Investigations into using HDF5 as An Alternative to STEP for Finite Element Modeling Data

Vailin Choi and Mike Folk

The HDF Group

April 2007

1 Introduction

This study continues previous investigations into the viability of using a general purpose format such as HDF5 for preserving engineering data.[1]

The previous effort focused on two product model data objects: Cartesian points and B-splines [1]. This study investigates another type of data object that presents scalability challenges: finite element modeling (FEM) data. FEM is a common method used in engineering design and can produce enormous amount of data. The main goal of this investigation is to explore how HDF5 might provide effective storage to meet the needs of ever-increasing growth of data volumes.

Using two sets of data collections, the objectives of this work are to (a) map finite element representations together with their related entities into HDF objects and structures, (b) explore different ways of mapping these data into HDF5 based on the finite element STEP data characteristics and HDF5 features, and (c) assess the benefit and costs of such HDF5 storage mappings.

2 Background

Current formats for managing finite element modeling data are based on the STEP format[2]. The semantics of data stored in STEP is described in the EXPRESS data description language, an object-oriented information modeling language [2].

Currently, EXPRESS objects are instantiated in STEP as ASCII text. Alternative XML implementations are also available [3]. STEP is a successful ISO standard[4], and is well accepted and supported. However, ASCII-based implementations do not adapt well for highly voluminous, complex data, such as data used in applications like finite element analysis and computational fluid dynamics. STEP is also not suitable for the heterogeneous supporting data found in collections of product model data, such as digital photographs, blueprints, and formatted text.

These shortcomings of STEP have led to a number of attempts to develop an alternate binary format. One such effort is EuroSTEP's EXPRESS/Binary Project [4], whose goal is to "standardize a mapping into a more efficient (i.e. binary) file representation" for EXPRESS data. It is hoped that this alternate file representation may satisfy the needs of several applications, such as Thermal Analysis and Finite Element Analysis, which utilize very large datasets for their information models. The EuroSTEP Project identified a number of criteria for such a new file representation:

- Available everywhere – open source, free
- Architecture-independent specification and extensibility
- Platform-independent implementation
- General purpose capability
  - Custom Structure Definition/Representation
  - Alignment with EXPRESS (datatypes, structures, rules)
  - Support for mixed content
- Fast read/write performance
- Out-of-core data access (partial I/O)
- Large files support, compression
- Applicability for long term archiving
- Pedigree, viability, widespread usage and some standardization
- Support, training, high quality documentation

The EXPRESS/Binary Project selected HDF5 (Hierarchical Data Format) as the binary format for initial investigation. The project has developed mappings from EXPRESS to HDF5, and has recently presented a prototype of these mappings to the International Standards Organization (ISO).

The work reported here builds on the EuroSTEP Project, and was done in collaboration with the Project. It uses some of the EXPRESS-HDF5 mappings to investigate

---

[1] This work is part of a project at the National Center for Supercomputing Applications (NCSA) and The HDF Group, supported by the National Archives and Records Administration (NARA), to investigate the use of scientific data formats to support long term preservation of high volume, complex federal records.

[2] STEP (**ST**andard for the **E**xchange of **P**roduct data) refers to the "International Standard ISO 10303 *Industrial systems and integration - Product data representation and exchange*."

the viability of using HDF5 as a binary format for storing finite element data.

3 Mapping of EXPRESS Data to HDF5

The two basic constructs of the EXPRESS language are **entities** and **types**. **Types** in EXPRESS include primitive types such as *integer* and *string*, as well as *enumeration*, *aggregation*, and *select* types. An **entity**, which represents a real-world object, is composed of named properties called attributes, each of which has an associated data type. An instance of an entity has an identifier as well as values associated with each declared attribute. A **schema** is a collection of EXPRESS entity declarations while a **data population** consists of entity instances that conform to the entity data type declarations.

