Request for Comments: HDF5 Dimension Scales
DRAFT—expires January 3, 2005

December 21, 2004

This document is a working draft. The first version of the specification will incorporate comments and changes to this document. The specification will be completed as soon as possible.

Contents

1 Introduction and overview

This proposal describes an implementation of dimension scales for HDF5 based on [5], which summarized earlier discussions in [1] [2] [3] and [4]. This earlier work covered different use cases, along with different proposals for implementation strategy.

In the design discussions, the most difficult issue 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.

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 is 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.

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 approach.

This proposal implements 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. One significant consequence of this design decision is that Dimension Scales will be visible as regular (albeit distinguishable) HDF5 Datasets, accessible through APIs that do not implement the profile defined here. For some uses this will be an advantage, and for others it is a disadvantage.

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. In a similar vein, support for units has been requested. As in the case of coordinates, this would seem to be beyond the scope of the current task.

2 The HDF5 Standard for Dimension Scales

Dimension scales will be stored as datasets, with additional metadata indicating that they are to be treated as dimension scales. Each dimension scale has 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.[1]

Datasets are linked to dimension scales. Each dimension of a Dataset may optionally have one or more associated Dimension Scales, as well as a label for the dimension. A Dimension Scale can be shared by two or more dimensions, including dimensions in the same or different dataset.

Relationships between dataset dimensions and their corresponding dimension scales are not be directly maintained or enforced by the HDF5 library. For instance, a dimension scale would not be automatically deleted when all datasets that refer to it are deleted.

[1] The name of a dimension scale does not have to be the same as the HDF5 path name for the dataset representing the scale.

A number of functions are proposed for dealing with dimension scales, such as a function to convert a dataset to a dimension scale. These functions will be implemented as 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. This feature is defined in a separate document [12].

3 Conceptual model

As discussed in on [5], 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 Definitions

This document refers to the standard objects of the HDF5 Abstract Data Model [10, 11]. Dimension Scales are implemented as an extension of these objects. In the HDF5 Abstract Data Model, a Dataset has a Dataspace, which defines a multi dimensional array of elements. Conceptually, a Dataspace has N dimension objects, which define the current and maximum size of the array in that dimension.

It is important to emphasize that the Dataspace of a Dataset has no intrinsic meaning except to define the layout in computer storage. Dimension Scales may be used to store application specific labels to the positions in the stored data array.

A Dimension Scale is an object associated with one dimension of a Dataspace.[2] The meaning of the association is left to applications. The values of the Dimension Scale are set by the application to reflect semantics of the data, for example, to associate coordinates of a reference system with positions on the dimension.

In general, these associations define a mapping between *values of a dimension index* and *values of the Dimension Scale dataset*. A simple case is where the Dimension Scale *s* is a (one dimensional) sequence of labels for the dimesion *ix* of Dataset *d*. In this case, Dimension Scale is an array indexed by the same index as in the dimension of the Dataspace. For example, for the Dimension Scale *s*, associated with dimension *ix*, the *ith* position of *ix* is associated with the value *s[i]*, so *s[i]* is taken as a label for *ix[i]*.

See [3] [4] and [5] for several other possible uses of Dimension Scales.

3.2 Entity Relationship Diagrams

Figure 1 shows UML to illustrate the relationship between a Dimension and a Dimension Scale object. Conceptually, each Dimension of a Dataspace may have zero or more Dimension Scales associated with it.

---

[2] Some of the use cases in [3] require associating more than one dimension with a Dimension Scale. This proposal does not directly address this requirement, but applications can extend the current proposal to support this use.

In turn, a Dimension Scale object may be associated with zero or more Dimensions (in zero or more Dataspaces).

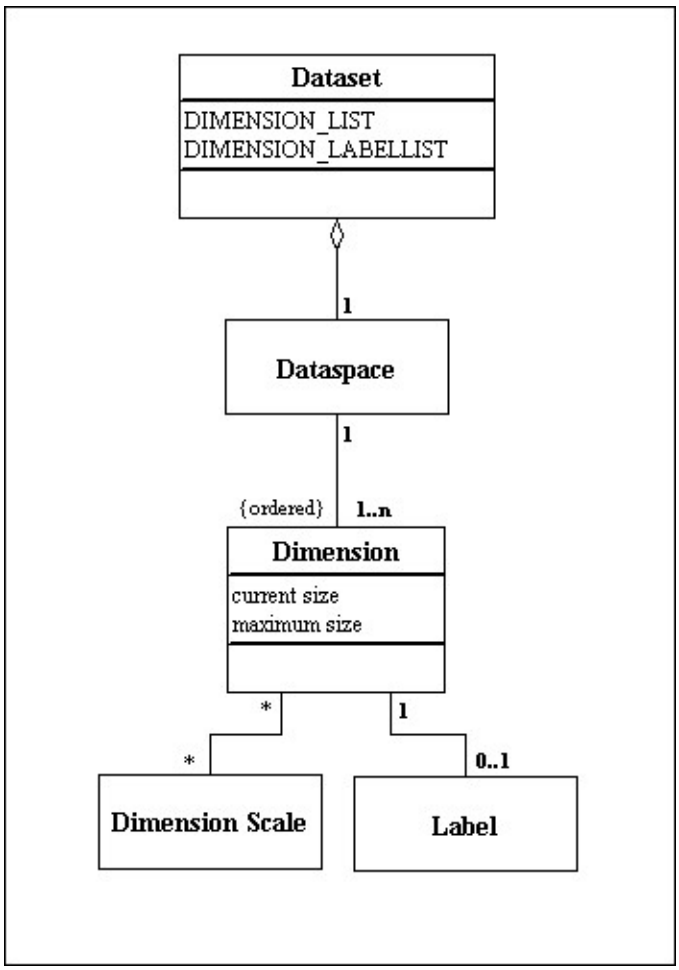

Figure 1. The relationship between a Dimension and a Dimension Scale.

Figure 2 illustrates the abstract model for a Dimension Scale object. A Dimension Scale is represented as a sub-class of a Dataset: a Dimension Scale has all the properties of a Dataset, with some specializations. A Dimension Scale dataset has an attribute "CLASS" with the value "DIMENSION_SCALE". (This is analogous to the Table, Image, and Palette objects [9].) The Dimension Scale dataset has other attributes, including an optional NAME and references to any associated Dataset, as discussed below.

