HDF5 for HDF4 Users: a short guide

National Center for Supercomputing Applications
University of Illinois, Urbana-Champaign
December 3, 2002

Contents

1. Introduction 1
2. Three Principles 2
Do what makes sense 2
Think of HDF5 as a completely new file format 2
Anything you can do in HDF4, you can do in HDF5 2
3. A short Summary of Differences in the Data Model and Format 2
4. Short Summary of Differences in the Programming Model 4
5. How Do I create HDF4 objects in HDF5? 4
SDS: Dataset 5
Vgroup: Group 7
Vdatas: Table Data 9
Raster Images (GR, DFR8, DF24) and Palettes 14
Attributes, Global Attributes, Annotations, etc. 17
5. How Do I… 18
...find out what is in an HDF5 file? 18
...find all the objects, etc? 19
...locate and access specific objects in an HDF5 file? 19
...find the tag and ref of an HDF5 object? 19
...create and use NT_xx data? 19
...create and use “String” data? 20
...create a Chunked/Compressed Dataset? 21
...read/write a sub-set or sub-sample of a Dataset? 22
...create/read Dimensions and Dimension Scales? 22
For More Information 22
Acknowledgements 23
References 23
Appendix 1: Why HDF5? 23

1. Introduction

HDF5 is a new format and library, unfamiliar to HDF4 users. HDF5 was designed based on lessons learned from HDF4, and is conceptually a superset of HDF4. (Almost) anything you can do with HDF4 you can do (better!) with HDF5. However, the details of HDF5 are substantially different from HDF4, so it may not be obvious how to use it.

Both HDF4 and HDF5 are supported by the NCSA HDF group. We will continue to maintain HDF4, as long as we are funded to do so. We do not plan to add any new features to HDF4, but we will fix bugs and build and test it on new operating system versions. We recommend using HDF5, especially if you are a new user and are not constrained to using HDF 4.x (HDF4). We also recommend that you consider migrating from HDF4 to HDF5 to take advantage of the improved features and performance of HDF5.

Earlier work has defined a conceptual mapping for HDF4 and HDF5 data objects [1], and software is available to convert files following these guidelines [2]. This note addresses another aspect of the transition from HDF4 to HDF5: the conversion of software and models.

Many users are familiar with HDF4, and have working software that uses the HDF4 library, and would now like to move forward to the new HDF5 library. For this task, it is natural to want to bring forward whatever has been done with HDF4 into HDF5. This note tries to provide some guidelines to help users learn how to do this.

This note is organized as follows. The first section presents some guiding principles for users. The subsequent sections explain differences between HDF4 and HDF5, in the Data Model and Formats, and the Programming Model. The final section presents a number of "How Do I…" cases: implementing common examples of HDF4 programs.

2. Three Principles

Do what makes sense

The most important guiding principle is "*do what is best for your needs*". This note will give advice and examples, but these are not proposed as hard and fast rules. In many cases, there will be more than one possible approach, and in some cases it will be best to use HDF5 in a totally different way than HDF4 was used. *Do what makes sense for you*.

Think of HDF5 as a completely new file format

While HDF Versions 1-4 were backward compatible (both format and APIs), HDF5 is a new format and library. In fact, it is best to think of HDF5 as a completely different format and library. HDF5 data cannot be read or written by HDF4. HDF4 data cannot be read or written by HDF5. In general, it will be necessary to rewrite software that uses HDF4.

Thus, the task of adopting HDF5 is similar to porting to another format; it is necessary to learn the new software and adapt old software to use the new.

Anything you can do in HDF4, you can do in HDF5

While HDF5 is new, it is highly compatible with HDF4. In fact, HDF5 is a conceptual superset of HDF4: in general, it is reasonable to expect that anything you can do with HDF4 you can do with HDF5; plus a lot more.[*]

The HDF4 to HDF5 Mapping [1] gives a detailed explanation of how to represent the data objects of HDF4 in HDF5 objects. This note explains how to do some common tasks.

3. A short Summary of Differences in the Data Model and Format

The HDF5 Format [9] is completely different from HDF4 [6]. For most users it will never be necessary to know about the file format for either format.

---

[*] There are a few things that HDF4 can do that are not yet available for HDF5, including reading netCDF, and compression other than GZIP.

HDF5 For HDF4 Users

Most users will work with the Data Model and APIs. The HDF4 and HDF5 Data Models and APIs are considerably different.

The HDF4 Data Model has six basic objects:

- Scientific dataset (SDS), a multidimensional array with dimension scales
- General raster image (GR), a 2-dimensional array of multi-component pixels (includes 8-bit and 24-bit raster images)
- 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

HDF5 includes two primary objects:

- Dataset, a multidimensional array of data elements
- 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 will not be considered here.

Overall, the HDF5 object model is simpler and more general than HDF4, so that every HDF4 object can be represented as an instance of the HDF5 objects. Table 1 lists suggested HDF5 objects for the common HDF4 objects.

Table 1. Representing HDF4 objects in HDF5. See ([1]) for details.

