XML Upgrade for h5dump

Robert E. McGrath
April 20, 2003

Contents

Overview

The HDF5 DTD and h5dump --xml option have been stable since 2001. It is time to update these products to keep up with current standards and tools. This note explains several relatively small changes that should be implemented immediately.

In addition, we need to add support for two features that are introduced or substantially changed in 1.6: fill values and filters.

This document describes the following changes:

1. creating an XML schema
2. changes to the h5dump utility to support the schema
   a. new header for XML output
   b. option for 'qualified' tags, on by default
   c. generating proper XML ID's and IDREF's
   d. creating new attributes for HDF5 path information
3. 1.6 Features
   a. <RequiredFilters>
   b. <FillValueInfo>

4. Other changes

1. Creating an HDF5 XML Schema

The XML Schema language is considerably more powerful and flexible than the DTD language, and many new tools (including the ESML from UAH) are using schema. The most important XML software (e.g., Apache Xerces (used by IBM) and XMLSpy) support both DTD and schema, so there is no loss of function moving to schema, and much to be gained.

Fortunately, it is technically possible to create a schema that precisely matches our current DTD, which means that the same dumper output can be validated with XMLschema with only trivial changes (see below). In fact, the XML Spy tool automatically generates a schema from a DTD. I have used this tool to create a schema that is equivalent to the DTD, which is on the web at:

http://hdf.ncsa.uiuc.edu/DTDs/HDF5-File.xsd.

2. Required Changes to the Dumper

In a minimal implementation, the dumper output (*h5dump --xml*) can be used with the schema developed above, with one simple change to the first few lines of the output. The change is necessary to inform the parser how to parse the file.

The DTD version is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HDF5-File PUBLIC "HDF5-File.dtd"
"http://hdf.ncsa.uiuc.edu/DTDs/HDF5-File.dtd">
<HDF5-File>
```

For schema, these lines must be replaced with text to define the XML namespaces. There are several permutations, depending on whether the HDF5 schema is defined as a namespace or not, and whether the elements of are required to be 'qualified' or not.

The simplest variant is unqualified, which accepts the same tags as the old DTD-based output. The header looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<HDF5-File
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
      "http://hdf.ncsa.uiuc.edu/DTDs/HDF5-File.xsd">
```

The rest of the output can be identical.

In order to mix with other schemas, it is likely that people will want to use the HDF5 schema as an XML namespace, and will want to qualify the tags, e.g., `<Data>` should be `<hdf5:Data>`. I recommend that we implement this.

To implement this, the schema will say:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace=
     "http://hdf.ncsa.uiuc.edu/DTDs/HDF5-File"
      xmlns:hdf5="http://hdf.ncsa.uiuc.edu/DTDs/HDF5-File"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="unqualified"
      attributeFormDefault="unqualified">
```

which declares the namespace that the schema should be known by (i.e., the schema is a definition of the XML namespace `http://hdf.ncsa.uiuc.edu/DTDs/HDF5-File`).

When this schema is used, an XML file that describes an HDF5 file will define an XML namespace for HDF5, and point to the schema. The heading of the XML file that uses the schema should be something like:

```
<?xml version="1.0" encoding="UTF-8"?>
<hdf5:HDF5-File xmlns:hdf5="http://hdf.ncsa.uiuc.edu/DTDs/HDF5-File"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
    "http://hdf.ncsa.uiuc.edu/DTDs/HDF5File
    http://hdf.ncsa.uiuc.edu/DTDs/HDF5-File.xsd">
```

which defines the namespace '*hdf5:*' which is defined by the HDF5 schema.

Within the file, the tags must then be qualified with the namespace, as in:

```
<hdf5:RootGroup OBJ-XID="xid_928" H5Path="/">
```

To implement the new header requires changes to a few lines of the h5dump.c. It is desirable for the schema to be the default output, so we should add a new option to output the old DTD when needed. Also, there must be an option to specify the XML namespace to use, or to omit the qualifier.

I propose the following options for the dumper:

