Transition from HDF4 to HDF5:

Notes about Compatibility, Conversion, and Interoperability of HDF and HDF-EOS

Robert E. McGrath and the NCSA HDF Group
NCSA
University of Illinois, Urbana-Champaign
mcgrath@ncsa.uiuc.edu
September 11, 2000

1. Introduction

The transition from HDF4- to HDF5-based software is a problem in compatibility (or incompatibility). There are several dimensions of compatibility/incompatibility which may arise. Any given application may face one or more of these issues.

For purposes of this discussion, "compatibility" is used as in common, informal usage. Two different pieces of software are "compatible" if it is easy to use either or both for the user's purposes. Note that this definition of "compatibility" is relative to the needs and goals of users, so there cannot be a single, absolute measure of "compatibility" that is correct for every possible use.

In this paper, we assume that most users are interested in continuing to "whatever they are already doing" with HDF4, and adopting HDF5 with a little work and loss as possible. In particular, this paper is concerned with users who are already using HDF4 and would like to use HDF5 for the same purposes with as little work as possible. This paper is not intended to provide a detailed analysis of user requirements, but we may briefly note some of the many possible ways that people may be using HDF4, including:

- Using HDF data
- Using tools that use HDF
- Writing tools that use HDF
- Designing and delivering data products with HDF

"Compatibility" and transition issues are different for different uses.

Some users may be designing new software and data products, and therefore may choose to implement with HDF5 from the start. Most users will need to continue to use existing HDF4-based software and HDF4 datasets, as well as HDF5. For this reason, practical software will need to deal with the conversion (translation) of data and with the "interoperation" of old and new datasets in the same environment.

Compatibility issues arise not only in the transition from the base HDF library, but also in transition from other software that uses HDF, such as the HDF-EOS library. HDF-EOS Version 1 and 2 is based on HDF4, HDF-EOS Version 3 will be based on HDF5. In the case of HDF-EOS, the higher level library may present its own compatibility issues as well as issues arising from the base HDF libraries and formats it uses, and interactions between the two.

This document will examine transition issues presented by three cases:

1. HDF4 and HDF5 base libraries (i.e., the libraries from NCSA [1])
2. The HDF-EOS library with HDF4 and with HDF5. [4]
3. "Hybrid" HDF-EOS files with HDF-EOS objects and also native HDF objects.

Section 2 briefly defines some of the key dimensions of compatibility issues that must be faced. Section 3 reviews the compatibility of the NCSA HDF4 and HDF5 formats and libraries, and Section 4 reviews the same issues for HDF-EOS Version 2 and Version 3.

Section 5 defines and discusses issues for interoperating multiple versions of HDF and HDF-EOS in a single program. Section 6 discusses conversion of HDF4 files to HDF5, and HDF-EOS Version 2 to HDF-EOS Version 3. Section 7 discusses the challenge of converting "hybrid" HDF-EOS files, those that contain both HDF-EOS and other HDF objects.

Finally, Section 8 sketches approaches to address transition issues.

2. Definitions: Types of Compatibility Issues

Table 1 lists the compatibility issues that will be discussed in this document. These are defined below. This document is not intended to be a complete treatise on the notion of "compatibility". The purpose, rather, is to note some of the more important issues facing HDF users.

Compatibility Issue	**Description**
Conceptual	The overall conceptual approach: the formal and informal models.
Format	Storage layout and other format details.
Software	Compatibility of the software
Programming model	The assumptions and approach to programming.
Features	The set of features supported.
Interfaces	The programming interfaces.

Table 1. Types of Compatibility Issues for HDF5 Transition

2.1. Conceptual

Two file formats or programs are *conceptually compatible* when they model similar objects and similar operations for those objects. For example, there are a great number of formats for storing two-dimensional raster images, including GIF, JPEG, TIFF, and HDF. The format and supporting software are different, but each format supports a very similar concept of an "image" with related color model. This conceptual compatibility allows image data from one format to be translated to and used along with images from other formats.

Note that this does not mean that there is a *perfect* translation, or that these formats have *identical* concepts of images. What it means is that for some uses, these formats provide *similar enough* conceptual models to be used for the same purposes by users. There may, of course, be uses which require details or features of one format that are not available in others. For these users, the formats would not be considered compatible.