HDF4 object	Corresponding HDF5 object	Description
SDS	Dataset	Only the first dimension of an HDF4 SDS can be unlimited, any or all can be unlimited in HDF5. Not all HDF4 storage properties are supported.
Image	Dataset	The HDF5 object should conform to the HDF5 Image specification [4].
Palette	Dataset	The HDF5 object should conform to the HDF5 Palette specification [4].
Vdata (table)	Dataset	The HDF5 object should conform to the HDF5 Table specification [5]. 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.
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 guidelines in Table 1 define the correspondence between the objects in and HDF4 file and the objects in an HDF5 file. The *h4toh5* utility converts all the objects in an HDF4 file to a default HDF5 file. The H4 to H5 library provides a conversion for individual objects. [2]

It is important to realize, though, that a default conversion preserves the design of the HDF4 file, which may not be an optimal design for the resulting HDF5 file. The H4 to H5 library can help users convert HDF4 objects to create the HDF5 file they want.

Some HDF4 objects might be converted to more than one possible HDF5 object, depending on the intention of the creator. Also, HDF5 has many more possibilities than HDF4, such as multidimensional tables (compound data), softlinks, object references, and stored (Named) datatypes. Applications should take advantage of these HDF5 features which had no counterpart in HDF4. See the HDF5 documentation for information about these objects and datatypes.

4. Short Summary of Differences in the Programming Model

HDF4 and HDF5 are implemented as C libraries, and have a similar overall programming model. Objects are created or opened, and then accessed by handles. The operations on different classes of object are implemented as C subroutines, with names that segregate the operations for the different classes.

That said, there are many important differences.

As already noted, HDF4 defines eight classes of object, each of which has certain properties and uses. The HDF5 model does not define such objects, although there are standard profiles recommended for Images and Tables [5]. These correspond to HDF4 GR, palette, and Vdata interfaces.

The HDF4 library reads and writes netCDF. This is not supported by HDF5. Also, HDF4 has JPEG and several other compression options. HDF5 only provides GZIP compression at this time.

Overall, the HDF5 API is a “lower level” interface, which enables much more powerful and detailed control over the file and data, at the cost of many more calls to the library. All other things equal, a program using HDF5 will have more calls to the library to implement the same operation, and will have many more lines of code. Several HL interfaces have been provided to encapsulate common operations. These libraries call the standard HDF5 library, but reduce the code that needs to be written in an application [5].

HDF5 has important features that are not present in HDF4:

- HDF5 is organized as a rooted, directed graph; every object can be addressed by one or more path names.
- HDF5 has Dataspace objects, which implement the definition and selection of data elements (i.e., the dimensionality of the array)
- HDF5 has Datatype objects and a fully developed model for defining, describing, and transforming datatypes.
- HDF5 has a general model for Property Lists, which is used to implement many types of parameters and settings.
- HDF5 has a variety of storage models, including MPI/IO, and can be extended by the addition of new file drivers
- HDF5 can be extended by adding new compression or other filters.

5. How Do I create HDF4 objects in HDF5?

As discussed above, HDF5 objects are not the same as HDF4. But it is possible to create objects that are equivalent to HDF4 objects.

SDS: Dataset

An HDF4 SDS is equivalent to an HDF5 Dataset with a numeric or character datatype. Figure 1 shows example code to create an HDF4 SDS, and Figure 2 shows code to create a similar HDF5 dataset.

```
int32 sd_id, sds_id;
int32 dim_sizes[2];
intn status;

sd_id = SDstart ("SDS.hdf", DFACC_CREATE);

dim_sizes[0] = 5;
dim_sizes[1] = 16;

sds_id = SDcreate (sd_id, "SDSexample", DFNT_INT32, 2, dim_sizes);

status = SDendaccess (sds_id);

status = SDend (sd_id);
```

Figure 1

```
hid_t      file_id, dataset_id, dataspace_id, datatype_id;
hsize_t    dims[2];
herr_t     status;

file_id = H5Fcreate("DS.h5", H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

dims[0] = 5;
dims[1] = 16;

dataspace_id = H5Screate_simple(2, dims, NULL);

datatype_id = H5Tcopy(H5T_STD_I32BE);

dataset_id = H5Dcreate(file_id, "/DSexample", datatype_id, dataspace_id,
                       H5P_DEFAULT);

status = H5Dclose(dataset_id);
status = H5Sclose(dataspace_id);
status = H5Tclose(datatype_id);
status = H5Fclose(file_id);
```

Figure 2

The two examples are similar, the same information is required. Notice that in HDF5 the description of the dimensions and data type is done with objects. Also, there are three HDF5 Property List objects, which are not used in the example (they are set to H5P_DEFAULT).

Accessing data in an HDF4 SDS and an HDF5 Dataset is similar. Figure 3 shows example code to write and read data from an SDS. Figure 4 shows example code to do similar operations for HDF5. One important difference is that HDF5 has no SD interface, or notion of the order or index of a dataset. In HDF5, the dataset is accessed by its path name "/DSexample"

Another important difference is that both the source and destination dataspace and datatype are described in HDF5. In HDF4, the memory type is assumed and the disk type is specified, and the source is assumed to be a contiguous array. In contrast, HDF5 lets the program specify any datatype for the source and destination, and may define the dataspace and selection for the source and destination. In the example, the whole array is written/read (H5S_ALL), so the original Dataspace is used.

```
int        dset_data[5][16];
sd_id = SDstart (“SDS.h5”, DFACC_WRITE);

sds_index = 0;
sds_id = SDselect (sd_id, sds_index);

start[0] = 0;
start[1] = 0;
edges[0] = 5;
edges[1] = 16;

status = SDwritedata (sds_id, start, NULL, edges, dset_data);

status = Sdreaddata (sds_id, start, NULL, edges, dset_data);

status = SDendaccess (sds_id);
status = SDend (sd_id);
```