HDF5 is a general purpose file format for storing large or complex volumes of scientific data. It consists of two primary objects: datasets and groups. "A **dataset** is essentially a multidimensional array of data elements, and a group is a structure for organizing objects in an HDF5 file. Datasets and groups can be stored in different ways so as to improve storage or I/O efficiency, and both can have associated metadata in the form of HDF5 **attributes**. Using these as basic building blocks, one can create and store almost any kind of scientific data structure in HDF5, such as images, arrays of vectors, and structured and unstructured grids." [5]

The representation of EXPRESS-driven data using HDF5 is specified by relating EXPRESS data concepts to HDF5 data concepts. Each entity declaration is represented as an HDF5 compound data type with the corresponding HDF5 data types for its attributes. The set of instances of an EXPRESS entity type is treated as a dataset in HDF5 and each population of an EXPRESS schema is represented as an HDF5 group. Details of mapping from EXPRESS data to HDF5 are described in *EXPRESS Data as HDF5 Mapping Specification Version 0.5*. [4]

4 Testing Data

4.1 Collections

Two sets of data from STEP files are used for this study:

A. The first set, **CONROD.stp** (henceforth "CONROD"), is populated based on the *structural_analysis_design* schema and contains finite element data for 10-node quadratic tetrahedra. The CONROD STEP file is about 1,206K (kilobytes).

B. The second set, **hull_mesh_1.stp** (henceforth "HULL"), is also populated based on the *structural_analysis_design* schema. It contains finite element data for 3-node linear triangles and 4-node linear quadrilaterals. The HULL STEP file is 1,368K.

4.2 Data characteristics

To investigate the appropriateness of HDF as the binary format for the finite element data, VOLUME_3D_ELEMENT_REPRESENTATION in the CONROD collection and SURFACE_3D_ELEMENT_REPRESENTATION in the HULL collection are identified. The former representations are for 10-node quadratic tetrahedra and the latter are for 3-node linear triangles and 4-node linear quadrilaterals.

4.3 Testing scalability with larger files

From an HDF5 perspective, the CONROD and HULL files are relatively small. Indeed both are small enough that they can easily fit into the memory of most computers today. Since the purpose of this study is to investigate the use of HDF5 for high volume FEM data, two larger files were created.

These two files are based on the CONROD and HULL files, and were created by increasing by a factor of 100 the number of those entity instances that seemed most likely to increase when the resolution of a FEM would increase, namely Cartesian points, nodes, and elements. The values stored for these entities were created with a random number generator. Table 2 lists the affected elements.

The larger files are referred to as CONROD* and HULL*.

5 Storage Formats in HDF

As observed from the above tables, **node, surface_3d_element_representation, volume_3d_element_representation, cartesian_point** and **direction** are entities which have relatively larger number of instances in either collection. They are selected for applying the HDF5 storage methods discussed below because some of their attributes are of significance in showing the appropriateness of HDF as the binary format for conversion. All five entities are also selected to be stored in HDF with compression and without compression for this study.

5.1 Method 1 – use exact EXPRESS-HDF5 mapping v. 6.

All the entities are converted to HDF counterparts following closely to what are described in the EXPRESS schema definition and in *EXPRESS Data as HDF5 Mapping Specification Version 0.5* [4]. Each entity with its attributes is represented as an HDF compound structure with the corresponding fields. See the column "**HDF5**

Method 1" in Tables 3-7 for the mapping of the five selected entities.

Table 1 lists the number of instances of these two element representations together with their associated entities.