When the Dimension Scale is associated with a dimension of a Dataset, the association is represented by attributes of the two datasets. In the Dataset, the DIMENSION_LIST is an array of pointers to scales (Figure 1), and in the Dimension Scale Dataset the REFERENCE_LIST is an array of pointers to Datasets (Figure 2).

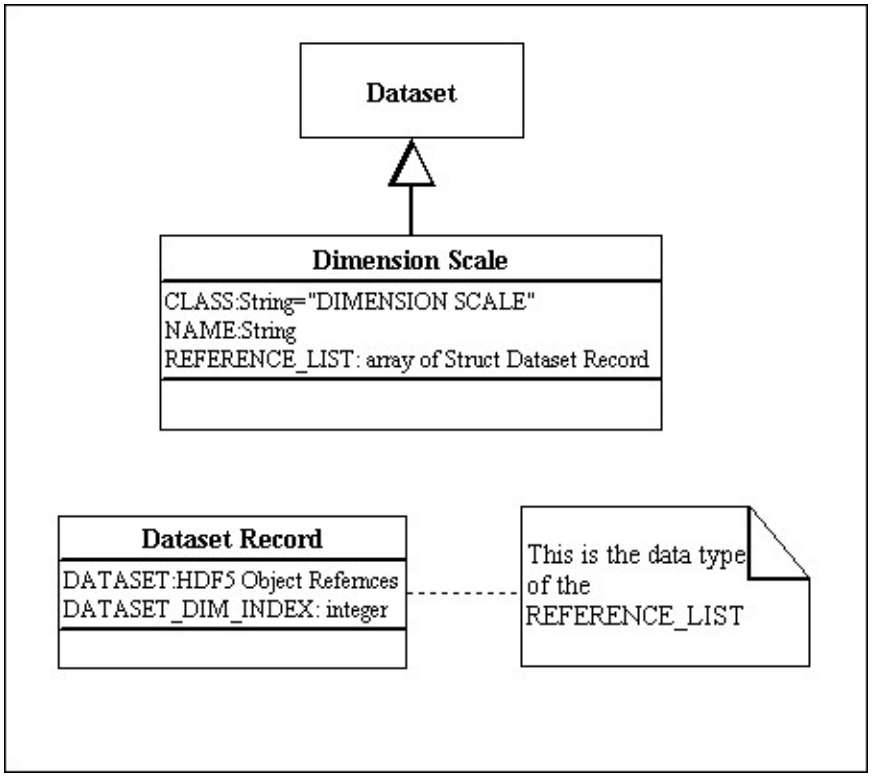

Figure 2. The definition of a Dimension Scale and its attributes.

3.3 What types of scales should be implemented?

As discussed in [5], 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." This proposal also includes scales that are higher dimensional arrays, as well. The four types of scales will be:

1. **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. In this case, the Dataset defines a Dimension Scale label for a dimension with no reference to a Dimension Scale dataset.
2. **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 the number of scale values could be less than, equal to, or greater than the corresponding dimension size.
3. **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. This is discussed in a separate proposal [12].
4. **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.

A number of use cases have been proposed in which more than one scale is needed for a given dimension. This proposal places no restrictions on the number of scales that can be associated with a dimension, nor on the number or identities of Dimensions that may share the same Dimension Scale.

As discussions in [5], there are use cases for storing many types of data in a scale, including, but not limited to integers, floats, and strings. Therefore, this proposal places no restrictions on the datatypes of scale values: *a Dimension Scale can have any HDF5 Datatype*. The interpretation of dimension scale values is left to applications.

3.4 Limitations of this Proposal

The model proposed here does not meet some requirements suggested by the use cases in [3]. These requirements must be implemented by other software, such as the netCDF 4 library [7]. These issues are discussed in detail in [4] and [5]. This section summarizes some of the limitations.

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. These mappings are not included in the model proposed here.

Visibility and Integrity. Since Dimension Scales are a specialization of a Dataset, it is "visible" and accessible as a regular Dataset through the HDF5 API. This means that an application program could alter the values of or delete a Dimension Scale object or required attributes without regard to any of the semantics defined in this document. This exposure has advantages and disadvantages, as discussed in [4].

One advantage it that the implementation requires no changes to the base library, which reduces the complexity of the code and the risk of side-effects. Also, the implementation builds on existing functions, which should improve the quality and reliability of the code.

An important disadvantage is that the core HDF5 library will not manage the semantics of Dimension Scales. In particular, applications or other software must implement:

1. Naming – the HDF5 library will impose no rules on the names of Dimension Scales
2. Consistency of references – e.g., if a Dataset (Dimension Scale) is deleted (e.g., with H5Gdelete(), any Dimension Scales (Datasets) that it refers to (refer to it) will not be updated by the HDF5 library.
3. Consistency of extents – the HDF5 library will not assure that a Dimension and associated Dimension Scale have the same extent (number of elements), nor that shared objects are consistent with each other. As in the case of delete, if a Dimension or Dimension Scale is extended (e.g., H5S…), any associated objects will not be automatically extended.

These are briefly summarized here.

Naming and Name Spaces. As discussed in [4] and [5], there are many potential schemes for naming dimensions, each suited for different uses. This specification does not impose any specific approach, so it may be used by different applications. However, the lack of restrictions has disadvantages as well.

For some purposes, it will be important to iterate through all the Dimension Scale objects in a file. This iterate operation is difficult with the design proposed here. This will be left to other software. For example, the HDF-EOS library [8] has its own mechanism for managing a set of dimensions, and the netCDF4 library [7] will implement this if it needs to.

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.

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.

Section 5 presents an API and programming model that implements some of these features. However, applications may ignore or bypass these APIs, to write or read the attributes directly.

4 Proposed Implementation

4.1 Brief Summary

A Dimension Scale is stored as an HDF5 Dataset.

- A Dimension Scale is an object that is associated with a dimension of a Dataset.
- A Dimension Scale can have at most one name.
- A Dimension Scale may be associated with zero, one, or many different dimensions in any number of Datasets.
- 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.