Figure 3

The data is written from “native”, i.e., machine memory format, and read back to memory. The disk format was set by the Datatype object when the Dataset was created (as in Figure 2). The HDF5 library will convert the data (byte swap, etc.) as needed.

```
hid_t      file_id, dataset_id;
herr_t     status;
int        dset_data[5][16];

file_id = H5Fopen(“DS.h5”, H5F_ACC_RDWR, H5P_DEFAULT);

dataset_id = H5Dopen(file_id, "/DSexample");

status = H5Dwrite(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
            dset_data);

status = H5Dread(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
            dset_data);

status = H5Dclose(dataset_id);
status = H5Fclose(file_id);
```

Figure 4

Vgroup: Group

The HDF5 Group object is very similar to the HDF4 Vgroup, and much simpler to use. Figure 5 shows example code to create two Vgroups in an HDF4 file. Figure 6 shows example code that creates two Groups in an HDF5 file. There is no “group interface”, so there is no need to start and end. Notice that the HDF5 Groups always have names, and are part of a rooted graph.

```
intn status_n;
int32 status_32, vgroup_ref,
vgroup1_id, vgroup2_id, file_id;

file_id = Hopen (“vgroup.hdf”, DFACC_CREATE, 0);
status_n = Vstart (file_id);

vgroup1_ref = vgroup2_ref = -1; /* create new group */

vgroup1_id = Vattach (file_id, vgroup1_ref, "w");
vgroup2_id = Vattach (file_id, vgroup2_ref, "w");

status_32 = Vdetach (vgroup1_id);
status_32 = Vdetach (vgroup2_id);

status_n = Vend (file_id);
status_n = Hclose (file_id);
```

Figure 5

```
hid_t      file_id, group1_id, group2_id;
herr_t     status;

file_id = H5Fcreate(“groups.h5”, H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

group1_id = H5Gcreate(file_id, "/MyGroup1", 0);

group2_id = H5Gcreate(file_id, "/MyGroup2", 0);

status = H5Gclose(group1_id);
status = H5Gclose(group2_id);
status = H5Fclose(file_id);
```

Figure 6

Inserting objects in groups is somewhat different in HDF4 and HDF5. In HDF4, the objects must be created and then explicitly added to a Vgroup. HDF4 objects do not necessarily belong to any Vgroup. In contrast, all HDF5 objects are members of at least one Group, and the object is created in a specific group.

```
intn status_n;
int32 status_32, sd_id,
sds_id,
sds_ref,
dim_sizes[1],
rank = 1,
vgroup_id, file_id;

file_id = Hopen (“Vgroup.hdf”, DFACC_CREATE, 0);

status_n = Vstart (file_id);

sd_id = SDstart (“Vgroup.hdf”, DFACC_WRITE);

dim_sizes[0] = 5;
dim_sizes[1] = 16;

sds_id = SDcreate (sd_id, “SDS”, DFNT_INT32, rank, dim_sizes);

vgroup_id = Vattach (file_id, -1, "w");
status_32 = Vsetname (vgroup_id, “MyGroup1”);
status_32 = Vsetclass (vgroup_id, “A group”);
sds_ref = SDidtoref (sds_id);
status_32 = Vaddtagref (vgroup_id, DFTAG_NDG, sds_ref);

status_n = SDendaccess (sds_id);
status_n = SDend (sd_id);

status_32 = Vdetach (vgroup_id);
status_n = Vend (file_id);
status_n = Hclose (file_id);
```

Figure 7

Figure 7 shows example HDF4 code that creates a SDS object and inserts it into a Vgroup. Note that the SDS is created, then inserted into the Group. Both the V and SD interfaces must be initialized and then released. In contrast, Figure 8 shows HDF5 code that creates a similar dataset in a group. The Dataset is created and linked to the group in the same step.

Once created, an HDF5 Dataset or other object can be inserted in additional groups, similar to the way that HDF4 inserts into a Vgroup. Figure 8 shows code to create a Group /MyGroup/X, and insert “dset1” in that group (with the name “ptr_to_dset1”). Note that HDF5 uses the path names of the objects, not tag/ref as in HDF4.

```
hid_t       file_id,  group_id, dataset_id, dataspace_id;
hsize_t     dims[2];
herr_t      status;
int         dset1_data[3][3], dset2_data[2][10];

file_id = H5Fopen("groups.h5", H5F_ACC_RDWR, H5P_DEFAULT);

group_id = H5Gcreate(file_id, "/MyGroup", 0);

/* Create the data space for the second dataset. */

dims[0] = 5;
dims[1] = 16;
dataspace_id = H5Screate_simple(2, dims, NULL);

/* Create a dataset in group "MyGroup". */

dataset_id = H5Dcreate(file_id, "/MyGroup/dset1", H5T_STD_I32BE, dataspace_id,
                  H5P_DEFAULT);

status = H5Sclose(dataspace_id);
status = H5Dclose(dataset_id);
status = H5Gclose(group_id);

group_id = H5Gcreate(file_id, "/MyGroup/X", 0);

status = H5Glink(file_id, H5G_LINK_HARD, "/MyGroup/dset1",
        "/MyGroup/X/ptr_to_dset1" );


status = H5Gclose(group_id);
status = H5Fclose(file_id);
```