2.2. Format

Format compatibility of two formats means that a storage format is either the same, one is a superset of the other, or there is a direct and simple correlation. “Compatibility” is not an either/or proposition, formats might be “partly” compatible (e.g., some files are compatible, others are not). Formats might be “backward compatible”, so that newer versions are supersets of older versions.

HDF4 is *format compatible* with HDF3 and earlier versions. The format of an HDF4 file follows the same basic rules as earlier versions, with additions and extensions not found in earlier versions. In contrast, HDF5 is not format compatible with HDF4 and earlier versions. The organization of an HDF5 file has no relation to the rules of HDF4.

2.3. Software

Software compatibility is a complicated concept. First, two pieces of software may be related or used in different ways, as compiled programs, as libraries to link, or as source code to compile with. Second, the “compatibility” of software has several dimensions, including programming model, features, and interfaces.

Two pieces of software may be compatible as running programs, in which case the two programs may run together, exchange data, and whatever else with no other actions. This sort of compatibility is often seen in commercial products.

Software that is provided as a linkable library can be compatible if a program can link to either of two versions without modifying or recompiling the calling program. In this case, the libraries must export the same compiled interface and otherwise be completely compatibly at link time.

Software may be compatible as source code. In this case, a program which uses one version may be compiled to use the other without substantial change.

Software may be compatible (incompatible) in several ways. Two pieces of software may have different ***programming models***. The programming model is used here to mean the overall approach and style of programming: the way programs and data are to be organized and accessed. When the programming model differs, then the same algorithm may be expressed substantially differently, and a conceptually identical program may have radically different code.

Software may also have different and incompatible ***features***. The features of software are the capabilities and provided, essentially the operations that are supported. When two pieces of software have different sets of features, a program using one piece of software may not even be expressible in another.

Software may be compatible by ***interface***. An interface is a set of visible data types, structures, and protocols (APIs). The “protocols” specify requirements for requests and results. Two different pieces of software may implement a particular interface in different ways, and thus be interface compatible. In this case, other programs can use either software equally well by using the common interface. Note that software may have many interfaces, so may be “partly compatible” by having some needed interfaces.

3. HDF4 and HDF5 Base Formats and Libraries

The HDF4 and HDF5 formats and libraries are conceptually compatible, but otherwise largely incompatible with each other. This section discusses these incompatibilities.

3.1. Conceptual

The HDF5 data model [2, 9, 10] is a *superset* of HDF4 [1]. Every data object that can be represented in HDF4 can be represented by a similar object in HDF5.[3] In addition, the HDF5 data model is much more general than HDF4, and encompasses many possibilities that cannot be represented in HDF4.

The conceptual compatibility ensures that, in principle, every HDF4 file can be transformed into a "semantically equivalent" HDF5 file. (See [3] and section 6 below.) Similarly, any application that uses HDF4 can be transformed to use HDF5.

3.2. Format

The HDF4 and HDF5 *file formats* are incompatible: the organization and layout of the files are completely different in HDF4 and HDF5. [1, 2] There is no relationship between the HDF4 file format and the HDF5 file format. This was a deliberate design decision: the HDF5 file format is designed to support larger files and more objects, and also is very general and flexible. [10]

The difference in file formats is usually not important for applications that use either HDF library, since the application need not access the file directly. Any application that directly accesses an HDF4 file would need to be redesigned to work with HDF5.

3.3. Software

The HDF4 and HDF5 libraries are completely different. The HDF5 library software is incompatible with HDF4 in essentially every way it could be. In particular, the interface is different, so both the source code and binary libraries are incompatible. A program using HDF4 will have to be rewritten to use HDF5.

3.3.1. Programming Model

Many aspects of the *programming model* of HDF5 are substantially different from HDF4. In general, HDF5 is much better designed and engineered than HDF4. However, these differences mean that programmers who know HDF4 must learn a new programming model in order to use HDF5.

For example, HDF4 uses "tags" and "refs" to identify objects, while HDF5 has no such concept. Also, the HDF5 model for defining data types is completely different from HDF4, and HDF5 makes extensive use of run-time type definitions. There are also many differences in practical aspects of the software, such as the configuration and make files.

