Mapping HDF4 Objects to HDF5 Objects

Mike Folk, Robert E. McGrath, Kent Yang
National Center for Supercomputing Applications, University of Illinois

February, 2000
Revised: October, 2000, August 2001

Note to reader: We present here some guidelines on how to represent HDF4 objects in HDF5 and how to interpret HDF5 objects as HDF4 objects. It is meant to help in implementing software that has to deal with both formats in some consistent way, such as converting HDF4 files to HDF5, or adapting HDF4 tools to HDF5. Please send comments and corrections to hdfhelp@ncsa.uiuc.edu.

1 Introduction

All versions of NCSA HDF from HDF1 through HDF4 are essentially the same. The HDF4 format and library are backward compatible with all earlier versions of HDF. HDF5 is different. Although it shares many features with earlier versions of HDF and is intended for essentially the same uses, HDF5 is a completely new file format, and the NCSA HDF5 API and library are also new and entirely different.

Many applications have been written for accessing, visualizing, and otherwise dealing with HDF4 objects and files. Few have as yet been written for HDF5. A great deal of development time and expense could be saved if some HDF4 applications could be adapted for dealing with HDF5 objects. The purpose of this paper is to facilitate such adaptations by establishing standard ways to (a) represent HDF4 objects in HDF5, and (b) interpret HDF5 objects *as* HDF4 objects.

Case (a) assumes that an application writes an HDF5 object intending for the object to be understood as a particular HDF4 object. It may add extra attributes to the object to make it conform as fully as possible to the corresponding HDF4 data model. Case (b) assumes that the HDF5 object was not created with HDF4 in mind, but nevertheless conforms to one or more HDF4 objects. (In the context of this paper, the term "conform" means that the characteristics of the HDF5 object are such that it would be meaningful and useful to an HDF4 application. It does not mean that the HDF5 object is exactly the same as a corresponding HDF4 object would be.)

It is not our intention to map all possible HDF4 objects into HDF5, and vice versa. In section 2 we identify those HDF4 and HDF5 objects that will be mapped.

It is also not our intention to cover all possible cases for mapping HDF4 to HDF5. This is a specification of recommended default mappings only. If a different mapping is more appropriate for a particular application, then it should be used.

Section 2 describes the types of HDF4 and HDF5 objects that can be mapped, and identifies those for which mappings are not recommended.

Sections 3-5 cover case (a), how to convert HDF4 files to HDF5. Section 3 recommends ways to explicitly represent HDF4 objects in HDF5. It describes the basic mappings that are recommended, and presents a set of rules to instantiate the mappings, including metadata attributes that make an object conform as fully as possible to the corresponding HDF4 data model.

Section 4 covers issues related to overall file organization, including how to map the organization and name structure of an HDF4 file into HDF5.

Section 5 covers other considerations, such as how to deal the HDF4 reference numbers and file-level information.

Section 6 deals with case (b), how to interpret HDF5 objects as if they were HDF4 objects when there is no explicit metadata.

This document relies on a number of documents, which the reader may need to refer to for background concerning HDF4 and HDF5 datatypes. This includes the HDF4 User's Guide, the HDF4 Specification and the HDF5 documentation cited in the reference section at the end of this document [2][3]. Also cited in the Reference section is the "HDF Configuration Record" specification [1], which provides a rigorous definition of all HDF4 objects.

2 HDF4 objects and HDF5 objects

The HDF4 format and library support the following eight basic objects:

- Scientific dataset (SDS), a multidimensional array with dimension scales
- 8-bit raster image (RIS8), a 2-dimensional array of 8-bit pixels
- 24-bit raster image (RIS24), a 2-dimensional array of 24-bit pixels
- General raster image (GR), a 2-dimensional array of multi-component pixels
- 8-bit color lookup table (palette), a 256 by 3 array of 8-bit integers
- Table (Vdata), a sequence of records
- Annotation, a stream of text that can be attached to any object
- Group, a structure for grouping objects

The HDF4 format also includes "primitive" objects that are used to construct these basic objects within an HDF4 file. These primitive objects are identified by "tags" within an HDF4 file. Since most HDF4 primitive objects have no counterpart in HDF5, nor are they accessed directly by most HDF4 users or applications, they will not be considered here. Exceptions to this are the HDF4 palette and annotation, which will be considered.

HDF5 includes two primary objects:

- Dataset, a multidimensional array of records
- Group, a structure for grouping objects

HDF5 objects can have "attributes", which are (usually) small, named datasets that are associated with groups or datasets. HDF5 includes other objects, such as named datatypes, but these have no counterparts in HDF4 and hence will not be considered here.

3 Representing HDF4 objects in HDF5

In this section we provide detailed rules for representing HDF4 objects in HDF5. All eight basic HDF4 objects can be represented in HDF5. Usually such representations require restrictions or extra metadata. In Table 1, a mapping is shown from HDF4 objects to their HDF5 counterparts.

As indicated in Table 1, in all cases except Vgroups and annotations HDF4 objects are mapped to HDF5 datasets with simple dataspaces. Vgroups are mapped to HDF5 groups, and annotations are mapped to HDF5 attributes. In the tables that follow we identify all components of an HDF4 object that an application is likely to use, and map it to a corresponding HDF5 component. This mapping includes only persistent objects and components. Items that are available only when accessing HDF4 files (e.g. file id and object index) are omitted.

All of the HDF5 objects except annotations have the following two optional attributes: HDF4_OBJECT_TYPE and HDF4_REF_NUM. HDF4_OBJECT_TYPE can be used to tell applications that the object is compatible with an HDF4 object. HDF4_REF_NUM is available for those applications that use reference numbers as identifiers for HDF4 objects.

Table 1. Representing HDF4 objects in HDF5.

HDF4 object	Corres-ponding HDF5 object	Restrictions
SDS	Dataset	HDF4 DIMENSIONLIST becomes an HDF5 attribute, if it exists. HDF4 dimension scales become HDF5 datasets. Only the first dimension can be unlimited. Not all HDF4 storage properties are supported.
Image	Dataset	The HDF5 dataset must be 2-dimensional. If the number of pixel components is 1, an HDF5 scalar datatype is used, otherwise a compound type is used. If a palette is present, HDF5 attributes are used to indicate this. Not all HDF4 storage properties are supported.
Palette	Dataset	The HDF5 dataset must be a 256 by 3 array of 8-bit integers. HDF5 attributes describe this dataset as a standard 8-bit palette.
Vdata (table)	Dataset	The HDF5 dataset must be 1-dimensonal, with a compound datatype equivalent to corresponding HDF4 field and record structure. Non-interleaved fields are not permitted in HDF5. (This last restriction could be lifted if a structure is created to store fields as separate datasets.)
Annotation	Attribute	HDF4 *file* annotations are attributes of the HDF5 root group. HDF4 *object* annotations are attributes of the corresponding HDF5 object. Only annotations on the HDF4 objects listed here are supported.
Vgroup	Group	

The mapping tables. In the following sub-sections, each of the six mappings from table 1 is described in detail with a table containing five columns:

- Column 1: a flag indicating whether the object is required ("R") in HDF5 in order for the object to conform to the corresponding HDF4 object. "O" (optional) means that it is not required.
- Column 2: components from HDF4 that are to be mapped to HDF5. Items with HDF4 names are in bold caps. Items in parentheses refer to information that is needed in the HDF5 version but do not have an HDF4 counterpart.
- Column 3: the HDF5 object that is mapped to.
- Column 4: information about the datatype, value, etc. of the HDF5 object.
- Column 5: additional information on how to perform the mapping.

