FCDMF data object
 *
 * MODIFICATIONS:
 *   JAN-20-2002  W Koegler  Created
 */

//-------------------------------------------------------------------
// Header Files
//-------------------------------------------------------------------

//---System Headers
//#include <stdio.h>
//#include <stdlib.h>
//#include <string.h>

//---FCDMF Headers
#include <fc.h>

//-------------------------------------------------------------------
// Global Variables
//-------------------------------------------------------------------
#define MAX_STR_LEN 1000
#define TEST_FILE "../data/can_fcdmf.saf"
#define CAN_ID  0 // the can will be the first mesh

// Special exit function for debugging
void Exit(int code, char *message) {
  printf("Error: %s\n", message);
  exit(code);
}

//-------------------------------------------------------------------
// Characterizations defined in fc.h
//-------------------------------------------------------------------
//int fc_getBoundingbox(fc_meshhandle mid, struct Vector *lowers,
//                      struct Vector *uppers);
//int fc_getMinMax(fc_variablehandle vid, float* min, int* minId,
//                 float* max, int* maxId);

//-------------------------------------------------------------------
// Main
//-------------------------------------------------------------------
int main(int argc, char **argv) {
  //-----------------------------------------------------------------
  // Variable declarations
  //-----------------------------------------------------------------
  int i;
  int errorCode;       // error return code from function calls
  // in order of use ...
  char datasetName[MAX_STR_LEN] = TEST_FILE;
  fc_datasethandle dataset;
  char* temp_name;
  int numMesh;
  char** meshNames;
  fc_meshhandle *meshes;
  int dim;
  int numElem, numVert;
  int numVertPerElem, numEdgePerElem, numFacePerElem;
  fc_elementType elemType;
  struct Vector lowers, uppers;
  int numSeq;
  char** seqNames;
  fc_sequencehandle *seqs;
  int numStep;
  fc_dataType seq_datatype;
  float *times;
  int canIndex;
  int numSeqVar;
  char** seqVarNames;
  int numVar;
  char** varNames;
  fc_variablehandle *vars;
  int num_temp;
  fc_dataType datatype;
```

```
fc_associationType assoc;
int* dim_length;

//-----------------------------------------------------------------------
// Open Dataset
//-----------------------------------------------------------------------

// Initialize library & open dataset
errorCode = fc_open_library();
if (errorCode != RET_OK)
  Exit(-1, "fc_init failed");
errorCode = fc_open_dataset(datasetName, &dataset);
if (errorCode != RET_OK)
  Exit(-1, "failed to open dataset");

//-----------------------------------------------------------------------
// Basic dataset query & report
//-----------------------------------------------------------------------

temp_name = NULL;
errorCode = fc_get_dataset_name(dataset, &temp_name);
if (errorCode != RET_OK)
  Exit(-1, "failed to get dataset name");
errorCode = fc_get_mesh_names(dataset, &numMesh, &meshNames);
if (errorCode != RET_OK)
  Exit(-1, "failed to get mesh names");

printf("\n");
printf("Dataset Summary: '%s'\n", temp_name);
printf("  Spatial Dimension = (query meshes for this info)\n");
printf("  No. Meshes        = %d\n", numMesh);
printf("\n");

free(temp_name);
temp_name = NULL;

//-----------------------------------------------------------------------
// Basic mesh query & report
//-----------------------------------------------------------------------

meshes = malloc(sizeof(fc_meshhandle)*numMesh);

for (i = 0; i < numMesh; i++) {
  // Open the meshes in the dataset
  errorCode = fc_open_mesh(dataset, meshNames[i], &meshes[i]);
  if (errorCode != RET_OK)
    Exit(-1, "failed to open mesh");
  errorCode = fc_describe_mesh(meshes[i], &numElem, NULL,
                               &elemType, NULL, NULL, &numVert, &dim);
  fc_vertices_per_element(elemType, &numVertPerElem);
  fc_edges_per_element(elemType, &numEdgePerElem);
  fc_faces_per_element(elemType, &numFacePerElem);

  printf("Mesh Data (%d): '%s'\n", i, meshNames[i]);
  printf("  Spatial Dimension        = %d\n", dim);
  printf("  No. Vertices             = %d\n", numVert);
  printf("  No. Elements             = %d\n", numElem);
  printf("  Element Type             = %s\n", fc_getElementTypeText(elemType));
  printf("  No. Vertices per Element = %d\n", numVertPerElem);
  printf("  No. Edges per Element    = %d\n", numEdgePerElem);
  printf("  No. Faces Per Element    = %d\n", numFacePerElem);
  printf("\n");
}

//-----------------------------------------------------------------------
// Determine bounding boxes
//-----------------------------------------------------------------------

// Compute bounding boxes for each mesh
printf("Bounding Boxes:\n");
for (i = 0; i < numMesh; i++) {
  printf("  Mesh '%s':\n", meshNames[i]);

  // determine & print bounding box
  fc_getBoundingbox(meshes[i], &lowers, &uppers);
  printf("    [ %g, %g, %g ]  -  [ %g, %g, %g ]\n",
         lowers.coords[0], lowers.coords[1], lowers.coords[2],
         uppers.coords[0], uppers.coords[1], uppers.coords[2]);
}

//-----------------------------------------------------------------------
```

```
// Query Sequences on Dataset & report
//--------------------------------------------------------------------

errorCode = fc_get_sequence_names(dataset, &numSeq, &seqNames);
if (errorCode != RET_OK)
  Exit(-1, "failed to get sequence names");

printf("\nSequence Summary on Dataset:\n");
printf("  No. Seqences = %d\n", numSeq);

seqs = malloc(sizeof(fc_sequencehandle)*numSeq);
for (i = 0; i < numSeq; i++) {
  printf("\nSequence (%d): `%s'\n", i, seqNames[i]);
  errorCode = fc_open_sequence(dataset, seqNames[i], &seqs[i]);
  errorCode = fc_describe_sequence(seqs[i], &numStep, &seq_datatype);
  errorCode = fc_get_sequence_coords(seqs[i], (void**)&times); // !test type!
  printf("  No. Steps = %d\n", numStep);
  printf("  Data Type = %s\n", fc_getDataTypeText(seq_datatype));
}

//--------------------------------------------------------------------
// Query Fields on can & report
//--------------------------------------------------------------------

canIndex = CAN_ID; // known a priori
errorCode = fc_get_variable_names(meshes[canIndex], &numVar, &varNames);
if (errorCode != RET_OK)
  Exit(-1, "failed to get variable names");
errorCode = fc_get_seq_variable_names(meshes[canIndex], &numSeqVar,
                                      &seqVarNames);

printf("\nField Summary on Can Mesh:\n");
if (numSeqVar > 0) {
  printf("  No. Seq Variables = %d (", numSeqVar);
  for (i = 0; i < numSeqVar -1; i++)
    printf("%s, ", seqVarNames[i]);
  printf("%s)\n", seqVarNames[numSeqVar-1]);
}
else
  printf ("  No. Seq Variables = 0\n");
if (numVar > 0) {
  printf("  Total No. Fields = %d (", numVar);
  for (i = 0; i < numVar -1; i++)
    printf("%s, ", varNames[i]);
  printf("%s)\n", varNames[numVar-1]);
}
else
  printf("  Total No. Fields = 0\n");

//--------------------------------------------------------------------
// Describe 1st time step of EQPS field
//--------------------------------------------------------------------

// open all of the EQPS fields on can mesh
errorCode = fc_open_seq_variable(meshes[canIndex],"EQPS_0", &numStep, &vars);
if (errorCode != RET_OK)
  Exit(-1, "failed to open sequence variable");

// describe 1st field
errorCode = fc_get_variable_name(vars[0], &temp_name);
errorCode = fc_describe_variable(vars[0], &datatype, &assoc, &num_temp,
                                 &dim, &dim_length);
printf("\n");
printf("Field Summary: '%s'\n", temp_name);
printf("  Data type             = %s\n", fc_getDataTypeText(datatype));
printf("  Association           = %s\n", fc_getAssociationTypeText(assoc));
printf("  Data Count            = %d\n", num_temp);
printf("  Data dimensionality   = %d\n", dim);
for (i = 0; i < dim; i++)
  printf("  Dimension length[%d] = %d\n", i, dim_length[i]);
printf("\n");

free(temp_name);
temp_name = NULL;

//--------------------------------------------------------------------
// Determine Basic Statistics of EQPS field for each time
//--------------------------------------------------------------------

printf("Basic Statistics for the field 'EQPS'\n");
printf("-------------------------------------------\n");
```

```
  printf("%12s %8s %8s %8s %8s\n",
         "StepValue", "Min", "Max", "Min ID", "Max ID");
  printf("-------------------------------------------\n");
  for (i = 0; i < numStep; i++) {
    float min, max;
    int minId, maxId;
    //errorCode = fc_get_variable_name(vars[i], &temp_name);
    errorCode = fc_getMinMax(vars[i], &min, &minId, &max, &maxId);
    printf("%12.6e %8.3g %8.3g %8d %8d\n", times[i], min, max, minId, maxId);
    free(temp_name);
    temp_name = NULL;
  }

  //---------------------------------------------------------------------
  // Cleanup
  //---------------------------------------------------------------------

  // Close variables that were opened
  for (i = 0; i < numStep; i++)
    fc_close_variable(vars[i]);

  // Close meshes in dataset
  for (i = 0; i < numMesh; i++)
    fc_close_mesh(meshes[i]);

  // Close Dataset & library
  fc_close_dataset(dataset);
  fc_close_library();

  // free allocated space
  fc_free_variable_names(varNames);
  free(meshes);
  fc_free_mesh_names(meshNames);

  // All done
  exit(0);
}
```

# Appendix C: FClib Test Output

```
Dataset Summary: '../data/can_fcdmf.saf'
  Spatial Dimension = (query meshes for this info)
  No. Meshes        = 2

Mesh Data (0): 'Block ID 1 - HEX'
  Spatial Dimension        = 3
  No. Vertices             = 6724
  No. Elements             = 4800
  Element Type             = cubes
  No. Vertices per Element = 8
  No. Edges per Element    = 12
  No. Faces Per Element    = 6

Mesh Data (1): 'Block ID 2 - HEX'
  Spatial Dimension        = 3
  No. Vertices             = 3364
  No. Elements             = 2352
  Element Type             = cubes
  No. Vertices per Element = 8
  No. Edges per Element    = 12
  No. Faces Per Element    = 6

Bounding Boxes:
  Mesh 'Block ID 1 - HEX':
    [ -5.2, 0, -15 ]  -  [ 5.2, 5.2, 0 ]
  Mesh 'Block ID 2 - HEX':
    [ -7.87846, 0, -0.462285 ]  -  [ 8.31258, 8, 4.7781 ]