Entity	# of instances for CONROD	# of instances for HULL
ALIGNED_SURFACE_3D_ELEMENT_COORDINATE_SYSTEM	0	2
CARTESIAN_POINT	15,235	5,500
CHARACTERIZED_OBJECT	1	1
DATA_ENVIRONMENT	1	1
DIMENSIONAL_EXPONENTS	2	3
DIRECTION	236	4
ELEMENT_MATERIAL	1	1
FEA_AXIS2_PLACEMENT_3D	15	2
FEA_LINEAR_ELASTICITY	1	1
FEA_MASS_DENSITY	1	1
FEA_MATERIAL_PROPERTY_REPRESENTATION	2	2
FEA_MODEL_3D	1	1
FEA_PARAMETRIC_POINT	0	2
MATERIAL_PROPERTY	1	0
MEASURE_REPRESENTATION_ITEM	1	1
NAMED_UNIT	5	6
NODE	1,260	5,497
PROPERTY_DEFINITION	0	1
PROPERTY_DEFINITION_REPRESENTATION	1	1
REPRESENTATION	3	3
REPRESENTATION_CONTEXT	2	1
SURFACE_3D_ELEMENT_DESCRIPTOR	0	2
SURFACE_3D_ELEMENT_REPRESENTATION	0	5,453
SURFACE_ELEMENT_PROPERTY	0	1
SURFACE_SECTION_FIELD_CONSTANT	0	1
UNCERTAINTY_MEASURE_WITH_UNIT	1	0
UNIFORM_SURFACE_SECTION	0	1
VOLUME_3D_ELEMENT_DESCRIPTOR	1	0
VOLUME_3D_ELEMENT_REPRESENTATION	546	0

Table 1. Elements converted from CONROD and HULL files to HDF5, including the number of instances for each entity.

Entity (Increased 100 times)	# of instances for CONROD*	# of instances for HULL*
CARTESIAN_POINT	1,523,500	550,000
NODE	126,000	549,700
SURFACE_3D_ELEMENT_REPRESENTATION	0	545,300
VOLUME_3D_ELEMENT_REPRESENTATION	54,600	0

Table 2. Elements whose numbers were increased in the larger files, including the number of instances for each entity.

5.2 Method 2 – substitute fixed length for variable-length elements

Adhering to the original STEP schema definition in [4], Method 1 uses variable length datatypes for certain attributes, such as the *name* attributes in Table 3 through Table 7. Because variable length representations in HDF5 have a significant overhead, Method 2 stores attributes with varying elements as fixed-length arrays in HDF5. This is based on the observation that all instantiations for that entity have the same number of elements for its variable-length attribute. See the column **HDF5 Method 2** in the tables below.

Changes to the mapping for these STEP entities are described below:

- CARTESIAN_POINT:
  - *name* is a variable length string. Since this attribute is at most 12 characters long for all the instances of this entity, it is represented as a character array of length 12.
- DIRECTION:
  - *name* is a variable length string. Since it is at most 12 characters long, it is represented as a character array of length 12.
- NODE:
  - *name* is a variable length string. Since this attribute is at most 6 characters long for all the instances of this entity, it is represented as a character array of length 6.
  - *items* has only one reference to a *cartesian_point* entity for all instances of this entity in both collection set A and B. It is therefore represented as an integer in the HDF compound structure for *node*.
- SURFACE_3D_ELEMENT_REPRESENTATION:
  - *name* is a variable length string. Since this attribute is at most 6 characters long for all the instances of this entity, it is represented as a character array of length 6.
  - *node_list* has at most 4 references to a *node* entity for all instances of this entity in collection set B. It is therefore represented as an integer array of 4 elements in the HDF5 compound structure for *surface_3d_element_representation*.
  - *items* has 2 references to *aligned_surface_3d_element_coordinate_system* and *fea_parametric_point* entities for all instances of this entity in collection set B. It is therefore represented as an integer array of 2 elements in the HDF5 compound structure for *surface_3d_element_representation*.
- VOLUME_3D_ELEMENT_REPRESENTATION:
  - *name* is a variable length string. Since it is at most 6 characters long, it is represented as a character array of length 6.
  - *node_list* has 10 references to a *node* entity for all instances of this entity in collection set A. It is therefore represented as an integer array of 10 elements in the HDF5 compound structure for *volume_3d_element_representation*.
  - *items* has 1 reference to an *fea_axis2_placement_3d* entity for all instances of this entity in collection set A. It is therefore represented as an integer in the HDF5 compound structure for *volume_3d_element_representation*.