The HCR definition of HDF4 was used to identify the HDF4 items that are to be mapped [1]. In the tables, we have tried to used the HCR terminology whenever possible. For instance, in column 2, DATATYPE refers to an HDF4 datatype. Non-terminals are shown in angle brackets (e.g. <name>). Most non-terminals are defined in the HCR documentation. Others that are used are:

- <string>: any legal quoted string
- <name>: any valid name
- <value>: any valid scalar value
- <HDF4 datatype>: any valid HDF4 datatype.
- <uint16>: a value of type DFNT_UINT16

3.1 SDS

The SDS mapping requires two types of HDF5 dataset, one for the SDS array and one for dimension scales. For each dimension, the creator of the HDF5 “SDS” must create a corresponding dimension dataset with a unique name. In the HDF5 “SDS” an attribute “DIMENSIONLIST” is created consisting of pointers to each of the respective dimension datasets and an attribute “DIMENSION_NAMELIST” with the full paths of the HDF5 dimension datasets.

Table 2. SDS mapping

	HDF4 object	HDF5 object or component	Datatype, value, etc.	Notes
R	<SDSArray>	Dataset		Objects with unlimited dimensions are stored using chunked storage.
O	<SDS Dimension with Name>	Dataset		(See note 1)
O	(HDF4 object type)	Attr	HDF4_OBJECT_TYPE = "SDS"	

	<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. See also section 5.4.
R	**DIMENSIONLIST**	Attr	DIMENSIONLIST = {object__ref1, object__ref2, … object__refn}	An array of object references that refer to the corresponding dimension datasets. See note 1.
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.
R	(Data)	Data		See section 5.5 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.

O	<User-defined attribute >	Attr	<name> = <value>*	rank = 1; size is fixed. Global attributes are covered below.
R	**NAME**	Attr name	<AttributeName>	
R	**DATATYPE**	Datatype	<Attributetype>	
R	**N_VALUES**	Num-values	<AttributeCount>	
R	**DATA**	Data	<AttributeData>	

O	<SDS pre-defined attribute >			Same names, datatypes, etc., as the hdf4 counterpart
O	**LONGNAME**	Attr	.	
O	**UNIT**	Attr		
O	**FORMAT**	Attr		
O	**COORDINATE_SYSTEM**	Attr		
O	**RANGE**	Attr		
O	**FILL_VALUE**	Attr	Set the fill value property to <FillValue>	See note 3.
O	**SCALE_FACTOR**	Attr		
O	**SCALE_FACTOR_ERROR**	Attr		
O	**ADD_OFFSET**	Attr		
O	**ADD_OFFSET_ERROR**	Attr		
O	**CALIBRATED_NT**	Attr		

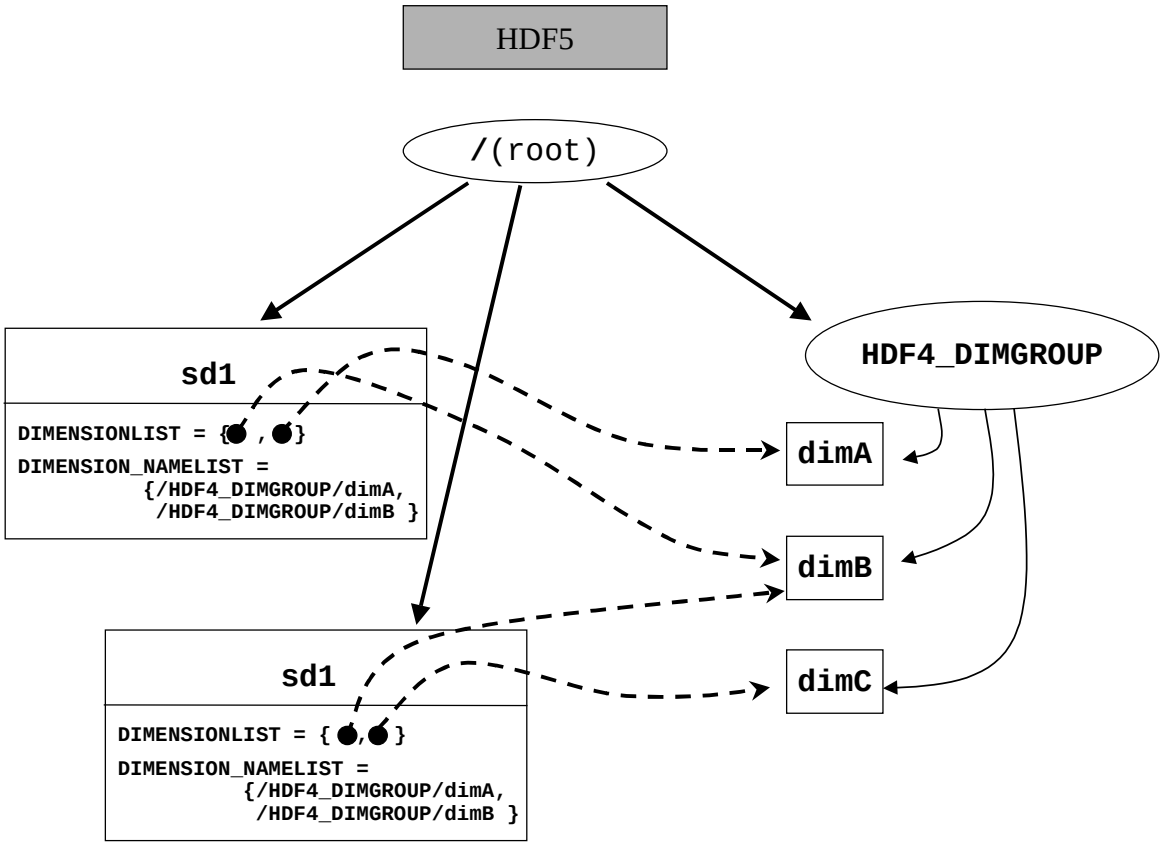
	<SDS Dimension with Name>	Dataset		
R	**NAME**	Name	<name>	See note 2.
R	**SIZE**	Dataspace		rank=1. Only the first dimension can be unlimited.
R	**DATATYPE**	Datatype	<HDF4 datatype>	
R	**DATA**	Data	<value>*	
O	<Dimension pre-defined attribute>			These are dimensions of an <SDS dimension with Name> dataset, not an SDS dataset
O	**LONGNAME**	Attr		
O	**UNIT**	Attr		
O	**FORMAT**	Attr		
O	<User-defined attribute>	Attr		Defined above.