Sequence Summary on Dataset:
  No. Seqences = 1

Sequence (0): `Time_Suite_0'
  No. Steps = 44
  Data Type = float

Field Summary on Can Mesh:
  No. Seq Variables = 4 (DISPLVEC_0, VELVEC_0, ACCLVEC_0, EQPS_0)
  Total No. Fields = 177 (coord field, DISPLVEC_0, DISPLVEC_1, DISPLVEC_2, DISPLVEC_3,
DISPLVEC_4, DISPLVEC_5, DISPLVEC_6, DISPLVEC_7, DISPLVEC_8, DISPLVEC_9, DISPLVEC_10, DISPLVEC_11,
DISPLVEC_12, DISPLVEC_13, DISPLVEC_14, DISPLVEC_15, DISPLVEC_16, DISPLVEC_17, DISPLVEC_18,
DISPLVEC_19, DISPLVEC_20, DISPLVEC_21, DISPLVEC_22, DISPLVEC_23, DISPLVEC_24, DISPLVEC_25,
DISPLVEC_26, DISPLVEC_27, DISPLVEC_28, DISPLVEC_29, DISPLVEC_30, DISPLVEC_31, DISPLVEC_32,
DISPLVEC_33, DISPLVEC_34, DISPLVEC_35, DISPLVEC_36, DISPLVEC_37, DISPLVEC_38, DISPLVEC_39,
DISPLVEC_40, DISPLVEC_41, DISPLVEC_42, DISPLVEC_43, VELVEC_0, VELVEC_1, VELVEC_2, VELVEC_3,
VELVEC_4, VELVEC_5, VELVEC_6, VELVEC_7, VELVEC_8, VELVEC_9, VELVEC_10, VELVEC_11, VELVEC_12,
VELVEC_13, VELVEC_14, VELVEC_15, VELVEC_16, VELVEC_17, VELVEC_18, VELVEC_19, VELVEC_20,
VELVEC_21, VELVEC_22, VELVEC_23, VELVEC_24, VELVEC_25, VELVEC_26, VELVEC_27, VELVEC_28,
VELVEC_29, VELVEC_30, VELVEC_31, VELVEC_32, VELVEC_33, VELVEC_34, VELVEC_35, VELVEC_36,
VELVEC_37, VELVEC_38, VELVEC_39, VELVEC_40, VELVEC_41, VELVEC_42, VELVEC_43, ACCLVEC_0,
ACCLVEC_1, ACCLVEC_2, ACCLVEC_3, ACCLVEC_4, ACCLVEC_5, ACCLVEC_6, ACCLVEC_7, ACCLVEC_8,
ACCLVEC_9, ACCLVEC_10, ACCLVEC_11, ACCLVEC_12, ACCLVEC_13, ACCLVEC_14, ACCLVEC_15, ACCLVEC_16,
ACCLVEC_17, ACCLVEC_18, ACCLVEC_19, ACCLVEC_20, ACCLVEC_21, ACCLVEC_22, ACCLVEC_23, ACCLVEC_24,
ACCLVEC_25, ACCLVEC_26, ACCLVEC_27, ACCLVEC_28, ACCLVEC_29, ACCLVEC_30, ACCLVEC_31, ACCLVEC_32,
ACCLVEC_33, ACCLVEC_34, ACCLVEC_35, ACCLVEC_36, ACCLVEC_37, ACCLVEC_38, ACCLVEC_39, ACCLVEC_40,
ACCLVEC_41, ACCLVEC_42, ACCLVEC_43, EQPS_0, EQPS_1, EQPS_2, EQPS_3, EQPS_4, EQPS_5, EQPS_6,
EQPS_7, EQPS_8, EQPS_9, EQPS_10, EQPS_11, EQPS_12, EQPS_13, EQPS_14, EQPS_15, EQPS_16, EQPS_17,
EQPS_18, EQPS_19, EQPS_20, EQPS_21, EQPS_22, EQPS_23, EQPS_24, EQPS_25, EQPS_26, EQPS_27,
EQPS_28, EQPS_29, EQPS_30, EQPS_31, EQPS_32, EQPS_33, EQPS_34, EQPS_35, EQPS_36, EQPS_37,
EQPS_38, EQPS_39, EQPS_40, EQPS_41, EQPS_42, EQPS_43)

Field Summary: 'EQPS_0'
  Data type            = float
  Association          = elements
  Data Count           = 4800
  Data dimensionality  = 0

Basic Statistics for the field 'EQPS'
-----------------------------------------------
   StepValue      Min      Max   Min ID   Max ID
-----------------------------------------------
0.000000e+00        0        0        0        0
1.000740e-04        0    0.865       15       80
1.999050e-04        0     1.22      351      160
2.999640e-04        0     1.26      540      440
4.000870e-04        0     1.32      540      600
```

| | | | | |
|---|---|---|---|---|
| 4.999190e-04 | 0 | 1.36 | 540 | 680 |
| 5.999350e-04 | 0 | 1.43 | 540 | 720 |
| 7.000490e-04 | 0 | 1.56 | 541 | 760 |
| 8.000350e-04 | 0 | 1.64 | 541 | 760 |
| 9.000610e-04 | 0 | 1.66 | 541 | 760 |
| 1.000010e-03 | 0 | 1.75 | 542 | 640 |
| 1.099980e-03 | 0 | 1.76 | 542 | 640 |
| 1.199930e-03 | 0 | 1.77 | 544 | 640 |
| 1.299990e-03 | 0 | 1.8 | 544 | 640 |
| 1.400090e-03 | 0 | 1.83 | 545 | 640 |
| 1.500040e-03 | 0 | 1.83 | 546 | 760 |
| 1.599920e-03 | 0 | 1.85 | 546 | 760 |
| 1.700000e-03 | 0 | 1.86 | 632 | 760 |
| 1.800070e-03 | 0 | 1.86 | 674 | 760 |
| 1.900020e-03 | 0 | 1.88 | 715 | 640 |
| 1.999990e-03 | 0 | 1.88 | 718 | 640 |
| 2.099930e-03 | 0 | 1.88 | 756 | 760 |
| 2.199920e-03 | 0 | 1.88 | 2275 | 760 |
| 2.299990e-03 | 0 | 1.89 | 2316 | 640 |
| 2.400050e-03 | 0.000169 | 1.89 | 759 | 640 |
| 2.499940e-03 | 0.000461 | 1.89 | 798 | 640 |
| 2.599920e-03 | 0.000662 | 1.9 | 798 | 640 |
| 2.699980e-03 | 0.000865 | 1.9 | 798 | 640 |
| 2.800000e-03 | 0.00144 | 1.9 | 838 | 640 |
| 2.899910e-03 | 0.00573 | 1.9 | 438 | 640 |
| 3.000020e-03 | 0.0107 | 1.9 | 1357 | 640 |
| 3.099980e-03 | 0.0124 | 1.9 | 1358 | 640 |
| 3.199990e-03 | 0.0129 | 1.9 | 1359 | 640 |
| 3.299940e-03 | 0.0138 | 1.9 | 1359 | 640 |
| 3.400020e-03 | 0.0146 | 1.9 | 1359 | 640 |
| 3.499970e-03 | 0.0189 | 1.9 | 1359 | 640 |
| 3.599990e-03 | 0.0201 | 1.9 | 3119 | 640 |
| 3.699960e-03 | 0.0201 | 2.15 | 3119 | 557 |
| 3.799990e-03 | 0.0202 | 2.56 | 3119 | 398 |
| 3.899980e-03 | 0.0203 | 2.7 | 3119 | 398 |
| 3.999990e-03 | 0.0208 | 2.8 | 3119 | 398 |
| 4.100010e-03 | 0.0209 | 2.84 | 3119 | 396 |
| 4.199970e-03 | 0.0214 | 2.87 | 3119 | 3486 |
| 4.299990e-03 | 0.0216 | 2.89 | 3119 | 3486 |

# Appendix D: PMO Makefile

```
# test of pmo
SRC = test_pmo.cxx
OBJ = ${SRC:.cxx=.o}
EXE = ${SRC:.cxx=}

# the data object
PMO_HOME         = /users/wkoegle/util/src/pmo-1.1.1
PMO_INCLUDE_PATH = -I${PMO_HOME}
PMO_LIB_PATH     = -L${PMO_HOME}/pmo-ref-tsc
PMO_LIBS         = -ltscpmo

# other needed libraries: MPICH & NetCDF
MPICH_HOME           = /usr/local
NETCDF_HOME          = /users/wkoegle/util
SUPPORT_INCLUDE_PATH = -I${MPICH_HOME}/include -I${NETCDF_HOME}/include
SUPPORT_LIB_PATH     = -L${MPICH_HOME}/lib  -L${NETCDF_HOME}/lib
SUPPORT_LIBS         = -lmpich -lpmpich -lsocket  -lnsl -lrt -lnetcdf

# flags
CXX      = g++
CFLAGS   = -g -Wall
INCLUDES = -I. ${SUPPORT_INCLUDE_PATH} ${PMO_INCLUDE_PATH}
LIBS     = ${SUPPORT_LIB_PATH} ${PMO_LIB_PATH} ${PMO_LIBS} ${SUPPORT_LIBS}

${EXE} : ${OBJ}
       ${CXX} -o $@ ${OBJ} ${LIBS}

regtest : ${EXE}
       rm -f test.out
       ./${EXE} > test.out
       diff test.out test.out.worked

%.o : %.cxx
       ${CXX} -c ${CFLAGS} ${INCLUDES} $<


clean :
       rm -f *.o *~ core *#
       rm -f ${EXE} test.out
```

# Appendix E: PMO Test Program

```
/*
 * Program for quick testing of PMO
 *
 * MODIFICATIONS:
 *   JAN-20-2003  W Koegler  Created
 */

//-------------------------------------------------------------------
// Header Files
//-------------------------------------------------------------------

//---System headers
//#include <string>
#include <cstdio>

//---MPI headers
#include <mpi.h>

//--TeraScale reference implementation headers
#include <pmo-ref-tsc/Mesh.h>
#include <pmo-ref-tsc/FieldManager.h>
#include <pmo-ref-tsc/EXOIIReader.h>
//#include <pmo-ref-tsc/EXOIIWriter.h>
//#include <pmo-ref-tsc/NodeList.h>


//-------------------------------------------------------------------
// Global Variables
//-------------------------------------------------------------------
#define TEST_FILE "../data/can.ex2"
#define CAN_ID 0 // the can will be the first mesh block
MPI_Comm comm;

// Special exit function for debugging
void Exit(int value, char* message) {
  // Shut down MPI if it was initialized
  if  (comm != -1)
    MPI_Finalize();
  printf("Error: %s\n", message);
  std::exit(value);
}

//-------------------------------------------------------------------
// Implemented Characterizations
//-------------------------------------------------------------------
int pmo_getBoundingBox(pmo::Mesh* mesh, std::string blockName,
                       double* &lowers, double* &uppers);
int pmo_getMinMax(pmo::Mesh* mesh, std::string issueName,
                  std::string scopeName, std::string extentName,
                  std::string fieldName,
                  double &min, int &minId, double &max, int &maxId);

//-------------------------------------------------------------------
// Main
//-------------------------------------------------------------------
int main( int argc, char *argv[]) {

  int errorCode;    // error return code from function calls
  std::string datasetName = TEST_FILE;

  //-----------------------------------------------------------------
  // Open Dataset
  //-----------------------------------------------------------------

