HDF4 to HDF5 Conversion Library: Programmers Notes

Kent Yang
Robert E. McGrath
January, 2002 (Revised June, 2002)

1. Introduction

The *h4toh5* library is a C library that converts individual objects from an HDF4 file to equivalent objects in an HDF5 file, following the default mapping defined in the specification document, *Mapping HDF4 Objects to HDF5 Objects* [1]. The *h4toh5* library uses both the HDF4 and HDF5 libraries.

The Users Guide and Reference Manual explain the API and how to use the *h4toh5* library [2,3]. This document presents more detailed information about the *h4toh5* library, including implementation decisions and advice for advanced users. This document assumes the reader is familiar with both HDF4 and HDF5 and the other documentation referenced here.

Section 2 discusses some design decisions in the implementation, what parameters to pass to the library, and how to deal with the structure of HDF4 files. Alternative implementations are explained and analyzed.

Section 3 and 4 discuss the conversion of objects with auxiliary objects, SDSs with Dimension scales and Images with palettes. Section 5 explains some difficult cases of file structures in which the Groups have a loop. We explain the algorithm used in the H4 to H5 conversion library.

Section 6 explains how number types must be converted between HDF4 and HDF5. Section 7 tells how chunking and compression are mapped.

Section 8 gives the algorithm for generating default names for HDF4 objects that do not have any name.

2. Implementation Choices

The *HDF4 to HDF5 Mapping Specification* [1] is the guideline for details of the conceptual mapping of HDF4 objects to HDF5 objects. This section will discuss two design choices for implementations: what input parameters to use; and the simplicity or complexity of the analysis of the file structure.

2.1. Input parameters

The *h4toh5* library API needs to be passed a reference to specific HDF4 objects to be converted, such as an individual SDS array. There are several forms of reference that might be used in this API. Similarly, the output HDF5 object must be specified; and there are several ways this might be done.

In HDF4 objects are uniquely identified by their reference and tag; which are used as keys to locate an HDF4 object internally. However, most users use the higher level interfaces (SDS, Raster Images, etc.), which use identifiers for the higher level objects of interest. It would be difficult, but possible, to use reference and tag as input parameters to the *h4toh5* library.

Another possible choice is to use HDF4 object names as input parameters. However, the object name is not always available (HDF objects do not have to have a name) and the object name is not unique (several objects may have the same name). For example, an HDF4 SDS may not have a name, and two different SDS objects may hold the same name. Clearly, to use HDF4 object name as the input parameter may cause ambiguity.

The current implementation of the h4toh5 library uses object identifiers returned by the HDF4 interfaces, not tags and references. Thus, an SDS is passed to the *h4toh5* library by the identifier returned by *SDselect()*.

The input HDF5 parameter can be either HDF5 object name or HDF5 object id (e.g., as returned from *H5Dopen()*). Also, the object name is actually a path, so it may be specified in a single absolute path or relative to a specified group.

We choose to use two parameters: HDF5 absolute path of the group and relative path of the dataset to the group to represent the HDF5 object. The reasons are as follows:

a) The combination of the absolute group path and relative path of the dataset to the group are equivalent to HDF5 object id.
b) H4toH5 conversion library users may not be familiar with HDF5 library; it is possibly inconvenient for them to use a series of HDF5 objects Open or Create calls to obtain HDF5 object id. To provide absolute group path and relative dataset path to the group is straightforward.
c) Users may want to keep the name of the converted HDF4 object as the HDF5 object name. Conversion library cannot automatically pass the default HDF4 object name by using HDF5 object id.

2.2. Dealing with the Structure of HDF4

An HDF4 file may have a very complex structure, with multiple objects of the same name and objects shared between multiple groups. The *h4toh5* library may deal with some or all of these issues, or leave them to the calling program, or some combination. This creates a design trade-off. A simple API would convert one object at a time, using

simple default rules, but this cannot handle complicated structures. A more comprehensive API covers the difficult cases, but requires a more complex library and API.

2.2.1. Simple conversion library

At the first glance, design of the HDF4 to HDF5 conversion library seems straightforward. The conversion library receives the input HDF4 object id and HDF5 names, and follows the *HDF4 to HDF5 Mapping* [1] to do the conversion object by object. This design is simple to understand and simple to use.

Detailed investigation of the HDF4 conventions and HDF4 and HDF5 file structures reveals that there are pitfalls that can result in incorrect or unexpected results.