Note 1. Dimension scales are to be stored in HDF5 as separate datasets. Hence, all of the information in this category is stored as part of the corresponding HDF5 dimension scale dataset. See section 3.1.1. for a detailed explanation.

Note 2. Global SDS attributes should be stored as attributes to the HDF5 root group.

Note 3. It is "semantically correct" to write out all the values of a HDF5 dataset, even if they were fill-values not stored in the HDF4 file. This will make the stored file much larger than the HDF4 file. For efficiency, it is recommended that fill values be used in HDF5 whenever they are used in HDF4. See section 3.1.1 for a detailed explanation.

Figure 1. How HDF4 dimension Scales should be stored in HDF5.



SDS global attributes

A special case is the SDS global attribute, which is an attribute that applies to all of the SDS in a file. An SDS global attribute is to be stored in HDF5 as an attribute on the root group with the suffix "GLO_SDS." Other than these differences, an SDS global attribute is treated the same as a normal SDS attribute.

Table 3. SDS global attribute mapping

	<SDS global attribute>	**HDF5 object or component**	**Datatype, value, etc.**	**Notes**.
R	**NAME**	Attr name	<AttributeName>_GLO_SDS	Attribute assigned to the root group.
R	**DATATYPE**	Datatype	<Attributetype>	Attribute assigned to the root group.
R	**N_VALUES**	Num-values	<AttributeCount>	Attribute assigned to the root group.
R	**DATA**	Data	<AttributeData>	Attribute assigned to the root group.

3.1.1. Storage of SDS as an HDF5 Dataset

SDS Dimensions and the HDF5 Dataspace

The dataspace for the HDF5 dataset should be constructed with dimensions that correspond to the dimensions of the SDS in HDF4, as reported by Sdgetinfo(). The last dimension is the fastest changing dimension in both cases. Figure 2 shows an example of C code to read the dimensions of an HDF4 SDS and create the correct HDF5 dataspace. It is important to note that the correct mapping for images is not simply a copy of the dimensions. See section 3.4 below.

Figure 2. Example code to create an HDF5 dataspace for and HDF5 SDS.

```
int data[X][Y];  /* Y is the fastest changing dimension  */
SDgetinfo(..., dims[], ...);   /* dims[0] == X; dims[1] == Y; */
/*  To create the correct HDF5 data set, declare the data space as follows,
   using the values of 'dims' returned above */
  H5Screate_simple(...,dims[], ..);
```

Dimension Scales[*]

Dimension scales are unique to the HDF4 file—no two dimension scales can have the same name in HDF4. In HDF5, all of the dimension scales are stored in the group called "`/HDF4_DIMGROUP`". Dimension names are <DimName1>, <DimName2>, …,<DimNameN> as defined in HCR. Dimension scale datasets are identified in the HDF5 SDS dataset in two ways: by the attribute DIMENSIONLIST, and by the attribute DIMENSION_NAMELIST. The attribute DIMENSIONLIST is an array of object references to the dimension scale datasets `DIMENSION_NAMELIST` is an array containing the names of the dimension scale datasets. Figure 1 illustrates this structure. In the figure there are two two-dimensional datasets, sd1 and sd2, each of which has two dimension scales. The datasets share the dimension scale `dimB`.

Uninitialized Data

In general, all data from an SDS should be written to the HDF5 dataset. If an SDS has no data in it, or only part of the data has been written, then the HDF5 dataset may optionally contain only the data actually written to the SDS. Data from the SDS that was never written will be read as fill values. In all cases, it is correct to write all the data values to the HDF5 file including the fill values. This can result in the HDF5 dataset using much more storage than the HDF4 SDS, because fill values are unnecessarily written to the file. For this reason, it is recommended that data that was not written to the SDS (i.e., fill values) should not be written to the HDF5 dataset if it can be avoided.

If the HDF4 SDS was created but has never had any data written the HDF5 dataset may be created with no data.

If the HDF4 SDS is chunked and some of the chunks have never been written, only the chunks that have been written need to be written to the HDF5 dataset. Chunks that have never been written (and would be entirely filled with fill values) may be written to the HDF5 dataset, or may be left empty.

However, in all cases it is correct to write all the data values to the HDF5 file including fill values.

SDS Storage Properties

In general, the HDF5 dataset is correct if it contains all the correct data values from the HDF4 SDS. However, if chunking and compression are used in HDF4 and not in HDF5, the HDF5 dataset may use much more storage and perform much poorer than the HDF4 SDS. For these reasons, the HDF5 dataset should use storage properties analogous to the HDF4 SDS, when possible.

The follow rules are recommended:

- If the HDF4 SDS is chunked, the HDF5 dataset should be chunked with the same chunk sizes
- If the HDF4 SDS is compressed, the HDF5 dataset should be compressed with the same compression method, if available.

[*] A specification and API for dimension scales is under development for HDF5. This specification will be modified in the future as needed.

- If the HDF4 SDS is compressed by any method not available with HDF5, the HDF5 dataset should be compressed with GZIP.

If the HDF4 SDS is compressed but not chunked, the HDF5 dataset should be "chunked" with the chunksize equal to the entire dataset. The HDF5 dataset should be compressed.

Table 4. Recommended conversion of chunking and compression

HDF4	HDF5
No chunking, No compression	No chunking, No compression
Chunking, No compression	Same chunking, no compression
Chunking, compression • GZIP compression • Other compression	Same chunking, same compression if available. • GZIP • GZIP
No chunking, compression • GZIP • Other compression	Chunk with chunk size = whole dataset, same compression if available. • GZIP • GZIP

3.2 Vdata

Vdatas are mapped to one-dimensional, extendable HDF5 datasets of compound datatype.

Table 5. Vdata mapping

	Vdata	HDF5 object or component	Datatype, value, etc.	Notes
R	**NAME**	Attr	HDF4_OBJECT_NAME = <string>	<string> is the same as the corresponding Vdata name
R	**CLASS**	Attr	HDF4_VDATA_CLASS = <string>	
O	**INTERLACEMODE**	NA		Full interlace always used in HDF5 version.
R	(Number of records)	Dataspace		Rank=1, curr_size = the number of records, maximum size=unlimited.
R	(Record)	Compound datatype		
R	(Field)	Member		Compound datatype member
R	**NAME**	Field name	<name>	
R	**DATATYPE**	Field datatype	<HDF4 datatype>	
R	**ORDER**	Num-values		
O	<User-defined attribute>	Attr	<FieldName>:<name> = <value>	<FieldName> is value of **NAME** for the field.
R	(Data)	Data		
O	<User-defined attribute >	Attr	<name> = <value>	rank = 1; size is fixed;
O	(HDF4 object type)	Attr	HDF4_OBJECT_TYPE = "Vdata"	
O	(Reference number)	Attr	HDF4_REF_NUM = <uint16>	
O	(External storage)	Storage prop		

3.3 Vgroup

Vgroups are mapped individually to HDF5 groups. See section 4 for details on how to deal with the graph structures defined by the collection of Vgroups within an HDF4 file.

Table 6. Vgroup mapping.

