

Mapping HDF5 to DAP2

Status of This Memo

This is a draft of the technical note on mapping HDF5 objects to DAP2.

Change Explanation

None

Copyright Notice

Copyright © 2010 United States Government as represented by the Administrator of the National Aeronautics and Space Administration. No copyright is claimed in the United States under Title 17, U.S. Code. All Other Rights Reserved.

Abstract

Data Access Protocol Version 2 (DAP2) and Hierarchical Data Format Version 5 (HDF5) are endorsed by NASA Headquarters as recommended earth science data systems standards. A significant number of HDF5 objects directly match DAP2 objects. However, because HDF5 and DAP2 have different data models, it can be difficult to fully map *all* HDF5 objects to DAP2 objects, yet several difficult-to-map HDF5 objects are essential for earth science applications. This technical note presents general HDF5-to-DAP2 mapping information, covering HDF5 objects, datasets, groups, and attributes. Considerable detail is presented for the representation of those crucial, difficult-to-map objects that are required for earth science applications. This document also includes information about special HDF5-to-DAP2 mapping that enables easy access to NASA HDF-EOS5 data via DAP2 clients.

ESDS-RFC-017
Category: Technical Notes
Updates: n/a

Yang, Gallagher, Lee
October 2010
Mapping HDF5 to DAP2

Table of Contents

1	Introduction	4
1.1	Motivation and purpose.....	4
1.2	Overview of HDF5.....	4
1.3	Overview of DAP2.....	4
1.4	How to access HDF5 data via OPeNDAP.....	5
2	Background.....	6
2.1	Introduction to HDF5 objects.....	6
2.2	Introduction to DAP2 objects.....	6
3	General HDF5-to-DAP2 Mapping	7
3.1	HDF5 datatypes that exactly map to DAP2	8
3.2	HDF5 datatypes that do not exactly map to DAP2	8
3.2.1	HDF5 datatypes that are rarely used by Earth Science applications	8
3.2.2	HDF5 datatypes that can be mapped to DAP2 with adaptation.....	9
3.2.2.1	8-bit signed integer	9
3.2.2.2	Object and region references	10
3.2.2.2.1	Object reference	10
3.2.2.2.2	Region reference	12
3.2.2.2.3	Limitations of using the DAP2 URL datatype for references	14
3.2.2.3	Compound datatype	14
3.3	HDF5 datasets	16
3.4	HDF5 groups	17
3.4.1	Evaluations of two options	17
3.4.2	General mapping procedures.....	21
3.4.3	Illustration with an example	21
3.4.4	Other discussions.....	22
3.5	HDF5 attributes	22
4	Special Mapping for HDF-EOS5 Objects	23
4.1	HDF-EOS5 grids	23
4.1.1	Overview	23
4.1.2	1-D projections.....	24
4.1.2.1	Mapping 1-D projection HDF-EOS5 Grid to DAP2 Grid	26
4.1.2.2	Mapping 1-D projection HDF-EOS5 Grid to DAP2 Array	27
4.1.3	2-D projections.....	28
4.2	HDF-EOS5 Swaths	30
4.3	Other issues related to mapping HDF-EOS5 to DAP2	34
4.3.1	Other coordinate variables.....	34
4.3.2	Missing attributes	34
4.3.3	Special characters.....	34
5	Summary.....	35
	Appendix: Displaying an HDF-EOS5 Grid Using IDV.....	38
	Glossary of Acronyms	39
	Informative References.....	39
	Authors.....	40

1 Introduction

1.1 Motivation and purpose

Data Access Protocol Version 2 (DAP2) and Hierarchical Data Format Version 5 (HDF5) are endorsed by NASA Headquarters as recommended earth science data systems standards. Appropriately mapping HDF5 to DAP2 can clearly improve the usability of NASA HDF5 data to be served by DAP2-compliant software. However, mapping HDF5 objects to DAP2 is challenging because they do not share the same data model. Furthermore, earth science data stored in HDF5 files may include special information that also needs to be carefully mapped in order for DAP2-compliant software to easily access the data.

The purpose of this technical note is not to provide a comprehensive mapping from HDF5 to DAP2, but rather to concentrate on the *necessary* mapping from NASA HDF5 data to DAP2. Given this narrower focus, readers should treat the information in this note as one way – but not the only way – to map HDF5 to DAP2. With this in mind, this technical note provides some general HDF5-to-DAP2 mapping information and documents how information specific to earth science, such as the geolocation information inside an HDF5 file, can be mapped to DAP2.

1.2 Overview of HDF5

HDF5 [1; 2] is a general-purpose library and file format for storing, managing, archiving, and exchanging scientific data. It can store almost every kind of scientific data structure, including images, arrays of vectors, structured and unstructured grids, and text. The HDF5 data model includes two primary types of objects, a number of supporting object types, and metadata describing how HDF5 files and objects are to be organized and accessed. The HDF5 file format describes how HDF5 data structures are represented in storage, in memory, or on other media. The format is self-describing in the sense that the structures of HDF5 objects are described within the file.

HDF5 was awarded the “R&D 100” award in 2002, as a “technologically significant” product that can change people’s lives for the better. It is in production use by academia, government, and commercial companies. The HDF Group, a nonprofit corporation, develops, owns, licenses, and distributes HDF5.

HDF5 has been embraced as an important data format for earth science and computational science. Hierarchical Data Format Earth Observing System Version 5 (HDF-EOS5) [3; 4], which is built on top of HDF5, is the primary data format for data from the Aura satellite. HDF5 is being used as the data format for the real-time data products produced from the tri-agency National Polar Orbiting Environmental Satellite System (NPOESS). The new generation of network Common Data Format (netCDF), netCDF-4, is built on top of HDF5.

1.3 Overview of DAP2

DAP2 [5; 6] is also a NASA community standard used to serve data within NASA from many of its data centers and from many other diverse sources and organizations. The Open-source Project for a Network Data Access Protocol (OPeNDAP), a nonprofit corporation, maintains, promotes and evolves DAP (currently DAP2), as well as other related technologies.

Data served using DAP2 are made available using simple descriptions built of basic computer science datatypes. These descriptions are thus nearly universal in their scope. In a typical deployment, DAP2 servers are written to transform local representations of data (i.e., data stored in specific data formats) into the DAP2 data model. Because each data format has been developed in concert with a certain set of requirements, each format requires special attention when developing the mapping between the local representation and the DAP2 representation. Throughout the rest of this technical note, the term ‘OPeNDAP’ represents client-server software that supports DAP2.

1.4 How to access HDF5 data via OPeNDAP

Figure 1 shows how to access HDF5 files remotely via OPeNDAP. The complete data-processing system consists of six components: (1) the remote HDF5 data (in this example, the HDF5 data are HDF-EOS5 files generated from instruments on the NASA Aura satellite), (2) the HDF5-OPeNDAP data handler to map the remote data into DAP, (3) an OPeNDAP data server (e.g., Hyrax), (4) the networking infrastructure to transport the data encoded using DAP2’s data, (5) the client-side DAP2 software to decode the data read from the data server, and (6) a visualization and analysis tool (e.g., IDV) built with the OPeNDAP client library to visualize and analyze the remote HDF5 data.

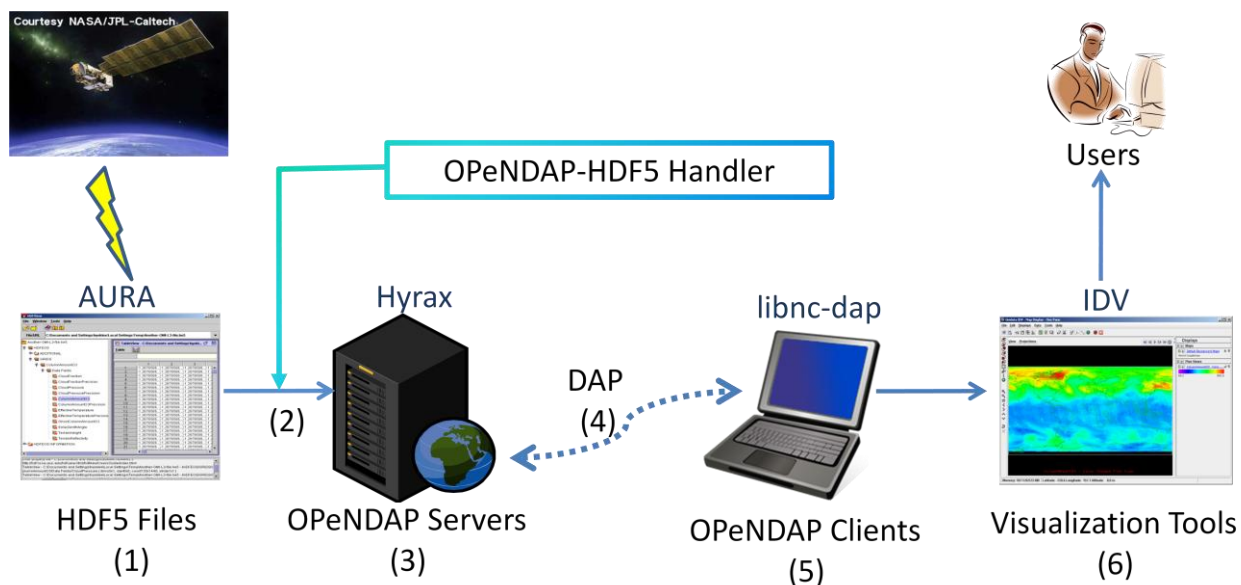


Figure 1. Components of the process for accessing HDF5 files via OPeNDAP

2 Background

2.1 Introduction to HDF5 objects

The HDF5 data model includes two primary types of objects: dataset and group. It also has other supporting object types, including attribute and datatype. Essentially, a dataset object consists of a data array with some metadata information (attributes) that describes the array. The array elements all have the same HDF5 datatype. A group object is used to show relationships among objects. An HDF5 file may be viewed as a directed graph with at least one group, called the root group. An object in HDF5 can belong to different groups, and HDF5 uses links to distinguish with which groups an object is associated.

One can compare HDF5's groups, datasets, and links to a UNIX file system: An HDF5 group is like a directory; an HDF5 dataset is like a file; and a link is like a local path to a directory or a file. A path of an object that starts from the HDF5 root group is called the absolute path of the object; a path that does not start from the root group is called a relative path of the object. Like a UNIX file system, the absolute path of the HDF5 root group is always a slash '/'. Also like UNIX, a '/' is used in the name of a path to separate an HDF5 group or an HDF5 dataset from their parent group.