5.3 Method 1-A or 2-A – remove repeated values

The third storage method is implemented together with either **Method 1** or **2**, and is referred to as **Method 1-A** or **2-A** respectively.

This method is based on the observation that some references from the following three entities to other entity instances are the same for all the instantiations of that entity. These references are therefore stored as metadata in the form of HDF attributes associated with the corresponding dataset. See the last column in tables 3-7.

Changes to the mapping for these STEP entities are described below:

- NODE:
  - *context_of_items* and *model_ref* references are the same for all instances of this entity in collection set A or B. They are therefore stored together as a compound structure in the form of HDF5 attribute associated with the *NODE* dataset.
- SURFACE_3D_ELEMENT_REPRESENTATION:
  - *context_of_items*, *model_ref*, *property* and *material* references are the same for all instances of this entity in collection set B. They are therefore stored together as a compound structure in the form of HDF5 attribute associated with the *surface_3d_element_repreasentation* dataset.
- VOLUME_3D_ELEMENT_REPRESENTATION:
  - *context_of_items*, *model_ref* , *element_descriptor* , and *material* references are the same for all instances of this entity in collection set A. They are therefore stored together as a compound structure in the form of HDF5 attribute associated with the *volume_3d_element_repreasentation* dataset.

5.4 Summary

Table 3 through Table 7 summarize the four different methods described above.

Attribute	EXPRESS types	HDF5 Method 1 (direct conversion from EXPRESS	HDF5 Method 2 (variable length elements stored as	HDF 5 Method 1-A/2-A (repeated values stored as HDF5
name	label (STRING)	variable length of H5T_C_S1	ARRAY[12] of H5T_C_S1	Same
coordinates	LIST[1:3] of length_measure (REAL)	ARRAY[3] of H5T_NATIVE_FLOAT	ARRAY[3] of H5T_NATIVE_FLOAT	Same

Table 3 Storage Methods for **cartesian_point**

Attribute	EXPRESS types	HDF5 Method 1	HDF5 Method 2	HDF 5 Method 1-A/2-A
name	label (STRING)	variable length of H5T_C_S1	ARRAY[12] of H5T_C_S1	same
direction_ratios	LIST[2:3] of REAL	ARRAY[3] of H5T_NATIVE_FLOAT	ARRAY[3] of H5T_NATIVE_FLOAT	

Table 4 Storage Methods for **direction**

Attribute	EXPRESS types	HDF5 Method 1	HDF5 Method 2	HDF 5 Method 1-A/2-A
name	label (STRING)	variable length of H5T_C_S1	ARRAY[6] of H5T_C_S1	Same
items	SET [1: ?] of representation_item (ENTITY)	variable length of H5T_NATIVE_INT	H5T_NATIVE_INT	Same
context_of_items	representation_context (ENTITY)	H5T_NATIVE_INT	Same	HDF5-attribute
model_ref	fea_model (ENTITY)	H5T_NATIVE_INT	Same	HDF5-attribute

Table 5 Storage Methods for **NODE**

Attribute	EXPRESS types	HDF5 Method 1	HDF5 Method 2	HDF 5 Method 1-A/2-A
name	label (STRING)	variable length of H5T_C_S1	ARRAY[6] of H5T_C_S1	same
items	SET[1:?] of representation_item (ENTITY)	variable length of H5T_NATIVE_INT	ARRAY[2] of H5T_NATIVE_INT	same
context_of_items	representation_context (ENTITY)	H5T_NATIVE_INT	Same	HDF5-attribute
node_list	LIST[1:?] of node_representation (ENTITY)	variable length of H5T_NATIVE_INT	ARRAY[4] of H5T_NATIVE_INT	Same
model_ref	fea_model_3d (ENTITY)	H5T_NATIVE_INT	Same	HDF5-attribute
element_descriptor	surface_3d_element_descriptor (ENTITY)	H5T_NATIVE_INT	Same	same
property	surface_element_property (ENTITY)	H5T_NATIVE_INT	Same	HDF5-attribute
material	element_material (ENTITY)	H5T_NATIVE_INT	Same	HDF5-attribute