	Vgroup	HDF5 object or component	Datatype, value, etc.	Notes
R	**NAME**	Attr	HDF4_OBJECT_NAME = <string>	<string> is the same as a normal Vgroup name
O	**CLASS**	Attr	HDF4_VGROUP_CLASS = <string>	
R	<Vgroup member>	Group member		HDF5 hard link
O	(HDF4 object type)	Attr	HDF4_OBJECT_TYPE=" Vgroup"	
O	<User-defined attribute>	Attr	<name> = <value>	rank = 1; size is fixed;
O	(Reference number)	Attr	REF_NUM = <uint16>	

Note: HCR defines three additional items: MEMBERTYPE, MEMBERNAME, and PALETTEINDEX. MEMBERTYPE and MEMBERNAME correspond to the HDF4_OBJECT_TYPE and HDF4_OBJECT_NAME of the member object. The PALETTEINDEX would be awkward to represent these in HDF5 groups, and hence they have been omitted. If it is found that they are needed, they will be added later.

3.4 Raster images

Raster images (8-bit, 24-bit, and general raster (GR)) are mapped to HDF5 datasets with simple 2D dataspaces. Each element of the dataset is a one-dimensional array of pixel components. The HDF5 Image should conform to the *HDF5 Image and Palette Specification*[4]. All attributes required by the specification should be included, even if not specified here. Notice that the HDF5 image conventions support additional information that is not supported in HDF4, such as image transparency. Table 7 shows the mapping for images.

GR global attributes

A special case is the GR global attribute, which is an attribute that applies to all of the GR in a file. A GR global attribute is to be stored in HDF5 as an attribute on the root group with the suffix "GLO_GR." Other than these differences, a GR global attribute is treated the same as a normal GR attribute.

3.4.1. Storage of GR images in HDF5

RI and GR Dimensions and the HDF5 Dataspace

The dataspace for the HDF5 dataset should be constructed with dimensions that correspond to the actual stored dimensions of the RI or GR in HDF4, which is the same order for all HDF5 datasets, including those converted from HDF4 SDSs. The last dimension is the fastest changing dimension in both cases. It is very important to note that the correct order for HDF5 is *not* the order of dimensions reported by the DFR8getdims() or GRgetiminfo(). Figure 4 and Figure 5 show C code to read HDF5 RI and GR image dimensions and create the correct HDF5 dataspace.

Also note that the HDF4 calls SDgetinfo() and GRgetiminfo() calls look similar, but the dimensions returned must be used differently. See section 3.1 above.

Table 7. Raster image mapping.

	Image	HDF5 object or component	Datatype, value, etc.	Notes
R	**NAME**	Attr	HDF4_OBJECT_NAME = <string>	<name> is the same as a GR name
R	(Pixel type)	Datatype		If N_COMPS=1, use atomic, else compound with 1 field.
R	**N_COMPS**	Num values		order of field, if N_COMPS > 1
R	**COMP_TYPE**	Atomic type	<HDF4 datatype>	
R	**DIMENSIONSIZE**	Dataspace		rank=2. See section 3.4.1 and section 5.4.
R	(image array)	data		
O	<User-defined attribute>	Attr	<name> = <value>	rank = 1; size is fixed;
R	(Class)	Attr	CLASS = "IMAGE"	Required by HDF5 image spec.
O	(HDF4 object type)	Attr	HDF4_OBJECT_TYPE = "raster8", "raster24" or "GR"	
O	(Reference number)	Attr	HDF4_REF_NUM = <uint16>	
R[3]	<Image palette>		PALETTE = {object__ref1, object__ref2, … object__refn}	PALETTE is an array of object references that refers to the corresponding palettes. See notes 1 and 2.
R[2]	(palette namelist)	Attr	PALETTE_NAMELIST = {palette_name1, palette_name2, …}	A list of the full path names of the HDF5 Palette datasets.
R[2]	(Image subclass)	Attr	IMAGE_SUBCLASS = "IMAGE_INDEXED"	See note 2.
R[3]	**INTERLACEMODE**	Attr	INTERLACE_MODE = ”INTERLACE_PIXEL” or “INTERLACE_PLANE”	For images with multiple components (e.g., RI24) the interlace mode should be the same as in theHDF4 image. See note 3 and section 3.4.1.
O	(Storage properties)			
O	Compression	Storage prop		If supported in HDF5. JPEG and RLE are not supported in HDF5.
O	Chunking	Storage prop		
O	External storage	Storage prop		

Note 1. In HDF5 there can be more than one palette in PALETTELIST, but in a translation from HDF4 to HDF5 there will be only one palette. In HDF5, palettes are stored in a special group called “/HDF4_PALGROUP”. Figure 3 illustrates this structure. In the figure there are two images, `image1` and `image2`, each of which has an attached palette.

Note 2. These are required if there is a palette associated with the image.

Note 3. Required for images with more than one component. Optional for images with a single component. See section 3.4.1.

Figure 3. How HDF4 palettes should be stored in HDF5 and referenced by images.