Figure 8

Vdatas: Table Data

An HDF4 Vdata is a (one dimensional) table of records, where each record is a sequence of fields, possibly of differing datatypes. The corresponding structure in HDF5 is a Dataset with a Compound Datatype. An HDF5 Compound Datatype defines a Datatype that is a sequence of named fields, each with a Datatype. Any Dataset can have a Compound Datatype, so HDF5 tables are not limited to one-dimension.

Figure 9 shows example code that creates a Vdata with 10 records. Each record has three fields, "a_name" (int), "b_name" (float), and "c_name" (double). The data is written from an array of C structures.

```
intn  status_n;
int32  status_32, file_id, vdata_id;
int32 vdata_ref = -1,
int32 num_of_records;
int16 rec_num;

typedef struct s1_t {
    int    a;
    float  b;
    double c;

} s1_t;

s1_t       data_buf[10];


file_id = Hopen ("vdata.hdf", DFACC_WRITE, 0);

status_n = Vstart (file_id);

vdata_id = VSattach (file_id, vdata_ref, "w");

status_32 = VSsetname (vdata_id, "Table 1");
status_32 = VSsetclass (vdata_id, "Some Data");

status_n = VSfdefine (vdata_id, "a_name", DFNT_INT32, 1 );
status_n = VSfdefine (vdata_id, "b_name", DFNT_FLOAT32, 1 );
status_n = VSfdefine (vdata_id, "c_name", DFNT_FLOAT64, 1 );

status_n = VSsetfields (vdata_id, "a_name","b_name","c_name");

num_of_records = VSwrite (vdata_id, data_buf, 10,
                          FULL_INTERLACE);

status_32 = VSdetach (vdata_id);
status_n = Vend (file_id);
status_32 = Hclose (file_id);
```

Figure 9

Figure 10 shows an example that creates an equivalent HDF5 Dataset. The Dataset is one dimensional, with a Compound Datatype with three members, as above. The Datatype is constructed from atomic Datatypes with a series of calls to H5Tinsert(). Then the Dataset is created and written in the same way as any other Dataset.

```
typedef struct s1_t {
    int    a;
    float  b;
    double c;

} s1_t;

s1_t       s1[10];

hid_t      s1_tid;

hid_t      file, dataset, space;
herr_t     status;
hsize_t    dim[] = {10};

space = H5Screate_simple(1, dim, NULL);

file = H5Fcreate("table.h5", H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

s1_tid = H5Tcreate (H5T_COMPOUND, sizeof(s1_t));

H5Tinsert(s1_tid, "a_name", HOFFSET(s1_t, a), H5T_NATIVE_INT);
H5Tinsert(s1_tid, "c_name", HOFFSET(s1_t, c), H5T_NATIVE_DOUBLE);
H5Tinsert(s1_tid, "b_name", HOFFSET(s1_t, b), H5T_NATIVE_FLOAT);

dataset = H5Dcreate(file, "/Table 1", s1_tid, space, H5P_DEFAULT);

status = H5Dwrite(dataset, s1_tid, H5S_ALL, H5S_ALL, H5P_DEFAULT, s1);

H5Tclose(s1_tid);
H5Sclose(space);
H5Dclose(dataset);
H5Fclose(file);
```

Figure 10

The HDF5 "Table" interface provides an alternative for creating and accessing one-dimensional arrays of compound data. This interface follows the conventions of the Table specification [8]. Figure 11 shows an example of how to create the same table as Figure 10 above using the Table interface (H5TBmake_table).

```
typedef struct s1_t {
    int    a;
    float  b;
    double c;

} s1_t;

s1_t       dst_buff[10];


/* Calculate the size and the offsets of the struct members in memory */
 size_t dst_size =  sizeof( s1_t );
 size_t dst_offset[3] = { HOFFSET( s1_t ),
                          HOFFSET( s1_t, b ),
                          HOFFSET( s1_t, c )};

 size_t dst_sizes[3] = { sizeof( dst_buf[0].a),
                         sizeof( dst_buf[0].b),
                         sizeof( dst_buf[0].c)};

const char *field_names[3]  = { "a_name", "b_name", "c_name" };
hid_t      field_type[3];

field_type[0] = H5T_NATIVE_INT;
field_type[1] = H5T_NATIVE_FLOAT;
field_type[2] = H5T_NATIVE_DOUBLE;


file = H5Fcreate(FILE, H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);


H5TBmake_table( "Table 1", file_id, "Table1", 3, 10, dst_size, field_names, dst_offset,
                field_type,  10, NULL, 0, dst_buff  );

H5Fclose(file);
```

Figure 11

Both HDF4 Vdata and HDF5 Compound data can be accessed by field. Figure 12 shows example code to show how to read the data from an HDF4 Vdata from field “b_name” into an array of floats. This example reads all the records, from 0 through 9.