```
-x, --xml            Output in XML using Schema (default)
-u, --use-dtd        Output in XML using DTD
-D U, --xml-dtd=U    Use the DTD or schema at U
-X N, --xmlns=N      (XML Schema) Use qualified names n the XML
                     ":": no namespace, default: "hdf5:"
```

The '--xml' option is changed to output the preamble for the 'schema' instead of DTD. A new option, tentatively called '-u' will override the schema and produce the same output

as the current default. The '-D' option specifies a particular file or URL to use as the DTD or schema. Currently this value is always a DTD. When the schema is used, the value will need to be the target namespace and URL for the schema. (This value cannot be checked by the dumper, it is up to the user to specify the correct value.) The final option specifies alternatives for the namespace qualifier.

Within the dumper code, every place that a tag for an element is output, the prefix must be used if defined. For example, code such as:

```
printf("<RootGroup OBJ-XID=\"%s\"\">\n",…);
```

should be changed to:

```
printf("<%sRootGroup OBJ-XID=\"%s\" H5Path=\"%s\">\n",
            xmlnsprefix,…);
```

where ‘xmlnsprefix’ is set to ‘hdf5:’, “”, or whatever the user selected.

A side-effect of this change is that the entire ‘xml_dataformat’ structure cannot be used, and should be deleted. This affects only the XML-specific code.

3. Fixing XML ID, IDREF attributes

In the process of checking the schema it became apparent that the newer XML parsers are much more thorough than what we used two years ago: they found several minor bugs in the DTD and one major error in the output of the dumper.

The major error is as follows: in the DTD we give each 'top level' object (Group, Dataset, etc.) an XML tag (in the DTD, type ID). Cross references are made in other XML tags (e.g., a Dataset points to the Groups it is a member of), these tags are type IDREF. These tags are used only by the XML parser and XML based software, they have no meaning outside the specific XML document. In the current version of the dumper, we fill these tags with a full path for the object, and earlier parsers were content.

However, XML has several strict requirements on these tags:

1. Each ID must be unique within the file
2. The ID is required to follow the production rule:
   `Name ::= (Letter | '_' | ':') (NameChar) *`

Our path names are unique, but are rejected because they have '/' and can have other prohibited characters. As a result, the output of the dumper is rejected by strict parsers, even though it is perfectly correct otherwise.

Proposed solution

There might be several solutions.

- We could eliminate these tags altogether.
- We could work out some sort of swizzling algorithm to escape all the prohibited characters
- We could create ID's not based on paths.

It seems a shame to remove a feature that may be useful in the future, so I looked at how to generate legal values.

Since the legal names are quite restricted compared to HDF5 names, the swizzling would have to be quite extensive.

The suggested approach is to generate unique identifiers using object ID's, e.g., *xid_nnnn*. These are unique, and have at least some relation to the HDF5 file, and are already available in the dumper--they are used as keys in the table of paths used to generate the ID strings.

There were two gotcha's encountered.

1. Soft links are objects in the data model and the XML model, but have no object id.
2. Objects with more than one link to them appear in the XML multiple times, each record needs a unique ID (although the record is actually a pointer to the object).

These problems were dealt with by creating fake object ids for softlinks, and disambiguating the ID in the XML when needed. These are very simple algorithms.

Retaining the Path Information

The approach proposed above has the side-effect of eliminating the potentially useful paths for the objects and their parents. These paths are useful to a program parsing the XML for at least two uses:

- An unambiguous identifier for the object (e.g., for printing)
- A path to open the object in the file.

While the path information can be obtained by navigating the XML back to the root, it seems desirable to have the path encoded in an attribute (of type xs:string, rather than xs:id), for applications that want to use it.

This can be accomplished by creating new XML attributes to hold the information that was previously encoded in the OBJ-XID. This is simple to implement.

The main implication of this change is that the new schema/DTD is no longer compatible with the old one. This is likely to be acceptable, because we should make other changes to bring the XML up to HDF5-1.6 in any case. See section 3 below.

Required changes

Basically, every place where the value of an ID or IDREF is generated, the code has to be changed from 'xml_escape_the_name(path)', to call 'xml_name_to_XID(path)'. The new routine creates a string of the format *'xid_%lu'*, with the object id or a fake object id. The path information will be added in new attributes, H5Path="{path to object}" and H5ParentPaths="{path to parent, path to parent2, etc.}".

There are about 20 places in the code to be changed. These changes are isolated in code dedicated to XML, they do not affect the normal output.

In three places, the object may be a duplicate, in which case the output is: 'xid_%ld-%d', where the last number is incremented each instance.

To illustrate the basic change, consider original XML output for a Dataset and its containing groups:

```
<Group Name="g1" OBJ-XID="/g1" Parents="/" >
   <Group Name="g1.1" OBJ-XID="/g1/g1.1" Parents="/g1" >
      <Dataset Name="dset1.1.1" OBJ-XID="/g1/g1.1/dset1.1.1"
         Parents="/g1/g1.1">
```

In the new algorithm, the XML would be:

```
<hdf5:Group Name="g1" OBJ-XID="xid_1576" H5Path="/g1"
        Parents="xid_928" H5ParentPaths="/">
   <hdf5:Group Name="g1.1" OBJ-XID="xid_3200" H5Path="/g1/g1.1"
         Parents="xid_1576" H5ParentPaths="/g1">
             <hdf5:Dataset Name="dset1.1.1" OBJ-XID="xid_5200"
                 H5Path="/g1/g1.1/dset1.1.1" Parents="xid_3200"
                    H5ParentPaths="/g1/g1.1">
```

The difference is in the 'OBJ-XID' elements and the 'Parents' elements which point to an OBJ-XID, and the new elements 'H5Path' and 'H5ParentPaths' which have the old path names.

In the case of objects with more than one link, the original output looked something like:

```
<Dataset Name="dset1" OBJ-XID="/dset1" Parents="root">
   <!-- ...description omitted />
</Dataset>
<Group Name="g1" OBJ-XID="/g1" Parents="/" >
   <Dataset Name="dset2" OBJ-XID="/g1/dset2" Parents="/g1">
      <DatasetPtr OBJ-XID="/dset1"/>
   </Dataset>
```

where /g1/dset2 is a hard link to /dset1. Note that while /g1/dset2 has a unique path name, it is really the same object as /dset1 so it has the same HDF5 object id.

In the new XML, the link /g1/dset2 is assigned a dummy object id as illustrated:

```
<hdf5:Dataset Name="dset1" OBJ-XID="xid_976" H5Path="/dset1"
   Parents="xid_928" H5ParentPaths="/">
             <!-- ...description omitted />
  </Dataset>
    <hdf5:Group Name="g1" OBJ-XID="xid_1632" H5Path="/g1"
         Parents="xid_928" H5ParentPaths="/">
       <hdf5:Dataset Name="dset2" OBJ-XID="xid_976-1"
           H5Path="/g1/dset2" Parents="xid_1632"
            H5ParentPaths="/g1">
         <hdf5:DatasetPtr OBJ-XID="xid_976" H5Path="/g1/dset2"/>
       </hdf5:Dataset>
```

So, the Dataset pointer *(/g1/dset2*) has an XML id based on the HDF5 object id of the dataset it points to (xid_976) with '-1' appended. Subsequent appearances of the same object would have a different integer added. (This is implemented by a counter that gives a new number for each such collision.)

As noted above, Softlinks have no object id. A fake ID is created for each Softlink. In the original XML a softlink would be something like:

```
      TargetObj="/somevalue" OBJ-XID="/slink1" Source="root"/>
```

The new version has a fake object ID for the OBJ-XID:

```
    <hdf5:SoftLink LinkName="slink1" OBJ-XID="xid_18446744073709551614"
     H5SourcePath="/slink1" TargetPath="somevalue"
     Parents="xid_928" H5ParentPaths="/"/>
```

The fake object id is generated starting from HADDR_MAX and counting down by one for each subsequent link. This will eventually collide with a real object id, but it is unlikely that XML (or the dumper) will encounter such a case.

The XML Schema for the old and new versions of the Group are shown in Table 1. Datasets and Named Datatypes are similar. The old and new Softlink are shown in Table 2.

Table 1

a)
```
<xs:element name="Group">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="GroupPtr"/>
      <xs:sequence>
        <xs:element ref="Attribute" minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="Group"/>
          <xs:element ref="Dataset"/>
          <xs:element ref="NamedDataType"/>
          <xs:element ref="SoftLink"/>
        </xs:choice>
      </xs:sequence>
    </xs:choice>
    <xs:attribute name="Name" type="xs:string" use="required"/>
    <xs:attribute name="OBJ-XID" type="xs:ID" use="required"/>
    <xs:attribute name="Parents" type="xs:IDREFS" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="GroupPtr">
  <xs:complexType>
    <xs:attribute name="OBJ-XID" type="xs:IDREF" use="required"/>
  </xs:complexType>
</xs:element>
```

b)
```
<xs:element name="Group">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="hdf5:GroupPtr"/>
      <xs:sequence>
        <xs:element ref="hdf5:Attribute" minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="hdf5:Group"/>
          <xs:element ref="hdf5:Dataset"/>
          <xs:element ref="hdf5:NamedDataType"/>
          <xs:element ref="hdf5:SoftLink"/>
        </xs:choice>
      </xs:sequence>
    </xs:choice>
    <xs:attribute name="Name" type="xs:string" use="required"/>
    <xs:attribute name="OBJ-XID" type="xs:ID" use="required"/>
    <xs:attribute name="H5Path" type="xs:string" use="required"/>
    <xs:attribute name="Parents" type="xs:IDREFS" use="required"/>
     <xs:attribute name="H5ParentPaths" type="xs:string"
        use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="GroupPtr">
  <xs:complexType>
    <xs:attribute name="OBJ-XID" type="xs:IDREF" use="required"/>
    <xs:attribute name="H5Path" type="xs:string" use="required"/>
    <xs:attribute name="Parents" type="xs:IDREFS" use="required"/>
   <xs:attribute name="H5ParentPaths" type="xs:string"
      use="required"/>
 </xs:complexType>
</xs:element>
```

Table 2

a)
```
<xs:element name="SoftLink">
  <xs:complexType>
    <xs:attribute name="LinkName" type="xs:string" use="required"/>