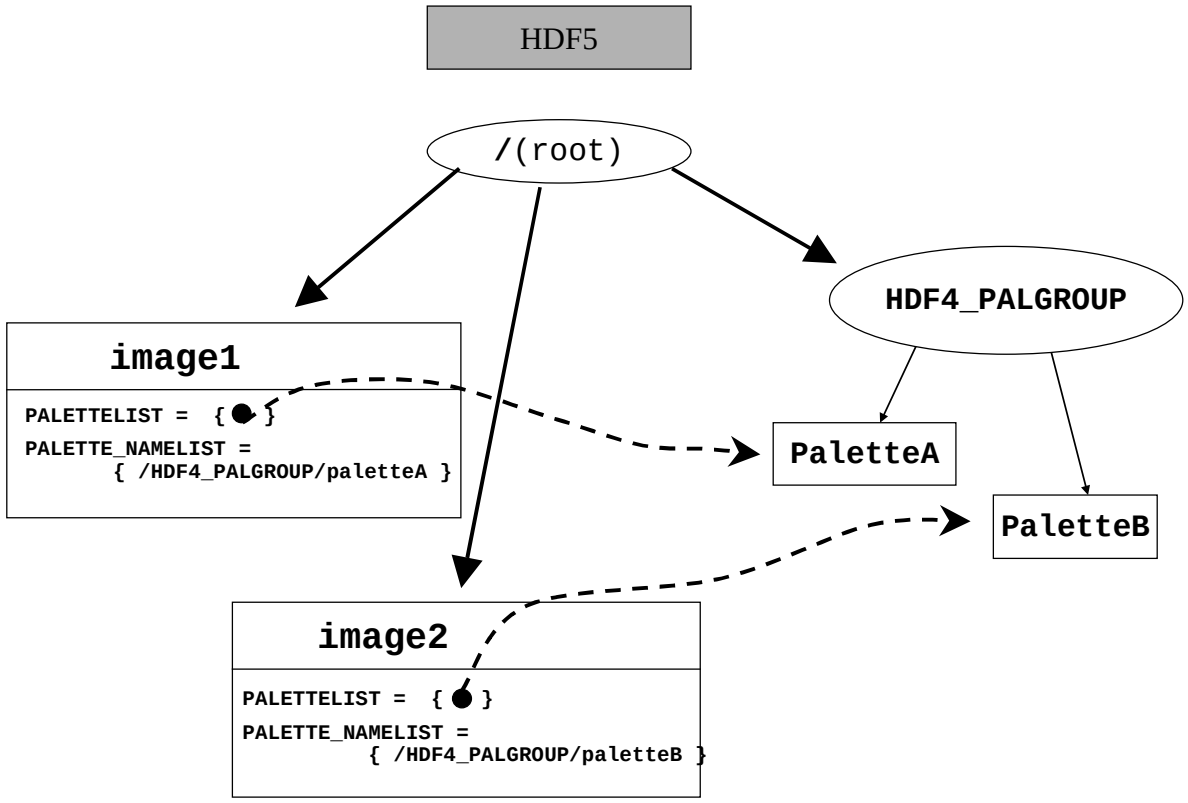


Figure 4. HDF5 dataspace for an RI8 image.

```
int dims[2];
int width;  /*  the fastest changing dimension */
int height;
DFR8getdims(…, &width, &height, …) ;
dims[0] = height;
dims[1] = width;  /* NOTE: width is the fastest changing dimension,
                         the second dimension */

/*  To create the correct HDF5 data set, declare the data space as follows, using
the values of 'dims' in the correct order */
H5Screate_simple(...,dims[], ..);
```

Figure 5. HDF5 dataspace for a GR image.

```
int h4dims[2];
GRgetiminfo(..., h4dims[], ...); /* h4dims[0] == ht; h4dims[1] == wid; */
 /*  To create the correct HDF5 data set, declare the data space as follows, using
 the values of 'dims' in the correct order */
 dims[0] =  h4dims[1];
 dims[1] = h4dims[0]
H5Screate_simple(...,dims[], ..);
```

Multiple Component Images

In the case of an image with more than one component per pixel (e.g., Red, Green, and Blue), the data may be arranged in one of two ways. Following HDF4 terminology, the data may be interlaced by pixel or by plane, which should be indicated by the HDF5 attribute INTERLACE_MODE. In both cases, the HDF5 dataset will have a dataspace with three dimensions, height, width, and components. The interlace modes specify different orders for the dimensions, as explained in the *HDF5 Image and Palette Specification[4]*.

"Line interlace' is not supported by HDF5. An HDF4 image that is stored with line interlace should be stored either as pixel or plane interlace.

Table 8. Storage of multiple component images.

HDF4 Interlace Mode (HDF5 Interlace)	Dimensions in the Dataspace
MFGR_INTERLACE_PIXEL (INTERLACE_PIXEL)	[height][width][pixel components]
MFGR_INTERLACE_COMPONENT (INTERLACE_PLANE)	[pixel components][height][width]
MFGR_INTERLACE_LINE (either)	either

For example, consider a 5 (rows) by 10 (column) image, with Red, Green, and Blue components. Each component is an unsigned byte. In HDF5, the datatype would be declared as an unsigned 8-bit integer. For pixel interlace, the dataspace would be a three dimensional array, with dimensions: [10][5][3]. For plane interlace, the dataspace would be three dimensions: [3][10][5].

In the case of images with only one component, the dataspace may be either a two dimensional array, or a three dimensional array with the third dimension of size 1. For example, a 5 by 10 image with 8 bit color indexes would be an HDF5 dataset with type unsigned 8-bit integer. The dataspace could be either a two dimensional array, with dimensions [10][5], or three dimensions, with dimensions either [10][5][1] or [1][10][5].

GR Storage Properties

In general, the HDF5 dataset is correct if it contains all the correct data values from the HDF4 GR image. However, if chunking and compression are used in HDF4 and not in HDF5, the HDF5 dataset may use much more storage and perform much poorer than the HDF4 GR. For these reasons, the HDF5 dataset should use storage properties analogous to the HDF4 GR.

The follow rules are recommended:

- If the HDF4 GR is chunked, the HDF5 dataset should be chunked with the same chunk sizes
- If the HDF4 GR is compressed, the HDF5 dataset should be compressed with the same compression method, if available.
- If the HDF4 GR is compressed by any method not available with HDF5, the HDF5 dataset should be compressed with GZIP.

If the HDF4 GR is compressed but not chunked, the HDF5 dataset should be "chunked" with the chunksize equal to the entire dataset. The HDF5 dataset should be compressed. These rules are summarized in Table 4, above.

3.5 Palette

Palettes are mapped to HDF5 datasets that are 2-D arrays of bytes with dimensions 256 x 3. The HDF5 Palette should conform to the *HDF5 Image and Palette Specification* [4]. All attributes required by the Palette Specification should be included, even if not specified here. Notice that the HDF5 palette conventions support additional information that is not supported in HDF4, such as color model and range index.

As indicated in section 3.4, it is recommended that in HDF5 all palette objects from an HDF4 file be stored in the group "/HDF4_PALGROUP/<palette_name>." The HDF5 link <palette_name> can be given any name, but if no name is available <palette_name> should be assigned according to the guidelines in section 4.

Table 9. Palette mapping.

	Palette	HDF5 object or component	Datatype, value, etc.	Notes
R	(Datatype)	Atomic datatype	<uint8>	
R	(Data)	Data		
R	(Rank & dimension sizes)	Dataspace		rank=2, dimension size = 256x3 (See note 1, and section 5.4)
O	(HDF4 object type)	Attr	HDF4_OBJECT_TYPE=" palette"	
O	<User-defined attribute>	Attr	<name> = <value>	rank = 1; size is fixed;
O	(Reference number)	Attr	HDF4_REF_NUM = <uint16>	
R	(Class)	Attr	CLASS = "PALETTE"	
R	(Palette type)	Attr	PAL_TYPE = "STANDARD8"	

Note 1: The HDF5 palette specification requires that a palette dataset have dimensions (`nentries` by `ncomponents`), where '`nentries`' is the number of colors (in this case 256) and '`ncomponents`' is the number of values per color (in this case 3).

3.6 Annotation

Annotations are mapped to HDF5 attributes. There are four kinds of HDF4 annotations: file labels and descriptions, and object labels and descriptions. File annotations will be attributes of the root group. Although object annotations can be associated with any tag/ref supported by HDF4, this specification supports only object annotations that are associated with HDF4 SDS, Vgroups, Vdatas, images, and palettes. Annotations will have the following HDF5 attribute names: FILE_LABEL<n>, FILE_DESCRIPTION<n>, OBJECT_LABEL<n>, OBJECT_DESCRIPTION<n>, where <n> is an integer used to distinguish one annotation from another. For instance, if an object had two object labels, the corresponding attribute names would be OBJECT_LABEL1 and OBJECT_LABEL2. Table 10 summarizes these guidelines.

Table 10. Annotation mapping.