Figure 13 shows how to read from one field of an HDF5 Compound Datatype. This is done by creating a Compound Datatype to describe the desired memory layout, i.e., the field(s) to be read. This datatype is created using the same mechanism as used to define the Dataset, inserting the appropriate field name, type, and offset. This example reads all the records (H5S_ALL).

Figure 14 shows the same read using the Table interface (H5TBread_fields_name). This call does the same read as Figure 13.

```
 intn  status_n;
 int32  status_32, file_id, vdata_id, vdata_ref, num_of_records, record_pos;
 float32 databuf[10];

file_id = Hopen ("vdata.hdf", DFACC_READ, 0);

status_n = Vstart (file_id);

vdata_ref = VSfind (file_id, "Table 1");

vdata_id = VSattach (file_id, vdata_ref, "r");

status_n = VSsetfields (vdata_id, "b_name");
record_pos = VSseek (vdata_id, 0);

num_of_records = VSread (vdata_id, databuf, 10, FULL_INTERLACE);

status_32 = VSdetach (vdata_id);
status_n = Vend (file_id);
status_32 = Hclose (file_id);
```

Figure 12

```
typedef struct s1_t {
    float  b;
} s1_t;

s1_t      s2[10];
hid_t     s2_tid;

dataset = H5Dopen(file, "/Table 1");

s2_tid = H5Tcreate(H5T_COMPOUND, sizeof(s1_t));
H5Tinsert(s1_tid, "b_name", HOFFSET(s1_t, b), H5T_NATIVE_FLOAT);

status = H5Dread(dataset, s2_tid, H5S_ALL, H5S_ALL, H5P_DEFAULT, s2);

H5Tclose(s2_tid);
H5Dclose(dataset);
H5Fclose(file);
```

Figure 13

```
float      s2[10];

size_t field_offset[1] = { 0 };

status = H5TBread_fields_name( file_id, "/Table 1", "b_name", 0, 10, sizeof( float ),
    field_offset, s2  ) ;
```

Figure 14

Some of the details of the HDF4 Vdata interface have no equivalent in HDF5, including:

- VDATA_CLASS
- Field Attributes
- Option for field interlace.

These features can be emulated in HDF5. For recommendations, see the Mapping Spec [1].

An HDF5 Table is an instance of a normal HDF5 Dataset, so all the features and functions of a Dataset can be used, including:

- Chunking
- Compression
- Selection of non-contiguous regions
- Any HDF5 Datatype
- Region References may point to records

In addition, HDF5 Compound data can be used in multidimensional arrays, although the Table interface only supports one dimensional tables.

These features are not available for HDF4 Vdatas.

Raster Images (GR, DFR8, DF24) and Palettes

HDF4 has several interfaces to store and retrieve "image" data. The GR interface provides a generic model for many kinds of imagery, which are stored as two or three dimensional arrays with associated palettes (color tables).

HDF5 does not have a specific object type for images. Instead, they are stored as regular Datasets with appropriate Datatype and Dataspace, plus conventional attributes to indicate that the data is to be interpreted as an image or palette. The conventions are defined in the "Image and Palette Specification" [7].

Figure 15 shows example code to create an HDF4 GR image. The image is 5x2 pixels, a 24-bit image, with three 8-bit components for each pixel. (No palette is needed.)

The HDF5 Image and Palette Specification defines how to store this image data:

- A Dataset with a two-dimensional Dataspace
- The Datatype should be an Array of 3 8-bit integers
- The Dataset should have conventional attributes, including:
    - CLASS="IMAGE"
    - IMAGE_SUBCLASS="IMAGE_TRUECOLOR"

Figure 16 shows an example code to create and write this dataset. Some of the code to create the Attributes is not presented here.

```
intn status;
int32 file_id,
gr_id,
ri_id,
start[2],
edges[2],
dim_sizes[2],
interlace_mode,
data_type,

int16 image_buf[2][5][3];

file_id = Hopen ("image.hdf", DFACC_CREATE, 0);

gr_id = GRstart (file_id);

data_type = DFNT_INT8;

dim_sizes[0] = 5;
dim_sizes[1] = 2;

ri_id = GRcreate (gr_id, "Image 1", 3, data_type,
                        MFGR_INTERLACE_PIXEL, dim_sizes);
start[0] = start[1] = 0;
edges[0] = 5;
edges[1] = 2;

status = GRwriteimage(ri_id, start, NULL, edges, (VOIDP)image_buf);

status = GRendaccess (ri_id);
status = GRend (gr_id);
status = Hclose (file_id);
```