Table 6 Storage methods for **SURFACE_3D_ELEMENT_REPRESENTATION**

Attribute	EXPRESS types	HDF5 Method 1	HDF5 Method 2	HDF 5 Method 1-A/2-A
name	label (STRING)	variable length of H5T_C_S1	ARRAY[6] of H5T_C_S1	same
items	SET[1:?] of representation_item (ENTITY)	variable length of H5T_NATIVE_INT	H5T_NATIVE_INT	same
context_of_items	representation_context (ENTITY)	H5T_NATIVE_INT	Same	HDF5-Attribute
node_list	LIST[1:?] of node_representation (ENTITY)	variable length of H5T_NATIVE_INT	ARRAY[10] of H5T_NATIVE_INT	Same
model_ref	fea_model_3d (ENTITY)	H5T_NATIVE_INT	Same	HDF5-attribute
element_descriptor	volume_3d_element_descriptor (ENTITY)	H5T_NATIVE_INT	Same	HDF5-attribute
material	element_material (ENTITY)	H5T_NATIVE_INT	Same	HDF5-attribute

Table 7 Storage methods for **VOLUME_3D_ELEMENT_REPRESENTATION**

5.5 Data compression

All of the above methods were applied a second time to the CONROD and HULL files with compression applied to the datasets. However, only the HDF5 files were compressed, not the STEP files. This requires some explanation, as follows.

No matter what format is used, data compression can be beneficial in saving storage space and improving transmission time. In comparing a text-based format with a binary format, such as STEP with HDF5, there are some special factors to consider.

Formats such as STEP and XML can compress very well, but when a STEP or XML file is compressed, it becomes difficult to access individual records without uncompressing the entire file. HDF5 avoids this problem by individually compressing datasets, and by including indexes that tell where the compressed datasets are located in the file. This makes it possible to access data by only uncompressing the datasets of interest, not the whole file. Very large arrays in HDF5 can also be broken into individually-compressed "chunks", so that only those chunks containing the data records of interest need be uncompressed.

For these reasons, and since our interest is in storing and accessing STEP data in which there are very large arrays of data, we only consider the case in which we would not compress a STEP file, but *would* compress individual objects in an HDF5 file.

6 Results

Table 8 shows the results from the study of the original CONROD and HULL files, including the sizes of the original STEP files, and the sizes of the corresponding HDF5 files with and without compression for each storage method. Figures 1 and 2 show the percent space savings from using the different HDF5 storage methods.

		HDF5 with no compression				HDF5 with compression			
Set		**Method 1 (variable len types)**		**Method 2 (fixed length types)**		**Method 1 (variable len types)**		**Method 2 (fixed length types)**	
	STEP	**Method 1**	**Method 1A**	**Method 2**	**Method 2A**	**Method 1**	**Method 1A**	**Method 2**	**Method 2A**
CONROD	1,206	1,145	1,126	572	554	785	785	386	386
HULL	1,368	1,739	1,608	663	533	1,057	1,057	307	307

Table 8 File sizes (K) for storage methods 1, 1-A, 2, 2-A. See Figure 1 and Figure 2.