2.2.1.1. Duplicate object names in different Vgroups

In some cases, the conversion library may not convert a legal HDF4 object that should be converted. One way this can happen is when two different HDF4 objects have the same name, and are to be stored in the same Group in HDF5.

Example1:

Two different SDS objects with the same name Figure 1 shows an example with two SDSs, both named **Uwind** under Vgroups **JAN** and **FEB**.

These two SDSs could be converted to HDF5 datasets with two calls the *h4toh5* library. However, if the user requests to store both datasets with their default name in the same group in the output file, this is a conflict. The simple conversion library will generate an error when the second SDS is converted.

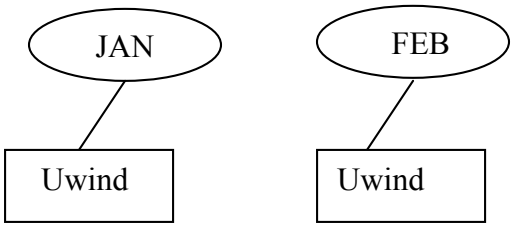

Figure 1 Two SDS objects (name: Uwind) hold the same name but belong to two Vgroup JAN and FEB.

Presumably, the second SDS should be converted since it does not violate HDF4 conventions, but it needs to be assigned a non-default name in the HDF5 file.

If there is no way to control this behavior of the *h4toh5* library, the user would have to go back to the original HDF4 file to change the name (if this is even possible) and redo the conversion. To figure out where and why the error happened would also be difficult and time consuming.

2.2.1.2. Aliasing, shared objects

If an object is a member of two Vgroups, the conversion library may convert the same HDF4 object more than once.

Example2:

Two Vgroups share the same SDS object. Figure 2 shows the layout of an HDF4 file with two Vgroups (**JAN** and **FEB**) that share a single SDS array, **Height**.

The members of the two Vgroups can be converted one at a time by a series of calls to the *h4toh5* library, placing their members in two HDF Groups. In this case, the SDS called Height would be converted twice, and duplicated in a second HDF5 group instead of being shared as in the original.

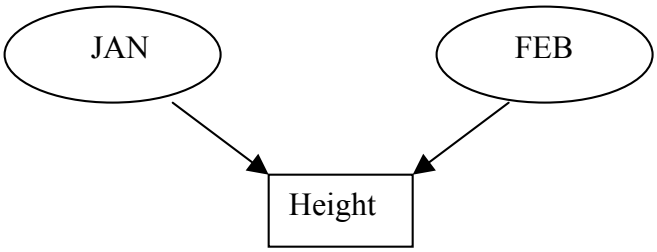

Figure 2. One SDS (Height) shared by two Vgroup JAN and FEB.

Figure 3 shows the converted HDF5 file structure.

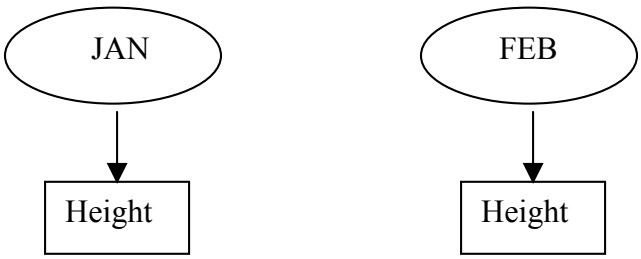

Figure 3. The converted HDF5 file structure of HDF4 file described in Figure 2 following the simple conversion library.

A simple conversion of the individual objects cannot detect that the SDS "Height" is shared by two Vgroups. Clearly, the intended result is that the HDF5 dataset 'Height' should also be shared by group **JAN** and group **FEB** in new HDF5 file.

The consequences of this behavior can be significant:

- The HDF5 file can have an inconsistent file structure compared to the original HDF4 structure.
- The HDF5 file can be unexpected large.
- The conversion can take unexpectedly long and use much more memory than expected.

2.2.1.3. Conclusion: Extra HDF4 or HDF5 calls will be needed to do the conversion

These examples have show that simply converting object by object may fail in common situations. The application program must implement several additional steps before calling the *h4toh5* library. The additional steps must include:

- Create or open an HDF5 group with the input group name.
- If the original HDF4 object name is used, extra HDF4 APIs to retrieve the object name is needed.

For example, in the case shown in Figure 1, the user wants to convert the **Uwind** under Vgroup **JAN** to group **JAN** with the same name Uwind.

