HDF5 Dimension Scale Proposal No. 3

Draft Version 3

Mike Folk

September 15, 2004

1 Introduction and overview

1.1 Background

This proposal is an attempt to combine the earlier dimension scale proposals [1] and [3]. It also draws on discussions from [2] and [4].

This is not an easy task, largely because the two proposals have different emphases and sometimes conflicting requirements. For example,

- The first proposal stresses the need for compatibility with HDF4 and netCDF. The second covers a much wider range of applications, including support for different coordinate systems.
- The first proposal recommends that no changes be required to the HDF5 storage model or library, and that the design should be similar to what the HDF4 to HDF5 mapping specifies. The second recommends significant change to the HDF5 storage model; it does not address library changes.
- The two proposals also differ in terms of the conceptual model that would be supported. The second proposal is fairly specific about how dimension scales are to be related to one another, and also proposes that coordinate system information be included. The first proposal is unspecific about these matters, leaving these decisions up to the application, or possibly to be determined as the model is fleshed out.

In this third proposal, we try to resolve the various views.

1.2 At what level should dimension scales be implemented?

The most difficult issue, for me, has been the extent to which dimension scales should be defined and implemented as high level objects (as with images, tables, and unstructured grids), versus the extent to which they should be defined and implemented as part of the basic HDF5 data model, format and library.

This issue arises because in one sense dimension scales are application-specific constructs, and in another they are application-independent data structure constructs. They are application-*specific* in the sense that their meaning derives from how they help us understand the scientific space to which they apply – for example, they help locate information in a geographic space. They are application-*independent* in the sense that they provide relevant information about a dataset's dataspace – for example, a dataspace includes information about each dimension, and it is useful to extend that information to include a dimension name.

Sharability is another concept that can fall on either side of the issue. When an application wants two dimensions to share a dimension scale, it might have different ideas about what that sharing means. For instance, one dimension might map to the entire scale, and another to a subset of the scale. On the other side, sharability for HDF5 means that two or more objects share another object that provides structural metadata, such as a datatype or a dataspace, and the relationships are strictly defined in the HDF5 model.

We can resolve the issue in four ways:

1. Do nothing.
2. Hybrid approach – implement some features within the HDF5 data model and library, and implementing some features as a high level model and library.
3. High level approach – implement completely as a high level model and library
4. Basic model approach. – implement completely within the basic HDF5 data model and library

1. Do nothing. We have ruled this out already – our users need and deserve some kind of support for dimension scales.

2. Hybrid approach. The hybrid approach would do some things at a high level, such as implementing dimension scales as datasets, and others by expanding the basic model and library, such as putting links to dimension scales into the dataspace header information.

This approach seemed to reflect the consensus of the group that emerged from our discussions last Spring, and most of my thinking during the past five months has assumed that this would be the approach. However, I now believe that it will be very hard to accommodate both views in an initial implementation, and I believe it could be very confusing to users if we do.

If we implement a dimension scale as a dataset (similar to image and table), and at the same time put dimension scale info into the base library and format, we are mixing high and low level information and functions.

In HDF4 we mix high and low level approaches where we use Vdatas as both high level and internal objects. This is somewhat different than what we are proposing here, but there may be some lessons from this experiences. The result was not terrible, but this use of Vdatas has led to some confusion by users and tool builders, and it probably has made the format specification more complicated than necessary. On balance, I believe the hybrid approach was not a good solution with HDF4.

Furthermore, there are as yet no other examples in which a high level object (dataset, group) is used in some specialized way by the basic data model and library, and this seems very reasonable.

Therefore, I recommend that we choose one or the other – the high level or the basic approach. There may be some exceptions to this, but they should be small and compelling.

Basic or high level? If we have to choose, which do we choose? Are we to think of dimension scales as just another use of datasets, like images, tables and unstructured grids? Or are we to think of them as fundamental components of datasets that can be shared.

If it's the former ("another use of datasets"), then a high level library seems like the best approach, with no changes to the format. This approach also has the advantage that we can treat the first implementation as a prototype that can be changed later with no harm done to the format or base library. On the other hand, if we choose the high level approach and then later change our mind, we will be stuck with legacy applications that depend on this approach, and new applications that depend on the new approach, leading to confusion and maintenance headaches. This could be significantly mitigated by labeling the approach as a prototype, but that strategy requires a plan, and resources, to do a full implementation in the near future.