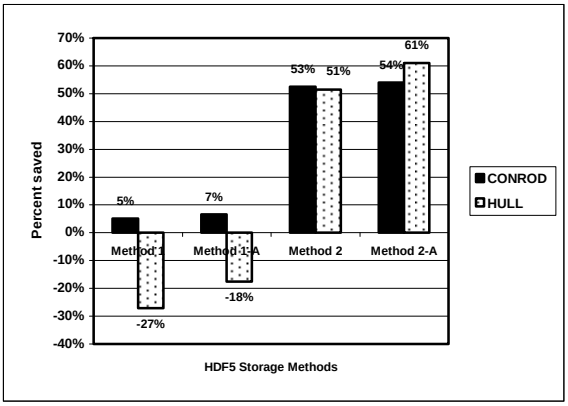

Figure 1. Percent saved between STEP and four HDF5 storage methods. No compression.

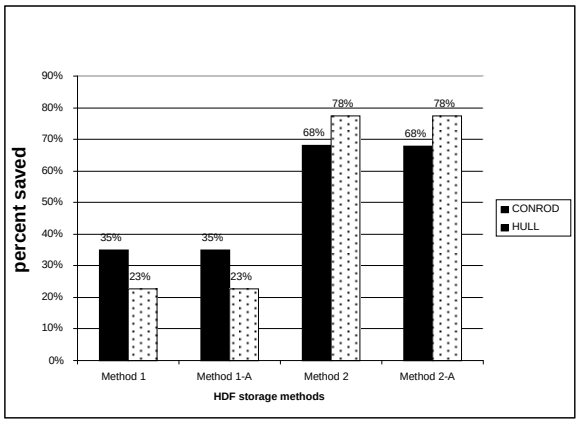

Figure 2. Percent saved between STEP and four HDF5 storage methods. With compression.

Table 9 shows the results from the study for CONROD* and HULL* (the large versions of the CONROD and HULL files), including the sizes of the original STEP files, and the sizes of the corresponding HDF5 files with and without compression for each storage method.

Figures 3 and 4 show the percent space savings from using the different HDF5 storage methods.

		HDF5 with no compression				HDF5 with compression			
Set		**Method 1 (variable len types)**		**Method 2 (fixed length types)**		**Method 1 (variable len types)**		**Method 2 (fixed length types)**	
	STEP	**Method 1**	**Method 1A**	**Method 2**	**Method 2A**	**Method 1**	**Method 1A**	**Method 2**	**Method 2A**
CONROD*	119,179	106,394	104,513	49,689	47,808	77,392	76,158	30,131	30,128
HULL*	138,545	164,384	151,262	59,202	46,080	96,963	95,674	23,538	23,531

Table 9. File sizes (K) for storage methods 1, 1-A, 2, 2-A, where high-multiple data are 100 times larger.

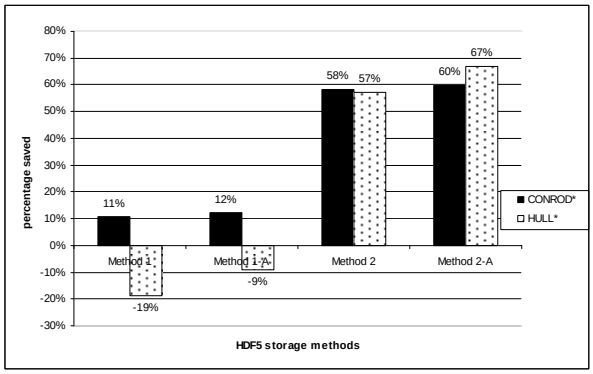

Figure 3. Percent saved on large files between STEP and four HDF5 storage methods. Certain entity instances are increased 100-fold. No compression.

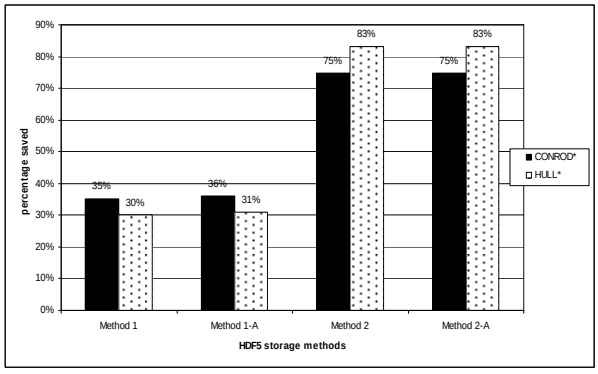

Figure 4. Percent saved on large files between STEP and four HDF5 storage methods. Certain entity instances are increased 100-fold. With compression.