The HDF5 data model defines a rich collection of atomic datatypes that includes integer, floating-point, string, enumeration, bitfield, array, reference, opaque, and variable-length datatypes. A compound datatype is conceptually similar to a C structure defining a heterogeneous collection of datatypes where each element can be individually addressed. A compound datatype may have any number of members, in any order, and the members may have any HDF5 datatype, including compound. Thus, complex nested compound datatypes can be created. Object reference and region reference are HDF5 datatypes that can be used to access HDF5 objects. In particular, an object reference can be used to access an HDF5 group or dataset, and a region reference can be used to access regions of an HDF5 dataset. More information about HDF5 objects can be found in the *HDF5 User's Guide* [7].

2.2 Introduction to DAP2 objects

The DAP2 data model treats every file it serves as a DAP2 dataset. DAP2 uses two objects to describe a dataset: the Dataset Descriptor Structure (DDS) and the Dataset Attribute Structure (DAS). The DDS describes the dataset's structure and the relationships between its variables; the DAS provides metadata information about the variables themselves. The Dataset Descriptor in XML (DDX) uses an XML notation to represent the combined information of the DDS and DAS.

The DAP2 data model includes atomic datatypes and constructor datatypes. Atomic datatypes include integer types, floating-point types and string types. The DAP2 integer datatypes include 8-bit unsigned char (*Byte*), 16-bit signed short integer (*Int16*), 16-bit unsigned short integer (*UInt16*), 32-bit signed integer (*Int32*), and 32-bit unsigned integer (*UInt32*). The DAP2

floating-datatypes include IEEE 32-bit floating-point (*Float32*) and IEEE 64-bit floating-point (*Float64*).

DAP2 URL is a string containing a Uniform Resource Locator (URL). An OPeNDAP URL can be used to indicate a data file on a remote host machine or, by adding a DAP2 Constraint Expression, to reference either part or all of one or more variables within that file.¹ This feature is used to map the HDF5 object reference and region reference datatypes to DAP2.

DAP2 constructor datatypes include Array, Structure, Grid, and Sequence. In DAP2, the Array and Structure types are conceptually equivalent to the corresponding types in C.²

A DAP2 Grid sets up an association of an N dimensional array with N one-dimensional arrays. These one-dimensional arrays are usually called map variables. The data values of these map variables describe dimensional information of the N dimensional array. In earth science applications, geolocation fields such as latitude and longitude are examples of map variables. For example, a two-dimensional array that stores atmospheric temperature data temp [180][360] and two one-dimensional arrays that store latitude and longitude lat[180] and lon[360] form a DAP2 Grid.

More information about DAP2 Data representation can be found in the *OPeNDAP User Guide* [8].

3 General HDF5-to-DAP2 Mapping

A significant number of HDF5 objects directly match DAP2 objects; therefore, a complete mapping of these matching objects is straightforward. However, because HDF5 and DAP2 have different data models, it can be difficult, or even impossible, to fully map *all* HDF5 objects to DAP2 objects.

Several difficult-to-map HDF5 objects are so crucial for earth science applications that representations of those objects in DAP2 should be provided and made available to the DAP2 clients that need them. The DAP2 representations should allow DAP2 clients that do not understand the HDF5 objects to simply ignore the corresponding DAP2 objects without negative consequences.

The following sections discuss how HDF5 objects are mapped to DAP2. Considerable detail is presented for the representation of difficult-to-map objects that are crucial and required for earth science applications. The mapping of HDF5 objects that are rarely used for earth science

¹ The word ‘file’ used here should not be taken literally since a URL can, in general, be used to reference any web-accessible resource. It happens that most DAP2 servers provide access to data stored in files.

² In DAP2, an Array is an N-dimensional datatype; in C, an array is limited to one dimension, and arrays of arrays of arrays (etc.) are used to represent N dimensions.

applications is not the focus of this document. Some ideas about mapping those rarely used objects are explored. These ideas should only be used as references for future discussions of complete HDF5-to-DAP2 mapping. HDF5 datatype objects are covered first, followed by datasets, groups, and attributes.

3.1 HDF5 datatypes that exactly map to DAP2

Table 1 lists the HDF5 datatypes that map exactly to DAP2. For each of these, the data representation of the DAP2 datatype is the same as the equivalent HDF5 datatype. However, because DAP2 is a transport standard for data, the datatypes use either IEEE or XDR encodings of values to ensure that values will be accurately represented regardless of the particulars such as byte- and word-order of the source or the destination machine. Additionally, HDF5 is a format standard for the storage of data, so its datatype representations more closely match those for a typical C-like programming language. The mappings shown in Table 1 pair datatypes from the two standards and indicate which can hold equivalent values.

Table 1. HDF5 datatypes that exactly map to corresponding DAP2 datatypes

HDF5 Datatype	DAP2 Datatype
8-bit unsigned integer	Byte
16-bit unsigned integer	UInt16
32-bit unsigned integer	UInt32
16-bit signed integer	Int16
32-bit signed integer	Int32
32-bit floating-point	Float32
64-bit floating-point	Float64
String	String

3.2 HDF5 datatypes that do not exactly map to DAP2

The mapping of HDF5 objects that are rarely used for earth science applications is not the focus of this document. Some ideas about mapping those rarely used objects are explored. These ideas should only be used as references for future discussions of complete HDF5-to-DAP2 mapping.

3.2.1 HDF5 datatypes that are rarely used by Earth Science applications

The discussion in this section explores ideas of the following HDF5 datatypes that are rarely used by Earth Science applications:

- Signed integer when the number of bits is greater than 32
- Unsigned integer when the number of bits is greater than 32

- Floating-point when the number of bits is greater than 64
- Bitfield
- Opaque
- Enumeration
- Array
- Variable Length
- Named datatype

For the HDF5 numeric types that use bit-widths larger than those available in DAP2, do not provide mapping to DAP2 since they are rarely used. The mapping will be straightforward if future DAP protocol provides these numeric types.

For BitField and Enumeration, represent them using an integer or integer array and an attribute. An opaque datatype can be mapped to DAP2 byte, and an attribute that marks the type is an opaque type and the number of bytes for this type. There are no corresponding DAP2 objects for the HDF5 array datatype. An HDF5 dataset that has N dimensions with an array datatype (M dimensions) can be mapped to a DAP2 array of M+N dimensions.

We did not encode HDF5 variable length types using DAP2 Sequence because Sequences in DAP2 must support relational constraints. Since variable length types in HDF5 do not, using Sequence would imply behavior that is not actually provided by the data source. However, a 1-D array of variable length string can be mapped to a DAP2 string; 1-D array of variable length string is used by some NASA data.

An HDF5 named datatype is an HDF5 primary object that stores the complete description of the datatype. Since it is rarely used and has no equivalent DAP2 datatype, this document does not discuss the mapping of the named datatype.

3.2.2 HDF5 datatypes that can be mapped to DAP2 with adaptation

HDF5 datatypes that can be mapped to DAP2 include:

- 8-bit signed integer
- Object and region references
- Compound

3.2.2.1 8-bit signed integer

Some NASA HDF5 files have data with an 8-bit signed integer datatype that has no corresponding DAP2 datatype. To access all variables via OPeNDAP, 8-bit signed integers are mapped to DAP2 Int16. The corresponding DAP2 data size will increase due to the increase in the number of bits from 8 to 16. Instead of using the single DAP2's Byte datatype in combination with an attribute indicating signedness, we chose this approach so that clients that are unaware of the attribute can still read the values correctly.

3.2.2.2 Object and region references

HDF5 object reference and region reference types do not have equivalent DAP2 types, but they are essential for future NPOESS HDF5 data. The HDF5 object reference and region reference types are represented in this mapping using DAP2's URL datatype. Because a DAP2 URL can contain a constraint expression, it is possible to use a DAP2 URL to reference either an entire HDF5 dataset or a specific part of an HDF5 dataset. Since the connection between HDF5 reference types and DAP2 URL may not be straightforward, we will use examples to show how HDF5 object reference and region reference datatypes can be mapped to DAP2 URL.

3.2.2.2.1 Object reference

A data element of an object reference points to an entire object in the current HDF5 file by storing the relative file address of the object header for the object pointed to. By dereferencing an object reference, one can retrieve the data (data shape, datatype, etc.) and metadata (attributes) information of the object that the object reference points to.

An HDF5 object reference type is mapped to a DAP2 URL type. A DAP2 application should obtain the DAP2 URL value mapped from the HDF5 object reference and retrieve the data value and other information to which the object reference refers.

The detailed steps are the following:

- 1) The OPeNDAP client issues a request to access the DAP2 DDS from an OPeNDAP server.
- 2) The object reference is displayed within the returned DDS.
- 3) The client requests the absolute HDF5 path(s) to which the object reference refers.
- 4) The OPeNDAP server returns the absolute path(s) of the HDF5 dataset(s) referenced by the requested DAP2 array element(s).
- 5) The OPeNDAP client retrieves the data in the referenced dataset using the absolute path returned by the OPeNDAP server.

Figure 2 illustrates the structure of an HDF5 file that includes an HDF5 dataset for which the datatype is the object reference. This file consists of one group, *grans1*, and four HDF5 datasets under this group. One dataset, *od1*, is an HDF5 object reference array. This object reference array includes three elements that refer to three other HDF5 datasets: *gran1*, *gran2*, and *gran3*. In the following figure, the file name is *grans1.h5*.

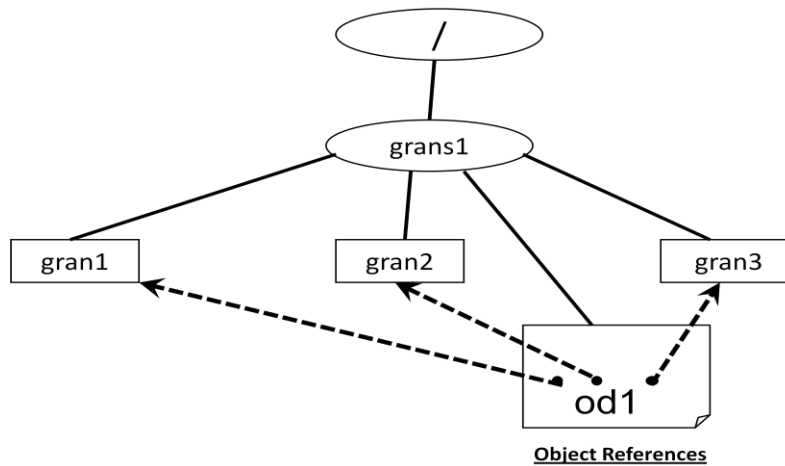


Figure 2. An HDF5 file that includes an object reference dataset

The following example, which refers to the HDF5 file in Figure 2, outlines the process for accessing data referenced by the HDF5 object reference via DAP2.