Figure 15

To facilitate the use of image data, the HDF5 Image interface implements the details of the specification. Figure 17 shows example code to create an image with similar data to Figure 16 above using the Image API.

```
hid_t       file_id, dataset_id, dataspace_id, datatype_id;
hsize_t     dims[2];
herr_t      status;
hsize_t       comp_dims[1] = { 3 };

unsigned char data[5][2][3]

file_id = H5Fcreate("image.h5", H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

dims[0] = 5;
dims[1] = 2;
dataspace_id = H5Screate_simple(2, dims, NULL);

datatype_id = H5Tarray_create( H5T_NATIVE_UINT8, 1, comp_dims, NULL );

dataset_id = H5Dcreate(file_id, "/Image 1", datatype_id, dataspace_id,
                                H5P_DEFAULT);

/* Create the required attributes */

attr_type = H5Tcopy( H5T_C_S1 );
attr_size = strlen( "IMAGE" ) + 1;
H5Tset_size( attr_type, (size_t)attr_size);
H5Tset_strpad( attr_type, H5T_STR_NULLTERM ) ;

attr_space_id = H5Screate( H5S_SCALAR );

attr_id = H5Acreate( dataset_id, "CLASS", attr_type, attr_space_id, H5P_DEFAULT );

H5Awrite( attr_id, attr_type, "IMAGE" ) ;
H5Aclose( attr_id );

/*  And so on for the required attributes:
   IMAGE_VERSION="1.2",  IMAGE_SUBCLASS="IMAGE_TRUECOLOR", and
   INTERLACE_MODE="INTERLACE_PIXEL" */

H5Sclose( attr_space_id ) ;

H5Dwrite( dataset_id, datatype_id, H5S_ALL, H5S_ALL, H5P_DEFAULT, data);

status = H5Dclose(dataset_id);
status = H5Sclose(dataspace_id);
status = H5Tclose(datatype_id);
status = H5Fclose(file_id);
```

Figure 16

```
hid_t        file_id;
unsigned char image_data[5][2][3];

file_id = H5Fcreate( "image.h5", H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT );

status = H5IMmake_image_24bit( file_id, "Image 1", 5, 2, image_data )

H5Fclose(file_id);
```

Figure 17

For indexed images, the image data is intended to be interpreted using one or more associated color tables (palettes). HDF4 has a Palette object and API functions to create Palettes and associate them with Images.

HDF5 does not have a object type for Palettes. The Image and Palette Specification defines conventions for storing Palette data as HDF5 Datasets, and associated Palettes with Image Datasets using HDF5 Object References. See the Mapping Specification for details [1]. The HDF5 Image API provides functions to create and manage Palette datasets [5].

Attributes, Global Attributes, Annotations, etc.

HDF4 has several categories of Attributes and Annotations, including:

- Attributes of objects (SDS, GR, Vdata)
- Attributes of fields (Vdata)
- Global Attributes (the SD interface, the GR interface)
- Global Annotations

The different types of object have different interfaces and conventions, but they are all methods of storing user-defined information associated with the file and objects within the file.

HDF4 has pre-defined attributes (such as Label, Units, and Format of an SDS). HDF5 does not define any attributes.

HDF5 has a single type of Attribute object, which can be used for any of the purposes. HDF5 Attributes are similar to Datasets, they have a Dataspace and a Datatype. Attributes may also have references to objects in the HDF5 file.

An HDF5 Attribute is always attached to an object (Dataset, Group, or Named Datatype). There are no “file” or “global” Attributes in HDF5. By convention, Attributes of the file should be stored as Attributes of the root Group.

Figure 18 shows example code to write a user-defined attribute to an HDF4 SDS. The attribute is called “VALID_RANGE”, and the value is two 32-bit floating point numbers. Figure 19 shows code to write a similar attribute to an HDF5 Dataset.

```
int attr_values[2];

sd_id = SDstart (“attr.hdf”, DFACC_WRITE);

sds_index = 0;
sds_id = SDselect (sd_id, sds_index);

n_values = 2;
status = SDsetattr (sds_id, “Valid_Range”, DFNT_FLOAT32, n_values,
          attr_values);

status = SDendaccess (sds_id);
status = SDend (sd_id);
```

Figure 18

```
hid_t      file_id, dataset_id, attribute_id, dataspace_id;
hsize_t    dims;
int        attr_data[2];
herr_t     status;

file_id = H5Fopen(“attr.h5”, H5F_ACC_RDWR, H5P_DEFAULT);

dataset_id = H5Dopen(file_id, "/dset");

dims = 2;
dataspace_id = H5Screate_simple(1, &dims, NULL);

attribute_id = H5Acreate(dataset_id, "ValidRange", H5T_NATIVE_INT, dataspace_id,
H5P_DEFAULT);

status = H5Awrite(attribute_id, H5T_NATIVE_INT, attr_data);

status = H5Aclose(attribute_id);
status = H5Sclose(dataspace_id);
status = H5Dclose(dataset_id);
status = H5Fclose(file_id);
```

Figure 19

5. How Do I…

This section gives some miscellaneous “how to” suggestions.

...find out what is in an HDF5 file?

The *h5dump* utility lists the contents of an HDF5 file [3]. This tool is equivalent to the *hdp* utility for HDF4, although much simpler to use.

...find all the objects, etc?

A program can discover the members of a Group using H5Giterate. The H5Gget_obj_info gives the type and information about an object. Each object type has inquiry functions to discover all the attributes of the object.

All the objects in an HDF5 file are organized in a rooted graph, and can be accessed by paths from the root group. There are no equivalents to “lone Vgroups” or similar objects in HDF4, every object is in the graph.

HDF4 has interfaces for accessing all the SDS, GR, etc. HDF5 has no equivalent.

...locate and access specific objects in an HDF5 file?

Objects are accessed by name, either through a full path or a path relative to a specific group.

If the name of the object is not known, it is possible to iterate through all the objects in the file. You have to traverse all the objects in the file using H5Giterate. For any object for which the link count is > 1, record it in a table, using "objno" as a key (See H5Gget_objinfo in the HDF5 Reference Manual). The first time you see it, there will be no entry in the table, and subsequent visits will find an entry.