A Dimension Scale can be associated with a dimension of an HDF5 dataset

- A dimension of a Dataset may have zero, one, or more Dimension Scales associated with it.
- Each scale is identified by an index value.

A dimension may have a label without a scale, and may have a scale with no label.

- The label need not be the same as the name of any associated Dimension Scales.

The implementation has two parts:

1. A storage profile
2. An API and programming model

This section specifies the storage profile. Section 5 proposes an API.

4.2 Storage Profile

This section specifies the storage profile for Dimension Scale objects and the association between Dimensions and Dimension Scales.

This profile is compatible with an earlier netcdf prototype [13] and the HDF4 to HDF5 Mapping [6]. This profile is also compatible with the netCDF4 proposal [7]. This profile may be used to augment the HDF-EOS5 profile, as well [8].

See Appendix 2 for a discussion of how to store converted HDF4 objects. See Appendix 3 for a discussion of netcdf4 issues. See Appendix 4 for a discussion of HDF-EOS5.

4.2.1 Dimension Scale Dataset

A Dimension Scale dataset is stored as an HDF5 dataset. There is no restriction on the dataspace or datatype, or storage properties of the dataset. Table 1 summarizes the stored data, i.e., the values of the scale.

Table 2 defines the required and optional attributes of the Dimension Scale Dataset. The attribute `REFERENCE_LIST` is a list of (dataset, index) pairs. Each pair represents an association defined by 'attach_scale'. These pairs are stored as an array of compound data. Table 3 defines this datatype.

The Dimension Scale Dataset has an attribute called `SUB_CLASS`. This string is intended to be used to document particular specializations of this profile, e.g., a Dimension Scale created by netCDF4.

Table 1. The properties of the Dimension Scale dataset

Field	Datatype	Dataspace	Storage Properties	Notes
<data>	Any[1]	Any[2]	Any[3]	These are the values of the Dimension Scale.

Notes:

1. The datatype of the scale does not have to be the same as the datatype of the Dataset(s) that use the scale. E.g., an integer dataset might have dimension scales that are string or float values.
2. The dataspace can be any rank and shape. It is not limited to one dimension, and is not restricted by the size of any dimension(s) associated with it. When a dimension is a associated with a one dimensional scale, the scale may be a different size from the dimension. In this case, it is up to the application to interpret or resolve the difference. When a dimension is associated with a scale with a rank higher than 1, the interpretation of the association is up to the application.
3. The Dimension Scale dataset can use any storage properties (including fill values, filters, and storage layout), not limited by the properties of any datasets that refer to it. When the Dimension Scale is extendible, it must be chunked.

Table 2. Standard Attributes for a stored Dimension Scale dataset.

Attribute Name	Datatype and Dimensions	Value	Required / Optional	Notes
CLASS	H5T_STRING, len = 16	"DIMENSION_SCALE"	R	This attributes distinguishes the dataset as a Dimension scale object.
NAME	H5T_STRING, len = <user defined>	<user defined>[1]	O Maximum of 1	The user defined name of the Dimension Scale.
REFERENCE_LIST	Array of Dataset Reference Type (Compound Datatype), variable length. See Table 3	[{dataset1, ind1 }, …] [,…] ….	O, required when scale is attached	This is set by attach_scale.
SUB_CLASS	H5T_STRING, len = <profile defined>	"HDF4_DIMENSION", "NC4_DIMENSION",	O, defined by other profiles	This is used to indicate a specific profile was used.
<Other attributes>			O	For example, **UNITS**.

Notes:

1. The name does not have to be the same as the HDF5 path name for the dataset. The name does not have to be related to any labels. Several Dimension Scales may have the same name.

Table 3. Dataset Reference Type. This is a pair, <dataset_ref, index>. This is created when the Dimension Scale is attached to a Dataset.

Field	Datatype	Value	Notes
DATASET	Object Reference.	Pointer to a Dataset that refers to the	This scale is attached to dataset *d*.
INDEX	H5T_NATIVE_INT	Index of the dataspace of the DATASET	This scale is attached to dimension *i* of dataset *d*.

4.2.2 Attributes of a Dataset with a Dimension Scale

A Dataset may have zero or more Dimension Scales associated with its dataspace. When present, these associations are represented by two attributes of the Dataset. Table 4 defines these attributes.

The DIMENSION_LIST is a two dimensional array with one row for each dimension of the Dataset, and a variable number of entries in each row, one for each associated array. This is stored as a one dimensional array, with the HDF5 Datatype variable length array of object references.

Table 4. Standard Attributes of a Dataset with associated Dimension Scale.[1]

Attribute Name	Datatype and Dimensions	Value	Required / Optional	Notes
DIMENSION_LIST	Array of object references [*rank*], where *rank* is the rank of the dataspace. The datatype is Variable Length H5T_STD_REF_OBJ	[[{object__ref1, object__ref2, … object__refn}, …] […] ..]	O, required if scales are attached	
DIMENSION_LABELLIST	Array of H5T_STRING [*rank*] where rank is the rank of the dataspace.	[<Label1>, <Label2>, …, <Label3>]	O, required for scales with a label	

Notes:

1. Note that there may be a label without a reference, and vice versa.

When a dimension has more than one scale, the scales are addressed by their index (e.g., the order they were attached to the dimension). When a scale is detached, it will be necessary to revise the DIMENSION_LIST array.

4.3 Dimension Scale Names and Labels

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. In addition, some applications will have a name but no associated data values for a dimension (i.e., just a label). To support this, we propose to have dimension labels, which may be but need not be the same as the name of the associated dataset.

Dimension Scale Name. Associated with the Dimension Scale object. A Dimension Scale may have no name, or one name.

Dimension Label. A optional label associated with a dimension of a Dataset.

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 [6], 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.

We propose to use the second approach, encoding names in attributes of the Dimension Scale or the Dataset that refers to a Dimension Scale.

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. This proposal leaves it to applications to enforce a policy of uniqueness when they need it.

Can a dimension have a label, without having an associated scale? Some applications may wish to name dimensions without having an associated scale. Therefore, a dataset may have a label for a dimension without having an associated Dimension Scale dataset.

Can a dimension have a scale, without having an associated label? Some applications may wish to assign a dimension scale with no label. Therefore, a dataset may have one or more Dimension Scales for a dimension without having an associated label.