If the answer is "fundamental components," then the format *will* have to be changed in certain ways, such as including more dimension information in dataspaces (dimension names, links to dimension scales, etc.), and perhaps also in the headers of datasets that are sub-classed as dimension scales. These changes have the same disadvantages as do those of the higher level approach, and they become even harder to alter later on than those in a high level prototype.

I believe that our users are best served if we choose the first option – to think of dimension scales as a high level use of datasets that have no special meaning in the HDF5 data model or library.

2 Proposal

2.1 Summary

It is recommended that coordinate systems not be included at this time.

It is recommended that dimension scales be stored as datasets, with metadata indicating that they are to be treated as dimension scales. Each dimension scale is to have an optional name. There is no requirement as to where dimension scales should be stored, nor must dimension scale names be unique within a file.

Datasets would be linked to dimension scales. Each dimension could optionally have one or more associated dimension scales, as well as a local name for the dimension scale. A dimension scale would be sharable by two or more dimensions, including dimensions in the same dataset.

Relationships between dataset dimensions and their corresponding dimension scales would not be directly maintained or enforced by the library For instance, a dimension scale would not be automatically deleted when all datasets are deleted that refer to it. It is tentatively recommended that the information that connects a dataset to a dimension scale would be store as an attribute in the dataset.

A number of functions are proposed for dealing with dimension scales, such as a function to convert a dataset to a dimension scale. For the most part, these functions should be high level functions.

In order to enable dimension scales to be represented as functions (a frequently requested feature), it is recommended that the dataset model be expanded to allow datasets to be represented by a generating function.

2.2 Detailed summary

Here is a more detailed summary of the recommendations in the new proposal:

1. Do not include coordinate systems at this time.
2. Dimension scales should have the following properties
    a. Dimension scales are to be stored as HDF5 datasets, with the following characteristics.
        i. There are no restriction on the size, shape, or datatype of dimension scales.
        ii. Read/write behavior is the same for dimension scales as it is for normal datasets.
        iii. Dimension scales are public. That is, they are as visible within a file as other datasets.
    b. A CLASS attribute is used to specify that a dataset is to be interpreted as a dimension scale, similar to the way it is used with HDF5 images. An HDF5 attribute is to be used for storing this attribute.
    c. A dimension scale can have at most one primary name. The name should be stored as an HDF5 attribute for the dimension scale dataset.
    d. Dimension scale names should not have to be unique within a file.
3. Datasets should have the following new properties
    a. Each dimension in a dataset may have any number of corresponding dimension scales.
    b. Each dimension may have any number of names, even if there is no corresponding dimension scale.
    c. If more than one scale or name are associated with a dimension, these are to be identified by index values.
    d. To accommodate the association of dimension names and scales, zero or more "dimension attribute records" may be associated with each dimension of a dataset, with the following information.
        i. Index value
        ii. Dimension name (optional)
        iii. Object reference to dimension scale (optional)

    We recommend storing this information as attributes, but we can see benefits in storing it in the dataspace header.
4. The following are recommended regarding relationships between dimensions and dimension scales.
    a. Two or more dimensions can share the same dimension scale. That is, a dimension scale may be associated with more than one dimension in more than one dataset.
    b. The relationships between dimension scales and dataset dimensions are not implicitly maintained or enforced by the library.

i. When a dataset dimension is extended, the HDF5 library is not required to automatically extend any corresponding dimension scales.
ii. The library does not maintain information about dimensions that are shared by more than one dataset.
iii. Dimension scales are not automatically deleted when datasets are deleted.
iv. The library does not enforce, nor require, that a dimension name stored in a dimension attribute record be the same as the name in the scale itself.