3.3.2. Features

Most of the features of the HDF4 library and format are available in HDF5, although the details may differ. However, the HDF5 format and library have new features not found in HDF4. [2, 10] Some of the new features of HDF5 are "automatic", such as the ability to have files larger than 2GB, and the ability to store more than 64,000 objects. Other features, such as using MPI/IO, may require substantial changes to the application before they can be used.

3.3.3. Interfaces

The HDF5 API is completely different from the HDF4 API. This reflects HDF5's more general conceptual model, and also the different programming model. While the APIs are incompatible, many HDF4 calls can be replaced with a corresponding series of HDF5 calls, to do an equivalent operation. This is usually a straightforward transformation, but the changes are not necessarily local or simple. It may not be feasible to automatically convert arbitrary HDF4 *code* to HDF5 *code*.

4. HDF-EOS with HDF4 and HDF5

The HDF-EOS library Version 1 and 2.x is based on NCSA HDF4. Version 3 of the HDF-EOS library uses HDF5.[4] HDF-EOS 3 (with HDF5) is incompatible with earlier versions of HDF-EOS in several ways. Some of the incompatibility is due to the incompatibility of HDF4 and HDF5, other aspects are due to differences in the two versions of HDF-EOS. [*]

4.1. Conceptual

The HDF-EOS Version 2 and Version 3 libraries are *conceptually very similar*. The HDF-EOS object model is conceptually identical, and the operations supported are only slightly different. Any program that uses HDF-EOS objects can use either version of the HDF-EOS library without conceptual redesign.

4.2. Format

HDF-EOS Version 2 stores its objects as a set of HDF4 objects, Version 3 stores its objects as a set of HDF5 objects. (See the discussions in Section 5, 6, and 7, below.) Therefore, the format of the files is incompatible, because the HDF4 and HDF5 formats are incompatible.

The representation of HDF-EOS objects in the HDF file has been designed to use HDF5 effectively. [8] The representation of HDF-EOS objects in HDF5 is not necessarily a simple translation of the equivalent HDF-EOS Version 2 (HDF4) file. In particular, it should be noted that the HDF-EOS Version 3 does not follow the defaults specified in the "Mapping HDF4 to HDF5". (See [3], and section 6 below.)

---

[*] This discussion is based on the beta release of HDF-EOS Version 3 from December 1999 and presentations by the developers ([8]).

4.3. Software

The HDF-EOS Version 2 and Version 3 software are very similar, but nevertheless incompatible. Note, too, that because the interfaces are so similar, the two libraries do not interoperate: a program must use one or the other version, but not both. [*]

4.3.1. Programming Model

The programming model for HDF-EOS Version 2 and Version 3 are essentially identical, reflecting the identical conceptual models.

4.3.2. Features

HDF-EOS Version 3 will have a few new features from earlier versions. For instance, HDF5 allows any dimension of a stored array to be UNLIMTED, and HDF-EOS Version 3 probably will allow users to use this feature in some cases.

In many cases, the new features are optional, and will not necessarily affect an existing program. However, when converting an application for Version 2 to Version 3, it would be advisable to review the program to determine if it needs to change the way it uses features, or should use new features.

4.3.3. Interfaces

The HDF-EOS interface for Version 3 is almost the same as earlier versions. However, HDF-EOS Version 2 and earlier used some HDF4 datatypes and constants. These must be changed in the HDF-EOS Version 3 interface, to use equivalent types from HDF5. As a result, a program that wants to convert from HDF-EOS Version 2 to Version 3 will have essentially the same code, but must change in many small ways to use the proper HDF5 types. So, it is not possible to simply relink with the new version of HDF-EOS, it is necessary to re-write the application program source code.

The Version 3 API is extremely similar to Version 2, with many subroutines having the same name and similar arguments. Because of this, it is not possible to use both Version 3 and Version 2 of HDF-EOS in the same program: many of the names collide, causing linkage errors. This means that a given program must use either Version 2 or Version 3 (i.e., either HDF4 or HDF5) *but not both*, as explained in the next section.