  // MPI initialization. Required for exodus II reader
  comm = MPI_COMM_WORLD;
  if (MPI_Init(&argc, &argv) != MPI_SUCCESS)
    Exit(-1, "MPI_Init failed");

  // Construct a mesh reader for the Exodus II file
  pmo::MeshReader *reader = new tsc::EXOIIReader(datasetName,&comm, errorCode);
  if(errorCode != pmo::SUCCESSFUL) {
    if(errorCode == pmo::NOTFOUND)
      Exit(-1, "new mesh reader failed; file not found");
    if(errorCode == pmo::GARBLEDFILE)
```

```
    Exit(-1, "new mesh reader failed; unable to open file");
}

// Construct a mesh object
pmo::Mesh *mesh = new tsc::Mesh();
if (!mesh)
  Exit(-1, "could not create a new mesh");
mesh->putCommunicator(&comm);
errorCode = mesh->putMeshName(datasetName);
errorCode = mesh->putMeshReader(reader);
if (errorCode != pmo::SUCCESSFUL)
  Exit(-1, "putMeshReader failed");

// Construct a field manager
int numState = 1; // we aren't doing a simulation so we don't need more
pmo::FieldManager *fieldManager = new tsc::FieldManager(numState);
if (!fieldManager)
  Exit(-1, "could not create a new field manager");
errorCode = fieldManager->putMeshReader(reader);
errorCode = mesh->putFieldManager(fieldManager);
if (errorCode != pmo::SUCCESSFUL)
  Exit(-1, "putFieldManager failed");

//-----------------------------------------------------------------------
// Basic mesh query & report
//-----------------------------------------------------------------------

int dim = mesh->getMeshDimension();
int numBlock = mesh->getNumElemBlocks();
int numNode = mesh->getNumLocalNodes();

printf("\n");
printf("Mesh Summary: '%s'\n", mesh->getMeshName().c_str());
printf("  Spatial Dimension = %d\n", dim);
printf("  No. Elem Blocks   = %d\n", numBlock);
printf("  No. Node Sets     = %d\n", mesh->getNumNodeSets());
printf("  No. Edge Sets     = %d\n", mesh->getNumEdgeSets());
printf("  No. Face Sets     = %d\n", mesh->getNumFaceSets());
printf("  No. Elem Sets     = %d\n", mesh->getNumElemSets());
printf("\n");

//-----------------------------------------------------------------------
// Basic Block queries & report
//-----------------------------------------------------------------------

std::string *blockNames = new std::string[numBlock];
//int *blockNumElems = new int[numBlock];
//int *blockNumNodes = new int[numBlock];
//int **blockNodeIds = new int*[numBlock];
errorCode = mesh->getElemBlockNames(numBlock, blockNames);
if (errorCode != pmo::SUCCESSFUL)
  Exit(-1, "getElemBlockNames failed");

// For each block
for (int i = 0; i < numBlock; i++) {
  // the element meta block data is gotten via the topology object:
  pmo::ElemTopology *et = mesh->getElemBlockTopology(blockNames[i]);
  if (et == NULL)
    Exit(-1, "ElemTopogy object is no good!");

  // number of elements
  int blockNumElem = mesh->getElemBlockNumLocalElems(blockNames[i]);

  // getting number of nodes (& their IDs--see bounding box) is a pain!
  //pmo::NodeList *nl;
  //errorCode = mesh->getNodeList(blockNames[i], nl); not implemented!
  int *conn, connLock;
  int temp_nodeIds[numNode];
  for (int j = 0; j < numNode; j++)
    temp_nodeIds[j] = 0;
  errorCode = mesh->getElemNodeConnect(blockNames[i], 0, blockNumElem - 1,
                                       1, pmo::RO, connLock, conn);
  if (errorCode != pmo::SUCCESSFUL)
    Exit(-1, "getElemNodeConnect failed");
  for (int j = 0; j < blockNumElem*et->getNumNodes(); j++)
    temp_nodeIds[conn[j]] = 1;
  int blockNumNode = 0;
  for (int j = 0; j < numNode; j++)
    if (temp_nodeIds[j])
      blockNumNode++;
```

```
  // print out the element block data
  printf("Element Block Data (%d): '%s'\n", i, blockNames[i].c_str());
  printf("  No. Nodes             = %d\n", blockNumNode);
  printf("  No. Elements          = %d\n", blockNumElem);
  printf("  Element Type          = %s\n", et->getName().c_str());
  printf("  No. Nodes per Element = %d\n", et->getNumNodes());
  printf("  No. Edges per Element = %d\n", et->getNumEdges());
  printf("  No. Faces per Element = %d\n", et->getNumFaces());
  printf("\n");
}

//-----------------------------------------------------------------------
// Determine bounding boxes
//-----------------------------------------------------------------------

printf("Bounding Boxes:\n");;

for (int i = 0; i < numBlock; i++) {
  printf("  Block: %s\n", blockNames[i].c_str());

  // determine bounding box
  double* lowers;
  double* uppers;
  errorCode = pmo_getBoundingBox(mesh, blockNames[i], lowers, uppers);
  if (errorCode != pmo::SUCCESSFUL)
    Exit(-1, "bounding box characterization failed");

  // print bounding box
  printf("    [ %g, %g, %g ]  -  [ %g, %g, %g ]\n",
         lowers[0], lowers[1], lowers[2],
         uppers[0], uppers[1], uppers[2]);

  delete [] lowers;
  delete [] uppers;
}

//-----------------------------------------------------------------------
// Query Time & Fields on Mesh and report
//-----------------------------------------------------------------------

int numIssue = fieldManager->readNumIssues();
std::string *issues = new std::string[numIssue];
errorCode = fieldManager->readIssueNames(numIssue, issues);
int numScope = 3;
std::string scopes[numScope];// = new std::string[numScope];
// Only these 3 scopes are supported by the exodus reader
scopes[0] = "GLOBAL";
scopes[1] = "NODAL";
scopes[2] = "ELEMBLOCK";
// NOTE! all of these values could vary issue to issue (but here assumed not)

printf("\nField Summary on Mesh:\n");
printf("  numIssue = %d\n", numIssue);
for (int i = 0; i < numScope; i++) {
  int numExtent = fieldManager->readNumExtents(issues[0], scopes[i]);
  if (numExtent > 0) {
    std::string extentNames[numExtent];
    errorCode = fieldManager->readExtentNames(issues[0], scopes[i],
                                              numExtent, extentNames);
    for (int j = 0; j < numExtent; j++) {
      int numField = fieldManager->readNumFields(issues[0], scopes[i],
                                                 extentNames[j]);
      if (numField > 0) {
        std::string fieldNames[numField];
        errorCode = fieldManager->readFieldNames(issues[0], scopes[i],
                                    extentNames[j], numField, fieldNames);
        printf("  No. %s Fields on extent '%s'= %d (", scopes[i].c_str(),
               extentNames[j].c_str(), numField);
        for (int k = 0; k < numField - 1; k++)
          printf("%s, ", fieldNames[k].c_str());
        printf("%s)\n", fieldNames[numField - 1].c_str());
      }
      else
        printf("  No. %s Fields on extent '%s'= 0\n", scopes[i].c_str(),
               extentNames[j].c_str());
    }
  }
  else
    printf("  No. %s Fields (no extents) = 0\n", scopes[i].c_str());
}
```

```
  //-------------------------------------------------------------------
  // Query Fields on Mesh Blocks
  //-------------------------------------------------------------------

  for (int i = 0; i < numBlock; i++) {
    // extents are the block's name
    printf("\nField Summary on Mesh Block %d: '%s'\n", i, blockNames[i].c_str());
    for (int j = 0; j < numScope; j++)
    printf("  No. %s Fields = %d\n", scopes[j].c_str(),
        fieldManager->readNumFields(issues[0], scopes[j], blockNames[i]));
  }

  //-------------------------------------------------------------------
  // Describe 1st time step of EQPS field
  //-------------------------------------------------------------------

  int canIndex = CAN_ID; // known a priori
  std::string fieldName;
  std::string mathType;
  std::string dataType;
  std::string persist;
  int numValue;
  int cardinality;
  fieldManager->readFieldNames(issues[0], "ELEMBLOCK", blockNames[canIndex],
                               1, &fieldName);
  fieldManager->readFieldRegistration(issues[0], "ELEMBLOCK",
                 blockNames[canIndex], fieldName, mathType, dataType,
                persist, numValue, cardinality);

  printf("\n");
  printf("Field Summary: '%s'\n", fieldName.c_str());
  printf("  Data type      = %s\n", dataType.c_str());
  printf("  Math type      = %s\n", mathType.c_str());
  printf("  Persistence    = %s\n", persist.c_str());
  printf("  No. Values     = %d\n", numValue);
  printf("  Cardinality    = %d\n", cardinality);

  //-------------------------------------------------------------------
  // Determine Basic Statistics of EQPS field for each time
  //-------------------------------------------------------------------

  printf("\n");
  printf("Basic Statistics for the field 'EQPS'\n");
  printf("-------------------------------------------------\n");
  printf("%14s %8s %8s %8s %8s\n",
        "IssueName", "Min", "Max", "Min ID", "Max ID");
  printf("-------------------------------------------------\n");
  for (int i = 0; i < numIssue; i++) {
    double min, max;
    int minId, maxId;
    errorCode = pmo_getMinMax(mesh, issues[i], "ELEMBLOCK",
                              blockNames[canIndex], fieldName,
                              min, minId, max, maxId);
    printf("'%12s' %8.3g %8.3g %8d %8d\n",
          issues[i].c_str(), min, max, minId, maxId);
  }

  //-------------------------------------------------------------------
  // Cleanup
  //-------------------------------------------------------------------
  delete mesh;

  if (comm != -1)
    MPI_Finalize();
}

//-------------------------------------------------------------------
// Bounding Box Characterization
//-------------------------------------------------------------------
int pmo_getBoundingBox(pmo::Mesh* mesh, std::string blockName,
                       double* &lowers, double* &uppers) {

  // Get nodal coords
  int numNode     = mesh->getNumLocalNodes();
  //bool nodalInterleave = mesh->getNodalInterleave();
  double *coords;
  int coordsLock;
  int errorCode = mesh->getNodalCoords(0, numNode-1, true, pmo::RO,
                                       coordsLock, coords);
  if (errorCode != pmo::SUCCESSFUL)
    return errorCode;
```

```
  // mark all nodes in that are in this block
  bool nodeInBlock[numNode];
  for (int i = 0; i < numNode; i++)
    nodeInBlock[i] = false;
  int numBlockElem = mesh->getElemBlockNumLocalElems(blockName);
  int numNodePerElem = mesh->getElemBlockTopology(blockName)->getNumNodes();
  int *conn, connLock;
  errorCode = mesh->getElemNodeConnect(blockName, 0, numBlockElem - 1,
                                       1, pmo::RO, connLock, conn);
  if (errorCode != pmo::SUCCESSFUL)
    return -1;
  for (int i = 0; i < numBlockElem * numNodePerElem; i++)
    nodeInBlock[conn[i]] = true;

  // Make space for returned values ! These will need to be deleted!
  int dim = mesh->getMeshDimension();
  lowers = new double[dim];
  uppers = new double[dim];