- 1) The OPeNDAP client issues a request to access the DAP2 DDS from an OPeNDAP server:

`http://dummy.localhost.com/opensdap/data/grans1.h5.dds`

This request can be parsed in the following way:

- a. `http://dummy.localhost.com/opensdap/data` represents an OPeNDAP server.
 - b. `grans1.h5` is the HDF5 file name.
 - c. `grans1.h5.dds` means to fetch the DDS for the file.
- 2) The object reference is displayed within the returned DDS:

`Url/grans1/od1[3]`

For compactness, other displayed information has been omitted in this example.

- 3) The client requests the absolute HDF5 path(s) to which the object reference refers.

The client can request the entire `/grans1/od1` array:

`http://dummy.localhost.com/opensdap/data/grans1.h5.dods?/grans1/od1`

Alternatively, it can request a single element of the array using the `[i]` syntax to specify the index of the desired element. When `i` is equal to 1, the second element is requested:

`http://dummy.localhost.com/opensdap/data/grans1.h5.dods?/grans1/od1[1]`

- 4) The OPeNDAP server returns the absolute path(s) of the HDF5 dataset(s) referenced by the requested array element(s).

The result for the first request (entire array) is:

```
{/grans1/gran1,/grans1/gran2,/grans1/gran3}
```

The result for the second request (element with index 1) is:

```
{/grans1/gran2}
```

- 5) The OPeNDAP client can now retrieve the data in the referenced dataset using the absolute path returned by the OPeNDAP server in the previous step:

```
http://dummy.localhost.com/opensdap/data/grans1.h5.dods?/grans1/gran2
```

- 6) The data in the *gran2* dataset will be returned.

3.2.2.2.2 Region reference

A data element with a dataset region reference datatype points to a selected region of a dataset. The region referenced is located by retrieving the coordinates of the areas in the region. By dereferencing a region reference, one can not only retrieve the data (data shape, datatype, etc.) and metadata (attributes) information of the object the region reference points to, but also the coordinates of the areas in the region.

The detailed steps are the following:

- 1) The OPeNDAP client issues a request to access the DAP2 DDS from an OPeNDAP server.
- 2) The region reference is displayed within the returned DDS.
- 3) The client requests the absolute HDF5 paths(s) and region(s) to which the reference refers.
- 4) The OPeNDAP server returns the requested information.
- 5) The OPeNDAP client retrieves the region of data in the referenced dataset using the absolute path and region bounds returned by the OPeNDAP server in the previous step.
- 6) The data in the specified region of the *gran4* dataset is returned.

Figure 3 illustrates the structure of an HDF5 file that includes an HDF5 dataset for which the datatype is a region reference. This file consists of one group, *grans2*, and three HDF5 datasets under this group. One dataset, *rd1*, is an HDF5 region reference array. This region reference array includes two elements that refer to subsets of two HDF5 dataset arrays. In this case, those subsets are */grans2/gran4 [1:4]* and */grans2/gran5 [0:1][1:2]*. In the following figure, the file name is *grans2.h5*.

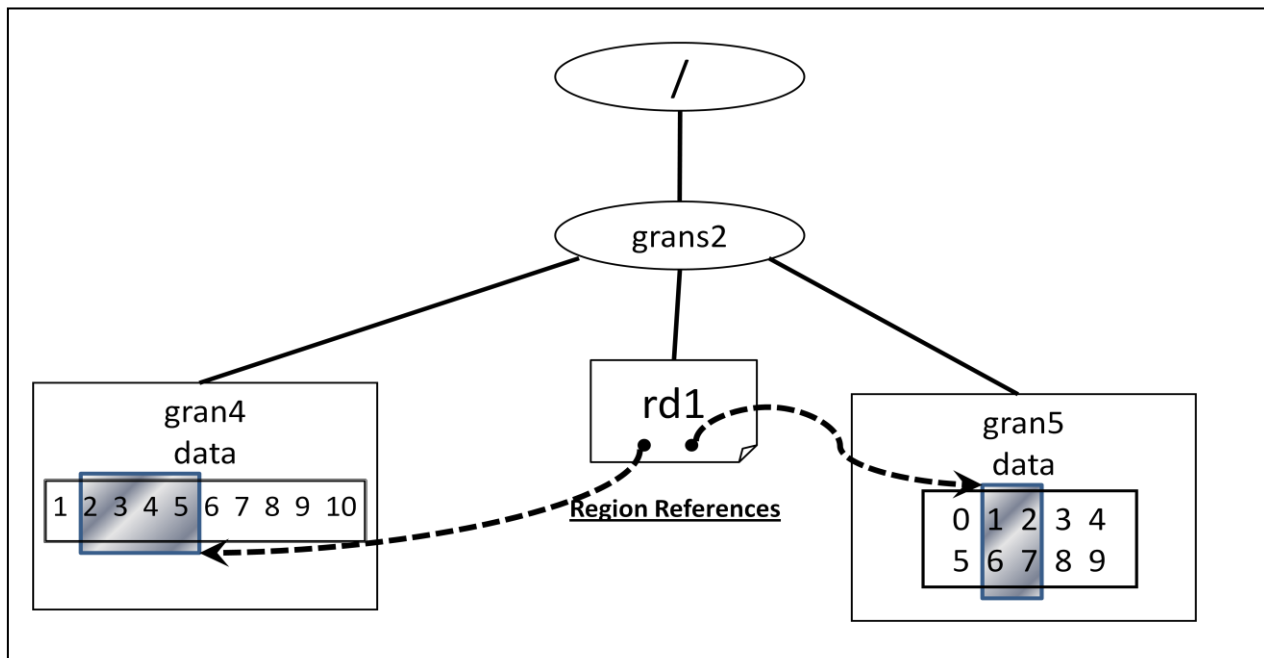


Figure 3. An HDF5 file that includes a region reference dataset

The following example, which refers to the HDF5 file in Figure 3, outlines the process for accessing data referenced by the HDF5 region reference via DAP2. The process is similar to the one just presented for object references, so some commentary is omitted in this example.

- 1) The OPeNDAP client issues a request to access the DAP2 DDS from an OPeNDAP server:

`http://dummy.localhost.com/opensdap/data/grans2.h5.dds`

- 2) The region reference is displayed within the returned DDS:

`Url /grans2/rd1[2]`

- 3) The client requests the absolute HDF5 paths(s) and region(s) to which the reference refers. The entire array or a single element can be requested. The entire array is requested here:

`http://dummy.localhost.com/opensdap/data/grans2.h5.dds?/grans2/rd1`

- 4) The OPeNDAP server returns the requested information. The result for this example is:

`{/grans2/gran4[1:4], /grans2/gran5[0:1][1:2]}`

- 5) The OPeNDAP client can now retrieve the region of data in the referenced dataset using the absolute path and region bounds returned by the OPeNDAP server in the previous step:

`http://dummy.localhost.com/opensdap/data/grans2.h5.dds?/grans2/gran4[1:4]`

- 6) The data in the specified region of the *gran4* dataset is returned:

`{2, 3, 4, 5}`

3.2.2.2.3 Limitations of using the DAP2 URL datatype for references

Unfortunately, while the DAP2 URL datatype can be used to represent many of the object and region references, there are cases that cannot be represented without some loss of information. First, in this mapping, there is no distinction made between object and region references, so some information is lost if a single file holds both kinds of HDF5 objects. It is possible to examine the value of the URL and determine which type of reference a given DAP2 URL corresponds to, but this is hardly a straightforward way to work with the data.

A second limitation occurs because the HDF5 specification allows references to use UTF-8 characters while DAP2 URLs are limited to ASCII characters. This difference means that some references might look strange because the ‘character code escaping’ mechanism would be employed to represent the multi-byte characters (one ‘%<hex digit><hex digit>’ group for each byte in the UTF-8 character).

The third limitation occurs because a data element with an HDF5 region reference datatype can point to a union of several regions, but the current mapping cannot address this case. The current mapping only addresses the mapping of a region reference that refers to one region.

Lastly, HDF5 version 1.8.0 and later support the creation of ‘external links’ where object and region references point to entities in other files. Furthermore, these external links are not necessarily HDF5 files. In each of these cases, mapping to a DAP2 URL can support the concept of the external link, but, in practice, the mechanism may fail. The crux of the issue is not whether the external link references an HDF5 file or not, because DAP2 hides the storage format of data referenced by a URL, but that the client must know that in order to access the referenced data, a second URL must be dereferenced. In other words, when an object or region reference is used for data within the file, the data values associated with that reference can be accessed either directly within the data source or by reading the reference URL and having a client perform a second access using that URL. However, if the reference is to an external file, the only way a client can get the data values is by performing the second access operation.

3.2.2.3 Compound datatype

A compound datatype is conceptually similar to a C structure defining a heterogeneous collection of datatypes where each element can be individually addressed. A compound datatype may have any number of members, in any order, and the members may have any HDF5 datatype, including compound. An HDF5 compound datatype is generally mapped to a DAP2 structure as long as the base members of the compound datatype can be mapped to DAP2. However, when the compound datatype has a base member whose datatype is either an object reference or region reference, the compound datatype cannot be mapped. If the datatype of the base member is a datatype rarely used by earth science applications (discussed in section 3.2.1), the compound datatype may also not be mapped. A nested HDF5 compound datatype can be mapped to DAP2 provided all of the compound types in the nesting hierarchy could be mapped.