5. Interoperation: Using HDF4 and HDF5 in the Same Environment

One kind of compatibility is the ability to use both HDF4 and HDF5 files in the same environment. There are two variations on this:

1. A single package that automatically recognizes and accesses HDF4 and HDF5 files, i.e., a single interface that hides the difference in formats.

---

[*] In contrast, because the APIs are different, HDF4 and HDF5 can be used in the same program with no problem.

2. An application that can access HDF4 files and also access HDF5 files with separate interfaces, and then use the data from each together.

One nearly ideal configuration would be to have a single library or application that automatically reads either HDF4 or HDF5. For example, a visualization program can read images from either HDF4 or HDF5 (as well as other formats). In the case of HDF4 and HDF5, this can be implemented *for specific applications*. But because the conceptual models are so different it would be difficult to create a general-purpose library to read *any arbitrary* HDF4 or HDF5 file.

In the case of HDF-EOS, the idea would be to have a single version of the HDF-EOS library capable of reading either HDF4 or HDF5. The user program would deal with HDF-EOS objects (Grid, Swath, etc.), and would do the same thing regardless of the version of HDF used for the storage. Figure 1 illustrates this type of interoperation for the case of HDF-EOS.

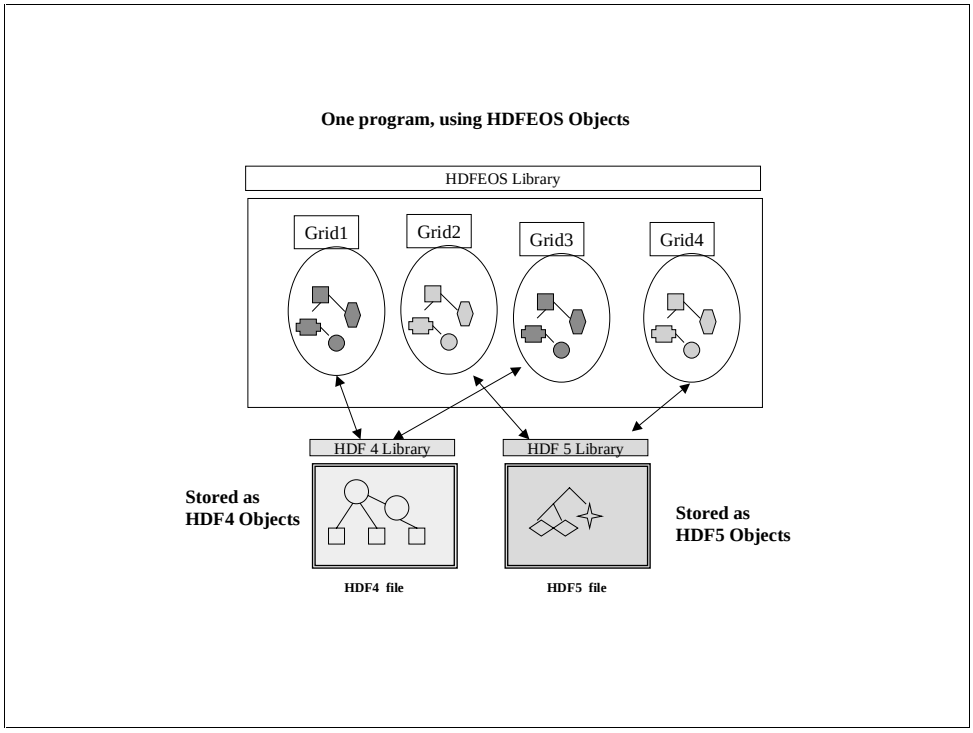

Figure 1. Interoperation of HDF-EOS Files. (*Grid1* and *Grid3* are from HDF-EOS V. 2; *Grid2* and *Grid4* are from HDF-EOS V. 3.)

This kind of interoperability is difficult to achieve with the current HDF-EOS libraries because the API "exposes" HDF4 and HDF5 constructs, such as HDF data type descriptors. These are incompatible, so the equivalent HDF-EOS calls would have to be translated to make sure the arguments are correct for each library. Of course, it would be perfectly possible to create a new HDF-EOS library that translates calls in this way, or has an API that is suitable to both HDF4 and HDF5 (e.g., by defining HDF-EOS types).