The calling program would have to do the following HDF4 and HDF5 calls to successfully convert the object.

- find the Vgroup name of which the SDS is a member (JAN)
- open an HDF5 group called JAN
- find the name of SDS object to be converted
- Call the h4toh5 library do the conversion
- Close the HDF4 and HDF5 objects

Other conversions require similar steps. It should also be noted that these steps are not necessarily trivial. For example, the target HDF5 group may or may not already exist, and the HDF4 objects may or may not have unique names.

2.2.2. A more comprehensive conversion library

Alternatively, the *h4toh5* library can be made more complex, so that it tracks the structure of the HDF4 and HDF5 file and therefore can detect and handle the kinds of problems described above. The conversion library will keep internal tables inside the library to keep track of all HDF4 and HDF5 object names that the library has been handled.

This implementation assures that,

1) The conversion library will successfully convert every "legal" HDF object specified at "Mapping HDF4 to HDF5 document" to an HDF5 object by following the name conventions of the Mapping HDF4 to HDF5 document [1]. See Section 8 below.
2) For shared HDF4 objects, the conversion library will also convert to shared HDF5 objects. The file structure can be maintained and the performance of conversion will be better compared with the simple conversion library.
3) Users do not have to make as many HDF4 and HDF5 calls to do the conversion. The calling program only needs to call open/create and close.
4) The conversion library may also help to check whether the name that the user specified has been used. If yes, it will also generate an error before further HDF5 functions are opened.

The *h4toh5* library implements such a "comprehensive conversion library". The most visible trade-off is that two extra functions (**h4toh5open** and **h4toh5close**) are added to initialize and delete the tables, and one extra parameter (*h4toh5id*) is used in each API function to pass the tables.

3. HDF4 SDS and Dimensions

In HDF4, SDS can have dimensions, which are associated with it by the HDF4 library. On the other hand, dimensions can be regarded as standalone objects (actually they are SDS datasets).

The conversion library will provide separate APIs to convert SDS and dimensions. Users can choose to convert only SDS objects or to convert SDS objects and their dimensions. The connection between SDS objects and dimensions in the HDF5 file is through object references from SDS object dataset to dimensional scale datasets.*

To fulfill different purposes and requirements, conversion library supports various options in the process of converting from SDS objects to HDF5 datasets.

1) SDS and the associated dimensions are both converted to HDF5 datasets simultaneously

   The dimensions are stored as HDF5 datasets and are connected through object references from SDS dataset to dimensions. This is the most common case the conversion library is used to convert an SDS object.

* The current implementation of the H4toH5 Conversion Library does not use HDF5 dimension scales, which became available in HDF5 Release 1.8.0; it still uses the schema for storing dimension scales described *HDF4 to HDF5 Mapping* [1]. Future versions of the H4toH5 Conversion Library may use HDF5 dimension scales.

2) SDS only is converted to HDF5 dataset. Dimensions are not required to be converted.

3) One SDS object dimension is converted to an HDF5 object. The converted dimension is regarded as a regular HDF5 dataset.

4) Followed by 3), the one dimension object converted from HDF4 has been connected to another HDF5 dataset via object reference. This HDF5 dataset may be a SDS-equivalent HDF5 dataset.

5) All dimensions of the SDS object can be converted to HDF5 objects without converting the SDS.

4. Images and Palettes

The relationship between HDF4 raster images and palettes is comparable with SDS and dimensions. An image (GR, 24-bit and 8-bit raster image) can have a palette to associate with it. In the HDF5 file, an object reference will be used to connect Image with palette, following the *HDF5 Image Specification* [4].

HDF provides three options to represent the color components in 24-bit raster images. These options consist of pixel interlacing, scan-line interlacing and scan-plane interlacing (See section 7.2.2.2 of *HDF4 User's Guide* [5]). HDF5 currently supports pixel interlacing and scan-plane interlacing [4]. So scan-line interlacing will be converted to pixel interlacing by default.

5. Vgroups

A single HDF4 Vgroup can be directly mapped to HDF5 group. The same object may belong to the multiple Vgroups, and may even belong to the same Vgroup more than once. So the structure of an HDF4 Vgroup can be very complicated; conceptually the membership of a Vgroup can form a directed graph, which may have cycles. The conversion should attempt to reproduce the structure in the HDF5 file.