Anonymous Dimensions. It is possible to have a Dimension Scale dataset with no name, and associate it with a dimension of a dataset with no label. This case associates an array of data values to the dimension, but no identifier.

A dimension with a label and a name. A dimension of a dataset can be associated with a Dimension Scale that has a name, and assigned a label. In this case, the association has two "names", the label and the dimension scale name. It is up to applications to interpret these names.

Table 5 summarizes the six possible combinations of label and name.

Table 5. Labels and scales of a dimension.

	No scale	Scale with no name	Scale with name
No label	Dimension has no label or scale (default)	Dimension has an anonymous scale	Dimension has scale, the scale is called "name"
Label	Dimension has label	Dimension has scale with a label.	Dimension has scale with both a label and name. A shared dimension has one name, but may have several labels

4.4 Shared Dimension Scales

Given the design described above, datasets can share dimension scales. 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.

This proposal defines attributes maintain back pointers along the lines of (a), which enable these kinds of cross referencing. Other software, such as NetCDF4, may well need a global table to track a set of dimensions. Such a table can be done in addition to the attributes defined here.

4.5 Example

This section presents an example to illustrate the data structures defined above.

Figure 3 shows a Dataset with a four dimensional Dataspace. The file also contains six Dimension Scale datasets. The Dimension Scale datasets are HDF5 objects, with path names such as "/DS1".

Figure 4 illustrates the use of dimension scales in this example. Each Dimension Scale Dataset has an optional NAME. For example, "/DS3" has been assigned the name "Scale3".

The dimensions of dataset D have been assigned zero or more scales and labels. Dimension 0 has two scales, Dimension 1 has one scale, and so on. Dimension 2 has no scale associated with it.

Some of the dimensions have labels as well. Note that dimension 2 has a label but no scale, and dimesion 3 has scales but no label.

Some of the Dimension Scales are shared. Dimension Scale DS1 is referenced by dimension 0 of D and by another unspecified dataset. Dimension Scale DS3 is referenced by dimension 1 and 3 of Dataset D.

These relationships are represented in the file by attributes of the Dataset D and the Dimension Scale Datasets. Figure 5 shows the values that are stored for the DIMENSION_LIST attribute of Dataset D. This is a two dimensional array, with some empty values. Table 6 shows the DIMENSION_LABELLIST for Dataset D. This is a one dimensional array with some empty values.

Each of the Dimension Scale Datasets has a name and other attributes. The references are represented in the REFERENCE_LIST attributes. Table 7 – Table 10 show the values for these tables. Note that Dimension Scale DS4 and DS6 have no references to them in this diagram.

We propose to store these tables as attributes of the Dimension Scale Dataset and the Datasets that refer to scales. Essentially, the association between a dimension of a Dataset and a Dimension Scale is represented by "pointers" in both of the associated objects. Since there can be multiple associations, there can be

multiple pointers stored at each object, representing the endpoints of the associations. These will be stored in tables, i.e., as an attribute with an array of values.

When dimension scales are attached or detached, the tables in the Dataset and the Dimension Scale must be updated. The arrays in the attributes can grow, and items can be deleted

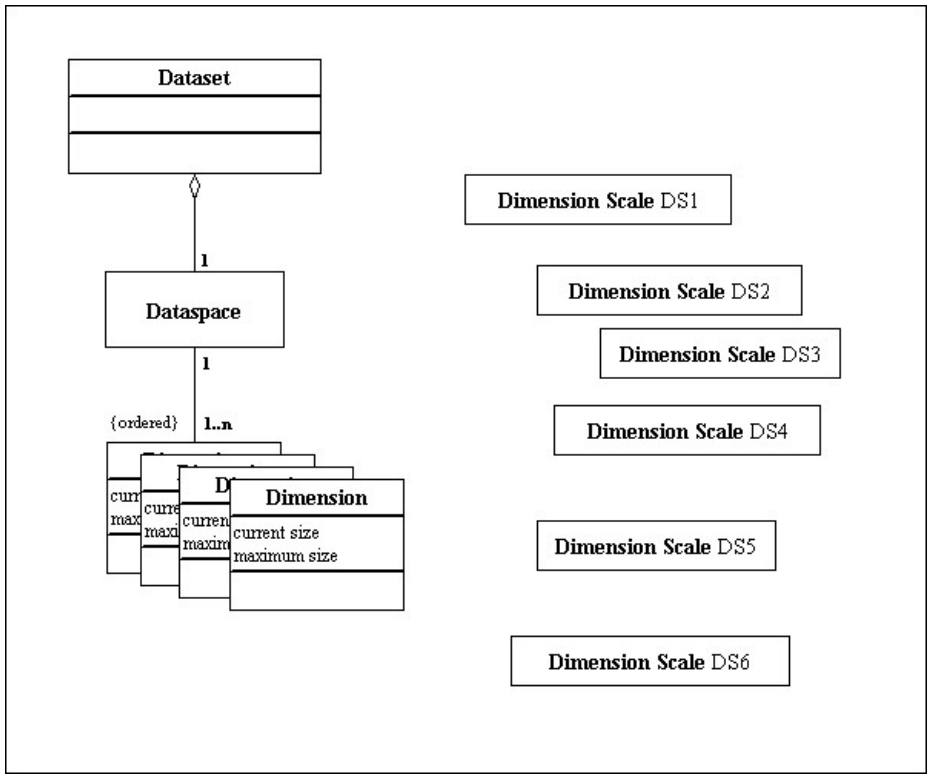

Figure 3. Example dataset and scales.