If you need a table of contents for the whole file, then collect all the information you need, and use "objno" as a key to avoid loops.

...find the tag and ref of an HDF5 object?

HDF5 does not use tags and refs, and has no conceptual equivalent. In general, objects are accessed via their path name. Table 2 lists common uses of the HDF4 tag/ref, along with the HDF5 equivalent.

Table 2

HDF4 operation	HDF5 operation
Insert in Vgroup	Create with path name, H5Glink
Open	Use path name
Store a pointer to an object, e.g., in an attribute	Use HDF5 Object Reference Datatype

...create and use NT_xx data?

HDF5 has a rich and complex model for describing and transforming Datatypes. Most of the Datatypes in HDF4 can be represented in HDF5.

In HDF5 it is necessary to define the storage type for the stored data, and also for the data when it is in memory. Data may be stored in any byte order (unlike HDF4). The HDF5 library provides predefined “H5T_NATIVE_XXX” Datatypes, which are correct type descriptions for each platform. Data is automatically transformed between the source and destination for any combination of Datatypes.

Table 3 gives the common HDF4 datatypes, along with the HDF5 storage and memory type.

Table 3

HDF4 type	Corresponding HDF5 type	HDF5 Memory type
DFNT_INT8	H5T_STD_I8BE	H5T_NATIVE_INT8
DFNT_UINT8	H5T_STD_U8BE	H5T_NATIVE_UINT8
DFNT_LINT8	H5T_STD_I8LE	H5T_NATIVE_INT8
DFNT_LUINT8	H5T_STD_U8LE	H5T_NATIVE_UINT8
DFNT_INT16	H5T_STD_I16BE	H5T_NATIVE_INT16
DFNT_UINT16	H5T_STD_U16BE	H5T_NATIVE_UINT16
DFNT_LINT16	H5T_STD_I16LE	H5T_NATIVE_INT16
DFNT_LUINT16	H5T_STD_U16LE	H5T_NATIVE_UINT16
DFNT_INT32	H5T_STD_I32BE	H5T_NATIVE_INT32
DFNT_UINT32	H5T_STD_U32BE	H5T_NATIVE_UINT32
DFNT_LINT32	H5T_STD_I32LE	H5T_NATIVE_INT32
DFNT_LUINT32	H5T_STD_U32LE	H5T_NATIVE_UINT32
DFNT_FLOAT32	H5T_IEEE_F32BE	H5T_NATIVE_FLOAT
DFNT_LFLOAT32	H5T_IEEE_F32LE	H5T_NATIVE_FLOAT
DFNT_FLOAT64	H5T_IEEE_F64BE	H5T_NATIVE_DOUBLE
DFNT_LFLOAT64	H5T_IEEE_F64LE	H5T_NATIVE_DOUBLE

HDF5 has many other Datatypes as well.

...create and use “String” data?

String attributes/datasets are created differently in HDF4 than they are in HDF5.

In HDF4, you typically used the DFNT_CHAR8 datatype to create the attribute/dataset and then it showed up as a string when viewed with a tool for reading HDF files. In HDF5, there is a similar datatype, H5T_NATIVE_CHAR, but if you use this then your data will NOT show up as a string with HDF5 tools. It will show up as individual characters. If you wish to use strings in HDF5, then you should use the H5T_C_S1 datatype. Figure 20 shows example code.

```
hid_t     file_id, dataset_id, space_id, stype, attr_id, grp_id;
herr_t    status;
size_t    size;

space_id = H5Screate (H5S_SCALAR);
stype = H5Tcopy (H5T_C_S1);
size = 21;                    /* Size of string for this example */
status = H5Tset_size (stype, size);  /* Set the length of the string */

/* Then use "stype" when calling H5Acreate or H5Dcreate.
   For a dataset it might look as follows:
      dataset_id = H5Dcreate (file_id, "Sdata", stype, space_id, H5P_DEFAULT);

   For an attribute it might look as follows:
      attr_id = H5Acreate (grp_id, "Adata", stype, space_id, H5P_DEFAULT);
*/
```

Figure 20

HDF5 also supports variable length data elements, including variable length strings. Figure 21 shows example code to write an array of C strings as an HDF5 Dataset with variable length data. Please see the HDF5 documentation for more details of how to manage variable length data.

```
    const char *wdata[3];
    hsize_t       dims1[] = {3};


    sid1 = H5Screate_simple(3, dims1, NULL);

    tid1 = H5Tcopy (H5T_C_S1);

      /* special: H5T_VARIABLE for strings */
    ret = H5Tset_size (tid1,H5T_VARIABLE);

    dataset=H5Dcreate(fid1,"Dataset1",tid1,sid1,H5P_DEFAULT);

    wdata[0] = “chocolate”;
    wdata[1] = “vanilla”;
    wdata[2] = “butterscotch”;

    ret=H5Dwrite(dataset,tid1,H5S_ALL,H5S_ALL,H5P_DEFAULT,wdata);
```

Figure 21

...create a Chunked/Compressed Dataset?