The general procedure of mapping an HDF5 compound datatype to DAP2 is to map a base member of the compound datatype to an individual field of the DAP2 structure. The following shows a DAP2 DDS that consists of an array of DAP2 structures mapped from a nested HDF5 compound datatype dataset.

```
Dataset {
  Structure {
    Int32 outer_int;
    Structure {
      Int32 inner_int;
      Float32 inner_float;
      String inner_name;
    } outer_com;
    String out_name;
  } ArrayOfStructures[3];
} compound.h5;
```

The dumped output of the original HDF5 compound datatype dataset by the h5dump dump tool is shown below. The data value is omitted for easy reading.

```
HDF5 "compound.h5" {
GROUP "/" {
  DATASET "ArrayOfStructures" {
    DATATYPE H5T_COMPOUND {
      H5T_STD_I32LE "outer_int";
      H5T_COMPOUND {
        H5T_STD_I32LE "inner_int";
        H5T_IEEE_F32LE "inner_float";
        H5T_STRING {
          STRSIZE 17;
          STRPAD H5T_STR_NULLTERM;
          CSET H5T_CSET_ASCII;
          CTYPE H5T_C_S1;
        } "inner_name";
      } "outer_com";
      H5T_STRING {
        STRSIZE 5;
        STRPAD H5T_STR_NULLTERM;
        CSET H5T_CSET_ASCII;
        CTYPE H5T_C_S1;
      } "out_name";
    }
    DATASPACE SIMPLE { ( 3 ) / ( 3 ) }
  }
}
}
```

3.3 HDF5 datasets

An HDF5 dataset is a dataset object that consists of a data array with metadata information, which stores a description of the data elements of the array as well as other information (such as data layout) that is necessary to write, read and interpret the stored data. The array elements all have the same HDF5 data type.

There are two types of HDF5 datasets: scalar and nonscalar. In a scalar dataset, the rank of the data space is zero, and there is only a single value. The most commonly used dataset is a nonscalar dataset of which the rank of the data space is not zero. HDF5 datasets can have attributes. In this section, we discuss the mapping of HDF5 datasets. Mapping attributes of HDF5 datasets is discussed in section 3.5.

The scalar dataset can only be mapped to a DAP2 variable. To map a scalar dataset to DAP2, map the datatype of the scalar dataset to the corresponding DAP2 datatype by following the previous discussions in section 3.1 and section 3.2. The name of the DAP2 variable will be the absolute path of the HDF5 dataset.

An HDF5 nonscalar dataset is mapped to a DAP2 array variable. A DAP2 array is similar to that defined by ANSI C. Multidimensional arrays **MUST** be stored in row-major order (as is the case with ANSI C). The datatype of the HDF5 nonscalar dataset is mapped to the corresponding DAP2 datatype by following the previous discussions in section 3.1 and section 3.2. The name of the DAP2 variable is the absolute path of the HDF5 dataset. The size and shape of the data array of the dataset is defined by an HDF5 data space. The number of dimensions and the number of elements of each dimension of the data array can be retrieved from HDF5 data space APIs.

Each dimension, and the number of elements of each dimension, is mapped to the dimension of a DAP2 array following the row-major order. This information, the name of the DAP2 variable, and the corresponding DAP2 datatype is then combined to form a DAP2 array. The DAP2 array is saved inside the DAP DDS. The data layout information, such as chunking and compression, is ignored when mapping to DAP2 because the processing in DAP2 is a read-only operation. The data is always decompressed before it passes to DAP2. The chunking storage may only affect the performance of retrieving the data. An HDF5 dataset must be under one or more HDF5 groups. Mapping an HDF5 group to DAP2 is discussed in the next section. So we will not discuss how the datasets are arranged in DAP2.

The following simple example shows how an HDF5 dataset can be mapped to a DAP2 array. The absolute path of an HDF5 dataset is `/HDF5_group/group2/dataset1`. The datatype of the dataset is a 32-bit signed integer. The data array of the dataset is a 2-D 10-by-10 array. This information can be retrieved by using HDF5 data space APIs. Assuming that there are no attributes associated with this dataset, the HDF5 dataset is mapped to the DAP2 array `Int32 /HDF5_group/group2/dataset1[10][10]`.

Since the total number of elements in a DAP2 array MUST NOT exceed $2^{31}-1$ [6], the HDF5 dataset with the number of elements greater than $2^{31}-1$ cannot be mapped to DAP2.

3.4 HDF5 groups

An HDF5 group object does not have an equivalent DAP2 object. Since HDF5 groups are used to show relationships among HDF5 objects, HDF5 group objects must be mapped to DAP2 to ensure that a DAP2 client can reconstruct the hierarchical relationships in an HDF5 file. There are two options to map the HDF5 group to DAP2. In this section, we present both options and recommend one. We provide an explanation for our choice and a detailed description of mapping an HDF5 group to DAP2 using that option.

3.4.1 Evaluations of two options

The following example illustrates the mapping of HDF5 groups to DAP2. Although this example shows only three levels of groups in the graph hierarchy, the DAP2 representation does not restrict the number of levels that can be mapped. The example also demonstrates the need for absolute paths in DAP2 variable names.

Figure 4 shows an HDF5 file, *group.h5*, with four groups: the root group (*/*), *g1*, *g2* and *g3*. Group *g1* includes one group, *g4*, and one dataset, *struct1*. Group *g2* includes two datasets, *d1* and *d2*. Group *g3* also includes two datasets, *d1* and *d4*.

Dataset *struct1* in group *g1* is a scalar value with the HDF5 compound datatype. *struct1* is respectively equivalent to the C structures *s1* shown here:

```
S1{
    int ai[100];
    float af[100];
}
```

Group *g4* in group *g1* has no members. It has an attribute with the name *group_attr*, and the value is a string, “group attribute value”.

Dataset *d1* in group *g2* is an integer array of 100 elements. Dataset *d2* in the same group is a floating-point array of 100 elements. Dataset *d1* in group *g3* is a scalar with an integer datatype. Dataset *d4* is a scalar with a floating-point datatype. Dataset *d4* has an attribute with the name *dset_attr*, and the value is a string “dataset attribute value”.

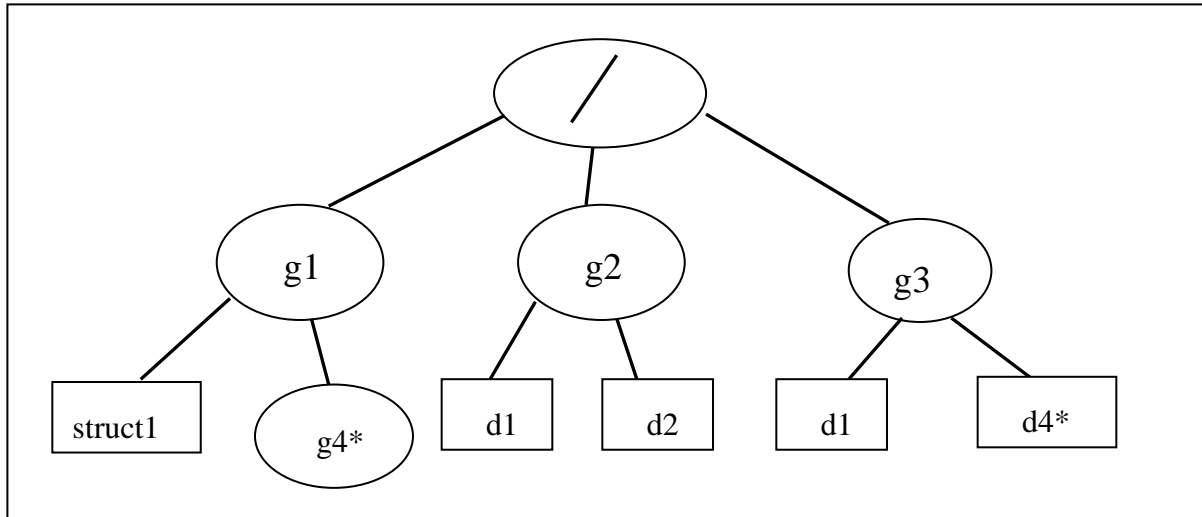


Figure 4. An HDF5 file that includes HDF5 groups, datasets and attributes. The object name with a * indicates that object has an at least one attribute.

3.4.1.1 Option 1: Map an HDF5 group to a DAP2 structure

The DAP2 structure can be used to represent grouping in the following way. One can define an HDF5 group as a single DAP2 structure. The name of the group is the name of the structure. All HDF5 objects within the group are defined as elements of the structure. Attributes of groups can be represented naturally as attributes of this structure. The whole HDF5 file is mapped to a DAP2 structure. There are substantial limitations to this approach. Firstly, the structure is a tree. Any loops or multiple links to the same object cannot be directly represented.

Secondly, with this approach every HDF5 file will contain exactly one object, the structure representing the root group. This structure will contain all the HDF5 objects regardless of the complexity of the file structure inside the HDF5 file. This is not intuitive to DAP2 clients, which expect a list of data arrays, not one big, complicated structure.

Finally, the fatal flaw for this approach is that it causes ambiguities for an HDF5 dataset using an HDF5 compound datatype. As discussed in section 3.2.2.4, an HDF5 compound datatype is mapped to a DAP2 structure. However, since an HDF5 group is also mapped to a DAP2 structure, it makes the DAP2 clients treat both the group and the dataset having compound datatype as DAP2 structures. This causes confusion.

The following shows the DDS output of the HDF5 file described in Figure 4.

```
Dataset {
  Structure {
    Structure {
      Structure {
        Int32 ai[100];
      }
    }
  }
}
```

```
        Float64 af[100];
    } struct1;
    Structure {
        } g4;
} g1;
Structure {
    Int32 d1[100];
    Float64 d2[100];
} g2;
Structure {
    Int32 d1;
    Float64 d4;
} g3;
} /;
} group.h5;
```

The compound dataset struct1 cannot be distinguished from group g2. If an HDF5 file is mapped in this way, it is impossible for an HDF5 client or a DAP2-to-HDF5 conversion tool to properly understand the structure of the HDF5 object.

3.4.1.2 Option 2: Map HDF5 hierarchical structure to a DAP2 attribute

Using this option, all HDF5 datasets are mapped to DAP2 variables, with information about the group structure mapped as DAP2 attributes. This approach is natural for most DAP2 applications; the file is presented as a set of DAP2 variables.

Different HDF5 objects under different groups can have the same “relative” name just as different files can have the same name under different directories on a file system. Therefore one can only use the absolute path to represent an HDF5 object without causing ambiguity.

We mapped the entire hierarchy of groups and datasets to a DAP2 attribute named *HDF5_ROOT_GROUP*. DAP2 allows the nested variable attributes so we could represent the subgroups under the root group as the nested variables. An attribute called “Datasets” provides the name list of the HDF5 datasets under a specific group.