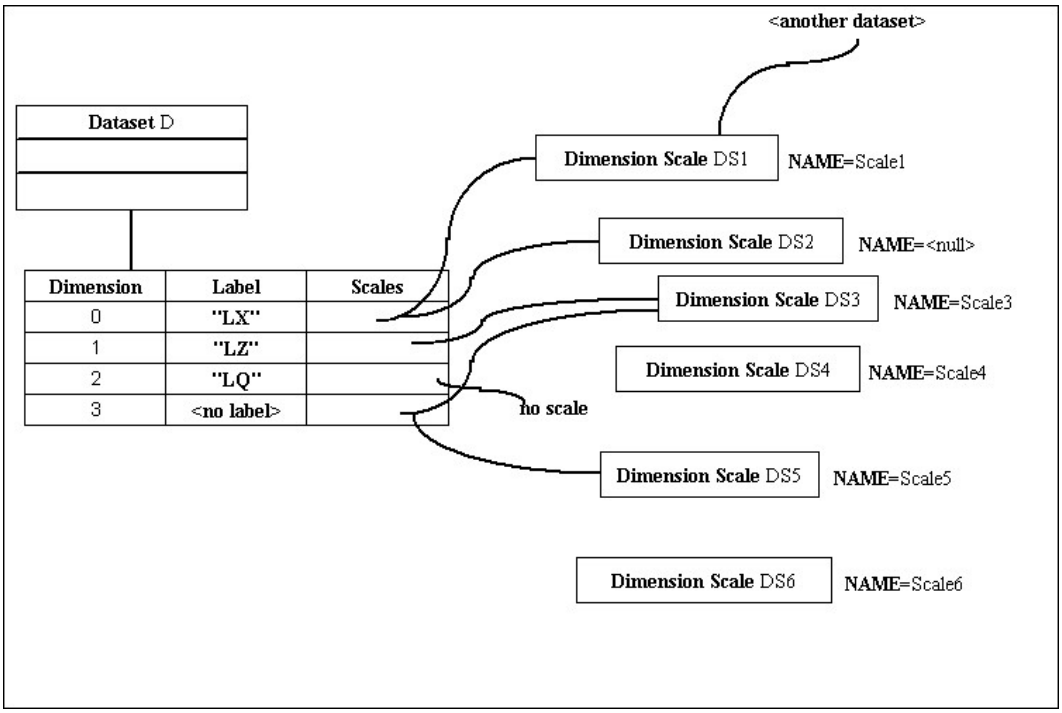

Figure 4. Example labels, names, and attached scales.

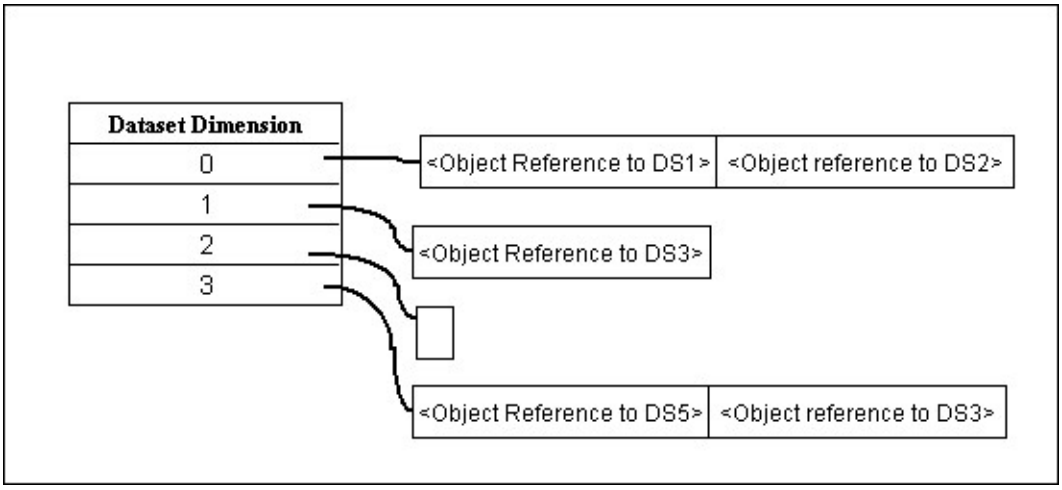

Figure 5. The table of dimension references, stored as an attribute of the Dataset.

Table 6. The table of dimension labels.

Dataset Dimension	Label
0	"LX"
1	"LZ"
2	"LQ"
3	""

Table 7. The reference list for DS1.

Reference	Dataset Reference Record
0	{Object reference to Dataset D, 0, 0}
1	{Object reference to other Dataset, ?, ?}

Table 8. Reference list for DS2

Reference	Dataset Reference Record
0	{Object reference to Dataset D, 0,1}

Table 9. Reference list for DS3

Reference	Dataset Reference Record
0	{Object reference to Dataset D, 1, 0}
1	{Object reference to Dataset D, 3, 1}

Table 10. Reference List for DS5

Reference	Dataset Reference Record
0	{Object reference to Dataset D, 3, 0}

The associations are identified by an index number which is stored in a back pointer and returned from an API. The detach function needs to be careful how it deletes an item from the table, because the entries at both ends of the association must be updated at the same time.

5 Programming Model and API

5.1 Programming Model

{Clean this up…}

Dimension Scales are HDF5 Datasets, so they may be created and accesses through any HDF5 API for datasets [10]. The HDF5 Dimension Scale API implements the specification defined in this document. The operations include:

- 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.
- Iterate through scales of (D, i) – get each scale attached.
- Get info (S) – get info about dimension scale S (existence, size, name, etc.)

This section outlines the programming model for using these operations with some example uses.

5.1.1 Create new Dimension Scale with Initial Values

1. Create dataset with for the Dimension Scale with H5Dcreate and other standard HDF5 calls.
2. Initialize the values of the Dimension Scale with H5Dwrite and other calls.
3. Convert the dataset to a Dimension Scale with H5DSmake_scale.
4. Close the Dimension Scale when finished with H5Dclose.

5.1.2 Attach Dimension Scale to Dataset

1. Create or open the Dataset, D, with H5Dopen, etc.
2. Create or open the Dimension Scale dataset, S, with H5Dopen or as above.
3. Attach the Dimension Scale S to dimension j of Dataset D with H5DSattach_scale
4. When finished, close the Dimension Scale and Dataset with H5Dclose.

5.1.3 Read Dimension Scale values

1. Open the Dataset D, with H5Dopen
2. If necessary, get the number of dimensions, H5Dget_space, etc.
3. For dimension *ix* of D, get the number of scales with H5DSget_num_scales
4. Get the datatype, dataspace, etc. of the Dimension Scale Dataset with H5Dget_space, H5Dget_type, H5Sget_ndims, etc.
5. Read the values of the Dimension Scale into memory with H5Dread, e.g. into dscalebuff.
6. When finished, close the Dimension Scale Dataset and other objects with H5Dclose etc.
7. When finished, close the Dataset D with H5Dclose.