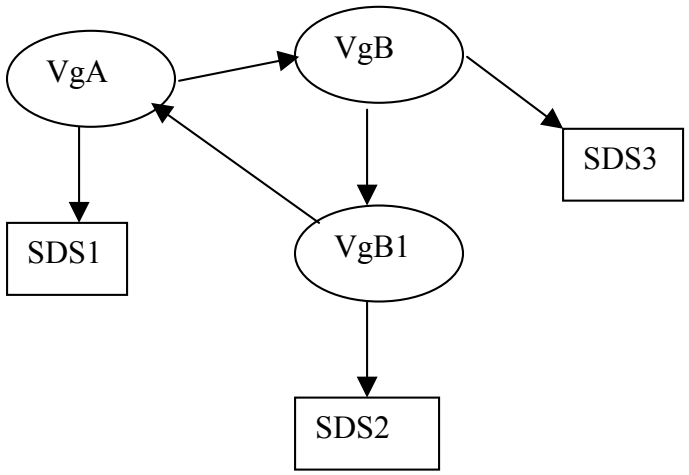

Figure 4: Vgroup conversion example

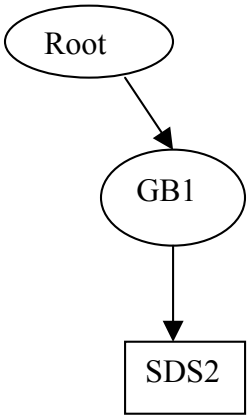

Figure 5. The expected HDF5 file converted from the example shown in Figure 4.

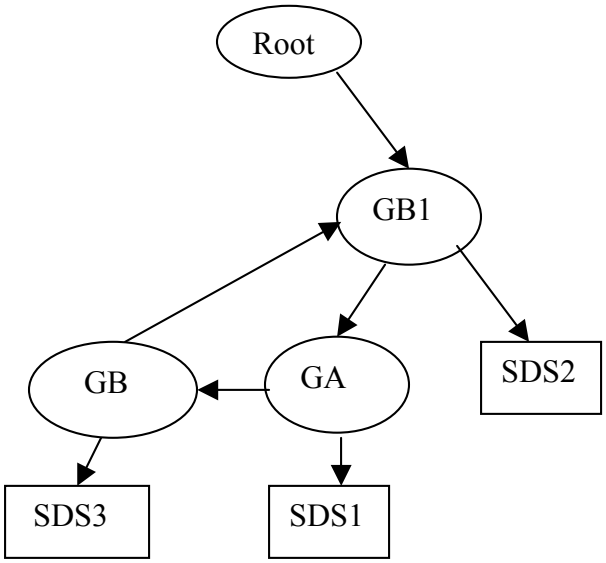

Figure 6. The HDF5 result converted from HDF4 example shown in Figure 4.

The conversion from HDF4 Vgroup to HDF5 group should be very clearly specified; otherwise the results might be unexpected. For example, suppose that the conversion of a Vgroup recursively converts all of its members, including any Vgroups and their children. This "deep copy" can produce unexpected results.

For example, Figure 4 shows the structure of an HDF4 file with three Vgroups (**VgA**, **VgB**, and **VGB1**), and three SDS datasets (**SDS1**, **SDS2**, and **SDS3**). Supposed the user

wants to convert Vgroup **VgB1** and its members; to a group **GB1** under the HDF5 root group. What objects should be created in the HDF5 file?

Often, the user would expect to convert **VgB1** and **SDS2** to HDF5 objects as shown in Figure 5. However, since **VgB1** connects with **VgA** and **VgA** connects with **VgB**, the final conversion result will end up with the conversion of the whole HDF4 file (Figure 6).

In the above example, the conversion library correctly follows the request to do the conversion recursively. The result, however, is not what the user expects.

To avoid such confusions, we provide three options when converting a Vgroup.

Option 1: Only the current Vgroup is converted to an HDF5 group. No members are converted.

Option 2: Convert the Vgroup and only the non-Vgroup members are converted to HDF5 objects. Vgroups in the current Vgroup are not converted.

Option 3: Convert the Vgroup and every member (including Vgroups and their members) to HDF5 objects. Users should be extremely cautious when choosing to use this option!

If we are using Option 2 to convert the HDF4 example shown in Figure 4, we can generate a user-expected output shown in Figure 5. However, by using option 3, the file structure cannot be preserved. The final result will be like Figure 6.

So we strongly recommend users to choose Option 1 and Option 2. Choosing Option 3 only when you know the HDF4 file structure.