7 Analysis

The following observations are made from the resulting data:

- Method 1, which uses variable-length datatypes for large volume types such as **surface_3d_element_representation,** is not effective, due to the high overhead of variable length types in HDF5. In the HULL collection, the uncompressed HDF5 file is actually larger. Even for CONROD, the percent of storage saved by using HDF5 is not high.
- Method 2, which replaces high-overhead variable length types with fixed-length types and padding, is a much better choice than Method 1. However, this is obtained with the pre-knowledge of the data pattern and also with the loss of self-description available when storing variable-length elements.
- Storing the unchanging values as HDF5 attributes, as in Method 1-A and 2-A, can sometimes improved storage efficiency. This is demonstrated with the small HULL collection, where Method 2-A saved 10% more than Method 2. This change seemed to be of no help for the compressed datasets, however.
- For compression, storage using fixed-length arrays as in Method 2 is highly beneficial. The percentage saved is at least doubled using Method 2 over Method 1.
- When the files were increased 100-fold in size, the same patterns appeared, with HDF5 modestly more effective for the large datasets than for the smaller ones.

8 Conclusion and future work

HDF5 is considered as a binary option to the text-based STEP formats, especially as an alternative to large STEP files such as those with FEM data that are too large to load into memory. Certain high-frequency entities, such as Cartesian points, nodes and element representations, can be stored and accessed efficiently in HDF5 by storing them as elements of HDF5 datasets.

This study found that how we choose to store these entities in HDF5 can make a notable difference in the size of the resulting file. For those entities that contain variable-length components, the recommended mapping had been to use HDF5 variable-length datatypes. This study found that the overhead associated with variable-length datatype storage in HDF5 leads to inefficient storage, and when compression is not used, HDF5 files were sometimes larger than their STEP counterparts.

An alternate datatype, which stores variable-length items in fixed size datatypes with padding, actually saved space, even without compression. And when compression was applied storage savings of greater than 80% could be achieved.

Future work. These results give confidence space can be saved by using HDF5 instead of text formats for high volume data. At the same time, HDF5 should also offer benefits when applications need to perform partial access on large datasets – that is, when datasets are too large to be loaded into a computer's memory. It is recommended that this assertion be investigated further.

9 Acknowledgments

We are grateful to Robert Chadduck and Mark Conrad of the National Archives and Records Administration (NARA) for supporting this work, including provision of STEP data files that were used in the testing. We also thank the EXPRESS-binary project that developed the material upon which this research is based, especially David Price, Keith Hunten, Denny Moore, and Steve Gordon. Steve and Denny provided FEM datasets that were crucial in doing the study.

10 References

[1] M. Folk and V. Choi. "Investigations into using HDF5 for product model data -- B-Spline and Cartesian Point data in HDF5." HDF Technical Report, January 2007. http://hdfgroup.org/projects/NARA/technical_reports.htm.

[2] P. Wilson. STEP and EXPRESS. http://deslab.mit.edu/DesignLab/dicpm/step.html.

[3] D. Rivers-Moore. XML and EXPRESS as schema definition languages. http://xml.coverpages.org/rivers-moore-k0603.html.

[4] Welcome to EXPRESS-Binary. http://www.exff.org/express_binary/index.html.

[5] What is HDF? http://hdfgroup.org.

This project is supported by the National Archives and Records Administration (NARA) through a supplement to The National Science Foundation PACI cooperative agreement CA #SCI-9619019. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation, the National Archives and Records Administration, or the U.S. government.