5. The following new functions are proposed for dealing with dimension scales. It is recommended that these be implemented as high level functions.
    a. Convert dataset to scale (D, name) – convert dataset D to a dimension scale.
    b. Attach scale (D, S, i) – attach dimension scale S to the ith dimension of D.
    c. Detach scale (D, i, j) – detach the jth scale from the ith dimension of D, and decrement the dim scale count for D.
    d. Get number of scales (D, i) – get the number of scales associated with the ith dimension of D.
    e. Get ID of scale (D, i, j) – get the ID for the jth scale associated with the ith dimension dataset D.
    f. Get scale info (S) – get info about dimension scale S (existence, size, name, etc.)
6. Expanding raw data options
    a. Extend the dataset model to allow a new storage option for datasets whereby datasets can be represented by a function, or formula.
    b. Study the possibility of allowing formulas to be used for attributes.

3 Details of conceptual model

Proposals [1] and [3] differ in their conceptual models. The following proposal includes (and excludes) features of both.

Our study of dimension scale use cases has revealed an enormous variety of ways that dimension scales can be used. We recognize the importance of having a model that will be easy to understand and use for the vast majority of applications. It is our sense that those applications will need either no scale, a single 1-D array of floats or integers, or a simple function that provides a scale and offset.

At the same time, we want to place as few restrictions as possible on other uses of dimension scales. For instance, we don't want to require dimension scales to be 1-D arrays, or to allow only one scale per dimension.

So our goal is to provide a model that serves the needs of two communities. We want to keep the dimension scale model conceptually simple for the majority of applications, but also to place as few restrictions as possible how dimension scales are interpreted and used. With this approach, it becomes the responsibility of applications to make sure that dimension scales satisfy the constraints of the model that they are assuming, such as constraints on the size of dimension scales and valid range of indices.

3.1 Should coordinate systems be represented in HDF5?

Perhaps the biggest difference between [1] and [3] is in the requirement in [3] that a coordinate system support be available. We argued in [4] that we felt it was premature to support coordinate systems in HDF5 at this time, and it was our sense that the reviewers agreed. Also, we have since discovered that the coordinate system design proposed in [3] is not the same as the dimension space that is defined in a netCDF file, so would not seem to be usable in the netCDF implementation.[1] For these reasons, it seems premature to support coordinate systems at this time.

[1] The proposal requires a different coordinate system for each different combination of dimensions. This complicates bookkeeping quite a bit, and could lead to confusion. This may be fixable, but that will take time.

Recommendation: do not include coordinate systems at this time.

3.2 What types of scales should be implemented?

There seems to be good agreement that the model should accommodate scales that consist of a stored 1-D list of values, certain simple functions, and "no scale." Higher dimensional arrays are more problematic, but we would recommend them as well:

No scale. Frequently no scale is needed, so it should not be required. In some of these cases, an axis label may still be needed, and should be available.

1-D array. Both fixed length and extendable arrays should be available. We recommend that the size not be required by HDF5 to conform to the size of the corresponding dimension, so that then number of scale values could be less than, equal to, or greater than the corresponding dimension size.

Simple function. At a minimum, a linear scale of the form A + Bx should be available. Beyond this, the initial scope is TBD, but should probably be linear.

Higher dimensional arrays. Proposal [3] makes a good case for including arrays with dimension greater than 1, and we are recommending these. The recommendations for 1-D arrays as to size and extendibility would seem to apply here as well.

Recommendation: support the following types of dimension scale: no scale, array of any dimension, simple function at least of the form A+Bx.

There is also the issue of whether there should be restrictions on the datatypes of scale values. Since the interpretation of dimension scale values will be left to applications, it seems reasonable not to restrict applications in this regard.

Recommendation: make no restriction of the datatypes of scale values.

A number of use cases have been proposed in which more than one scale is needed for a given dimension. If this can be done without overly complicating the model, it seems to be a valuable feature, and hence is recommended.

Recommendation: make no restrictions on the number of scales that can be associated with a dimension.

4 Implementation details

4.1 Dimension scales as HDF5 datasets

It is proposed that dimension scales be stored as datasets. This approach would seem to satisfy the data model describe above, and has the advantages of simplifying the API and concentrating much of the code on one common structure, rather than two. Details of the special characteristics and operations for dimension scale datasets are covered in the next several sections.

Recommendation: store dimension scales as HDF5 datasets.