    <xs:attribute name="Target" type="xs:string" use="required"/>
    <xs:attribute name="TargetObj" type="xs:IDREF"/>
    <xs:attribute name="OBJ-XID" type="xs:ID" use="required"/>
    <xs:attribute name="Source" type="xs:IDREF" use="required"/>
  </xs:complexType>
</xs:element>
```

b)
```
<xs:element name="SoftLink">
  <xs:complexType>
    <xs:attribute name="LinkName" type="xs:string" use="required"/>
    <xs:attribute name="TargetPath" type="xs:string"
        use="required"/>
    <xs:attribute name="TargetObj" type="xs:IDREF"/>
    <xs:attribute name="OBJ-XID" type="xs:ID" use="required"/>
    <xs:attribute name="H5Path" type="xs:string" use="required"/>
    <xs:attribute name="Parents" type="xs:IDREF" use="required"/>
    <xs:attribute name="H5ParentPaths" type="xs:string"
        use="required"/>
  </xs:complexType>
</xs:element>
```

4. New Features For HDF5-1.6

Along with the changes above, there are several revisions that should be made to implement new features in HDF5-1.6.

Revisions to the <Compression> tag: support <RequiredFilters>

With HDF5-1.6.0, there are significant new filters, both internal and external. The earlier DTD had a 'Compression' tag, which is not adequate to describe the filter pipeline. Therefore, a new tag is proposed, 'RequiredFilters', with sub-elements for different filters, not limited to compression. The old 'Compression' tag should be denigrated. (It was not fully implemented in the dumper.)

Figure 1 shows a diagram of the tags to describe the filters. (These apply only to chunked datasets.) For a given dataset, there will be zero or more instances of `<RequiredFilter>`, in the order of the filters (i.e., the order they will be used when writing data). Table 3 gives a summary of the new tags and their attributes. Note that these tags use XML schema data types.

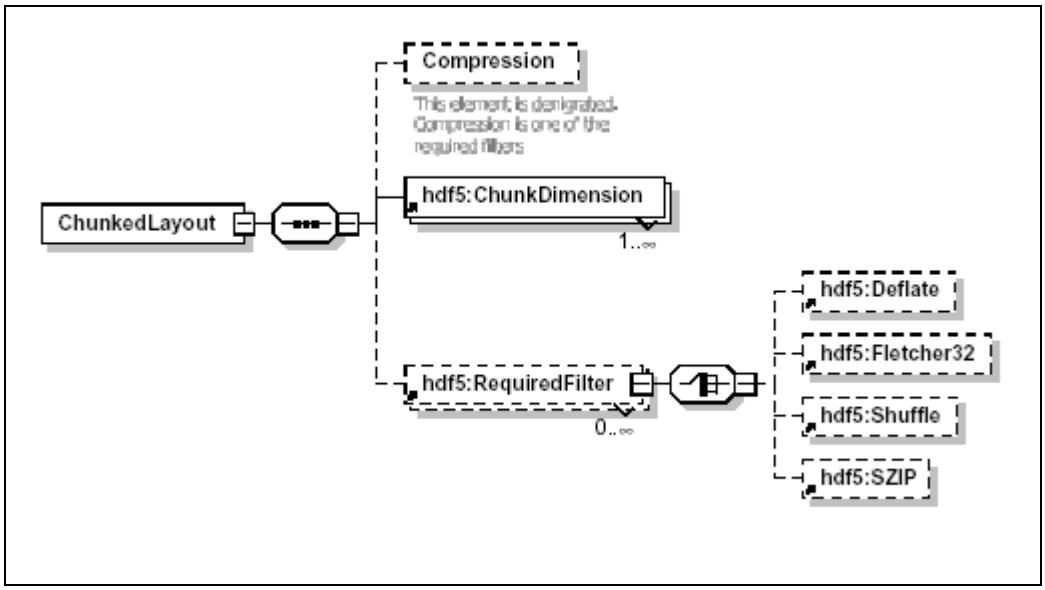

Figure 1

Table 3

Filter	Summary of XML Description
Deflate (gzip)	`<hdf5:Deflate level=”xs:int [1-9]”/>`
Fletcher32 checksum	`<hdf5:Fletcher32 />`
Shuffle (shuffle bytes)	`<hdf5:Shuffle />`

Szip	`<hdf5:SZIP`
	`Pixels_per_block="xs:int"`
	`[Mode=”Hardware|K13“]`
	`Coding="NN|Entropy" ByteOrder="LSB|MSB"`
	`[Header="Raw"]/>`

These tags are implemented in the dumper by new code that analyses the dataset creation property list to discover the required filters, and print the information about each. Figure 2 shows an example XML description of a dataset that was create using Szip compression. If the dataset had other filters (e.g., Shuffle), there would be additional lines between lines 7 and 10.