5.1.4 Write or Update Dimension Scale values

1. Open the Dimension Scale Dataset S with H5open
2. Get the datatype, dataspace, etc. of S with H5Dget_space, H5Dget_type, H5Sget_ndims, etc.
3. If needed, read the values of S into memory with H5Dread. Note, may read selected values using a selection
4. Write updated values to S with H5Dwrite. Note, may write selected values using a selection.
5. When finished, close the Dimension Scale Dataset and other objects with H5Dclose etc.

5.1.5 Create a label for a dimension

1. Open the Dataset D with H5open
2. Add write a label for dimension i of D, with H5DSset_label.
3. When finished, close the Dimension Scale Dataset and other objects with H5Dclose etc.

5.1.6 Delete a Dimension Scale:

1. Open the Dimension Scale to be deleted.
2. Read the REFERENCE_LIST attribute into memory with H5Aread etc.
3. For each entry in the list:

a. Dereference the dataset reference
b. Detach the scale with H5DSdetach_scale {????}
c. Close the dataset reference

4. Delete the Dimension Scale Dataset {{ HOW }}}

5.1.7 Clean up Dimension Scales when deleting a Dataset

1. Open the Dataset to be deleted, with H5Dopen.
2. Read the DIMENSION_LIST, if any, with H5Aread etc.
3. For each entry in the dimension list, detach the scale with H5DSdetach_scale
4. Delete the Dataset, with H5Gunlink.

5.1.8 Extending a Dimension with a Dimension Scale attached

When an extendible Dataset has Dimension Scales, it is necessary to coordinate when the dimensions change size.

1. Open the Dataset to be extended, with H5Dopen.
2. Extend the dimension(s) with H5Dextend
3. For each dimension that was extended, get the number of scales with H5DSget_num_scales. For each dimension with scales
   a. Get each scale with H5DSget_scale_id
   b. Extend the scale to the new size of the dimension with H5Dextend
   c. Write new values to the extended scale with H5Dwrite, etc.
   d. Close the Dimension Scale Dataset with H5Dclose
4. When finished, close the Dataset with H5Dclose

5.2 Programming API: H5DS

This section proposes a new programming API, to be added to the HDF5 High Level APIs

---

Name: H5DSmake_scale
Signature:

herr_t H5DSmake_scale(*hid_t dset, char *dimname*)

Purpose:

Convert dataset D to a dimension scale.

Description:

The dataset *dset* is converted to a Dimension Scale dataset, as defined above. Creates the CLASS attribute and an empty DIMENSION_INDEX_LIST.

If *dimname* is specified, then an attribute called NAME is created, with the value *dimname*.

Fails if:

- Bad arguments
- If dset is already a scale
- If dset is a dset which has dimension scales (??)

If the dataset was created with H5TBmake_table, it is not recommended to convert to a Dimension Scale.

Parameters:

hid_t dset; IN: the dataset to be made a Dimension Scale

*char *dimname; IN: the dimension name (optional), NULL if the dimension has no name.*

Returns:

Zero if succeed, negative if fail.

Name: H5DSattach_scale
Signature:

herr_t H5DSattach_scale(*hid_t dset, unsigned ind, hid_t scale*)

Purpose:

Attach dimension scale S to dimension ind of D.

Description:

Define Dimension Scale *scale* to be associated with dimension *ind* of Dataset *dset.*

Entries are created in the DIMENSION_LIST and REFERENCE_LIST attributes, as define in section 4.2.

Fails if:

- Bad args
- If *scale* is not a Dimension Scale
- If Dset is a Dimension Scale (A Dimension Scale cannot have scales.)

Note: The DS can be attached to the same dimension more than once.

Parameters:

hid_t dset; IN: the dataset

unsigned ind; IN: the dimension of dset

hid_t scale; IN: the scale to be attached

Returns:

Zero if succeed, negatiems if fail.

Name: H5DSdetach_scale
Signature:

herr_t H5DSdetach_scale(*hid_t dset, unsigned ind, hid_t scale*)

Purpose:

Detach dimension scale *scale* from the dimension *ind* ofDataset *dset*.

Description:

If possible, deletes the *scale* association of dimension *ind* of Datase *dset*. This deletes the entries in the DIMENSION_LIST and REFERENCE_LIST attributes, as define in section 4.2.

Fails if:

- Bad arguments

Parameters:

hid_t dset; IN: the dataset

unsigned ind; IN: the dimension of dset

hid_t scale; IN: the scale to be attached

Returns:

Zero if succeed, negative if fail.

Name: H5DSinterate_scales
Signature:

herr_t H5DSiterate_scales(*hid_t dset, unsigned dim, int* *idx, *H5DS_iterate_t* visitor, *void* *visitor_data)

Purpose:

Iterates the operation *visitor* through the scales attached to dimension *dim*.

Description:

H5Dsiterate_scales iterates over the scales attached to dimension dim of dataset dset. For each scale in the list, the visitor_data and some additional information, specified below, are passed to the visitor function. The iteration begins with the idx object in the group and the next element to be processed by the operator is returned in idx. If idx is NULL, then the iterator starts at the first group member; since no stopping point is returned in this case, the iterator cannot be restarted if one of the calls to its operator returns non-zero.

The prototype for H5DS_iterate_t is:

typedef *herr_t* (*H5DS_iterate_t)(*hid_t dset, unsigned dim,* hid_t scale, *void* *visitor_data);

The operation receives the Dimension Scale dataset identifier, scale, and the pointer to the operator data passed in to H5Dsiterate_scales, visitor_data.

The return values from an operator are:

- Zero causes the iterator to continue, returning zero when all group members have been processed.
- Positive causes the iterator to immediately return that positive value, indicating short-circuit success. The iterator can be restarted at the next group member.
- Negative causes the iterator to immediately return that value, indicating failure. The iterator can be restarted at the next group member.

H5Dsiterate_scales assumes that the scalse of the dimension identified by dim remains unchanged through the iteration. If the membership changes during the iteration, the function's behavior is undefined.

Parameters:

hid_t dset; IN: the dataset

unsigned dim; IN: the dimension of dset

int *idx; IN/OUT: input the index to start iterating, output the next index to visit. If NULL, start at the first position.