  // set uppers & lowers to first node
  int first;
  for (first = 0; first < numNode; first++) {
    // skip if this node isn't in the block
    if (!nodeInBlock[first])
      continue;
    else {
      for (int j = 0; j < dim; j++)
        lowers[j] = uppers[j] = coords[first*dim + j];
      break;
    }
  }

  // expand uppers & lowers
  for (int j = first+1; j < numNode; j++) {
    // skip if this node isn't in the block
    if (!nodeInBlock[j])
      continue;
    for (int k = 0; k < dim; k++) {
      // FIX index depends on type of nodal interleave
      double temp_double = coords[j*dim + k];
      if (temp_double < lowers[k])
        lowers[k] = temp_double;
      else if (temp_double > uppers[k])
        uppers[k] = temp_double;
    }
  }

  errorCode = mesh->releaseNodalCoords(coordsLock);
  return errorCode;
}

//-----------------------------------------------------------------------
// Field Statistics Characterization
//-----------------------------------------------------------------------
int pmo_getMinMax(pmo::Mesh* mesh, std::string issueName,
                  std::string scopeName,
                  std::string extentName, std::string fieldName,
                  double &min, int &minId, double &max, int &maxId) {

  // assumes reading from permanent storage

  // get the field manager
  pmo::FieldManager* fieldManager = mesh->getFieldManager();
  if (!fieldManager)
    return -1;

  std::string mathType;
  std::string dataType;
  std::string persist;  // don't care
  int numValue;
  int cardinality;

  // read field meta data
  fieldManager->readFieldRegistration(issueName, scopeName, extentName,
                                      fieldName, mathType, dataType, persist,
                                      numValue, cardinality);
  // FIX test math type
  // FIX? test cardinality

  // get the field and do the calculations
```

```
  int stateIndex = 0; // FIX? what should this be?
  int num = fieldManager->readFieldSize(issueName, scopeName, extentName,
                                        fieldName);

  if (dataType == "INT") {
    // get field
    int* field = new int[num];
    fieldManager->readIntField(issueName, scopeName, extentName, fieldName,
                               stateIndex, num, field);

    // do it
    min = max = field[0];
    minId = 0; maxId = 0;
    for (int i = 1; i < numValue; i++) {
      if (field[i] < min) {
        min = field[i];
        minId = i;
      }
      else if (field[i] > max) {
        max = field[i];
        maxId = i;
      }
    }

    // cleanup
    delete[] field;
  }
  else if (dataType == "REAL") {
    // get field
    double* field = new double[num];
    fieldManager->readRealField(issueName, scopeName, extentName, fieldName,
                                stateIndex, num, field);

    // do it
    min = max = field[0];
    minId = 0; maxId = 0;
    for (int i = 1; i < numValue; i++) {
      if (field[i] < min) {
        min = field[i];
        minId = i;
      }
      else if (field[i] > max) {
        max = field[i];
        maxId = i;
      }
    }

    // cleanup
    delete[] field;
  }

  return 0;
}
```

# Appendix F: PMO Test Output

```
WARNING: EXOIIReader::getGlobalAttributeDouble(...), did not find a
         length for "nemesis_file_version" or "nemesis file version"
         global attribute. It must not be defined in the file.
         We will use a default value of: 0
         nc_strerror: Attribute not found
WARNING: EXOIIReader::getGlobalAttributeDouble(...), did not find a
         length for "nemesis_api_version" or "nemesis api version"
         global attribute. It must not be defined in the file.
         We will use a default value of: 0
         nc_strerror: Attribute not found
WARNING: EXOIIReader::EXOIIReader(...), could not find the "num_processors" dimension.
         Setting a default value of one.
         nc_strerror: Invalid dimension id or name
WARNING: EXOIIReader::EXOIIReader(...), could not find the "num_procs_file" dimension
         Setting a default value of one.
         nc_error: Invalid dimension id or name
WARNING: EXOIIReader::EXOIIReader(...), could not find the "num_nodes_global" dimension.
         Setting a default value of number local nodes.
         nc_error: Invalid dimension id or name
WARNING: EXOIIReader::EXOIIReader(...), could not retrieve "num_n_cmaps" variable
         nc_error: Invalid dimension id or name
WARNING: EXOIIReader::EXOIIReader(...), could not find the "num_elems_global" dimension.
         Setting a default value of number local elements.
         nc_error: Invalid dimension id or name
WARNING: EXOIIReader::EXOIIReader(...), could not find the "num_el_blk_global" dimension.
         Setting a default value of number local element blocks.
         nc_error: Invalid dimension id or name
WARNING: EXOIIReader::EXOIIReader(...), did not find the "el_blk_cnt_global" variable id
         Setting the default fill to -1
         nc_error: Variable not found
WARNING: EXOIIReader::EXOIIReader(...), did not find the "el_blk_ids_global" variable id
         Setting the default fill to -1
         nc_error: Variable not found
WARNING: EXOIIReader::EXOIIReader(...), could not retrieve the "num_ns_global" dimension
         Setting the default value to number local node sets.
         nc_error: Invalid dimension id or name
WARNING: EXOIIReader::EXOIIReader(...), did not find the "ns_node_cnt_global" variable id
         Setting the default value to -1 for all node sets
         nc_error: Variable not found
WARNING: EXOIIReader::EXOIIReader(...), did not find the "ns_ids_global" variable id
         Setting the default fill to -1 for all node sets
         nc_error: Variable not found
WARNING: EXOIIReader::EXOIIReader(...), did not find the "ns_df_cnt_global" variable id
         nc_error: Variable not found
         Setting the default fill to -1
WARNING EXOIIReader::EXOIIReader(...), could not retrieve the "num_ss_global" dimension.
        Setting the default value to 0
        nc_error: Invalid dimension id or name
>>>>                Mesh Meta Data
>>>> ----------------------------------------------------------------
>>>> Model Independent Conventions:
>>>>   len_string     = 33
>>>>   len_line       = 81
>>>>   qa_size        = 4
>>>>   num_issues     = 44
>>>> Generic Data:
>>>>   num_processors = 1
>>>>   num_qa_rec     = 6
>>>>   num_info       = 4
>>>>   num_procs_file = 1
>>>> Annotation:
>>>>     "ElemBlock:1" = "1"
>>>>     "ElemBlock:2" = "2"
>>>>     "INFO1" = "01/ Gen3D: can_half.g2d"
>>>>     "INFO2" = "01/ can_half.g3d"
>>>>     "INFO3" = "02/ Gen3D: block_half.g2d"
>>>>     "INFO4" = "02/ block_half.g3d"
>>>>     "INFO:NUM_INFO" = "4"
>>>>     "NodeSet:1" = "1"
>>>>     "NodeSet:2" = "100"
>>>>     "QA1:CODE" = "FASTQ"
>>>>     "QA1:DATE" = "05/16/94"
>>>>     "QA1:TIME" = "14:09:31"
>>>>     "QA1:VERSION" = " 2.3"
>>>>     "QA2:CODE" = "Gen3D"
```

```
>>>>     "QA2:DATE" = "05/16/94"
>>>>     "QA2:TIME" = "14:09:41"
>>>>     "QA2:VERSION" = "   1.7"
>>>>     "QA3:CODE" = "FASTQ"
>>>>     "QA3:DATE" = "05/16/94"
>>>>     "QA3:TIME" = "14:09:56"
>>>>     "QA3:VERSION" = " 2.3"
>>>>     "QA4:CODE" = "Gen3D"
>>>>     "QA4:DATE" = "05/16/94"
>>>>     "QA4:TIME" = "14:10:01"
>>>>     "QA4:VERSION" = "   1.7"
>>>>     "QA5:CODE" = "GJoin"
>>>>     "QA5:DATE" = "05/16/94"
>>>>     "QA5:TIME" = "14:10:03"
>>>>     "QA5:VERSION" = "  1.12"
>>>>     "QA6:CODE" = "PRONTO3D"
>>>>     "QA6:DATE" = "05/16/94"
>>>>     "QA6:TIME" = "14:10:17"
>>>>     "QA6:VERSION" = "7.0.1"
>>>>     "QA:NUM_QA" = "6"
>>>>     "SideSet:1" = "4"
>>>>     "exoii:api_version" = "2.010000e+00"
>>>>     "exoii:floating_point_word_size" = "4.000000e+00"
>>>>     "exoii:nemesis_api_version" = "0.000000e+00"
>>>>     "exoii:nemesis_file_version" = "0.000000e+00"
>>>>     "exoii:title" = " self contact test problem"
>>>>     "exoii:version" = "2.010000e+00"
>>>> Nodal Data:
>>>>   dim              = 3
>>>>   num_nodes        = 10088
>>>>   num_nodes_global = 10088
>>>> SHARED NODE LISTS:
   There are no shared nodes in this mesh.
>>>> Element Data:
>>>>   num_elem          = 7152
>>>>   num_elems_global  = 7152
>>>>   num_el_blk        = 2
>>>>   num_el_blk_global = 2
>>>>   Elements Block Local:
>>>>     (block index: status, number elems, number nodes, block ID, type)
>>>>     0: 1, 4800, 8, 1, "HEX"
>>>>     1: 1, 2352, 8, 2, "HEX"
>>>>   Element Blocks Global:
>>>>     (block index: number elems, block ID)
>>>>     block 0: 0, -1
>>>>     block 1: 0, -1
>>>> Node Sets:
>>>>   num_node_sets = 2
>>>>   num_ns_global = 2
>>>>   Node Sets Local:
>>>>     (set index: status, set size, setID)
>>>>     0: 1, 444, 1
>>>>     1: 1, 164, 100
>>>>   Node Sets Global:
>>>>    (set index:, set size, setID, distFac size)
>>>>     0: -1, -1, -1
>>>>     1: -1, -1, -1
>>>> Side Sets:
>>>>   num_sidesets  = 1
>>>>   num_ss_global = 0
>>>>   Side Sets Local:
>>>>     (set index: status, set size, setID, distFac size)
>>>>     0: 1, 120, 4, 480
>>>>   Side Sets Global: NONE FOUND
>>>> Results Summary:
>>>>   Number of Global Variables  = 6
>>>>   Number of Nodal Variables   = 9
>>>>   Number of Element Variables = 1
>>>>   Global Variables:
>>>>         "KE"    "NSTEPS"    "TMSTEP"    "XMOM"    "YMOM"    "ZMOM"

>>>>   Nodal Variables:
>>>>         "ACCLX"    "ACCLY"    "ACCLZ"    "DISPLX"    "DISPLY"    "DISPLZ"
>>>>         "VELX"    "VELY"    "VELZ"
>>>>   Element Variables:
>>>>         "EQPS"
>>>>   Block 0: 1,
>>>>   Block 1: 1,
>>>>   Number of Issues = 44
>>>>      Issue Names:
>>>>       "0.000000e+00"
```