```
 1 <hdf5:Dataset Name="datasetF32" OBJ-XID="xid_976"
 2   H5Path="/datasetF32" Parents="xid_928" H5ParentPaths="/">
 3   <hdf5:StorageLayout>
 4     <hdf5:ChunkedLayout Ndims="2">
 5     <hdf5:ChunkDimension DimSize="100"/>
 6     <hdf5:ChunkDimension DimSize="100"/>
 7      <hdf5:RequiredFilter>
 8       <hdf5:SZIP Pixels_per_block="16" Coding="NN"
 9         ByteOrder="LSB" Header="Raw"/>
10      </hdf5:RequiredFilter>
11     </hdf5:ChunkedLayout>
12    </hdf5:StorageLayout>
13    <hdf5:FillValueInfo FillTime="FillOnAlloc" AllocationTime="Incremental">
14     <hdf5:FillValue>
15      <hdf5:Data>
16       <hdf5:DataFromFile>
17               "0.000000"
18       </hdf5:DataFromFile>
19     </hdf5:Data>
20    </hdf5:FillValue>
21   </hdf5:FillValueInfo>
22   <hdf5:Dataspace>
23     ...
24   </hdf5:Dataspace>
25   <hdf5:DataType>
26     ...
27   </hdf5:DataType>
28   <hdf5:Data>
29     ...
```

Figure 2

Add New Tags for <FillValueInfo>

Fill values were not implemented at all in the earlier DTD. HDF5-1.6.0 has new features for controlling storage allocation and filling. See [http://hdf.ncsa.uiuc.edu/RFC/Fill_Value/FillValue.html] for a detailed explanation. Figure 3 shows a diagram of the proposed tags to describe the fill value and allocation settings. Table 4 shows the tags and attributes.

Figure 2 above shows example XML description of a dataset with FillTime=FillOnAlloc and AllocTime=Incremental, and a fill value of '0' (lines 13-20).

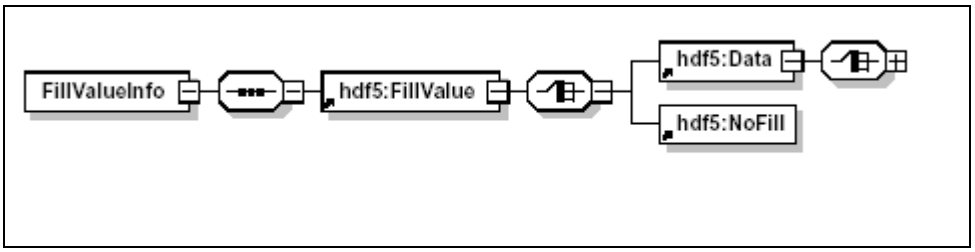

Figure 3

Table 4

Tag	Summary
FillValueInfo	`<hdf5:FillValueInfo FillTime="FillOnAlloc|FillNever" AllocationTime="Early|Late|Incremental">`
FillValue	`<Data>` Same as Dataset data
	`<NoFill>` No fill value is defined

5. Other Changes Related to XML

This document has explained the major changes in detail. Numerous other small changes will be made to correct errors or refine the schema. This section briefly lists them.

Revisions to the test output and documentation

The test output will need to be modified to match the new output. This affects only the h5dump XML test output, other tests will not be affected.

Some documentation for the dumper will need to be updated. This has not been investigated, but should be covered by parts of this note.

HDF5-1.6 Changes to the DTD (?)

It will be necessary to make equivalent changes to add the 1.6 features and updates to the DTD. The goal will be to assure that the output of the 'h5dump - - xml – u' option can be correctly parsed and validated with the DTD.

The changes fall into two logical categories:

- Addition of new tags for the 1.6 features described in this document
- Corrections and refinements

These changes are analogous to the schema described here.

It is important to note that the XML schema has some specifications that cannot be expressed or enforced in XML DTD. Notably, some attributes are declared to have data types (e.g., 'xs:unignedInt'), and some also have ranges specified (1-32). Also, the XML schema states rules such as "there can be 1-32 instances of this element". These concepts cannot be expressed in a DTD. Essentially, the new schema is more specific than the old DTD grammar.

The consequence is that it is possible to construct XML that would be accepted by the DTD but rejected by the schema. For example, the DTD would accept a 'DimSize' of "128" or even "a crazy value", while these would be correctly rejected by the schema.

XML generated by the h5dump utility will be correct, so this problem will occur only if users generate their own XML or modify the dumper output.

This is not likely to be a major problem.

Changes to the h5gen program (the reader)

The XML parsing needs minor modifications to work with the new Xerces. This is not very difficult, and works with schema with minimal changes.

The *h5gen* program was written to use the old Java object model, it will need to be rewritten to use the new object model. This is a big project which is not considered here.

Without this tool, we have no way to generate HDF5 from XML.

Miscellaneous Future Changes

We will probably want to add new features to the XML Schema in the future. These include:

- new objects such as 'image', 'table', 'palette', and 'dimension' (the latter will appear in HDF5-1.8)
- use of powerful features of XML schema, such as mapping XML datatypes to HDF5 datatypes, defining numeric attributes (such as "DimSize") as appropriate numbers.
- incorporation of ideas from XDF or similar array definition languages to mark up data elements
- implementing pointers (extended URLS) to objects in the binary HDF file

This document does not consider these or other possible enhancements.

There are a number of minor changes that might be made to make the XML reflect the current design of HDF5. These include:

- constraining the number of dimensions of a dataspace to 32
- delete the ComplexDataspace tag
- delete the DimPerm attribute

Acknowledgements

This report is based upon work supported in part by a Cooperative Agreement with NASA under NASA grant NAG 5-2040. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Aeronautics and Space Administration.

Other support provided by NCSA and other sponsors and agencies
(http://hdf.ncsa.uiuc.edu/acknowledge.html)

Some of the diagrams in this paper were prepared using XMLSpy IDE from Altova, Inc.
(www.altova.com).