HDF5_ROOT_GROUP only preserves the HDF5 file hierarchy. Other HDF5 group attributes are mapped to DAP2 as if the HDF5 group is a DAP2 variable. The DAS of the HDF5 file shown in Figure 4 is:

```
Attributes {
    HDF5_Root_Group {
        g1 {
```

```
        String Datasets "/struct1";
        g4{
        }
    }
    /g2 {
        String Datasets "d1 d2";
    }
    /g3 {
        String Datasets "d1 d4";
    }
}
/g1/g4/ {
    String group_attr "group attribute value";
}
/g3/d4 {
    String dset_attr "dataset attribute value";
}
}
```

The DDS of the HDF5 file shown in Figure 4 is:

```
Dataset {
    Structure {
        Int32 ai[100];
        Float64 af[100];
    } /g1/struct1;
    Int32 /g2/d1[100];
    Float64 /g2/d2[100];
    Int32 /g3/d1;
    Float64 /g3/d4;
} group.h5;
```

In this way, we can clearly see that the DAP2 structure */g1/struct1* is mapped from an HDF5 compound datatype dataset *struct1*, and this dataset is under the group *g1*.

3.4.1.3 Recommendations

We recommend using option 2 to map the group into a DAP2 attribute so that DAP2 clients can easily access HDF5 data regardless of their understanding of the HDF5 group concept. This attribute can be ignored for clients that do not need to use or to understand the group information; whereas, it should be used for clients that need to reconstruct the HDF5 group hierarchy. Mapping the group hierarchy into a DAP2 variable may cause clients to erroneously treat the group hierarchy as real data rather than metadata.

3.4.2 General mapping procedures

Since every group (and dataset) in an HDF5 file is part of a hierarchy originating from the HDF5 root group, the HDF5 groups and the relationships they establish can be mapped using a single DAP2 attribute. This attribute, named *HDF5_ROOT_GROUP*, expresses the entire hierarchy of groups and datasets in an HDF5 file.

Note that the described DAP2 mapping representation for HDF5 groups does not handle cycles in the graph hierarchy, which, while allowed by the HDF5 format, are rarely used by earth science applications.

As mentioned in section 3.3, an HDF5 dataset is mapped to a DAP2 variable. Since different HDF5 datasets under different groups can have the same name, the absolute path of an HDF5 dataset should be used as the name of the corresponding DAP2 variable in order to avoid ambiguity.

Step 1. Map all HDF5 datasets to DAP2 variables. The DAP2 variable name is the absolute path of the HDF5 dataset name.

Step 2. Create the *HDF5_ROOT_GROUP* DAP2 attribute. This attribute expresses the entire hierarchy of groups and datasets in an HDF5 file. Since DAP2 allows nested variable attributes, one can represent the subgroups under the root group as the nested variables. If that group consists of datasets, a special attribute named “Datasets” will include the list of HDF5 dataset names. If the group does not have any datasets, the attribute will be an empty set. This is allowed in DAP2. Note that the group does not appear at DDS. It only appears at the DAP2 DAS.

Step 3. Map HDF5 group attributes to DAP2 attributes. An HDF5 group may contain attributes. Those attributes are mapped to DAP2. Using the same logic as Step 2, the group can be treated as a DAP2 variable. Only the DAP2 variable name should be the absolute path of the group to avoid any ambiguity.

3.4.3 Illustration with an example

See section 3.4.1.2 for an example of how to map HDF5 groups to DAP2. There we use an example HDF5 file to illustrate how an HDF5 file hierarchy can be preserved by storing the HDF5 group information with a DAP2 attribute, *HDF5_ROOT_GROUP*.

3.4.4 Other discussions

An HDF5 object may have multiple absolute paths. In HDF5's terms, an HDF5 object having multiple absolute paths may have more than one hard link and soft link. One can find the differences between hard links and soft links from the HDF5 RFC[2]. Since current earth science HDF5 data uses neither hard link nor soft link, the discussion of the mapping should be only used as a reference for future applications.

For objects holding multiple hard links, one of them is used to represent the DAP2 variable name. The absolute paths of other links are saved as DAP2 attributes. The mapping of soft links is not discussed in this document.

3.5 HDF5 attributes

HDF5 group and dataset objects can have attributes. An HDF5 attribute that is either a scalar or a one-dimensional array can be mapped to a DAP2 attribute as long as the datatype of the HDF5 attribute can be mapped to DAP2. DAP2 preserves the datatype information and conveys the attribute value. The information of the shape of the attribute is not explicitly preserved. The DAP2 attribute mapped from an attribute of an HDF5 group needs to be distinguished from the DAP2 attribute mapped from an attribute of an HDF5 dataset. In order to do this, a slash '/' is added at the end of the name of a DAP2 variable mapped from an HDF5 group object when representing the attribute of that variable. This convention allows clients to distinguish whether the attribute is mapped from an HDF5 group or from an HDF5 dataset.

For example, the HDF5 file `group.h5` shown in Figure 4 includes one group attribute for group `/g1/g4` and one dataset attribute for the dataset `/g3/d4`. The attribute name of `/g1/g4` is `group_attr`. The attribute name of `/g3/d4` is `dsset_attr`. The datatype of both attributes is HDF5 String. Mapping this HDF5 file to DAP2 provides the following DAP2 DAS table. Please note the '/' added at the end of `/g4`. This will make clients understand this attribute is an HDF5 group attribute. For compactness, the `HDF5_ROOT_GROUP` attribute that expresses the entire hierarchy of the HDF5 file is omitted in this DAS table.

```
Attributes {  
  /g1/g4/ {  
    String group_attr "group attribute value";  
  }  
  /g3/d4 {  
    String dsset_attr "dataset attribute value";  
  }  
} group.h5;
```

4 Special Mapping for HDF-EOS5 Objects

One goal of mapping HDF5 to DAP2 is to enable easy access to NASA HDF-EOS5 data via DAP2 clients. Many widely used OPeNDAP visualization clients, such as Unidata’s Integrated Data Viewer (IDV) [9] and the Grid Analysis and Display System (GrADS) [10], require the representation of geolocation information for the data to follow the CF conventions [13]. The general HDF5-to-DAP2 mapping was designed to handle HDF5 files from all domains and did not provide the required representation. A special mapping for NASA HDF-EOS5 data is necessary for OPeNDAP clients to visualize and analyze these data.

Two types of HDF-EOS5 objects — grids and swaths — are widely used to store NASA HDF-EOS5 data. More than 99% of NASA HDF-EOS5 data are either grids or swaths. In rare cases, an HDF-EOS5 file includes both a grid object and a swath object. This technical note only considers cases where an HDF-EOS5 file includes either a grid or a swath, but not both.

In this document, the terms “data field” and “variable” are interchangeable. The terms “dimension variable” and “coordinate variable” are also interchangeable.

The order of dimensions in the grid and swath examples shown in the following sections is in row-major order (as with C or C++), with the last dimension being the most rapidly incrementing dimension.

4.1 HDF-EOS5 grids

4.1.1 Overview

An HDF-EOS5 grid is a data object that is organized by geographic spacing that can be computed through projection information. An HDF-EOS5 grid consists primarily of physical data fields and attributes that describe those fields. The physical data fields are usually multidimensional data arrays and are stored as HDF5 datasets. Although some HDF-EOS5 grid files optionally store latitude and longitude fields in the file, all HDF-EOS5 grid files provide the latitude and longitude information implicitly by projection parameters contained in a special HDF5 scalar dataset called *StructMetadata.0*. Special data processing needs to be done to retrieve the latitude and longitude values of a physical variable for an HDF-EOS5 grid file in which no optional latitude and longitude fields are stored.

The HDF-EOS5 library allows an application to store the grid data in one of the more than a dozen projections. There are two classes of projections for describing the dimensionality of latitude and longitude: 1-D projections and 2-D projections.

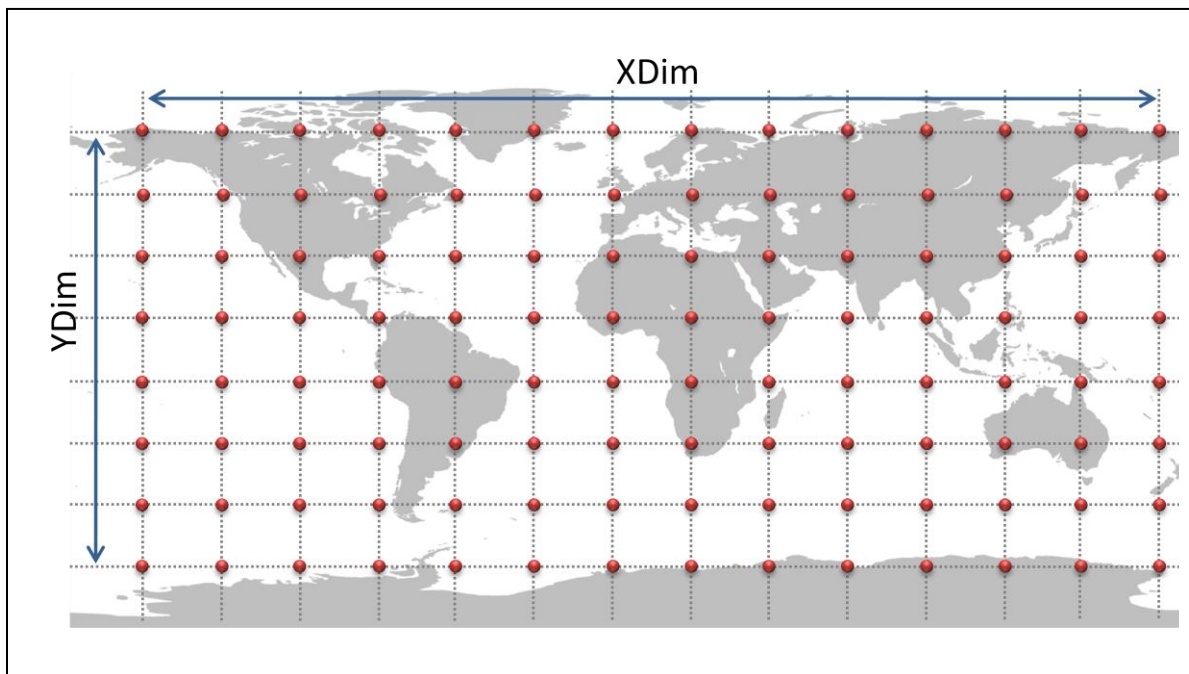
Mapping an HDF-EOS5 grid of 1-D projections to DAP2 is different than mapping an HDF-EOS5 grid of 2-D projections. The detailed mapping information for 1-D projections and 2-D projections is given in the following two sections.