4.2 Structures and relationships

This section covers the structures recommended for handling dimension scale and their relationships.

Recommendation: in the proposed model, support the following structures and relationships:

- *A dimension scale is an object that is associated with a dimension of a dataset.*
- *A dimension scale can have at most one primary name.*
- *A dimension scale may be associated with more than one dimension in more than one dataset.*
- *Every dimension of a dataset may have one or more dimension scales associated with it. If there are more than one scale associated with a dimension, each scale is identified by an index value.*
- *Unless otherwise specified, a dimension scale inherits the properties of an HDF5 dataset.*

- *There are no restrictions on the size, shape, or datatype of a dimension scale.*

Comments::

- *On the recommendation to restrict dimension scales to having one primary name*. netCDF requires this, and it seems to be a fairly common assumption of applications. If an application needs to assign more than one name to a dimension scale, it can use HDF5 attributes to do so, but this concept will not be dealt with by the API. See section 4.4.4 for further discussion of this issue.
- *Allowing more than one scale to be associated with a dimension.* The index value of a given dimension scale will not be persistent when deletions and additions are performed. The exact behavior in this case needs to be determined.
- *On letting dimension scale inherit the properties of an HDF5 dataset, unless otherwise specified.* We have no specific restrictions in mind, but this leaves open the possibility that some will arise.
- *On allowing scales that are not 1-D.* No distinction will be made in the API between the simple case (one 1-D array per dimension) and the general case (any number of arrays of any dimension) described above. It will be left to the application to manage this.
- *[3] proposes to assign different dimension scale combinations, depending on coordinate system (examples 7 and 10 (maybe) in [3]).* Because we don't include coordinate systems in the proposed model, the model does not provide all of the information needed in examples 7 and 10 in [3]. In these cases, the coordinate system in [3] specifies which of several scales go with each dimension. The model proposed here would require the application to keep track of this information.
- *Should attributes and array datatypes be allowed to have dimension scales?* We know of no request for this feature, which we believe would complicate the model for users, so we do not at this time feel that attributes and array datatypes should support dimension scales.

4.3 Operations

In this section we recommend basic operations to be covered by a dimension scales API.

Recommendation: make the following new functions available for dealing with dimension scales

- *Convert dataset to scale (D, name) – convert dataset D to a dimension scale.*
- *Attach scale (D, S, i) – attach dimension scale S to the i^{th} dimension of D.*
- *Detach scale (D, i, j) – detach the j^{th} dimension scale from the i^{th} dimension of D, and decrement the dim scale count of D.*
- *Get number of scales (D, i) – get the number of scales associated with the i^{th} dimension of D.*
- *Get ID of scale (D, i, j) – get the ID for the j^{th} scale associated with the i^{th} dimension dataset D.*
- *Get info (S) – get info about dimension scale S (existence, size, name, etc.)*
- *The operations available for datasets are also available for dimension scales.*

In [1] there are three additional operations listed: destroy scale, change size of scale, and iterate through scales. The destroy and change-size operations can be done with existing HDF5 dataset operations; on the other hand, it could be argued that dimension scale-specific versions would be more natural for users. We are not recommending them, but are open to arguments to the contrary.

The iterate operation becomes difficult with the implementation that we are proposing, and seems to us to be best left to the applications. We are, of course, open to opposing views as to whether these operations should be included.

(We have not yet developed a programming model for operating on dimension scales. Nor have we tested these functions by showing how they would apply in various use cases.)

Operations not recommended. Because dimension scales add meaning to datasets, it is reasonable to look for ways to maintain the proper relationships between datasets and their corresponding dimension scales. Two operations that might be desired involve (1) automatically extending dataset dimensions, and (2) automatically deleting dimension scales when all datasets that use them are deleted. We recommend against supporting these operations in the library, and letting applications enforce them according to their needs. A discussion of this follows

Automatically extending dataset dimensions. When a dimension of a dataset is extended, should the library automatically extend the corresponding dimension scale, or should this be left to the application? Since a dimension scale can be shared among many datasets, this raises a number of issues that are difficult to address in a general way. For instance, which dimension scale should be extended when only one dataset is extended, and what values are to be added? We have seen no compelling reason to implement an automatic extension of dimension scales when dataset dimensions are extended, so we suggest letting applications be responsible for this operation.