It should be noted that using these options in different orders will lead to different results if no further convention is provided. For example, consider the following two sequences of conversions of the HDF4 file shown in Figure 4:

Sequence 1:

1. Convert **VgB1** following the option 1. Only **VgB1** is converted. **SDS2** is not converted.
2. Convert **VgB** following the option 2. **VgB** and **SDS3** are converted.
3. Find Vgroup members of **VgB** and convert these Vgroup members following the option 2. While checking **VgB1**, we find **VgB1** has already been visited, so only hard link from **VgB** to **VgB1** is created. **SDS2** is not converted. The final conversion result is shown in Figure 7.

Sequence 2

1. Convert **VgB** following the option 2. **VgB** and **SDS3** are converted.
2. Find Vgroup members of **VgB** and convert these Vgroup members following the option 2. **VgB1** is found. **VgB1** and **SDS2** are converted.
3. Convert **VgB1** following the option 1. A hard link is created. The final conversion result is shown in Figure 8.

Comparing the results in Figure 7 to Figure 8, we find the conversion results are substantially different using the two different orders of conversion, even with the identical input file. Such inconsistency can be avoided if we consistently use an extra convention.

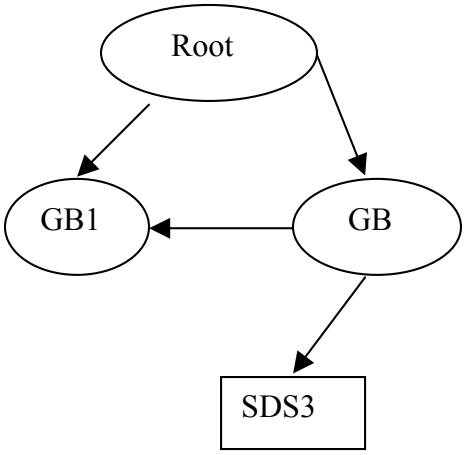

Figure 7. The HDF4 conversion result following Sequence 1.

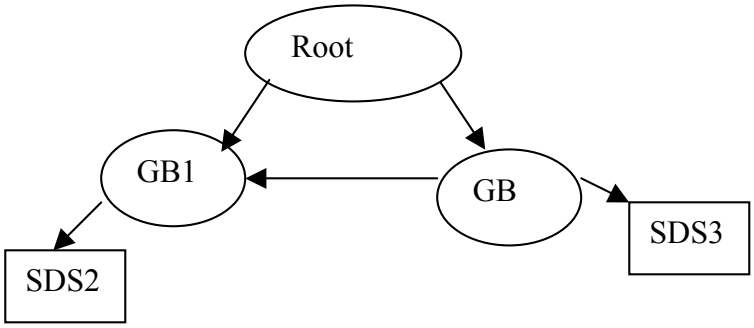

Figure 8. The HDF4 conversion result following Sequence 2.

To minimize these problems, we recommend using the *h4toh5* library in conventional orders. One convention would be to demand the same option for all conversions. Another convention would be to check the option number of the converted Vgroup and change the option number to the highest number if a conflict happens.

In the above example, when following the order of Sequence 1 to do the conversion, at step 3, when checking the option number, we find although **VgB** has been visited before, but the option number is 1. However, the current option number is 2. So we still have to

check whether **VgB** has any non-Vgroup members and convert them into HDF5 if we find any. Under this case, **SDS2** will be converted and put it under group **GB1**. The conversion result will be like that shown in Figure 8.

While following the order of Sequence 2 to do the conversion, at step 3, the current option number is less than the previous number, so nothing is changed.

6. Data Type Conversions from HDF4 to HDF5

Both HDF4 and HDF5 are portable file formats, they ensure that numbers are represented correctly when moved from machine to machine [5]. An HDF4 file is stored in big-endian order in disk by default. It can be stored in little-endian order by setting an option. HDF4 only allows the user to specific the data type of the HDF4 data stored in disk, so called file data type. It will automatically convert the file data type to native data type of the machine that handles the HDF file and store the data to memory (so called memory data type). This memory data type is hidden from the user. While writing the data to the disk, HDF4 will also automatically match data type and data type size stored in disk. Users don't have much control over the operations of HDF4 file. HDF4 will automatically adjust data type size in memory when running on different machines [5].