In the following discussions, we always assume that the latitude and longitude need to be retrieved through *StructMetadata.0*. For HDF-EOS5 grid files that provide latitude and longitude fields, the DAP2 output of latitude and longitude values may be directly obtained from the latitude and longitude fields of the HDF-EOS5 grid files. Occasionally, only the subsets of the latitude and longitude fields of the HDF-EOS5 grid files are needed for the DAP2 output. For example, some 1-D projection HDF-EOS5 grid latitude and longitude fields are 2-D arrays. Only the values along one column or one row of these 2-D arrays are needed for the DAP2 output.

4.1.2 1-D projections

Figure 5 shows part of an HDF-EOS2 grid file for a 1-D projection, which is a geographic projection. The Network Common Data Form Language (CDL)-like description of the grid structure for this HDF-EOS5 grid file is also inserted in the figure. The grid structure information can be extracted from the *StructMetadata.0*. The example data field is an 8-by-14, two-dimensional array represented by red dots.

All data points along each horizontal line (parallel to XDim) share the same latitude and all data points along each vertical line (parallel to YDim) share the same longitude. Therefore, a one-dimensional array with 14 elements can be used to describe the longitude of this 2-D data field. A one-dimensional array with 8 elements can be used to describe the latitude of this 2-D data field.



```
HDF-EOS5 INFORMATION

StructMetadata.0 =
"... XDim= 14 YDim = 8
UpperLeftPointMtrs =(-156.0, 71.0) LowerRightMtrs =(180.0, -75.0)
... Projection = HE5_GCTP_GEO ..."

HDF-EOS5 GRIDS:
Float32 Ozone(YDim, XDim)
```

Figure 5. An HDF-EOS2 grid containing a 1-D geographic projection

The example in Figure 5 shows that, for a 1-D projection, it is sufficient to use a 1-D array to describe the latitude of the data field and a second 1-D array to describe the longitude. The sizes of the 1-D arrays correspond to the sizes of the horizontal dimensions of the data field.

There are two ways to map a 1-D projection HDF-EOS5 grid file to DAP2. The first is to map the HDF-EOS5 grid to a DAP2 Grid. The second is to map the HDF-EOS5 grid to a DAP2 Array variable. The following two sections describe each way in detail.

4.1.2.1 Mapping 1-D projection HDF-EOS5 Grid to DAP2 Grid

To ensure that OPeNDAP visualization clients can access HDF-EOS5 data, several key DAP2 attributes need to be provided according to CF conventions. The “units” of latitude and longitude must be provided, and the attribute name and the corresponding values must follow CF conventions. The “units” of longitude must be presented as “degrees_east” and the “units” of latitude must be “degrees_north”. Most HDF-EOS5 grid files do not have latitude and longitude fields, therefore no “units” attributes of latitude and longitude are provided. For those HDF-EOS5 grid files that also provide latitude and longitude values, some may provide “units” for latitude and longitude. Furthermore, it is possible that the value of “units” may not be “degrees_east” for longitude and “degrees_north” for latitude. In such cases, the “units” of latitude and longitude must be created or corrected to ensure that CF conventions are followed.

CF conventions also highly recommend the usage of “long_name” for a variable that contains a long descriptive name for the attribute. Since the latitude and longitude field name values can be used as the value of “long_name” if there is no real value of the “long_name” attribute, it is always a good idea to add the “long_name” attribute to make the DAP2 output follow CF conventions.

```
Attributes {  
    lon {  
        String units "degrees_east";  
        String long_name "longitude";  
    }  
    lat {  
        String units "degrees_north";  
        String long_name "latitude";  
    }  
}
```

A DAP2 Grid can be used to associate latitude and longitude with physical variables. For a 1-D projection HDF-EOS5 grid, a DAP2 Grid that consists of the physical variable and the associated latitude and longitude needs to be generated. The latitude and longitude values can be computed through special processing of the information provided in the HDF5 *StructMetadata.0* dataset. One may need to adjust the latitude and longitude values to ensure that the “units” of the latitude and longitude follow CF conventions, as described previously.

The DDS of the DAP2 file mapped from the HDF-EOS5 grid containing a 1-D projection shown in Figure 5 is:

```
Dataset {
  Grid {
    Array:
      Float32 Ozone [lat=8] [lon= 14];
    Maps:
      Float32 lat[lat = 8];
      Float32 lon[lon = 14];
  }
  Ozone;
}
```

4.1.2.2 Mapping 1-D projection HDF-EOS5 Grid to DAP2 Array

One can also map 1-D projection HDF-EOS5 Grid to DAP2 Array. In this case, the latitude and the longitude, as well as other data fields, are mapped to DAP2 arrays. The mapping still needs to follow CF conventions to ensure the maximum access of OPeNDAP client tools to HDF-EOS5 data. The latitude and longitude still need to be either computed or retrieved from the provided latitude and longitude fields as described in the previous section. The DDS output with this mapping option is:

```
Dataset {
  Float32 Ozone [lat=8] [lon= 14];
  Float32 lat[lat = 8];
  Float32 lon[lon = 14];
}
```

To follow the CF conventions, the “units” and “long_name” attributes still need to be provided for the latitude and longitude variables as described in the previous section. Another attribute called “coordinates” should also be provided for each physical variable. This attribute should specify the name list of the corresponding coordinate variables of the physical variable. In the above example, the coordinate variables of the “Ozone” are “lat” and “lon”. So an attribute for “Ozone” should be added to the DAP2 variable “Ozone”. The DAS of the DAP2 file mapped from the HDF-EOS5 grid shown in Figure 5 to DAP2 arrays is:

```
Attributes {
  lon {
    String units "degrees_east";
    String long_name "longitude";
  }
  lat {
    String units "degrees_north";
    String long_name "latitude";
  }
  Ozone {
    String coordinates "lon lat";
  }
}
```

4.1.3 2-D projections

Figure 6 shows part of an HDF-EOS5 grid file for 2-D projection, which is a polar stereographic projection. A CDL-like description of the grid structure for this HDF-EOS5 grid file is also inserted in the figure. The example data field is a 9-by-7, two-dimensional array.

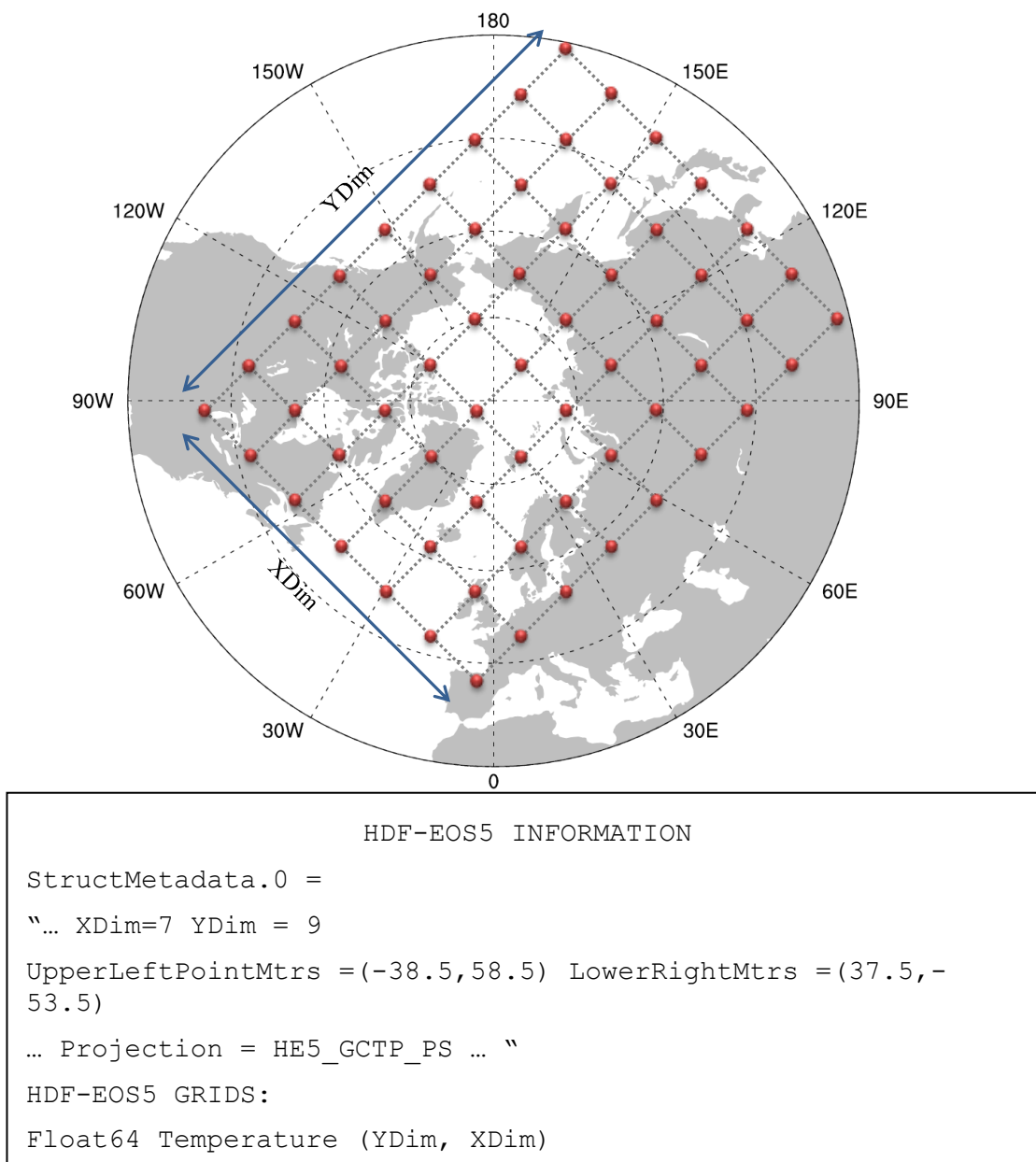


Figure 6. An HDF-EOS5 grid containing a 2-D polar stereographic projection

For this polar projection, no data points (red dots) along the grid lines parallel to XDim and YDim share the same latitude or longitude values. Therefore, a 9-by-7, two-dimensional array is needed to describe the latitude of this 2-D data field, and another 9-by-7, two-dimensional array is needed to describe the longitude.