Note, though, that "hybrid" files (that contain both HDF-EOS and other HDF objects) would be difficult to handle correctly in this configuration. See Section 7 for a discussion of hybrid files.

6. Conversion of HDF4 files to HDF5 files

Many users will want to create new applications that use HDF5, and then will want to convert older HDF4 files into HDF5 to use the new software. This section discusses the process of translating HDF4 to HDF5.

6.1. Converting HDF4 to HDF5

As explained above, HDF5 is conceptually a superset of HDF4. Every HDF4 object can be represented with an analogous and semantically equivalent HDF5 object. This means that any HDF4 file can be converted to a semantically equivalent HDF5 file. [*] Figure 2 illustrates the conversion process: each HDF4 object is read from the HDF4 file with the HDF4 library, and an equivalent HDF5 object is written to an HDF5 file using the HDF5 library.

While every HDF4 object (file) can be converted to an equivalent HDF5 object (file), there is no single right way to do this. A given HDF4 object might be represented in HDF5 in several ways, each with advantages for some purposes. In order to ease the design of translators and promote standardization, NCSA has defined a specification for a standard, default mapping of HDF4 to HDF5.[3] The NCSA *h4toh5* utility implements this specification, and will convert all objects in an HDF4 file to the default equivalents in an HDF5 file.[7]

6.2. Converting HDF-EOS Version 2 to HDF-EOS Version 3

As explained above, HDF-EOS Version 3 (with HDF5) is conceptually identical to HDF EOS Version 2 (with HDF4). Each HDF-EOS object can be created and manipulated in the same way with either version of the HDF-EOS library, so any HDF-EOS file can be converted from Version 2 to Version 3, and vice versa. Figure 3 illustrates the conversion process. Each HDF-EOS object is read from the HDF-EOS (Version 2) file with the HDF-EOS library and HDF4. An equivalent HDF-EOS object is written to an HDF-EOS (Version 3) file using the HDF-EOS library and HDF5. Raytheon is creating a tool to perform this conversion. [6]

This approach is conceptually simple and is guaranteed to create the correct results. Unfortunately, because the two versions of HDF-EOS have identical APIs, it is not possible to implement this with the standard libraries, because each implements the same names (e.g., each library has a routine called *GDopen()* , etc.). A conversion can be done with a special version of one of the two HDF-EOS libraries, with a modified API that does not have name conflicts. [6]

---

[*] Many, but not all, HDF5 objects and structures can be converted to equivalent HDF4 objects. However, there are many possible HDF5 objects that have no equivalent in HDF4, so it is often not possible to completely convert an HDF5 file to HDF4. Also, some storage options supported by HDF4 (e.g., compression schemes) may not be available for HDF5. Please see [3] for a detailed explanation.