	Annotation	HDF5 object or component	Datatype, value, etc.	Notes
R	(File label)	Attr	FILE_LABEL<n> = <string>	Root group attribute.
R	(File description)	Attr	FILE_DESCRIPTION<n> = <string>	Root group attribute.
R	(Object label)	Attr	OBJECT_LABEL<n> = <string>	Attribute for corresponding HDF5 object. (See note 1.)
R	(Object description)	Attr	OBJECT_DESCRIPTION<n> = <string>	Attribute for corresponding HDF5 object. (See note 1.)

Note 1. Although any HDF4 object that can be represented by a tag can have associated object labels and descriptions, this specification only requires that annotations be converted for the following tags:

1. SDS-related tags DFTAG_SD, DFTAG_SDG, DFTAG_NDG
2. Image-related tags DFTAG_RI, DFTAG_RI8, DFTAG_RIG
3. Vgroup tag DFTAG_VG
4. Vdata-related tags DFTAG_VH, DFTAG_VS
5. Palette tag DFTAG_LUT

4 HDF4 and HDF5 File Organization

In addition to mapping individual objects, it is necessary to map the organization and name structure of the HDF4 file to the HDF5 file. The following guidelines are covered in this section:

1. Object names should be translated in a standard way. Objects that have no name in HDF4 (such as palettes) must be assigned a default name in the HDF5 file. If two or more HDF4 objects have the same name, they must be assigned unique names in HDF5.
2. The hierarchical structure of the HDF4 file should be accurately reproduced in the HDF5 file, but objects used internally by the HDF4 library (e.g., "CDF0.0") should *not* be represented in the HDF5 file.
3. Shared objects (such as Named Dimensions) should be stored once and shared in the HDF5 file.

4.1 Object Names

The treatment of object names in HDF 4 (and earlier versions) varies depending on the type of object. The HDF4 User's Guide provides details on naming HDF objects. In summary:

- Palettes and annotations do not have names.
- Vgroup and Vdata names are optional.
- Images may or may not be assigned names, depending on the version of the library, and in any case are optional.
- SDS objects are treated differently by different versions of the library. Early versions of the HDF library (before HDF 3.2) did not require an SDS to have a name, and they allowed two or more SDSs to have the same name. In later versions every SDS was required to have a unique name, and in addition it was possible to assign a second name, called LONGNAME, that did not have to be unique.
- Objects of different classes may have the same name. For example, there might be a Vgroup named "DATA" and also an SDS with name "DATA".

Hence, in HDF4, names may occur in a variety of ways:

- An HDF4 object may have a unique name
- An HDF4 object may have no name
- More than one HDF4 object may have the same name

For the purposes of this paper, we can think of the name of an object in HDF4 (if one exists) as a kind of object attribute. It may not be stored specifically as an attribute, but it nevertheless plays a similar role. We refer to this as NAME in this paper.

In HDF5, the name of an object is the final component of its path within the file. For instance, if an HDF5 object has the path `/foo/bar`, its HDF5 name is "bar."

4.1.1 Objects with unique names

In general, if an HDF4 object has a unique NAME attribute, the corresponding HDF5 object's path should use that name. In addition, as the tables in Section 3 indicate, that same name can be stored as an attribute of the HDF5 object with the name "HDF4_OBJECT_NAME."

4.1.2 Objects with no name

When an HDF4 object with no name must be stored in HDF5, it can be assigned either a default name, or a name provided by an application. A default name should be constructed from a default string with the object's reference

number concatenated on the end. For instance, an unnamed Vgroup whose reference number is 2 would be named HDF4_VGROUP_2. Table 11 shows templates for the default HDF5 names for HDF4 objects.

Table 11. Default HDF5 names for HDF4 objects.

HDF4 Object Type	HDF5 Object Name
Vgroup	HDF4_VGROUP_<ref>
Vdata	HDF4_VDATA_<ref>
SDS	HDF4_SDS_<ref>
Image	HDF4_IMAGE_<ref>
Palette	HDF4_PALETTE_<ref>

4.1.3 Name Collisions

When two HDF4 objects have the same name and will be written to the same HDF5 group, it is necessary to change the pathname of one or both of the objects. For instance, suppose two HDF4 SDSs have the name "mouse", and both belong to the top-level Vgroup "cat." Using the rule suggested in section 4.1.1, these would both be stored as HDF5 datasets with the path "/cat/mouse." Since no two objects in HDF5 can have the same path, it is necessary to change the pathname of one or both of these datasets. The following rules are used for dealing with this situation.

When there is a name collision, applications are free to change the names to anything they like. If no name is assigned, then all but one of the colliding objects in the HDF5 file should be given a default name, using the templates in Table 11.

Notice that the original non-unique HDF4 name can still be stored in the attribute HDF_OBJECT_NAME associated with the corresponding HDF5 object.

4.2 Organization of the HDF4 and HDF5 Files

The grouping structure of an HDF4 file should be represented in the corresponding HDF5 file, but with some exceptions, as described here.

> *In general, a user-defined HDF4 Vgroup should be mapped to an HDF5 Group with a corresponding path name. Lone Vgroups should be children of the HDF5 root group.*

HDF4 grouping structures.

An HDF4 file can consist of a set of individual objects, but it may also be organized as a directed graph, with Vgroups as nodes and any HDF4 objects as nodes. Any HDF4 object may be a child of no Vgroup, one Vgroup, or more than one Vgroup. Since HDF5 can also be organized as a directed graph (using the HDF5 group structure), a mapping from HDF4 to HDF5 should replicate the graph structure as much as possible. Hence, as a general rule, *HDF4 objects that are members of a Vgroup are stored as children of a corresponding HDF5 Group.*

One difference between HDF4 and HDF5 is that there is always at least one group in HDF5, the root group, and all objects must be descendants of the root group. For purposes of the HDF4-to-HDF5 mapping, the root group in HDF5 can be thought of as the same as an HDF4 file. Hence, *HDF4 objects that are not members of a Vgroup should be stored as children of the root group in the HDF5 file.*

A special case of the HDF4 grouping structure that requires attention is a cyclic structure, where the links between groups form a cycle. In this case there is no "top" group, and hence no obvious rule for which group should be assigned to the HDF5 root group. The rule proposed for handling this situation is to arbitrarily choose any Vgroup in the ring and place it under root group. That is, *when Vgroups form a cycle, the cycle of groups should be converted into HDF5 by arbitrarily choosing any Vgroup in the cycle and placing it under the root group.*

An HDF4 file that contains SDS or GR objects is likely to contain extra Vgroups that were not defined by the application. These are special Vgroups and other structures used by the HDF library to organize the SDS and GR objects. These extra objects are hidden from the SDS and GR API, and are used only to help describe the internal organization of the HDF4 file. For example, Vgroups of class "CDF0.0" and "RI0.0" contain all of the SDSs and GRs in an HDF file, together with global SDS attributes and global GR attributes, respectively. *Since these internal objects are not needed to describe HDF4 objects in HDF5, they should be ignored in the conversion process.*

Figure 6 provides examples of how some of these structures occur in an HDF4 file:

- The normal Vgroups VgroupA and VGBa2 both contain the dataset SDS1. That relationship is replicated in the HDF5 file.
- The Vgroups CDF0.0 and RI0.0 are written by the HDF4 library, as a way of organizing the HDF4 file. If an application accesses the file using the SD or GR interfaces, the application never sees these two groups, and consequently they should be ignored in any conversion to HDF5.
- In the figure there is a Vgroup cycle between VgroupB and VGBa. When converting to HDF5, one of the groups in the cycle is attached to the root, in this case Vgroup B.