The example in Figure 6 indicates that, for 2-D projections, two 2-D arrays are needed to describe the latitude and longitude of the data field (one array for each dimension). The sizes of these 2-D arrays correspond to the sizes of the horizontal dimensions of the data field. The latitude and longitude values can be computed through special processing of the information provided in the HDF5 *StructMetadata.0* dataset.

As in the 1-D projection case, CF conventions need to be followed to ensure the maximum access of HDF-EOS5 data via OPeNDAP clients. However, there is no corresponding DAP2 object, like DAP2 grid, to describe both coordinate information and the physical variable. Both latitude/longitude fields and physical fields need to map to DAP2 arrays. As described in section 4.1.2.2, additional attributes called “coordinates” should be provided for each physical variable that has corresponding coordinate variables to describe it.

For the 2-D projection HDF-EOS5 Grid shown in Figure 6, DAP2 variables should be generated to store longitude and latitude. The DAP2 variable name for longitude is “lon”, and the DAP2 variable name for latitude is “lat”. The physical variable “Temperature” should have an attribute called “coordinates” to specify the geolocation information of the variable. The “coordinates” attribute should include the name list of the corresponding DAP2 coordinate variables “lat” and “lon”.

However, only “units” and “long_name” must be specified in “lon” and “lat”.

The DDS of the DAP2 file mapped from the HDF-EOS5 grid containing a 2-D projection shown in Figure 6 is:

```
Dataset {  
    Float32 lat[lat= 9][lon= 7];  
    Float32 lon[lat= 9][lon= 7];  
    Float64 Temperature[lat= 9][lon= 7];  
}
```

As in the 1-D projection case, the “units” and “long_name” of latitude and longitude should be provided according to CF conventions. The DAS of the DAP2 file mapped from the HDF-EOS5 grid shown in Figure 6 is:

```
Attributes {  
  
    lon {  
        String units "degrees_east";  
        String long_name "longitude";  
    }  
    lat {  
        String units "degrees_north";  
        String long_name "latitude";  
    }  
    Temperature {  
        String coordinates "lon lat";  
    }  
}
```

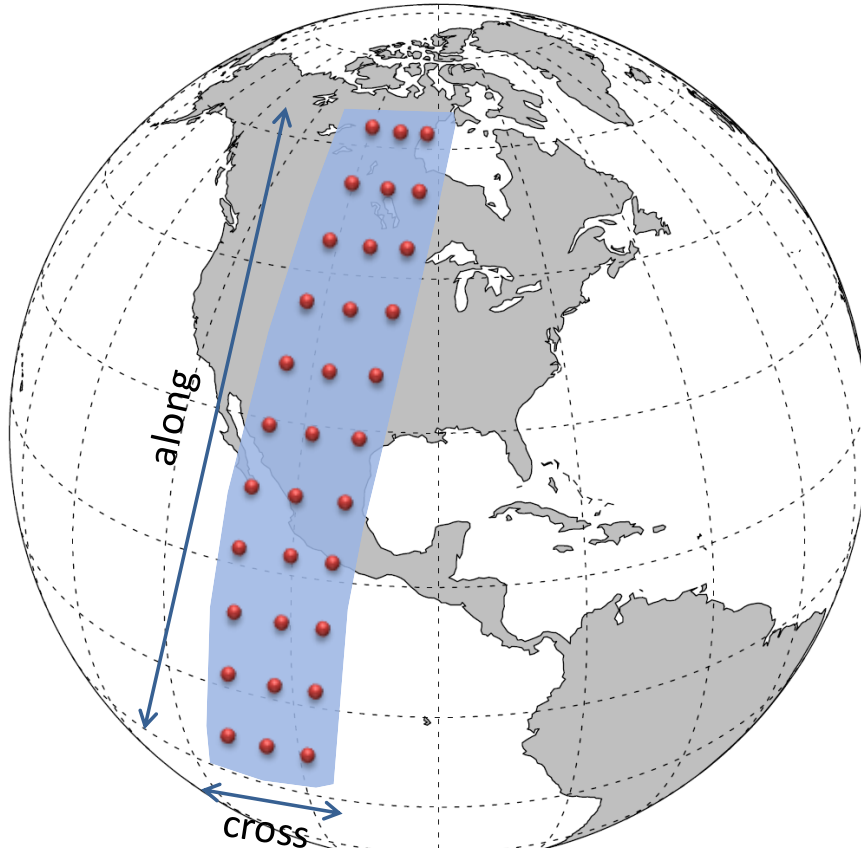
Other than latitude and longitude fields, there are currently no specific HDF-EOS5 APIs that one can use to retrieve values of coordinate variables such as altitude and pressure. At the time of this writing, this issue is still being investigated, and further discussion can be found in section 4.3

4.2 HDF-EOS5 Swaths

The concept of an HDF-EOS5 swath is based on a typical satellite swath, where an instrument takes a series of scans perpendicular to the ground track of the satellite as it moves along that ground track. An HDF-EOS5 swath also consists of physical data fields and attributes that describe those fields. Like an HDF-EOS5 grid, data are stored as multidimensional arrays described by HDF5 datasets. Unlike an HDF-EOS5 grid, horizontal geolocation fields are also stored as multidimensional arrays described by HDF5 datasets.

In addition, HDF-EOS5 defines a concept called a dimension map. When dimension maps are used, the geolocation fields do not necessarily have the same extent as the physical data fields. A dimension map provides sufficient information to allow the geolocation information at each data point to be derived from the geolocation fields by interpolation.

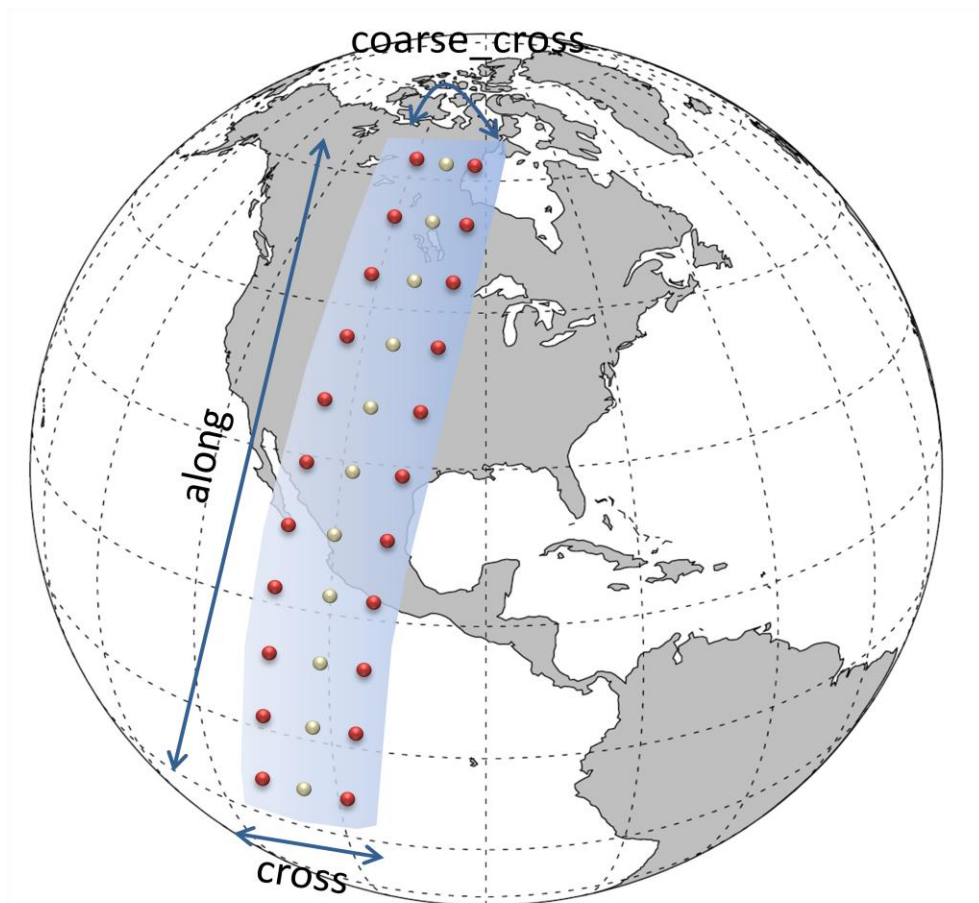
HDF-EOS5 swath examples are illustrated in the graphical portions of Figures 7 and 8. Figure 7 shows an HDF-EOS5 swath without a dimension map. Both physical data and geolocation data values are provided at data points represented by red dots. The geolocation fields are comprised of latitude and longitude. The latitude and longitude are both 11-by-3, two-dimensional arrays; 11-by-3 is the same extent as the physical data field “CloudCover”.



```
HDF-EOS5 INFORMATION
StructMetadata.0 =
" ... OBJECT=Dimension_1 DimensionName="along" size= 11
  OBJECT=Dimension_2 DimensionName="cross" size= 3 ..."

HDF-EOS5 Swath:
Data fields:
    Float32 CloudCover(along,cross)
Geolocation fields:
    Float32 lat(along,cross)
    Float32 lon(along,cross)
```

Figure 7. An HDF-EOS5 swath without dimension maps



```
HDF-EOS5 INFORMATION
StructMetadata.0 =
" ... OBJECT=Dimension_1 DimensionName="along" size= 11
  OBJECT=Dimension_2 DimensionName="cross" size= 3
  OBJECT=Dimension_3 DimensionName="coarse_cross" size= 2..."

HDF-EOS5 Swath:
Data fields:
    Float32 CloudCover(along,cross)
Geolocation fields:
    Float32 lat(along,coarse_cross)
    Float32 lon(along,coarse_cross)
```

Figure 8. An HDF-EOS5 swath with dimension maps. Both geolocation and data values are provided at red dots; only data values are provided at white dots.

In Figure 8, the latitude and longitude are 11-by-2 arrays shown by red dots. In this swath example, the physical data field “CloudCover” is an 11-by-3 array shown by both red and white dots. Latitude and longitude for the points in the mid-column of the data files are not provided directly and must be interpolated based on information provided by the dimension map.