Recommendation: do not automatically extend dimension scales when dataset dimensions are extended.

Automatically deleting dimension scales. Should a dimension scale be deleted when all datasets that use it have been deleted? This is another case where different applications might have different requirements, so a general policy would be difficult to devise. Furthermore, enforcing a deletion policy, even a simple one, adds complexity to the library, and could also affect performance. Deletion policies seem best left to applications.

Recommendation: do not automatically delete dimension scales in circumstances involving dataset deletion.

4.4 Dimension scale datasets – behavior and content

In this section we recommend characteristics of datasets that are to be treated as dimension scales.

4.4.1 Read/write behavior vs. normal dataset operations

Should the read/write behavior for dimension scales be different from that of other datasets?

There may be some reasons for this, such as how fill values are treated, and how dataset extension is dealt with. However, we are not aware that the lack of such special behavior would offset the advantages of letting the behavior be the same for dimension scales and for datasets that are not dimension scales.

Recommendation: make read/write behavior the same for dimension scales as it is for normal dataset.

4.4.2 Public and private dimension scales

Should we have both public and private dimension scales?

It might be useful in some case to be able to hide dimension scales from the top level, but this creates a more complicated model for users. Sometimes all dimension scales would be visible, sometimes some would be visible and some not, and sometimes none would be visible.

Recommendation: make all dimension scales public.

(We recognize that most of the HDF staff recommended otherwise, so this is probably a controversial recommendation.)

4.4.3 Dimension scale class

How do we avoid confusing dimension scales with other datasets?

One disadvantage of using datasets for dimension scales is that dimension scales might be confused as being something other than what they are. To lessen the confusion, we could specify a dimension scale class attribute (e.g. CLASS = "DIMENSION_SCALE"), or it could be a header message.

Recommendation: use a CLASS attribute to specify that a dataset is to be interpreted as a dimension scale.

Should attributes be used for storing the CLASS attribute?

Since a similar approach is used to identify images and tables, we recommend that attributes be used. It also would make this information apparent to higher level views of datasets, such as those provided by HDFView.

Recommendation: use an attribute for storing the CLASS attribute..

4.4.4 Dimension scale name

Dimension scales are often referred to by name, so we have recommended that dimension scales have names. Since some applications do not wish to apply names to dimension scales, we recommended that dimension scale names be optional.

We also recommend that a dimension scale should have at most one name. If a dimension scale has one name, bookkeeping involving the scale name may be significantly simplified, as would the model that applications must deal with. On the other hand, some applications may want to use different names to refer to the same dimension scale, but we do not consider this a capability that the HDF5 library needs to provide. (See further discussion of this in section 0.)

Recommendation: Dimension scales should have no name or one name.

How is a name represented?

Three options seem reasonable: (1) the last link in the pathname, (2) an attribute, (3) a header message.

1. Last link in the pathname. The h4toh5 mapping uses this approach, but there could be more than one path to a dataset, leading to ambiguities. This could be overcome by enforcing conventions.
2. Attribute. This exposes this information at the top level, making it accessible to any viewer that can see attributes. It also makes it easy for applications to change the name, which could be dangerous, or valuable.
3. Header message. This approach makes the name a little less available at the top level, but firmly pushes the concept into the base format and library. Since it also requires applications to change the name through a function call, it leaves open the possibility that the form of the name could be altered later without requiring application codes to change. On the other hand, if we treat names this way, it means that the "name" attribute is being treated differently from the "class" attribute, which could be confusing.

Recommendation. We tentatively recommend that the name be stored as an attribute.

Should dimension scale names be unique among dimension scales within a file?

We have seen a number of cases in which applications need more than one dimension scale with the same name. We have also seen applications where the opposite is true: dimension scale names are assumed to be unique within a file. One way to address this is for the library not to enforce uniqueness, and leave it to applications to enforce a policy of uniqueness when they need it. We recommend this approach.

Recommendation: Dimension scale names need not be unique within a file.

Can a dimension have a name, without having an associated scale?