Figure 6. HDF4 example, showing special internal groups (CDF0.0 and RI0.0), and cyclic grouping structure.

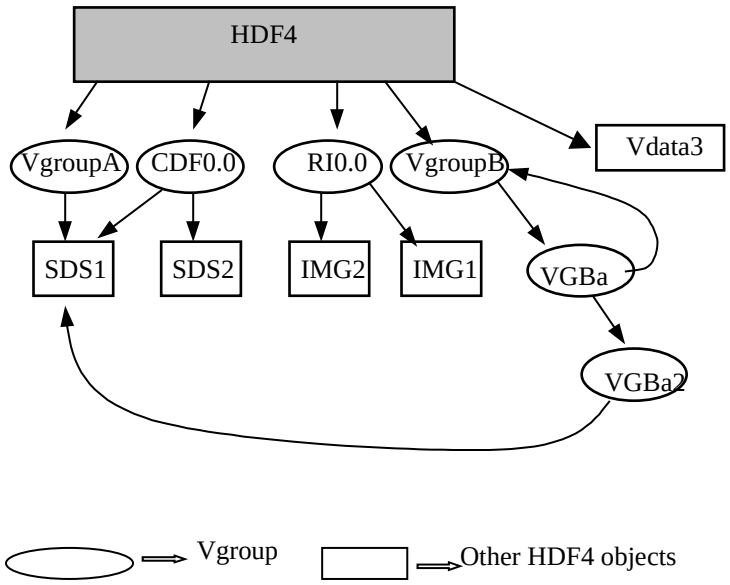
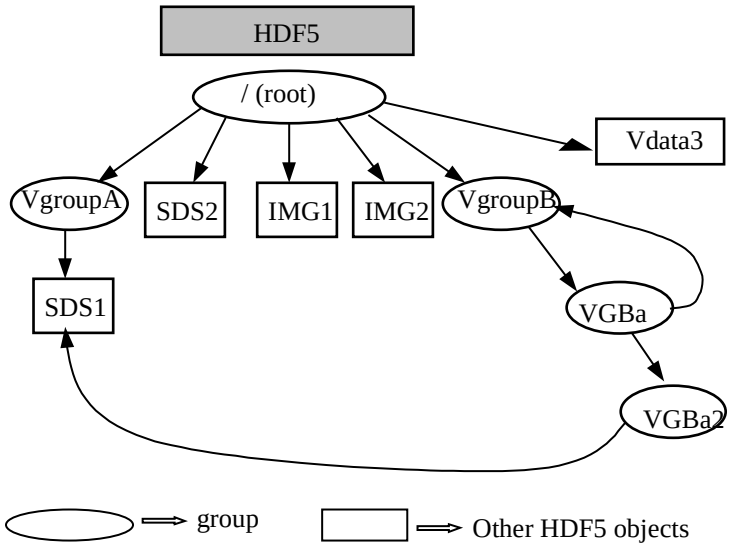


Figure 7 shows how the file in Figure 6 would look in HDF5 using these conventions.

Figure 7. Illustrative Example of HDF5 Structure



4.3 Shared HDF4 objects: dimensions and palettes

Dimension scales (coordinate variables) and palettes can be shared in HDF4. The following rules apply to storing these objects in HDF5.

- Two extra Groups are added to the HDF5 root group: "HDF4_DIMGROUP" and "HDF4_PALGROUP". These groups are used to store the HDF5 datasets constructed for HDF4 Named Dimension Scales and Palettes, respectively. These objects are globally unique to the HDF4 file, so one copy of each object is stored in the respective group.
- Attributes that are global attributes of the SDS and GR interfaces are stored as attributes of the HDF5 root group.

See Section e.1 and 3.4 for more details of how these objects should be stored.

5 Other considerations

Most applications that deal with HDF4 files deal with the eight basic objects, but there is other information that sometimes must be considered.

5.1 Dealing with HDF4 reference numbers

Reference numbers should not, in general, be used by applications as identifiers for HDF4 objects. Nevertheless, since some applications use reference numbers in this way, it would be useful to have an unambiguous way to store equivalent identifiers with HDF5 objects. Hence, if an application must use a reference number in connection with an HDF5 object we propose using an attribute "REF_NUM" whose type is unsigned 16-bit integer to indicate that the corresponding object is to be interpreted as having the given reference number. It is the responsibility of the application to provide a method of assigning valid values for such reference numbers.

5.2 File-level information

HDF4 and HDF5 both have certain file-level information. This includes information that is stored in the file, such as version number, and information about the file, such as its size. It is recommended that the following types of file characteristics be treated as indicated:

- Version number: the library version number of the HDF5 file should be ignored.
- File size: HDF5 files that are larger than 2^31-1 cannot be fully represented in HDF4. It is the responsibility of the application to decide how to deal with HDF5 files that are larger than 2^31-1.
- HDF5 user-defined header: to be treated as an HDF4 file description.
- HDF5 offset datatype and other datatypes: should be ignored.

5.3 Storage properties

When translating an HDF4 object to HDF5, it may be possible to determine certain storage properties used for storing the HDF4 object. For example, valid storage properties for an SDS are compression, chunking, and external storage. If special storage properties are used in HDF4, and if those storage properties are available in HDF5, then they should be used to store the corresponding HDF5 object. For instance, if an HDF4 SDS is chunked, then the corresponding HDF5 datasets should be chunked in the same way. If it is not possible to determine an HDF4 storage property, then the HDF5 dataset can be stored without applying that property. See section 3.1.1 for details.

5.4 Dataspaces

The HDF5 "dataspace" concept generalizes and formalizes the ideas of dimensionality that are implicit in HDF4 objects. Determining the appropriate dataspace of the HDF5 object is straightforward. This mapping was given for each object in Section 2, and is summarized in Table 12.

Table 12. Summary of HDF5 dataspaces for HDF4 objects.

HDF4 Object	HDF5 Dataset	Notes
SDS: rank, dimensions	Simple dataspace, rank, dimensions. maxdims[0] = H5S_UNLIMITED or maxdims[0] = dimsizes[0]	The first dimension of the HDF4 SDS may be UNLIMITED. In this case, the first dimension of the HDF5 dataspace would then have maxdims = H5S_UNLIMITED
GR (ncomp = 1), RI8	Simple dataspace, rank = 2, dimensions = dimsizes	The HDF5 dimensions should be the reverse of the dimensions returned by the GR interface. See section 3.4.
GR (ncomp > 1), RI24	Simple dataspace, rank = 3, dimensions = dimsizes, third dimension is ncomps.	The HDF5 dimensions should be the reverse of the dimensions returned by the GR interface. See section 3.4See section 3.4.1 for interlace modes.
Palette	Simple dataspace, rank = 2, dimensions= {256, 3}	
Vdata	Simple dataspace, rank = 1, dimension = H5S_UNLIMITED or dimension = <maximum records>	Datatype is compound.