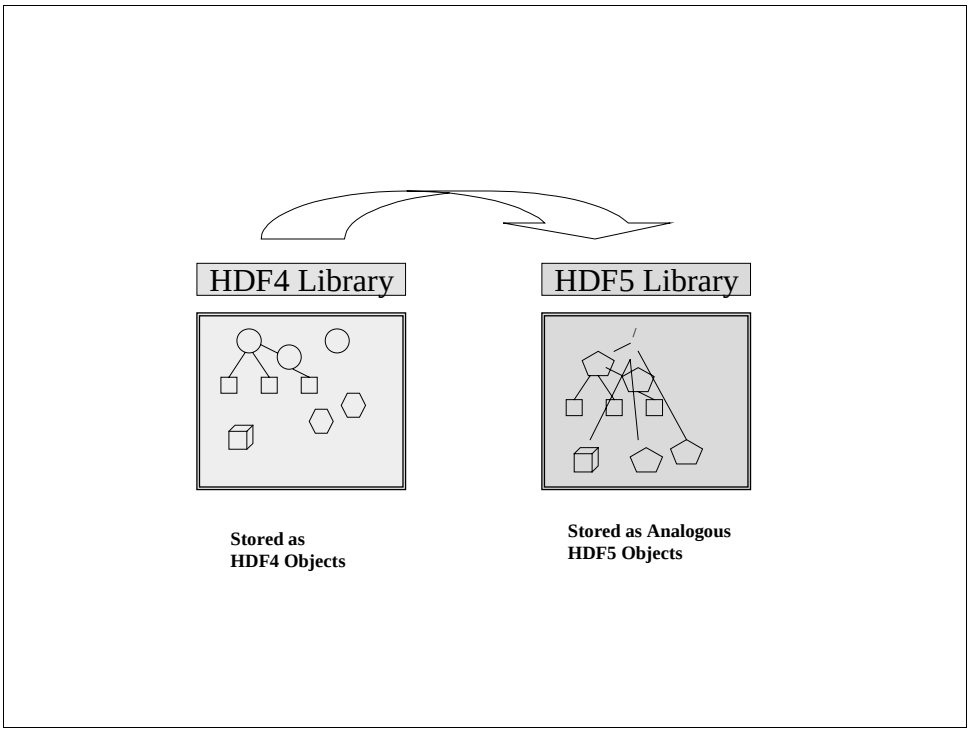

Figure 2. Conversion of HDF4 to HDF5.

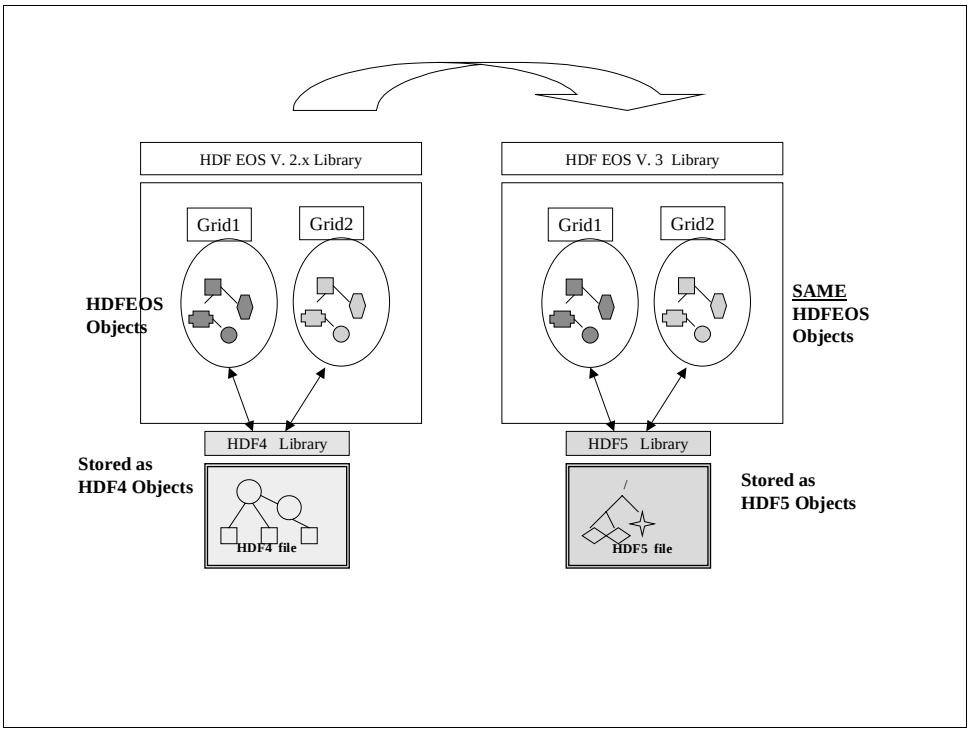

Figure 3. Converting HDF-EOS Version 2 (HDF4) to Version 3 (HDF5).

7. "Hybrid" files that use HDF-EOS and HDF

Applications may use both HDF-EOS Version 2 and HDF4 (or HDF-EOS Version 3 and HDF5), creating both HDF-EOS objects and standard HDF objects (e.g., see Section 7.4 of the "HDF-EOS Users Guide" [5]). In this case, the resulting file is a "hybrid", with some of the HDF objects that are components of the HDF-EOS objects, and other HDF objects directly representing application data. The former can only be accessed via the HDF-EOS library, the latter must be accessed directly through the HDF API.

Figure 4 illustrates a "hybrid" HDF-EOS file. The HDF-EOS objects (Grids, etc.) are stored as aggregates of HDF for objects (datasets, tables, groups, etc.). These objects should only be written and read through the HDF-EOS library.