H5DS_iterate_t visitor; IN: the visitor function

void *visitor_data; IN: arbitrary data to pass to the visitor function.

Returns:

Returns the return value of the last operator if it was non-zero, or zero if all scales were processed.

Name: H5DSset_label
Signature:

herr_t H5DSset_label(*hid_t dset, unsigned ind, const char *label*)

Purpose:

Set label for the dimension *ind* of *dset*.

Description:

Sets the DIMENSION_LABEL_LIST for dimension *ind* of dataset *dset*. If the dimension had a label, the new value replaces the old.

Fails if:

- Bad arguments

Parameters:

hid_t dset; IN: the dataset

unsigned ind; IN: the dimension

*char *label: IN: the label*

Returns:

Zero if succeed, negative if fail.

Name: H5DSget_label
Signature:

ssize_t H5DSget_label(*hid_t dset, unsigned ind, char *label, size_t *labellen*)

Purpose:

Read the label for dimension *ind* of *dset*.

Description:

Returns the value of the DIMENSION_LABEL_LIST for dimension *ind* of dataset *dset*, if set. Up to *labellen* characters of the name are copied into the buffer label. The parameter *labellen* is set to the size of the returned *label*.

If *dset* has no label, the return value of *label* is NULL.

Fails if

- Bad arguments

Parameters:

hid_t dset; IN: the dataset

unsigned ind; IN: the dimension

*char *label: OUT: the label*

size_t labellen: IN/OUT: the length of the buffer label;, on return, the size of the label.

Returns:

Zero if succeed, negative if fail.

Name: H5DSdelete_label
Signature:

herr_t H5DSdelete_label(*hid_t dset, unsigned ind*)

Purpose:

Remove label from dimension *ind* of *dset*.

Fails if

- Bad arguments

Parameters:

hid_t dset; IN: the dataset

unsigned ind; IN: the dimension

Returns:

Zero if succeed, negative if fail.

Name: H5DSget_num_scales
Signature:

herr_t H5DSget_num_scales(*hid_t dset, unsigned ind, unsigned *nscales*)

Purpose:

Get the number of scales associated with dimension *ind* of dataset *dset*.

Description:

Return the number of entries in DIMENSION_LIST for dimension *ind*.

Parameters:

hid_t dset; IN: the dataset

unsigned ind; IN: the dimension

*unsigned *nscales; OUT: the number of scales associated with dimension ind.*

Returns:

Zero if succeed, negative if fail.

Name: H5DSget_scale_by_id
Signature:

hid_t H5DSget_scale_by_id(*hid_t dset, unsigned ind, unsigned scaleid*)

Purpose:

Get the *hid_t* for scale *scaleid* associated with dimension *ind* dataset *dset*.

{??? Delete this function???}

Description:

Returns the id for scale *scaleid* of dimension *ind* dataset *dset*. If the scale doesn't exist, returns NULL.

Parameters:

hid_t dset; IN: the dataset

int ind; IN: the dimension

int scaleid: IN: the index of the scale

Returns:

The open hid_t for the Dimension Scale dataset, or NULL.

Name: H5DSget_scale_name
Signature:

herr_t H5DSget_scale_name(*hid_t scale, char *name, size_t *namelen*)

Purpose:

Read the name of scale *scale*.

Description:

Read the value of the NAME attribute for scale *scaleid*. Reads up to *namelen* characters into *name*. On return, *namelen* is the number of characters read into *name*.

Parameters:

int scaleid: IN: the index of the scale

*char *name: OUT: the name, or NULL if there is no name set.*

*size_t *namelen: IN/OUT: the maximum characters to copy into name. On return, the number of characters copied into name.*

Returns:

Zero if succeed, negative if fail.

---

6 Other Language Interfaces

This document presents the C API. Fortran, C++, and other language interfaces will be implemented later.

7 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.
5. Folk, Mike. "HDF5 Dimension Scale Proposal No. 3 Draft Version 3", September 15, 2004. . http://hdf.ncsa.uiuc.edu/RFC/H5DimScales/H5dimscale_prop_No3.v3.pdf
6. Mike Folk, Robert E. McGrath, Kent Yang, "Mapping HDF4 Objects to HDF5 Objects Version 3" August, 2003. http://hdf.ncsa.uiuc.edu/HDF5/doc/ADGuide/H4toH5Mapping.pdf
7. Unidata, "Shared Dimensions in NetCDF-4", http://my.unidata.ucar.edu/content/staff/russ/shared-dimensions.html.
8. NASA, "The HDF-EOS Information Center", http://hdfeos.gsfc.nasa.gov/hdfeos/index.cfm
9. HDF, "HDF5 High Level APIs", http://hdf.ncsa.uiuc.edu/HDF5/hdf5_hl/.
10. HDF, "HDF5 User's Guide", September 2004. http://hdf.ncsa.uiuc.edu/HDF5/doc/UG/.
11. HDF, "HDF5 Abstract Data Model", 1999. http://hdf.ncsa.uiuc.edu/HDF5/papers/presentations/ADM/ADM_EOS_Sep99/EOSpresentation/index.html
12. HDF, "Expanding raw data options: "formula" datasets and attributes: Request for Comment", To appear, 2005.
13. Nancy Yeager, "Design of NetCDF-H5 Prototype", May, 1999. http://hdf.ncsa.uiuc.edu/apps/netcdfh5/design.html.

8 Appendix 1: Implications for tools

The introduction of Dimension Scales will require that tools using HDF5 add new features to correctly use Dimension Scales.

Since the Dimension Scales and associations are represented by Datasets, object references, and Attributes; HDF5 tools will "see" these objects and treat them as regular objects. This will omit the meaning of the Dimension Scales, and may be quite puzzling for users.

Therefore, tools should be updated to recognize Dimension Scales, display and process them appropriately.

Tool	Changes
H5dump, H5ls	TBD
H5DFView	Do not display Dimensions as regular datasets. When dimension are present use as labels for Data table and image displays. Will need to deal with dimension names, labels, and possibly with editing features (add dimension, change label, etc.)
H5diff	Should do a logical comparison, i.e., Datasets that have the same dimensions should be "the same", even though the object references differ. Dimensions must have same path name, same name, etc. to be same dimension. Need to define the details of how to deal with order of datasets, how to report differences in names or labels. Algorithm is TBD.
H5repack	Should do a logical copy of the dimensions, and then attach them to the datasets appropriately. Need to decide about "compaction" policies and possibly renumbering the indexes.