```
>>>>       "1.000006e-03"
>>>>       "1.000737e-04"
>>>>       "1.099981e-03"
>>>>       "1.199933e-03"
>>>>       "1.299987e-03"
>>>>       "1.400093e-03"
>>>>       "1.500041e-03"
>>>>       "1.599919e-03"
>>>>       "1.699999e-03"
>>>>       "1.800073e-03"
>>>>       "1.900023e-03"
>>>>       "1.999051e-04"
>>>>       "1.999986e-03"
>>>>       "2.099930e-03"
>>>>       "2.199921e-03"
>>>>       "2.299993e-03"
>>>>       "2.400054e-03"
>>>>       "2.499943e-03"
>>>>       "2.599921e-03"
>>>>       "2.699980e-03"
>>>>       "2.800000e-03"
>>>>       "2.899911e-03"
>>>>       "2.999644e-04"
>>>>       "3.000021e-03"
>>>>       "3.099979e-03"
>>>>       "3.199995e-03"
>>>>       "3.299942e-03"
>>>>       "3.400023e-03"
>>>>       "3.499968e-03"
>>>>       "3.599992e-03"
>>>>       "3.699962e-03"
>>>>       "3.799987e-03"
>>>>       "3.899982e-03"
>>>>       "3.999986e-03"
>>>>       "4.000865e-04"
>>>>       "4.100011e-03"
>>>>       "4.199974e-03"
>>>>       "4.299989e-03"
>>>>       "4.999192e-04"
>>>>       "5.999351e-04"
>>>>       "7.000492e-04"
>>>>       "8.000353e-04"
>>>>       "9.000607e-04"
>>>> ----------------------------------------------------------------

WARNING: EXOIIReader::getNodalGlobalIDs(...), could not find the node_num_map variable id
    nc_error: Variable not found
WARNING: EXOIIReader::getLengthNodalSharedProcs(...), could not retrieve nodal global IDs.
        pmo_error: NODATADEFINED

Mesh Summary: '../data/can.ex2'
  Spatial Dimension = 3
  No. Elem Blocks   = 2
  No. Node Sets     = 2
  No. Edge Sets     = 0
  No. Face Sets     = 1
  No. Elem Sets     = 0

Element Block Data (0): '1'
  No. Nodes             = 6724
  No. Elements          = 4800
  Element Type          = HEX8
  No. Nodes per Element = 8
  No. Edges per Element = 12
  No. Faces per Element = 6

Element Block Data (1): '2'
  No. Nodes             = 3364
  No. Elements          = 2352
  Element Type          = HEX8
  No. Nodes per Element = 8
  No. Edges per Element = 12
  No. Faces per Element = 6

Bounding Boxes:
  Block: 1
    [ -5.2, 0, -15 ]  -  [ 5.2, 5.2, 0 ]
  Block: 2
    [ -7.87846, 0, -0.462285 ]  -  [ 8.31258, 8, 4.7781 ]

Field Summary on Mesh:
```

```
  numIssue = 44
  No. GLOBAL Fields on extent 'UNKNOWN'= 6 (KE, XMOM, YMOM, ZMOM, NSTEPS, TMSTEP)
  No. NODAL Fields on extent 'ALL'= 9 (DISPLX, DISPLY, DISPLZ, VELX, VELY, VELZ, ACCLX, ACCLY,
ACCLZ)
  No. ELEMBLOCK Fields on extent '1'= 1 (EQPS)
  No. ELEMBLOCK Fields on extent '2'= 1 (EQPS)

Field Summary on Mesh Block 0: '1'
  No. GLOBAL Fields = 6
  No. NODAL Fields = 9
  No. ELEMBLOCK Fields = 1

Field Summary on Mesh Block 1: '2'
  No. GLOBAL Fields = 6
  No. NODAL Fields = 9
  No. ELEMBLOCK Fields = 1

Field Summary: 'EQPS'
  Data type      = REAL
  Math type      = SCALAR
  Persistence    = STATE
  No. Values     = 4800
  Cardinality    = 1

Basic Statistics for the field 'EQPS'
```

| IssueName | Min | Max | Min ID | Max ID |
|---|---|---|---|---|
| '0.000000e+00' | 0 | 0 | 0 | 0 |
| '1.000737e-04' | 0 | 0.865 | 15 | 80 |
| '1.999051e-04' | 0 | 1.22 | 351 | 160 |
| '2.999644e-04' | 0 | 1.26 | 540 | 440 |
| '4.000865e-04' | 0 | 1.32 | 540 | 600 |
| '4.999192e-04' | 0 | 1.36 | 540 | 680 |
| '5.999351e-04' | 0 | 1.43 | 540 | 720 |
| '7.000492e-04' | 0 | 1.56 | 541 | 760 |
| '8.000353e-04' | 0 | 1.64 | 541 | 760 |
| '9.000607e-04' | 0 | 1.66 | 541 | 760 |
| '1.000006e-03' | 0 | 1.75 | 542 | 640 |
| '1.099981e-03' | 0 | 1.76 | 542 | 640 |
| '1.199933e-03' | 0 | 1.77 | 544 | 640 |
| '1.299987e-03' | 0 | 1.8 | 544 | 640 |
| '1.400093e-03' | 0 | 1.83 | 545 | 640 |
| '1.500041e-03' | 0 | 1.83 | 546 | 760 |
| '1.599919e-03' | 0 | 1.85 | 546 | 760 |
| '1.699999e-03' | 0 | 1.86 | 632 | 760 |
| '1.800073e-03' | 0 | 1.86 | 674 | 760 |
| '1.900023e-03' | 0 | 1.88 | 715 | 640 |
| '1.999986e-03' | 0 | 1.88 | 718 | 640 |
| '2.099930e-03' | 0 | 1.88 | 756 | 760 |
| '2.199921e-03' | 0 | 1.88 | 2275 | 760 |
| '2.299993e-03' | 0 | 1.89 | 2316 | 640 |
| '2.400054e-03' | 0.000169 | 1.89 | 759 | 640 |
| '2.499943e-03' | 0.00046 | 1.89 | 798 | 640 |
| '2.599921e-03' | 0.000662 | 1.9 | 798 | 640 |
| '2.699980e-03' | 0.000865 | 1.9 | 798 | 640 |
| '2.800000e-03' | 0.00144 | 1.9 | 838 | 640 |
| '2.899911e-03' | 0.00573 | 1.9 | 438 | 640 |
| '3.000021e-03' | 0.0107 | 1.9 | 1357 | 640 |
| '3.099979e-03' | 0.0124 | 1.9 | 1358 | 640 |
| '3.199995e-03' | 0.0129 | 1.9 | 1359 | 640 |
| '3.299942e-03' | 0.0138 | 1.9 | 1359 | 640 |
| '3.400023e-03' | 0.0146 | 1.9 | 1359 | 640 |
| '3.499968e-03' | 0.0189 | 1.9 | 1359 | 640 |
| '3.599992e-03' | 0.0201 | 1.9 | 3119 | 640 |
| '3.699962e-03' | 0.0201 | 2.15 | 3119 | 557 |
| '3.799987e-03' | 0.0202 | 2.56 | 3119 | 398 |
| '3.899982e-03' | 0.0203 | 2.7 | 3119 | 398 |
| '3.999986e-03' | 0.0208 | 2.8 | 3119 | 398 |
| '4.100011e-03' | 0.0209 | 2.84 | 3119 | 396 |
| '4.199974e-03' | 0.0214 | 2.87 | 3119 | 3486 |
| '4.299989e-03' | 0.0216 | 2.89 | 3119 | 3486 |

# Appendix G: DOL Makefile

```
# test of dol
SRC = test_dol.cxx
OBJ = ${SRC:.cxx=.o}
OBJ2 = ${SRC:.cxx=2.o}
EXE = ${SRC:.cxx=}
EXE2 = ${SRC:.cxx=2}

# the data object
DOL_HOME         = /users/wkoegle/Work/fcdmf/dol
DOL_INCLUDE_PATH = -I${DOL_HOME}/include
DOL_LIB_PATH     = -L${DOL_HOME}/lib
DOL_LIBS         = -ldolsafio -ldol
DOL_LIBS2        = -ldolexoio -ldol

# other needed libraries: SAF (& HDF5) or Exodus II (& netCDF)
SAF_HOME              = /users/wkoegle/util
EXODUS_HOME           = /net/troi/apps/Access
SUPPORT_INCLUDE_PATH = -I${SAF_HOME}/include -I${EXODUS_HOME}/inc
SUPPORT_LIB_PATH     = -L${SAF_HOME}/lib -L${EXODUS_HOME}/lib
SUPPORT_LIBS          = -lsafapi -lvbt -ldsl \
                        -lhdf5 -lz -lm
SUPPORT_LIBS2         = -lnemIc -lexoIIv2c -lnetcdf -lm

# flags
CXX      = g++
CFLAGS   = -g -Wall
INCLUDES = -I. ${SUPPORT_INCLUDE_PATH} ${SUPPORT_INCLUDE_PATH2} \
           ${DOL_INCLUDE_PATH}
LIBS     = ${SUPPORT_LIB_PATH} ${DOL_LIB_PATH} ${DOL_LIBS} ${SUPPORT_LIBS}
LIBS2     = ${SUPPORT_LIB_PATH} ${DOL_LIB_PATH} ${DOL_LIBS2} ${SUPPORT_LIBS2}

${EXE} : ${OBJ}
       ${CXX} -o $@ ${OBJ} ${LIBS}

${EXE2} : ${OBJ2}
       ${CXX} -o $@ ${OBJ2} ${LIBS2}

regtest : ${EXE}
       rm -f test.out
       ./${EXE} > test.out
       diff test.out test.out.worked

regtest2 : ${EXE2}
       rm -f test2.out
       ./${EXE2} > test2.out
       diff test2.out test2.out.worked

${OBJ2} : ${SRC}
       ${CXX} -o ${OBJ2} -c ${CFLAGS} -DEXO_FILE ${INCLUDES} $<

%.o : %.cxx
       ${CXX} -c ${CFLAGS} ${INCLUDES} $<


clean :
       rm -f *.o *~ core *#
       rm -f ${EXE} ${EXE2} test.out test2.out
```

# Appendix H: DOL Test Program

```
/*
 * Program for quick testing of DOL
 *
 * MODIFICATIONS:
 *   JAN-21-2003  W Koegler  Created
 */

//-------------------------------------------------------------------
// Header Files
//-------------------------------------------------------------------

//---System
//#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//---DOL helpers
//#include "sw_util.h"
//#include "sw_err.h"
//#include "sw_values.h"
//---DOL set, field & relations
#include "sw_field.h"
//#include "sw_rel.h"
//#include "sw_rel_p.h"
#include "sw_set.h"
//#include "sw_tuples.h"
//---DOL mesh
//#include "mesh.h"
#include "mesh_block.h"
#include "mesh_cells.h"
//#include "mesh_block_par.h"
//#include "mesh_model.h"
//#include "mesh_p.h"
#include "mesh_par.h"  // need for DOFT_SAF definition (data object file type)
//---DOL DataObject
#include "do.h"
#include "do_mesh.h"
#include "do_model.h"
#include "do_file.h"

//-------------------------------------------------------------------
// Global Variables
//-------------------------------------------------------------------
#define MAX_STR_LEN 1000

#ifdef EXO_FILE

#define TEST_FILE "../data/can.ex2"
#define FILE_TYPE DOFT_EXODUSII
#define CAN_ID 2 // the can will be the third mesh block