The application may also create standard HDF objects (images, datasets, etc.). For Version 2 these would be HDF4 objects, for Version 3 these would presumably be HDF5 objects. These objects are unknown to the HDF-EOS library, and are written and read through the standard HDF interface. However, all the objects are stored in the same HDF file. From the HDF file itself, there is not necessarily a way to distinguish which objects are components of HDF-EOS objects, and which ones are separate objects.

Converting a hybrid file requires two kinds of conversion, HDF-EOS Version 2 to Version 3, and HDF4 to HDF5. The HDF-EOS objects must be converted, and the HDF4 objects *not part of HDF-EOS objects* must be identified and converted to HDF5 objects. This can only be done with knowledge of the HDF-EOS library or the application that created the file, or most likely both. The critical point to repeat is that *from the HDF4 file itself* there is no way to reliably determine which objects are components of HDF-EOS objects and which are application defined objects.

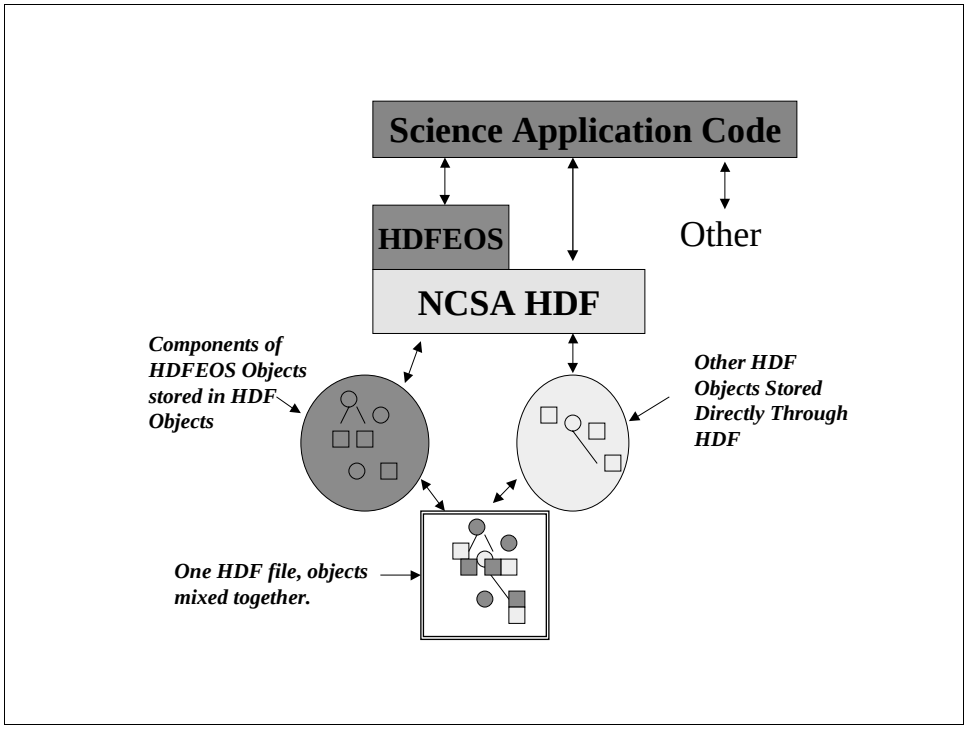

Figure 4. "Hybrid" HDF-EOS/HDF files.

8. Addressing Transition Issues

It should be clear that the transition from HDF4 to HDF5 can raise a variety of challenges, depending on the application and the goals of the users. There is no single technical fix that will meet all these challenges. This section discusses some technical approaches that address some of the transition issues. These may address some of the needs of some applications and users.

8.1. Conceptual Compatibility

Issues of conceptual compatibility affect the design of applications and software. The primary challenge is to map concepts from applications or models to HDF5 concepts. Obviously, there are a great many ways this may be done.

8.1.1. Mapping Application-Specific Concepts to HDF5

For a given application that uses HDF4 (or not), the conceptual model(s) of the application can be mapped to HDF5 concepts, and code can then be written to implement the concepts with the HDF5 library. This provides nearly ideal transition to HDF5—application by application. The HDF-EOS library Version 3 is an example of such an application-specific mapping: the HDF-EOS object model is mapped to appropriate aggregates of HDF5 objects. [8]

Since HDF5 concepts are a superset of HDF4, most applications that use HDF4 can easily map their concepts to HDF5.