HDF5 provides substantially more flexibility by giving users more control over data type. The memory data type is a parameter that the user can define or change according to purposes of applications. HDF5 will also make sure the data type size correct in memory when running on different machines [6]. HDF5 supports a superset of the datatypes supported by HDF4.

When converting from HDF4 to HDF5, the conversion library must attempt to faithfully keep the original HDF4 type, size, and byte order to store the number as an equivalent HDF5 type, size, and byte order. The HDF4 to HDF5 Mapping Specification give the equivalent types ([1], Section 5.5).

1) The memory data type of HDF5 converted from HDF4 should always be NATIVE since that is how HDF4 installed the data.
2) Numbers should be represented correctly while converting HDF4 to HDF5 across platforms. The conversion library will find the exact size of HDF4 memory data type on the platform it is running and pass this information to HDF5. This information is essential for
    - Correctly allocate the memory space for data
    - Find the correct HDF5 native data type on the platform
    - Memory data type and data size have to be calculated correctly for every field of HDF4 Vdata and these have to be fed into the corresponding HDF5 compound data type in order

7. Chunking and compression

The *h4toh5* library will always preserve the same chunking and compression features of the HDF4 object in the converted HDF5 file as long as HDF5 supports these features. HDF4 supports various compression methods including simple RLE, NBIT, skip Huffman, gzip and JPEG. Currently HDF5 only supports gzip compression method. The current implementation of the conversion library is as follows:

1) If the conversion library finds the HDF4 object is in gzip compression, conversion library will keep the gzip compression information. If it is the other compression in HDF4 file, conversion library will ALWAYS use gzip compression in HDF5 and set the compression level to 9.
2) Conversion library will keep the same chunking information in the HDF4. For extensible HDF4 objects (with an unlimited dimension) HDF4 file, the HDF5 dataset must be chunked ev3en if the HDF4 object was not. In this case, a default chunk size has to be provided when converting to HDF5.

8. Object Name Conventions

HDF4 uses object reference and object tag to distinguish objects, "names" are optional attributes. Two different HDF4 objects can have the same name and an object may not have a name at all. All HDF5 objects, however, must have a name, although the same object may have several names.

While doing the HDF4 to HDF5 conversion, we have to make a smooth transition from HDF4 to HDF5 when a name conflict is discovered. In other words, we have to find a unique representation of the HDF4 object name if the name is used more than once or no HDF4 name is found.

The current implementation of the conversion library generates names using the following convention:

<FILE TYPE>_<OBJECT TYPE>_<ref>

Where FILE TYPE is always {HDF4} and OBJECT TYPE is one of
{SDS, VGROUP, PALETTE, VDATA, IMAGE, DIMSCALE}

For example, an HDF4 palette with reference number '12' would be assigned the name:

HDF4_PALETTE_12.

HDF4 annotations, there could be either a "file label" or a "file description" with the same name, so the convention is:

<FILETYPE>_<OBJECTTYPE>_<ANNOTATIONTYPE>_<ANNOTATIONINDEX>

where ANNOTATIONTYPE is either “LABEL” or “DESC”.

For example, an SDS label annotation with index number ‘2’ would be:

HDF4_SDS_LABEL_2.

This name convention is a simple way to assign a unique default name to any HDF4 object. However, the names generated are arbitrary and may be confusing.

It may be noted that the *h5view* Java utility can be used to edit the output file, to change the names of HDF5 objects or attributes [7].

Acknowledgments

This report is based upon work supported in part by a Cooperative Agreement with NASA under NASA grants NAG5-2040 and NCC5-599. 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.

References

1. *Mapping HDF4 Objects to HDF5 Objects*:
http://hdfgroup.org/HDF5/doc/ADGuide/H4toH5Mapping.pdf

2. *H4TOH5 Conversion Library API User's Guide*,
http://hdfgroup.org/h4toh5/h4toh5lib_UG.html

3. *H4TOH5 Conversion Library API Reference Manual*,
http://hdfgroup.org/h4toh5/h4toh5lib_RM.html

4. *HDF5 Image and Palette Specification*,
http://hdfgroup.org/HDF5/doc/ADGuide/ImageSpec.html

5. *HDF4 Documentation*, http://hdfgroup.org/doc.html

6. *HDF5 Documentation*, http://hdfgroup.org/HDF5/doc/

7. NCSA H5View, http://hdfgroup.org/hdf-java-html/hdfview/

For Further Information

Web: http://hdfgroup.org/h4toh5

Email: help@hdfgroup.org

Last modified: 4 November 2010