#else // default: use the saf file

#define TEST_FILE "../data/can_dol.saf"
#define FILE_TYPE DOFT_SAF
#define CAN_ID 1 // the can will be the second mesh block

#endif

// Special exit function for debugging
void Exit(int code, char *message) {
  printf("Error: %s\n", message);
  exit(code);
}

//-------------------------------------------------------------------
// Implemented Characterizations
//-------------------------------------------------------------------
int dol_getBoundingBox(MeshDataObject* meshObj, MESH_BLOCK* mesh_block,
                       int &dim, double* &lowers, double* &uppers);
int dol_getMinMax(MeshDataObject* meshObj, SW_FIELD* field, int instance_index,
                  double &min, int &minId, double &max, int &maxId);

//-------------------------------------------------------------------
```

```
// Main
//-------------------------------------------------------------------
int main(int argc, char *argv[]) {

 int errorCode;  // error return code from function calls

  //-----------------------------------------------------------------
  // Open Dataset
  //-----------------------------------------------------------------

  // Open the data file object
  DO_FileObject *fileObject;
  char hostname[MAX_STR_LEN] = "melon";
  char fileName[MAX_STR_LEN] = TEST_FILE;
  int fileType = FILE_TYPE;
  fileObject = DO_FileObject::Open(hostname, fileName, fileType, FALSE,
                                   -1, NULL);
  if (fileObject == NULL)
    Exit(-1, "couldn't open file object");

  // Create a mesh model
  MESH_MODEL* mesh_model = MeshModelCreate(fileName);
  if (mesh_model == NULL)
    Exit(-1, "Could not create a mesh model");

  // Read description from file into mesh_model
  errorCode = fileObject->ReadMeshDesc(mesh_model);
  if (errorCode != 0)
    Exit(-1, "Couldn't read description from file into the mesh model");

  // Create a model object
  DataObjectList do_list;
  ModelDataObject* modelObject = new ModelDataObject(&do_list, fileName,
                                                     mesh_model, fileObject);
  if (modelObject == NULL)
    Exit(-1, "Could not create ModelDataObject for file");

  // Get mesh objects
  int numMeshObj = modelObject->GetMeshObjectCount();
  MeshDataObject** meshObjs = modelObject->GetMeshObjects();

  // We only expect one mesh object (top set)
  if (numMeshObj < 0)
    Exit(-1, "did not obtain a mesh object");
  else if (numMeshObj > 1)
    printf("Warning: %d mesh objects, but only the first is examined\n",
           numMeshObj);
  MeshDataObject* meshObj = meshObjs[0];

  //-----------------------------------------------------------------
  // Basic MeshObject query & report
  //-----------------------------------------------------------------

  int numMeshBlock = meshObj->GetMeshBlockCount();
  int numSet = meshObj->GetSetCount();
  SW_SET** sets = meshObj->GetSets();

  printf("\n");
  printf("Mesh Summary: '%s'\n", meshObj->GetName());
  printf("  Spatial Dimension = (query blocks for this info)\n");
  printf("  No. Blocks        = %d\n", numMeshBlock);
  printf("  No. Coord Fields  = %d\n", meshObj->GetCoordFieldCount());
  printf("  No. Sequence Sets = %d\n", meshObj->GetSequenceSetCount());
  printf("  No. Sets          = %d (", numSet);
  for (int i = 0; i < numSet - 1; i++)
    printf("%s, ", sets[i]->name);
  if (numSet > 0)
    printf("%s)\n", sets[numSet-1]->name);
  printf("  No. Relations     = %d\n", meshObj->GetRelCount());
  printf("  No. Fields        = %d\n", meshObj->GetFieldCount());
  if (sets)
    free(sets);

  //-----------------------------------------------------------------
  // Meshblock query & report
  //-----------------------------------------------------------------

  MESH_BLOCK **mesh_blocks = meshObj->GetMeshBlocks();

  // For each block
  for (int i = 0; i < numMeshBlock; i++) {
```

```
  SW_SET* set;
  MESH_CELLS* nodes = mesh_blocks[i]->nodes;
  MESH_CELLS* edges = mesh_blocks[i]->edges;
  MESH_CELLS* faces = mesh_blocks[i]->faces;
  MESH_CELLS* zones = mesh_blocks[i]->zones;

  // get dimensionality from the coord field
  SW_FIELD_REF* field_ref;
  int dim = -1;
  if (nodes) {
    LLSCANTO(*nodes->set->field_ref_list, field_ref,
             field_ref->field->is_coord);
    if (field_ref)
      dim = field_ref->field->value_count;
  }

  printf("\n");
  printf("Block Data (%d): '%s'\n", i, mesh_blocks[i]->name);
  printf("  Spatial Dimension = ");
  if (dim == -1)
    printf("no coordinate field\n");
  else
    printf("%d\n", dim);
  if (nodes) {
    set = nodes->set;
    printf("  No. Nodes         = %d\n", swSetGetMemberCount(set, -1));
  }
  else
    printf("  No. Nodes         = %d\n", 0);
  if (edges) {
    set = edges->set;
    printf("  No. Edges         = %d\n", swSetGetMemberCount(set, -1));
  }
  else
    printf("  No. Edges         = %d\n", 0);
  if (faces) {
    set = faces->set;
    printf("  No. Faces         = %d\n", swSetGetMemberCount(set, -1));
  }
  else
    printf("  No. Faces         = %d\n", 0);
  if (zones) {
    set = zones->set;
    printf("  No. Zones         = %d\n", swSetGetMemberCount(set, -1));
    printf("  Zone type         = %s\n",
                               MESH_CELL_TYPE_TEXT(zones->cells_type));
  }
}

//-----------------------------------------------------------------------
// Determine bounding boxes
//-----------------------------------------------------------------------

printf("\n");
printf("Bounding Boxes:\n");
for (int i = 0; i < numMeshBlock; i++) {
  printf("  Block: %s\n", mesh_blocks[i]->name);
  int dim;
  double* lowers;
  double* uppers;
  errorCode = dol_getBoundingBox(meshObj, mesh_blocks[i], dim,
                                 lowers, uppers);
  if (errorCode != 0) {
    printf("    no coordinate field\n");
    //Exit(-1, "bounding box characterization failed");
    continue;
  }

  // print bounding box
  printf("    [ %g, %g, %g ]  -  [ %g, %g, %g ]\n",
         lowers[0], lowers[1], lowers[2],
         uppers[0], uppers[1], uppers[2]);

  delete [] lowers;
  delete [] uppers;
}

//-----------------------------------------------------------------------
// Query Time & Fields on Mesh and report
//-----------------------------------------------------------------------
```

```
// Get the time Fields - sequences?
int numField = meshObj->GetFieldCount();
SW_FIELD** fields = meshObj->GetFields();
SW_FIELD* time;
int numTimeField = 0;
for (int i = 0; i < numField; i++) {
  if (fields[i]->is_time) {
    time = fields[i];
    numTimeField++;
  }
}

printf("\n");
printf("Field Summary on MeshObject: '%s'\n", meshObj->GetName());
printf("  No. Fields         = %d (", meshObj->GetFieldCount());
for (int i = 0; i < numField - 1; i++)
  printf("%s, ", fields[i]->name);
if (numField > 0)
  printf("%s)\n", fields[numField-1]->name);
printf("  No. Time Fields    = %d\n", numTimeField);
printf("  No. Sequence Sets = %d\n", meshObj->GetSequenceSetCount());
printf("  No. Coord Fields   = %d\n", meshObj->GetCoordFieldCount());
free(fields);

printf("\n");
SW_SET** seqSets = meshObj->GetSequenceSets();
printf("Sequence Set Query: '%s'\n", seqSets[0]->name);
printf("  Type Name             = %s\n", seqSets[0]->type_name);
printf("  Set Type              = %s\n", SW_SET_TYPE_TEXT(seqSets[0]->type));
printf("  No. Members           = %d\n", swSetGetMemberCount(seqSets[0], -1));
if (seqSets)
  free(seqSets);

printf("\n");
printf("Time Field Query: '%s' (& it's set = '%s')\n", time->name,
       time->set->name); // should check to make sure that's not null
printf("  No. Entries           = %d\n", swSetGetMemberCount(time->set, -1));
printf("  No. Instances         = %d\n", swFieldGetInstanceCount(time));
printf("  No. Values per Entry = %d\n", time->value_count);
printf("  Data type             = %s\n", SW_VALUE_TYPE_TEXT(time->value_type));
printf("  Data Organization     = %s\n",
       SW_FIELD_ORG_TYPE_TEXT(time->org_type));

//---------------------------------------------------------------------
// Query Fields on Mesh Blocks
//---------------------------------------------------------------------

SW_FIELD_REF* field_ref;
int numCoordField;
int numSeqField;
char meshCellsNames[4][1024] = { "Nodes", "Edges", "Faces", "Zones" };

for (int i = 0; i < numMeshBlock; i++) {
  printf("\n");
  printf("Field Summary on Mesh Block %d: '%s'\n", i,
         mesh_blocks[i]->name);

  MESH_CELLS* mesh_cells[4];
  mesh_cells[0] = mesh_blocks[i]->nodes;
  mesh_cells[1] = mesh_blocks[i]->edges;
  mesh_cells[2] = mesh_blocks[i]->faces;
  mesh_cells[3] = mesh_blocks[i]->zones;

  for (int j = 0; j < 4; j++) {
    numField = 0;
    numCoordField = 0;
    numTimeField = 0;
    numSeqField = 0;
    if (!mesh_cells[j])
      printf("  Total No. Fields on %s = 0\n", meshCellsNames[j]);
    else {
      LLSCAN(*mesh_cells[j]->set->field_ref_list, field_ref) {
        if (field_ref->field->is_time)
          numTimeField++;
        if (field_ref->field->is_coord)
          numCoordField++;
        if (field_ref->field->sequence_set_id != -1)
          numSeqField++;
        numField++;
      }
      if (numField > 0) {
```

```
        printf("  Total No. Fields on %s = %d", meshCellsNames[j], numField);
        printf(" (");
        LLSCAN(*mesh_cells[j]->set->field_ref_list, field_ref) {
          printf("%s, ", field_ref->field->name);
        }
        printf(")\n");
        printf("    No. Sequence Fields = %d\n", numSeqField);
        printf("    No. Non-sequence Fields = %d\n", numField - numSeqField);
        printf("    No. Time Fields = %d\n", numTimeField);
        printf("    No. Coord Fields = %d\n", numCoordField);
      }
      else
        printf("  Total No. Fields on %s = 0\n", meshCellsNames[j]);
    }
  }
}

//-----------------------------------------------------------------------
// Describe 1st time step of EQPS field
//-----------------------------------------------------------------------