8.1.2. Default Standards for Conceptual Mapping

NCSA has defined a specification for a standard default mapping of HDF4 objects and concepts to HDF5. (“Mapping HDF4 to HDF5”, [3]) The mapping specification defines default translations of any HDF4 object to an equivalent HDF5 object. Applications that conform to this standard will be able to exchange converted files, because the HDF5 files converted from HDF4 will be consistent with other conforming HDF5 files.

The default mapping will be adequate for many applications, but for some applications it may be advantageous to design an application-specific mapping.

8.1.3. Design Tools that Capture Expert Knowledge

When mapping from HDF4 to HDF5 concepts, there are many choices to be made. The mapping specification [3] describes some of these choices and provides defaults for many of them. However, for any given application, expert knowledge may be required to use HDF5 to the best advantage. It would be useful to create documentation and/or tools that encapsulate such expertise.

8.2. File Conversion and Interoperability

Many applications simply need to be able to read both HDF4 and HDF5 files into equivalent data structures. Others may need to convert HDF4 files to HDF5, to be able to use HDF5-based tools. The former is “interoperation”, that latter is “conversion”.

8.2.1. HDF5 Readers/Writers

For an application that already reads HDF4, it is usually fairly easy to add similar reading and writing capabilities for HDF5. This requires new software to use the HDF5 library. Note, too, that this usually requires a conceptual mapping between the application and HDF5, as described above in section 8.1.1. Developers should use the mapping specification [3] as a guide for how to design a new HDF5 reader or writer, and standard software, such as described below, may help.

8.2.2. File Translation

For many applications, the best approach is to create new versions of tools that use HDF5, and convert HDF4 file to HDF5. The conversion process was discussed in Sections 5, 6, and 7. In some cases, the conversion may follow the defaults of the HDF Mapping standard [3], others may customize the conversion to suit the application.

The NCSA *h4toh5* tool [7] implements the HDF4 to HDF5 mapping standard with all the defaults. The routines of the *h4toh5* library [7] can be used by applications to convert files in application specific ways. The HDF-EOS Grid and Swath converter [6] is an example of a customized converter, which maps HDF-EOS objects to HDF-EOS objects. (The HDF-EOS converter does not use *h4toh5*, or the defaults of the mapping specification.)

8.3. Source Code Conversion and Interoperation

Another aspect of transition between HDF4 and HDF5 is the conversion of programs. Again, since HDF5 concepts are a superset of HDF4, code that uses HDF4 can be converted to use HDF5 fairly easily. However, the programming model and APIs are rather different, so the conversion is not necessarily trivial or limited to a few lines of code. For example, an operation that takes one call to HDF4 might require a series of calls to HDF5 to accomplish the same operation.

References

1. "The NCSA HDF4 Home Page", http://hdf.ncsa.uiuc.edu/hdf4.html

2. "HDF5 - A New Generation of HDF", http://hdf.ncsa.uiuc.edu/HDF5/doc/RM_H5.html

3. "Mapping HDF4 to HDF5 Objects", http://hdf.ncsa.uiuc.edu/HDF5/papers/H4-to-H5MappingGuidelines.pdf

4. "HDF-EOS and Related Software", http://HDF-EOS.gsfc.nasa.gov/HDF-EOS/SoftwareDist.html.

5. "HDF-EOS Users Guide", http://HDF-EOS.gsfc.nasa.gov/HDF-EOS/hdf.html.

6. Ray Milburn, Personal communication, July, 2000.

7. "H4toh5", in progress, September, 2000. See http://hdf.ncsa.uiuc.edu

8. David Wynne, Personal communication, July, 2000.

9. “HDF Abstract Data Model”, http://hdf.ncsa.uiuc.edu/HDF5/papers/ADM/ADM_EOS_Sep99/EOSpresentation/index.html

10. “Introduction to HDF”, http://hdf.ncsa.uiuc.edu/HDF5/papers/HDF5_overview/index.htm