9 Appendix 2: HDF4 to HDF5 Mapping

The HDF4 to HDF5 Mapping V3 defined an initial representation of HDF4 Dimension Scales in a converted HDF5 file [8]. This specification differs in some details, but HDF4 Dimension Scales can be converted into objects that conform to this specification. Essentially, HDF4 is a special case, because a Dataset can have only one Dimension Scale per dimension.

The HDF4 ot HDF5 Mapping and h4h5 software will be updated to conform to this Dimension Scale specification.

	<SDS Dimension with Name> (See note 1)	Dataset		
R	**NAME**	Name	<name>	See note 1.
R	**SIZE**	Dataspace		rank=1. Only the first dimension can be unlimited.
R	**DATATYPE**	Datatype	<HDF4 datatype>	
R	**DATA**	Data	<value>*	
O	<Dimension pre-defined attribute>			These are dimensions of an <SDS dimension with Name> dataset, not an SDS dataset
O	***LONGNAME**	Attr		
O	**UNIT**	Attr		
O	**FORMAT**	Attr		
O	<User-defined attribute>	Attr		Defined above.
O	****NAME**	Attr	<value of LONGNAME, if set>	
R	****CLASS**	Attr	"DIMENSION_SCALE"	
R	****REFERENCE_LIST**	Attr	Array of Dimension Record Type	Required if the dimension is used by any datasets.

Notes:

* Redundant with new **NAME** attribute. This attribute should be set for backward compatibility.

** New attributes to conform to the Dimension Scale Specification.

	<SDSArray>			
R	**NAME**	Attr	HDF4_OBJECT_NAME = <SDSArrayName>	See Section 4 for details on how **NAME** is used as a link in HDF5.
R	**DATATYPE**	Datatype	<HDF4 datatype>	
R	**DIMENSIONRANK & DIMENSIONSIZE**	Dataspace		Dimension sizes are also part of dimension information.
R	***DIMENSIONLIST**	Attr	DIMENSIONLIST = {object__ref1, object__ref2, … object__refn}	An array of object references that refer to the corresponding. dimension datasets.
R	****DIMENSION_NAMELIST**	Attr	DIMENSION_NAMELIST = {<DimName1, <DimName2>, …,<DimNameN>}	The absolute paths of dimensions are stored. Dimension names are defined in the HCR specification. See note 1.
O	*****DIMENSION_LABELLIST**	Attr		Same as list DIMENSION_NAMELIST, follows the DimensionScalespecification
R	(Data)	Data		See [8] for details on how to handle datatypes.
O	<User-defined attribute >	Attr		rank = 1; size is fixed. Global attributes: see note 2.
O	<SDS pre-defined attribute >	Attr		
O	(Reference number)	Attr	HDF4_REF_NUM = <uint16>	
O	(Storage properties)			
O	Compression property	Storage prop		Use if supported in HDF5.
O	Chunk property	Storage prop		Use if supported in HDF5.
O	External storage	Storage prop		Use if supported in HDF5.

Notes:

* DIMENSION_LIST was a one dimensional array. It is now a special case of a two D array.

** Redundant with the DIMENSION_LABEL_LIST attribute, should be stored for backward compatibility.

*** New attribute to conform to the Dimension Scale specification.

10 Appendix 3: netCDF-4 Dimensions

An important use of Dimension scales will be to represent coordinate variables in netCDF-4. It should be realized that this profile does not support all of the semantics of netCDF-4 dimensions. However, it is general enough to be used to store coordinate variables, and to store the associations between dimensions of a netCDF variable and coordinate variables [7].

HDF5 Dimension Scales can be used:

- A coordinate variable is stored as an HDF5 Dimension Scale (in the appropriate group defined by netCDF-4).
- The attribute “NAME” can hold a netCDF-4 defined name
- The attribute “SUB_CLASS” can be used to store a string indicating that this is a netCDF –4 coordinate variable. (There is no need to code this in the name of the dataset.)

- When a netCDF-4 Variable is created, the relevant coordinate variables should be attached.
- When a dimension is extended, the attached coordinate variables should be extended as well. An algorithm for this was sketched in Section 5.1.8 above, netCDF-4 should implement an appropriate variation of this approach.

Note that these operations can use the standard HDF5 APIs or can use the API proposed here.

HDF5 Dimension Scales do not implement several aspects of netCDF-4 dimensions.

- This specification does not enforce any rules about the names of Dimension Scales or where they are placed in the file. These rules will be enforced by the netCDF-4 library.
- This specification does not maintain the consistency of the extents of Dimension Scales and associated dimensions. However, this can be implemented.
- This specification does not provide a mechanism for finding all the Dimension Scales.

These and other semantics may require additional data structures in the netCDF-4 file. This specification certainly does not preclude the use of such data structures.

HDF5 Dimension Scales provide some features not needed by netCDF-4. These can be ignored.

- NetCDF-4 may choose not to use labels or dimension names.
- NetCDF-4 may choose not to have multiple Dimension Scales for a give dimension of a variable.

In summary, this specification provides constructs that can and should be used to implement netCDF-4 Coordinate Variables.

11 Appendix 4: HDF-EOS5 Issues

The HDF-EOS5 library implements a model of dimensions similar to HDF-EOS2 and HDF4 [8]. For example, the HDF-EOS library API has several functions that create dimensions global to the file, and assign dimensions to define the shape of a Grid.

These functions can be modified to create an HDF5 Dimension Scale object for each scale, perhaps in a reserved directory. When the Grid is created, the related scales would be attached to the Datasets representing the fields of the Grid. (There is one HDF5 Dataset per HDF-EOS Grid Field.)

The rest of the HDF-EOS library would not need to change, although other changes could be made to use the HDF5 attributes in the HDF-EOS code.

The advantage of adding this to the HDF-EOS library is that applications that do not use HDF-EOS but do use the HDF5 Dimension Scales would be able to locate and interpret the dimension scales in the HDF-EOS file.