The storage properties, including chunking and compression, are controlled with an HDF5 Property List. The Storage Properties for a Dataset are set when the Dataset is created (H5Dcreate). Note that in HDF4, the Dataset is created first, and then the storage properties are set. In HDF5, the storage properties are set (in the property list) and then the Dataset is created.

Figure 22 shows an example of a chunked and compressed SDS. Figure 23 shows a similar chunked and compressed HDF5 dataset.

```
int32 comp_type;   /* Compression flag */
HDF_CHUNK_DEF c_def;   /* Chunking definitions */

  sds_id = SDcreate (sd_id, “sds”, DFNT_INT32, 2, dim_sizes);

  c_def.chunk_lengths[0] = 3;
  c_def.chunk_lengths[1] = 2;

  c_def.comp.comp_type = COMP_CODE_DEFLATE;
  c_def.comp.cinfo.deflate.level = 6;
  comp_flag = HDF_CHUNK | HDF_COMP;

  status = SDsetchunk (sds_id, c_def, comp_flag);

/* write and read the dataset… */
```

Figure 22

```
hid_t       create_props;
hsize_t      chunk_dims[2] = { 3, 2};

dataspace = H5Screate_simple (2, dims, maxdims);

/* set  dataset creation properties */
create_props = H5Pcreate (H5P_DATASET_CREATE);
status = H5Pset_chunk ( create_props, 2, chunk_dims);  /* set chunking */
status = H5Pset_deflate (create_props, 6);                    /* set gzip compression */

/* Create a new dataset within the file using  the creation properties.  */
dataset = H5Dcreate (file, "/dataset", H5T_NATIVE_INT, dataspace, create_props);

/* write and read the dataset … */
```

Figure 23

For more information see the HDF5 tutorial for an example:

http://hdf.ncsa.uiuc.edu/HDF5/doc/Tutor/extend.html - Example using Chunking

...read/write a sub-set or sub-sample of a Dataset?

HDF5 supports a very flexible and general set of data selection features, which control both the source and destination. These are implemented by specifying selections on the source and destination Dataspace for a read or write.

HDF5 supports:

Hyperslabs, including repeated blocks as well as strides
Unions of hyperslabs
Sets of points

See the HDF5 tutorial for examples:

http://hdf.ncsa.uiuc.edu/HDF5/doc/Tutor/select.html - Hyperslab selection
http://hdf.ncsa.uiuc.edu/HDF5/doc/Tutor/selectc.html - Point selection

...create/read Dimensions and Dimension Scales?

HDF5 does not currently support Dimensions or Dimension Scales. This will be added in 2003.

For More Information

Information and HDF software is available from:

http://hdf.ncsa.uiuc.edu

Questions and problems should be sent to:

hdfhelp@ncsa.uiuc.edu

Acknowledgements

This report is based upon work supported in part by a Cooperative Agreement with NASA under NASA grant NAG 5-2040 and NAG NCCS-599. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Aeronautics and Space Administration.

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

References

1. Mike Folk, Robert E. McGrath, and Kent Yang, *Mapping HDF4 Objects to HDF5 Objects*, National Center for Supercomputing Applications, University of Illinois, July, 2002. http://hf.ncsa.uiuc.edu/doc/ADGuide/H5toH5Mapping.pdf

2. "HDF (4.x) and HDF5", http://hdf.ncsa.uiuc.edu/h4toh5/

3. "HDF5 Tools", http://hdf.ncsa.uiuc.edu/HDF5/doc/Tools.html#Tools-Dump

4. "H4toH5 Conversion Library", http://hdf.ncsa.uiuc.edu/h4toh5/libh4toh5.html

5. "HDF5: High Level APIs", http://hdf.ncsa.uiuc.edu/HDF5/hdf5_hl/doc/

6. "HDF Specification and Developer's Guide v4.1r5", ftp://ftp.ncsa.uiuc.edu/HDF/HDF/Documentation/HDF4.1r5/Spec/

7. "Image and Palette Specification", http://hdf.ncsa.uiuc.edu/HDF5/doc/ADGuide/ImageSpec.html

8. "HDF5 Table Specification", http://hdf.ncsa.uiuc.edu/HDF5/hdf5_hl/doc/RM_hdf5tb_spec.html

9. "HDF5 File Format Specification", http://hdf.ncsa.uiuc.edu/HDF5/doc/H5.format.html

Appendix 1: Why HDF5?

Versions 1-4 of HDF maintained backward compatibility with earlier versions of the format and library, dating back to 1988. Why is HDF5 a new and incompatible format and library?

Despite the success and widespread use of HDF4, analysis of HDF4 showed a number of shortcomings, including:

- limits on object and file size (< 2GB)
- limits on the number of objects in a file (< 20K)
- rigid data models
- I/O performance issues

In addition, new demands and requirements have emerged, including:

- Bigger, faster machines and storage systems
    - Massive parallelism, teraflop speeds
    - Parallel file systems, terabyte storage

- Greater complexity and power features
    - Complex data structures (including variable length data, meshes, etc.)
    - Complex subsetting, sampling, etc.
    - Support for data transformations
    - Easier to extend, e.g., with custom compression
- Emphasis on remote and distributed access

After careful study, it was clear that these shortcomings and requirements could not be met with the HDF4 format and library. It was necessary to start fresh, to build a new format and library based on the lessons learned from HDF4.