It has been suggested that some applications may wish to name dimensions without having an associated scale. This seems like a reasonable recommendation.

Recommendation: a dataset may contain one or more names for a dimension, even when there is no corresponding dimension scale.

4.5 Information connecting datasets with dimension scales

4.5.1 What information is used to connect a dataset to its corresponding dimension scales?

The following have been suggested at one time or another:

- Reference to dimension scale
- Dimension scale name
- Units for dimension
- Current and maximum sizes of dimension scale
- Mapping between dimension and dimension scale

Reference to dimension scale

A dataset needs some way to identify a dimension scale. A dataset reference provides an unambiguous identifier, and is very compatible with the HDF5 programming model. Because of the recommendation that datasets may have named dimensions without corresponding scales, this reference should be optional.

Recommendation: a reference to a dimension scale be stored optionally in a dataset.

Dimension names and dimension scale names

It has been asserted that it would be useful to allow dimension scales to have a local name – a name that is stored in a dataset together with the reference to a dimension scale. This would allow easy look-up of the dimension scale name, it would allow a dataset to name a dimension without pointing to a dimension scale, and it would also allow applications to assign local names that differ from a dimension scale's own name.

Regarding the first reason (easy look-up), we do now know how important this would be. Without it, a dimension scale's header must be read in order to find a dimension scale's name. If this extra look-up is a problem, then applications could, of course, provide their own solutions to this problem in a variety of ways.

Regarding the second reason (axis name without attached scale), this seems to be a fundamental need, and is cited in example #5 in [2].

Regarding the third reason (alternate names), we have seen no compelling example of this requirement. (We welcome examples showing why a different local name would be good.)

The arguments against using local names have to do with complexity and bookkeeping requirements. The use of a different local name could make the model more confusing for users, and would probably require an extra query function for both the local name and the real name. The bookkeeping requirement would seem to be minor if there is no requirement that the local name be the same as the real name.

Recommendation: Allow dimension names to be stored optionally, even when there is no corresponding dimension scale.

Recommendation: The library should not enforce consistency between the name stored in the dataset and the name in the scale itself.

Units

Support for units has been requested, but this would seem to be beyond the scope of the current task, so it is not recommended at this time.

Current and max size

The main argument for including these is quick look-up. The main arguments for omitting them are costs in complexity and potential performance of synchronizing dataset dimensions with their corresponding dimension scales, without any compelling argument as to the value of such synchronizing. We are not recommending that this information be tracked at this time.

One-to-many mapping

When there are fewer values in a dimension scale than in the corresponding dimension, it is useful to have a mapping between the two. For example, mappings are used by HDF-EOS to map geolocation information to dimensions. On the other hand, the way that mappings are defined can be very idiosyncratic, and it would seem to be challenging to provide a mapping model that satisfied a large number of cases. Hence, we are not recommending that mappings be included in the model.

4.5.2 How is this information to be specified and organized?

This question is similar to the question of how dimension scale names are to be specified in dimension scales. Two options seem reasonable:

1. As a dataset attribute. This exposes this information at the top level, making it accessible to any viewer that can see attributes. It also makes it easy for applications to change the dimension number, name and reference, which could be dangerous, or valuable.
2. As part of the dataspace object. This is proposed in [3], and does seem like a natural place for this information. Since it also requires applications to change the values through a function call, it leaves open the possibility that their format could be altered later without requiring application codes to change. However, this approach firmly pushes the concept into the base format and library, which we are trying to avoid, and for that reason we do not recommend it.

Tentative recommendation: Put the information that connects datasets with dimension scales in an attribute.

As to how this information should be organized, one option would be to create an attribute in the dataset for every dimension scale that is the dataset links to. The attribute would contain all of the local information described above.

Recommendation: For each dimension scale that is linked to from a dataset, store the following information as an attribute in the dataset: the index value, an optional dimension name, and an optional object reference to the corresponding dimension scale.

4.6 Shared dimensions – additional capabilities

Given the design described above, datasets can share dimensions. The following additional capabilities would seem to be useful.