// Get the EQPS field on the can
int canIndex = CAN_ID; // known a priori
LLSCANTO(*mesh_blocks[canIndex]->zones->set->field_ref_list, field_ref,
        !strcmp(field_ref->field->name, "EQPS"));
if (!field_ref)
  Exit(-1, "didn't find EQPS field");
SW_FIELD* field = field_ref->field;

printf("\n");
printf("Field Summary: '%s'\n", field->name);
printf("  No. Entries          = %d\n", swSetGetMemberCount(field->set, -1));
printf("  No. Instances        = %d\n", swFieldGetInstanceCount(field));
printf("  No. Values per Entry = %d\n", field->value_count);
printf("  Data type            = %s\n",
       SW_VALUE_TYPE_TEXT(field->value_type));
printf("  Data Organization    = %s\n",
       SW_FIELD_ORG_TYPE_TEXT(field->org_type));

//-----------------------------------------------------------------------
// Determine Basic Statistics of EQPS field for each time
//-----------------------------------------------------------------------

// get time values
int numTime = swSetGetMemberCount(time->set, -1);
SW_FIELD_INST* time_inst = meshObj->GetFieldInst(time, -1);
double times[numTime];
switch(time->value_type) {
case SW_VT_INT: {
  int* cast_values = (int*)time_inst->values;
  for (int i = 0; i < numTime; i++)
    times[i] = cast_values[i];
} break;
case SW_VT_FLOAT: {
  float* cast_values = (float*)time_inst->values;
  for (int i = 0; i < numTime; i++)
    times[i] = cast_values[i];
} break;
case SW_VT_DOUBLE: {
  double* cast_values = (double*)time_inst->values;
  for (int i = 0; i < numTime; i++)
    times[i] = cast_values[i];
} break;
default:
  Exit(-1, "cannot handle the data type of the time field");
}

printf("\n");
printf("Basic Statistics for the field 'EQPS'\n");
printf("-----------------------------------------------\n");
printf("%12s %8s %8s %8s %8s\n",
       "Step Value", "Min", "Max", "Min ID", "Max ID");
printf("-----------------------------------------------\n");
for (int i = 0; i < numTime; i++) {
  double min, max;
  int minId, maxId;
  errorCode = dol_getMinMax(meshObj, field, i, min, minId, max, maxId);
  printf("%12.6e %8.3g %8.3g %8d %8d\n", times[i], min, max, minId, maxId);
}

//-----------------------------------------------------------------------
```

```
  // Cleanup
  //-------------------------------------------------------------------

}

//-------------------------------------------------------------------
// Bounding Box Characterization
//-------------------------------------------------------------------
int dol_getBoundingBox(MeshDataObject* meshObj, MESH_BLOCK* mesh_block,
                       int &dim, double* &lowers, double* &uppers) {
  MESH_CELLS* nodes;
  SW_SET* nodes_set;
  SW_FIELD_REF* field_ref;
  SW_FIELD* coord_field;

  // test input

  // get the nodes' set
  nodes = mesh_block->nodes;
  if (!nodes)
    return -1;
  nodes_set = nodes->set;

  // get coord field reference
  LLSCANTO(*nodes_set->field_ref_list, field_ref,
           field_ref->field->is_coord);
  if (field_ref)
    coord_field = field_ref->field;
  else {
    return -1;
  }

  // get info
  int numNode = swSetGetMemberCount(nodes_set, -1);
  int numInst = swFieldGetInstanceCount(coord_field);
  //printf("numInst = %d\n", numInst);
  dim = coord_field->value_count;

  // get the data
  int inst_index;
  if (numInst == 1)
    inst_index = -1;
  else
    inst_index = 0;
  SW_FIELD_INST* field_inst = meshObj->GetFieldInst(coord_field, inst_index);
  if (!field_inst) {
    printf("bounding box error couldn't get instance\n");
    return -1;
  }
  if (field_inst->type != FT_SIMPLE) {
    printf("error: bounding box can't handle '%s' fields yet\n",
           SW_FIELD_TYPE_TEXT(field_inst->type));
    return -1;
  }

  // do it
  lowers = new double[dim];
  uppers = new double[dim];
  switch(coord_field->value_type) {
  case SW_VT_INT:
    {
      int* cast_values = (int*)field_inst->values;
      int index;
      for (int j = 0; j < dim; j++) {
        index = 0*dim + j;
        lowers[j] = uppers[j] = cast_values[index];
      }
      for (int i = 0; i < numNode; i++)
        for (int j = 0; j < dim; j++) {
          index = i*dim + j;
          if (cast_values[index] < lowers[j])
            lowers[j] = cast_values[index];
          else if (cast_values[index] > uppers[j])
            uppers[j] = cast_values[index];
        }
    } break;
  case SW_VT_FLOAT:
    {
      float* cast_values = (float*)field_inst->values;
      int index;
      for (int j = 0; j < dim; j++) {
```

```
      index = 0*dim + j;
      lowers[j] = uppers[j] = cast_values[index];
    }
    for (int i = 0; i < numNode; i++)
      for (int j = 0; j < dim; j++) {
        index = i*dim + j;
        if (cast_values[index] < lowers[j])
          lowers[j] = cast_values[index];
        else if (cast_values[index] > uppers[j])
          uppers[j] = cast_values[index];
      }
  } break;
case SW_VT_DOUBLE:
  {
    double* cast_values = (double*)field_inst->values;
    int index;
    for (int j = 0; j < dim; j++) {
      index = 0*dim + j;
      lowers[j] = uppers[j] = cast_values[index];
    }
    for (int i = 0; i < numNode; i++)
      for (int j = 0; j < dim; j++) {
        index = i*dim + j;
        if (cast_values[index] < lowers[j])
          lowers[j] = cast_values[index];
        else if (cast_values[index] > uppers[j])
          uppers[j] = cast_values[index];
      }
  } break;
default:
  printf("bounding box error: don't know the value type\n");
  return -1;
}

return 0;
}

//-----------------------------------------------------------------------
// Field Statistics Characterization
//-----------------------------------------------------------------------
int dol_getMinMax(MeshDataObject* meshObj, SW_FIELD* field, int instance_index,
                  double &min, int &minId, double &max, int &maxId) {
  // test input

  // get the data
  int numNode = swSetGetMemberCount(field->set, -1);
  if (field->value_count != 1) {
    printf("getMinMax error: only works on scalar fields\n");
    return -1;
  }

  // get the data
  SW_FIELD_INST* field_inst = meshObj->GetFieldInst(field, instance_index);
  if (!field_inst) {
    printf("getMinMax error couldn't get instance\n");
    return -1;
  }

  switch(field->value_type) {
  case SW_VT_INT: {
    int* cast_values = (int*)field_inst->values;
    min = max = cast_values[0];
    minId = 0; maxId = 0;
    for (int i = 1; i < numNode; i++) {
      if (cast_values[i] < min) {
        min = cast_values[i];
        minId = i;
      }
      else if (cast_values[i] > max) {
        max = cast_values[i];
        maxId = i;
      }
    }
  } break;
  case SW_VT_FLOAT: {
    float* cast_values = (float*)field_inst->values;
    min = max = cast_values[0];
    minId = 0; maxId = 0;
    for (int i = 1; i < numNode; i++) {
      if (cast_values[i] < min) {
        min = cast_values[i];
```

```
        minId = i;
      }
      else if (cast_values[i] > max) {
        max = cast_values[i];
        maxId = i;
      }
    }
  } break;
  case SW_VT_DOUBLE: {
    double* cast_values = (double*)field_inst->values;
    min = max = cast_values[0];
    minId = 0; maxId = 0;
    for (int i = 1; i < numNode; i++) {
      if (cast_values[i] < min) {
        min = cast_values[i];
        minId = i;
      }
      else if (cast_values[i] > max) {
        max = cast_values[i];
        maxId = i;
      }
    }
  } break;
  default:
    printf("getMinMax error: don't know the value type\n");
    return -1;
  }

  return 0;
}
```

# Appendix I: DOL Test Output – Exodus Version

```
Mesh Summary: 'General Mesh'
  Spatial Dimension = (query blocks for this info)
  No. Blocks        = 4
  No. Coord Fields  = 1
  No. Sequence Sets = 1
  No. Sets          = 7 (Times, Top_elems, Top_nodes, Domain 0 elems, Domain 0 elems,
Exodus_Domain_0_Block_1_elems, Exodus_Domain_0_Block_2_elems)
  No. Relations     = 6
  No. Fields        = 16

Block Data (0): 'Top_Set'
  Spatial Dimension = no coordinate field
  No. Nodes         = 10088
  No. Edges         = 0
  No. Faces         = 0
  No. Zones         = 7152
  Zone type         = OtherPolyhedron

Block Data (1): 'Domain 0'
  Spatial Dimension = 3
  No. Nodes         = 10088
  No. Edges         = 0
  No. Faces         = 0
  No. Zones         = 7152
  Zone type         = OtherPolyhedron

Block Data (2): 'Exodus_Domain_0_Block_1'
  Spatial Dimension = no coordinate field
  No. Nodes         = 0
  No. Edges         = 0
  No. Faces         = 0
  No. Zones         = 4800
  Zone type         = Hexahedron

Block Data (3): 'Exodus_Domain_0_Block_2'
  Spatial Dimension = no coordinate field
  No. Nodes         = 0
  No. Edges         = 0
  No. Faces         = 0
  No. Zones         = 2352
  Zone type         = Hexahedron

Bounding Boxes:
  Block: Top_Set
    no coordinate field
  Block: Domain 0
error: bounding box can't handle 'Field Decomposed' fields yet
    no coordinate field
  Block: Exodus_Domain_0_Block_1
    no coordinate field
  Block: Exodus_Domain_0_Block_2
    no coordinate field

Field Summary on MeshObject: 'General Mesh'
  No. Fields        = 16 (GLOBAL_TIME, DISPLX, DISPLY, DISPLZ, VELX, VELY, VELZ, ACCLX, ACCLY,
ACCLZ, EQPS, EQPS, X, Y, Z, domain_coords)
  No. Time Fields   = 1
  No. Sequence Sets = 1
  No. Coord Fields  = 1

Sequence Set Query: 'Times'
  Type Name            = time support
  Set Type             = Set
  No. Members          = 44

Time Field Query: 'GLOBAL_TIME' (& it's set = 'Times')
  No. Entries          = 44
  No. Instances        = 1
  No. Values per Entry = 1
  Data type            = FLOAT
  Data Organization    = VE

Field Summary on Mesh Block 0: 'Top_Set'
  Total No. Fields on Nodes = 0
  Total No. Fields on Edges = 0
```

```
  Total No. Fields on Faces = 0
  Total No. Fields on Zones = 0

Field Summary on Mesh Block 1: 'Domain 0'
  Total No. Fields on Nodes = 13 (DISPLX, DISPLY, DISPLZ, VELX, VELY, VELZ, ACCLX, ACCLY, ACCLZ,
X, Y, Z, domain_coords, )
    No. Sequence Fields = 9
    No. Non-sequence Fields = 4
    No. Time Fields = 0
    No. Coord Fields = 1
  Total No. Fields on Edges = 0
  Total No. Fields on Faces = 0
  Total No. Fields on Zones = 0

Field Summary on Mesh Block 2: 'Exodus_Domain_0_Block_1'
  Total No. Fields on Nodes = 0
  Total No. Fields on Edges = 0
  Total No. Fields on Faces = 0
  Total No. Fields on Zones = 1 (EQPS, )
    No. Sequence Fields = 1
    No. Non-sequence Fields = 0
    No. Time Fields = 0
    No. Coord Fields = 0