5.5 Datatypes

When converting SDSs, Vdatas, and images from HDF4 files to HDF5 files, the HDF4 data elements need to be converted to appropriate corresponding HDF5 data types. The following rules describe how HDF4 datatypes should be converted to HDF5:

5.5.1. "Standard" numeric datatypes. HDF4 standard integer and floating point datatypes should be converted to corresponding HDF5 datatypes. The size, type (integer vs. float), "endianness," etc., should be maintained. Table 13 describes the correspondence between standard HDF4 type definitions and HDF5 type definitions.

Table 13. Correspondence between standard numeric datatypes.

HDF4 type	Corresponding HDF5 type
DFNT_INT8	H5T_STD_I8BE
DFNT_UINT8	H5T_STD_U8BE
DFNT_LINT8	H5T_STD_I8LE
DFNT_LUINT8	H5T_STD_U8LE
DFNT_INT16	H5T_STD_I16BE
DFNT_UINT16	H5T_STD_U16BE
DFNT_LINT16	H5T_STD_I16LE
DFNT_LUINT16	H5T_STD_U16LE
DFNT_INT32	H5T_STD_I32BE
DFNT_UINT32	H5T_STD_U32BE
DFNT_LINT32	H5T_STD_I32LE
DFNT_LUINT32	H5T_STD_U32LE
DFNT_LINT64	H5T_STD_I64LE
DFNT_LUINT64	H5T_STD_U64LE
DFNT_FLOAT32	H5T_IEEE_F32BE
DFNT_LFLOAT32	H5T_IEEE_F32LE
DFNT_FLOAT64	H5T_IEEE_F64BE
DFNT_LFLOAT64	H5T_IEEE_F64LE

5.5.2. "Native" datatypes (e.g. DFNT_NINT16). Because HDF4 files contain no information about the actual architecture that a native type was written from, it may not be possible for an application to decipher the value of an HDF4 native type for conversion. Therefore, HDF4 native format types should be treated with caution. If a converter can determine the value of a native type, it should choose a corresponding HDF5 type and perform the conversion. If it can not, the proper strategy is left to the discretion of the application.

5.5.3. Character datatype. For the HDF4 datatypes DFNT_CHAR8 and DFNT_UCHAR8, there is no corresponding datatype in HDF5. If the values in an HDF4 SDS, raster image, palette or dimensional scale data of SDS are of one of these two types, it is recommended that they be converted to H5T_STD_I8BE or H5T_STD_U8BE. If the values of a Vdata field or attribute are of one of these types, it is recommended that those values be interpreted as constituting a string, and that they be converted to H5T_STRING.

6 Interpreting HDF5 files as HDF4 files when there is no explicit metadata

Sections 3-5 deal with how to represent HDF4 objects in HDF5. If these guidelines are followed in creating HDF5 files, it should be possible for an "HDF4 application" to make reasonable use of data stored in HDF5. But what happens if an HDF4 application encounters an HDF5 file that does not contain the extra information specified in these guidelines? This section provides guidelines on how an HDF4 application can interpret a "raw" HDF5 file.

If an HDF4-based application encounters and object in an HDF5 file that does not contain the metadata described in the previous sections, three possible outcomes can occur:

1. No HDF4 counterpart exists to the HDF5 object
2. There are more than one possible HDF4 counterparts to the HDF5 object (the ambiguous case)
3. The HDF5 object has an unambiguous corresponding HDF4 counterpart

6.1 Case 1: No HDF4 counterpart

Only HDF5 datasets, attributes and groups can have HDF5 counterparts. All other HDF5 objects should be assumed to have no HDF4 counterpart. For instance, the HDF5 "named datatype" object has no HDF4 counterpart.

Certain HDF5 datasets, attributes and groups also have no HDF4 counterpart. These include

- Any HDF5 dataset or attribute whose datatype is not equivalent to an HDF4 datatype. HDF4 datatypes include unsigned and signed 8-, 16-, 32- and 64-bit integers, and 32- and 64-bit IEEE floats.
- Datasets with datatypes of multiplicity greater than 1, unless they map to Vdatas.
- Any HDF5 object whose size is greater than 2^{31}-1.
- Any HDF5 attribute that is associated with an HDF5 dataset whose HDF4 counterpart is a palette.
- Any HDF5 soft link that does not point to a corresponding HDF4-compatible object.

6.2 Case 2: The ambiguous case

There are a number of cases where an HDF5 object could be interpreted as any of a number of HDF4 objects. Here are some such cases.

6.2.1 HDF5 "datasets"

According to Table 1 HDF5 datasets can represent SDS, images, palettes, and Vdatas. This can lead to certain ambiguities. For example, a 2-D HDF5 dataset of some HDF4-compatible datatype could be converted either to an HDF4 SDS, GR raster, or Vdata. We propose the following convention to resolve this ambiguous case:

Unless there is metadata to indicate otherwise, an HDF5 datasets with an HDF4-compatible scalar datatype is assumed to be an HDF4 SDS.

6.2.2 The root group

Another ambiguous case is the HDF5 root group, which has no precise counterpart in HDF4. The HDF5 root group could always be mapped to a corresponding HDF4 Vgroup, with all HDF4 objects descending from that group, but this might in some cases create a view that was unnatural for a particular application. It might be more natural, for instance, to ignore the root group and to treat all of its attributes as HDF4 file annotations. Therefore, with respect to the root group, we propose the convention:

An application can treat an HDF5 root group in whatever way best fits with its view of HDF4 files.

6.2.3 Strings

Since strings are not explicitly supported in HDF4, it is recommended that any HDF5 string be mapped into an array of characters in HDF4. Thus, for example, an HDF5 one-dimensional array of H5T_STRING should be translated to an HDF4 two-dimensional array of type 'char.'

HDF5 strings may be null-terminated, or may have padding (0-padding or space-padding.) In general, a corresponding HDF4 array should be terminated or padded in the equivalent fashion, if the HDF4 application can handle it. Otherwise, the treatment of an HDF5 string is left to the application.

6.3 Case 3: The mapping is unambiguous

Any HDF5 dataset that is not covered in case 1 or case 2 should map to and HDF4 SDS or Vdata. HDF5 groups map to Vgroups, with the exception of the root group, covered above.

Any HDF5 dataset or group attribute becomes an attribute to the corresponding HDF4 object, if the object supports attributes. Otherwise it is interpreted as an HDF4 annotation for the corresponding HDF4 object.

7 References

1. *HCR (HDF Configuration Record) Specification.* ftp://ftp.ncsa.uiuc.edu/HDF/pub/HCR/Doc/HCR-Definitions/
2. *HDF4 Documentation.* http://hdf.ncsa.uiuc.edu/doc.html
3. *HDF5 Documentation.* http://hdf.ncsa.uiuc.edu/HDF5/doc/
4. *HDF5 Image and Palette Specification.* http://hdf.ncsa.uiuc.edu/HDF5/doc/ImageSpec.html