1. When a dimension scale is deleted, remove the reference to the dimension scale in all datasets that refer to it.
2. Determine how many datasets are attached to a given dimension scale
3. Determine what datasets are attached to a given dimension scale

These capabilities can be provided in several ways:

a) Back pointers. If every dimension scale contained a list of back pointers to all datasets that referenced it, then it would be relatively easy to open all of these datasets and remove the references, as well as to answer questions #2 and #3. This would require the library to update the back pointer list every time a link was made.

b) Alternatively, such lists could be maintained in a separate table. Such a table could contain all information about a number of dimension scales, which might provide a convenient way for applications to gain information about a set of dimension scales. For instance, this table might correspond to the coordinate variable definitions in a netCDF file.

c) If no such list were available, an HDF5 function could be available to search all datasets to determine which ones referenced a given dimension scale. This would be straightforward, but in some cases could be very time consuming.

d) Finally, it could be argued that these capabilities are so unlikely to occur that HDF5 need not provide a solution. When a dimension is deleted, dangling references would occur, but that is already possible for any dataset that uses object references. And if this were a problem, an application could always implement any of the three solutions on its own, although this would require more knowledge of HDF5 than might be available.

Recommendation: These three additional capabilities should not be supported in the library.

5 Expanding raw data options

5.1 Formula datasets

One dataset feature has been recommended that is not currently available – datasets represented by functions of the indices, which we tentatively call a *formula dataset*.

More formally, consider the following definitions

D – a dataset array.

r – the rank of D.

dim_i – the size of the i^{th} dimension of D, where $0 < i \leq r$.

v – an index vector for D. v is a vector of r integers, were $v = (i_1, i_2, \ldots i_r)$, $i_j \leq dim_j$. i_j refers to the j^{th} element of v. The components of v identify the coordinates of an element in D.

If D is an r-dimensional standard dataset, D(v) is the element of D whose coordinates are $(i_1, i_2, \ldots i_r)$.

If D is an r-dimensional formula dataset, there is a function f(v) defined for D, and D(v) is $f(i_1, i_2, \ldots i_r)$.

What formulas should be allowed? The design proposed in [3] has three options: linear, logarithmic, and other. We propose that the design should accommodate expansion, but the first implementation be *linear expression* only. That is an expression of the form

$f(v) = a_1 i_1^{e_1} + a_2 i_2^{e_2} + \ldots + a_r i_r^{e_r} + k.$, where the a_i and k are constants. <<Check with Elena.>>

Example.

Let A be a standard dataset with rank r = 2, $dim_1 = 30$ and dim_2 40, and v = (5, 2).

Then A(v) is the element A(5,2).

Example.

Let A be a formula dataset with rank r = 2, $dim_1 = 30$ and dim_2 40, and v = (5,2).

Since r = 2, $f(v) = a_1 i_1^{e_1} + a_2 i_2^{e_2} + k$.

We define $f(v) = 3i_1 + 4i_2^{\,2} + 13$, in which case $a_1=3$, $e_1=1$, $a_2=4$, $e_2=2$, and k=13.

Then A(v) is the same as f(5,2), or $3\times5 + 4\times2^2 + 13$, or 44.

Recommendation: extend the dataset model to allow a new storage option for datasets whereby datasets can be represented by a function, or formula.

Further study would seem to be advisable before proceeding with this option.

5.2 Formula attributes

If formulas are permitted for dataset, it may also be valuable to enable attributes to have the "formula" option.

Recommendation: Study the possibility of allowing formulas to be used for attributes.

6 References

1. McGrath, Robert E. "Dimension Scales in HDF5: Preliminary Ideas." May 2001. http://hdf.ncsa.uiuc.edu/RFC/ARCHIVE/DimScales/H5dimscales.htm.
2. McGrath, Robert E. "Needed: A convenience API to Support Dimensions in HDF5." July 2001. http://hdf.ncsa.uiuc.edu/RFC/ARCHIVE/DimScales/H5dims.htm.
3. Koziol, Quincey. "Coordinate Systems in HDF5." A set of slides. March 11, 2004.
4. Folk, Mike. "Should Dimension Scales be basic HDF5 constructs or higher level constructs?" May 2004. http://hdf.ncsa.uiuc.edu/RFC/ARCHIVE/DimScales/How_H5dimscales.htm.