Field Summary on Mesh Block 3: 'Exodus_Domain_0_Block_2'
  Total No. Fields on Nodes = 0
  Total No. Fields on Edges = 0
  Total No. Fields on Faces = 0
  Total No. Fields on Zones = 1 (EQPS, )
    No. Sequence Fields = 1
    No. Non-sequence Fields = 0
    No. Time Fields = 0
    No. Coord Fields = 0

Field Summary: 'EQPS'
  No. Entries          = 4800
  No. Instances        = 44
  No. Values per Entry = 1
  Data type            = FLOAT
  Data Organization    = VE

Basic Statistics for the field 'EQPS'
-------------------------------------------------
  Step Value      Min      Max   Min ID   Max ID
-------------------------------------------------
0.000000e+00        0        0        0        0
1.000737e-04        0    0.865       15       80
1.999051e-04        0     1.22      351      160
2.999644e-04        0     1.26      540      440
4.000865e-04        0     1.32      540      600
4.999192e-04        0     1.36      540      680
5.999351e-04        0     1.43      540      720
7.000492e-04        0     1.56      541      760
8.000353e-04        0     1.64      541      760
9.000607e-04        0     1.66      541      760
1.000006e-03        0     1.75      542      640
1.099981e-03        0     1.76      542      640
1.199933e-03        0     1.77      544      640
1.299987e-03        0      1.8      544      640
1.400093e-03        0     1.83      545      640
1.500041e-03        0     1.83      546      760
1.599919e-03        0     1.85      546      760
1.699999e-03        0     1.86      632      760
1.800073e-03        0     1.86      674      760
1.900023e-03        0     1.88      715      640
1.999986e-03        0     1.88      718      640
2.099930e-03        0     1.88      756      760
2.199921e-03        0     1.88     2275      760
2.299993e-03        0     1.89     2316      640
2.400054e-03 0.000169     1.89      759      640
2.499943e-03  0.00046     1.89      798      640
2.599921e-03 0.000662      1.9      798      640
2.699980e-03 0.000865      1.9      798      640
2.800000e-03  0.00144      1.9      838      640
2.899911e-03  0.00573      1.9      438      640
3.000021e-03   0.0107      1.9     1357      640
3.099979e-03   0.0124      1.9     1358      640
3.199995e-03   0.0129      1.9     1359      640
3.299942e-03   0.0138      1.9     1359      640
3.400023e-03   0.0146      1.9     1359      640
3.499968e-03   0.0189      1.9     1359      640
```

| | | | | |
|---|---|---|---|---|
| 3.599992e-03 | 0.0201 | 1.9 | 3119 | 640 |
| 3.699962e-03 | 0.0201 | 2.15 | 3119 | 557 |
| 3.799987e-03 | 0.0202 | 2.56 | 3119 | 398 |
| 3.899982e-03 | 0.0203 | 2.7 | 3119 | 398 |
| 3.999986e-03 | 0.0208 | 2.8 | 3119 | 398 |
| 4.100011e-03 | 0.0209 | 2.84 | 3119 | 396 |
| 4.199974e-03 | 0.0214 | 2.87 | 3119 | 3486 |
| 4.299989e-03 | 0.0216 | 2.89 | 3119 | 3486 |

# Appendix J: DOL Test Output – SAF Version

```
digraph "../data/can_dol.saf" {
rankdir=LR
"TOP_CELL";
"Block ID 1 - HEX";
"Block ID 2 - HEX";
"TOP_CELL" -> "Block ID 2 - HEX" [label="nodes"]
"TOP_CELL" -> "Block ID 2 - HEX" [label="elems"]
"TOP_CELL" -> "Block ID 1 - HEX" [label="elems"]
"TOP_CELL" -> "Block ID 1 - HEX" [label="nodes"]
"Time_Suite_0";
"Block ID 1 - HEX" -> "Block ID 1 - HEX" [label="elems->nodes" color="green" fontcolor="green"
dir="back"]
"Block ID 2 - HEX" -> "Block ID 2 - HEX" [label="elems->nodes" color="green" fontcolor="green"
dir="back"]
}

Mesh Summary: 'TOP_CELL'
  Spatial Dimension = (query blocks for this info)
  No. Blocks        = 3
  No. Coord Fields  = 2
  No. Sequence Sets = 9
  No. Sets          = 15 (TOP_CELL, Cells Block ID 1 - HEX, Cells Block ID 2 - HEX, TOP_CELL,
Cells Block ID 1 - HEX, Cells Block ID 2 - HEX, Times, DISPLVEC, DISPLVEC, VELVEC, VELVEC,
ACCLVEC, ACCLVEC, EQPS, EQPS)
  No. Relations     = 14
  No. Fields        = 11

Block Data (0): 'TOP_CELL'
  Spatial Dimension = no coordinate field
  No. Nodes         = 10088
  No. Edges         = 0
  No. Faces         = 0
  No. Zones         = 7152
  Zone type         = Hexahedron

Block Data (1): 'Block ID 1 - HEX'
  Spatial Dimension = 3
  No. Nodes         = 6724
  No. Edges         = 0
  No. Faces         = 0
  No. Zones         = 4800
  Zone type         = Hexahedron

Block Data (2): 'Block ID 2 - HEX'
  Spatial Dimension = 3
  No. Nodes         = 3364
  No. Edges         = 0
  No. Faces         = 0
  No. Zones         = 2352
  Zone type         = Hexahedron

Bounding Boxes:
  Block: TOP_CELL
    no coordinate field
  Block: Block ID 1 - HEX
    [ -5.2, 0, -15 ]  -  [ 5.2, 5.2, 0 ]
  Block: Block ID 2 - HEX
    [ -7.87846, 0, -0.462285 ]  -  [ 8.31258, 8, 4.7781 ]

Field Summary on MeshObject: 'TOP_CELL'
  No. Fields        = 11 (Time_State, DISPLVEC, DISPLVEC, VELVEC, VELVEC, ACCLVEC, ACCLVEC, EQPS,
EQPS, coords, coords)
  No. Time Fields   = 1
  No. Sequence Sets = 9
  No. Coord Fields  = 2

Sequence Set Query: 'Times'
  Type Name           = time support
  Set Type            = Set
  No. Members         = 44

Time Field Query: 'Time_State' (& it's set = 'Times')
  No. Entries         = 44
  No. Instances       = 1
  No. Values per Entry = 1
```

```
  Data type            = FLOAT
  Data Organization    = VE

Field Summary on Mesh Block 0: 'TOP_CELL'
  Total No. Fields on Nodes = 0
  Total No. Fields on Edges = 0
  Total No. Fields on Faces = 0
  Total No. Fields on Zones = 0

Field Summary on Mesh Block 1: 'Block ID 1 - HEX'
  Total No. Fields on Nodes = 4 (DISPLVEC, VELVEC, ACCLVEC, coords, )
    No. Sequence Fields = 3
    No. Non-sequence Fields = 1
    No. Time Fields = 0
    No. Coord Fields = 1
  Total No. Fields on Edges = 0
  Total No. Fields on Faces = 0
  Total No. Fields on Zones = 1 (EQPS, )
    No. Sequence Fields = 1
    No. Non-sequence Fields = 0
    No. Time Fields = 0
    No. Coord Fields = 0

Field Summary on Mesh Block 2: 'Block ID 2 - HEX'
  Total No. Fields on Nodes = 4 (DISPLVEC, VELVEC, ACCLVEC, coords, )
    No. Sequence Fields = 3
    No. Non-sequence Fields = 1
    No. Time Fields = 0
    No. Coord Fields = 1
  Total No. Fields on Edges = 0
  Total No. Fields on Faces = 0
  Total No. Fields on Zones = 1 (EQPS, )
    No. Sequence Fields = 1
    No. Non-sequence Fields = 0
    No. Time Fields = 0
    No. Coord Fields = 0

Field Summary: 'EQPS'
  No. Entries          = 4800
  No. Instances        = 44
  No. Values per Entry = 1
  Data type            = FLOAT
  Data Organization    = VE

Basic Statistics for the field 'EQPS'
-------------------------------------------------
  Step Value      Min      Max   Min ID   Max ID
-------------------------------------------------
0.000000e+00        0        0        0        0
1.000740e-04        0    0.865       15       80
1.999050e-04        0     1.22      351      160
2.999640e-04        0     1.26      540      440
4.000870e-04        0     1.32      540      600
4.999190e-04        0     1.36      540      680
5.999350e-04        0     1.43      540      720
7.000490e-04        0     1.56      541      760
8.000350e-04        0     1.64      541      760
9.000610e-04        0     1.66      541      760
1.000010e-03        0     1.75      542      640
1.099980e-03        0     1.76      542      640
1.199930e-03        0     1.77      544      640
1.299990e-03        0      1.8      544      640
1.400090e-03        0     1.83      545      640
1.500040e-03        0     1.83      546      760
1.599920e-03        0     1.85      546      760
1.700000e-03        0     1.86      632      760
1.800070e-03        0     1.86      674      760
1.900020e-03        0     1.88      715      640
1.999990e-03        0     1.88      718      640
2.099930e-03        0     1.88      756      760
2.199920e-03        0     1.88     2275      760
2.299990e-03        0     1.89     2316      640
2.400050e-03 0.000169     1.89      759      640
2.499940e-03 0.000461     1.89      798      640
2.599920e-03 0.000662      1.9      798      640
2.699980e-03 0.000865      1.9      798      640
2.800000e-03  0.00144      1.9      838      640
2.899910e-03  0.00573      1.9      438      640
3.000020e-03   0.0107      1.9     1357      640
3.099980e-03   0.0124      1.9     1358      640
3.199990e-03   0.0129      1.9     1359      640
```

| | | | | |
|---|---|---|---|---|
| 3.299940e-03 | 0.0138 | 1.9 | 1359 | 640 |
| 3.400020e-03 | 0.0146 | 1.9 | 1359 | 640 |
| 3.499970e-03 | 0.0189 | 1.9 | 1359 | 640 |
| 3.599990e-03 | 0.0201 | 1.9 | 3119 | 640 |
| 3.699960e-03 | 0.0201 | 2.15 | 3119 | 557 |
| 3.799990e-03 | 0.0202 | 2.56 | 3119 | 398 |
| 3.899980e-03 | 0.0203 | 2.7 | 3119 | 398 |
| 3.999990e-03 | 0.0208 | 2.8 | 3119 | 398 |
| 4.100010e-03 | 0.0209 | 2.84 | 3119 | 396 |
| 4.199970e-03 | 0.0214 | 2.87 | 3119 | 3486 |
| 4.299990e-03 | 0.0216 | 2.89 | 3119 | 3486 |

## Distribution

| | | |
|---|---|---|
| 1 | MS 9915 | W. S. Koegler |
| 3 | MS 9018 | Central Technical Files, 8945-1 |
| 1 | MS 0899 | Technical Library, 9616 |
| 1 | MS 9021 | Classification Office, 8511 for Technical Library, MS 0899, 9616 |
| | | DOE/OSTI via URL |