Mapping an HDF-EOS5 swath to DAP2 is very similar to mapping an HDF-EOS5 grid with a 2-D projection to DAP2. In fact, the layouts of the DAP2 DAS and DDS are the same as for the 2-D projection HDF-EOS5 grid case. Several key attributes requested by CF conventions need to be provided to ensure that OPeNDAP clients can access HDF-EOS5 data.

The DDS of the DAP2 file mapped from the HDF-EOS5 swath shown in Figure 7 is:

```
Dataset {  
    Float32 lat[lat= 11][lon= 3];  
    Float32 lon[lat= 11][lon= 3];  
    Float32 CloudCover[lat= 11][lon= 3];  
}
```

The DAS of the DAP2 file mapped from the HDF-EOS5 swath shown in Figure 7 is:

```
Attributes {  
  
    lon {  
        String units "degrees_east";  
        String long_name "longitude";  
    }  
    lat {  
        String units "degrees_north";  
        String long_name "latitude";  
    }  
    CloudCover {  
        String coordinates "lon lat";  
    }  
}
```

The DAS and DDS of the DAP2 file mapped from the HDF-EOS5 swath shown in Figure 8 are the same as those for the swath shown in Figure 7. However, that longitude and latitude must be interpolated using the dimension maps for the HDF-EOS5 swath that is shown in Figure 8.

4.3 Other issues related to mapping HDF-EOS5 to DAP2

This subsection covers the special HDF-EOS5 mapping issues not discussed in the previous two subsections. Since most issues discussed here are still under investigation, the main purpose of this subsection is not to provide comprehensive solutions but merely to bring up these issues. Some discussions are terse. Ideas and thoughts presented here are intended to benefit future work in these areas.

4.3.1 Other coordinate variables

Some HDF-EOS5 data have coordinate variables other than latitude and longitude. For example, the coordinate variables “pressure” and “time” are frequently seen in some HDF-EOS5 files. However, HDF-EOS5 does not provide a way to identify coordinate variables other than “latitude” and “longitude,” even though there are coordinate variables such as “time” and “pressure”. In other words, variables such as time and pressure cannot be identified as coordinate variables just by analyzing the HDF-EOS5 file alone. They must be specified by other means. Without knowing other coordinate variables, OPeNDAP clients will fail to visualize the variable associated with these coordinate variables.

One way to solve this problem is to provide a supplement file that provides the mapping. For example, the supplement file could be “coordinate variable names”, “pressure”, “time”. In this way, pressure and time can be identified as coordinate variables and be passed to OPeNDAP visualization clients.

4.3.2 Missing attributes

The current version of CF conventions lists more than 30 attributes[13]. In previous sections, we discussed how to provide several key attributes, such as “coordinates” for variables, “units” for latitude and longitude, and “long_name” for variables. However, to correctly visualize and analyze the HDF-EOS5 data, other attributes need to be provided (attributes such as “units” for general variables, “FillValue” for variables that have missing or undefined data, and “scale_factor” and “add_offset” for data to be multiplied by the “scale_factor” and then added by the “add_offset”).

Although HDF-EOS5 files provide “units” and “FillValue” if the data is applied, it should be expected that some other attributes required by OPeNDAP clients to correctly visualize and analyze the data may still be missing in future HDF-EOS5 files. Furthermore, some attribute values may not follow CF conventions. One solution is to provide missing attributes in a supplement file, as discussed in the previous section.

4.3.3 Special characters

According to CF conventions, variable, dimension and attribute names should begin with a letter and be comprised of letters, digits and underscores, but some HDF-EOS5 variables do not follow CF conventions. For example, the variable name may start with a number and contain special

characters, such as a period. Currently most OPeNDAP visualization clients are only partially following CF conventions: many client tools allow variable names to start with underscores. However, most client tools do not allow the special character period (.) inside a variable name. To ensure that most OPeNDAP visualization client tools can access HDF-EOS5 files, the problem with special characters needs to be addressed. One solution is to change the unconventional special character, such as a period, in a variable name to a CF-conventional character, such as an underscore, and keep the original name as the value of the attribute “long_name”.

5 Summary

Mapping HDF5 to DAP2 is not straightforward because HDF5 and DAP2 have different data models. Some HDF5 objects are not mapped to DAP2 because the mapping is difficult, or in some cases impossible. Furthermore, it is typically the case that the earth science community rarely uses the objects that are not mapped. To ensure that OPeNDAP clients have access to NASA HDF-EOS5 data, future developers of DAP2 data handlers and servers should adopt the special mappings for HDF-EOS5 grid and swath objects described in this technical note.

Tables 2–4 summarize the HDF5-to-DAP2 mapping information discussed in this technical note.

Table 2. General HDF5-to-DAP2 mapping

General Mapping	
HDF5 Datatype	DAP2 Datatype
8-bit unsigned integer	Byte
8-bit signed integer	Int16
16-bit unsigned integer	UInt16
32-bit unsigned integer	UInt32
16-bit signed integer	Int16
32-bit signed integer	Int32
32-bit floating-point	Float32
64-bit floating-point	Float64
String	String
Object reference	URL
Region reference	URL
Compound ¹	Structure
Dataset ²	Variable
Attribute ³	Attribute
Group ⁴	1. Special attribute, HDF5_ROOT_GROUP, to represent the HDF5 group structure 2. Absolute path of the HDF5 dataset as the DAP2 variable name
1. HDF5 compound can be mapped to DAP2 under the condition that the base members (excluding object/region references) of compound can be mapped to DAP2. 2. HDF5 dataset can be mapped to DAP2 under the condition that the datatype of the HDF5 dataset can be mapped to DAP2. 3. HDF5 attribute can be mapped to DAP2 under the condition that the datatype of the HDF5 dataset can be mapped to DAP2, and the data is either scalar or one-dimensional array. 4. HDF5 group can be mapped to DAP2 under the condition that the file structure is a tree structure.	

Table 3. HDF-EOS5-to-DAP2 special mapping

Special HDF-EOS5 Mapping	
HDF-EOS5 grid with 1-D projection	<p>Solution 1:</p> <ol style="list-style-type: none"> 1. Map a data variable and the associated coordinate variables to a DAP2 Grid. 2. Ensure that the attributes of latitude and longitude follow CF conventions. <p>Solution 2:</p> <ol style="list-style-type: none"> 1. Map a data variable to a DAP2 Array. Add the DAP2 “coordinates” attribute for the data variable. The “coordinates” attribute provides the name list of the associated coordinate variables. It should be presented by following CF conventions. 2. Ensure that the attributes of latitude and longitude follow CF conventions.
HDF-EOS5 grid with 2-D projection	<ol style="list-style-type: none"> 1. Map data variables to DAP2 Array variables. 2. Generate two DAP2 variables that represent longitude and latitude. 3. Ensure that attributes of latitude and longitude to follow CF conventions. 4. Add an attribute, coordinates, for each DAP2 variable. The “coordinates” attribute provides the name list of the associated coordinate variables. It should be presented by following CF conventions.
HDF-EOS5 swath	<ol style="list-style-type: none"> 1. Map all fields to DAP2 Array variables. 2. Generate two DAP2 variables to represent longitude and latitude. 3. Ensure that attributes of latitude and longitude to follow CF conventions. 4. Add an attribute, coordinates, for each DAP2 variable. The “coordinates” attribute provides the name list of the associated coordinate variables. It should be presented by following CF conventions.

Table 4. HDF5 datatypes briefly discussed within this technical note

Signed integer when the number of bits is greater than or equal to 64
Unsigned integer when the number of bits is greater than or equal to 64
Floating-point when the number of bits is greater than 64
Bitfield
Opaque
Enumeration
Array
Variable Length
Named datatype

Appendix: Displaying an HDF-EOS5 Grid Using IDV

The example in Figure 9 demonstrates how an OPeNDAP IDV client accesses HDF-EOS5 aura data.

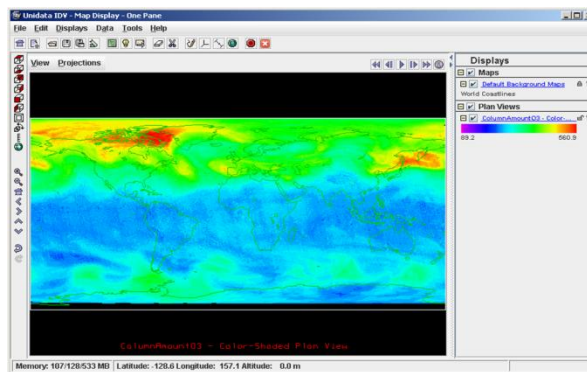


Figure 9. Displaying a NASA AURA OMI HDF-EOS5 file using IDV via the HDF5-OPeNDAP handler configured with the CF option. The physical field is “ozone” and the unit is “DU”.

Glossary of Acronyms

ANSI	American National Standards Institute
API	Application Programming Interface
CDL	Common Data form Language
CF	Climate and Forecast
DAP	Data Access Protocol
DAP2	Data Access Protocol Version 2
DAS	Dataset Attribute Structure
DDS	Dataset Descriptor Structure
DDX	Dataset Descriptor in XML
EOS	Earth Observing System
GrADS	Grid Analysis and Display System
HDF	Hierarchical Data Format
HDF5	Hierarchical Data Format Version 5
IDV	Integrated Data Viewer
IEEE	Institute of Electrical and Electronics Engineers
NASA	National Aeronautics and Space Administration
NetCDF	Network Common Data Form
NPOESS	National Polar-orbiting Operational Environmental Satellite System
OMI	Ozone Monitoring Instrument
OPeNDAP	Open-source Project for a Network Data Access Protocol
R&D	Research and Development
URL	Uniform Resource Locator
XML	Extensible Markup Language

Informative References

- [1] HDF website: <http://hdfgroup.org>
- [2] ESDS-RFC-007 HDF5: <http://www.esdswg.com/spg/rfc/ese-rfc-007/ESDS-RFC-007v1.pdf>
- [3] HDF-EOS website: <http://www.hdfeos.org>
- [4] ESDS-RFC-008 HDF EOS 5:
<http://www.esdswg.com/spg/rfc/ese-rfc-008/ESDS-RFC-008v1.0.pdf>

- [5] OPeNDAP website: <http://opendap.org>
- [6] ESDS-RFC-004 The Data Access Protocol – DAP 2.0:
<http://www.esdswg.com/spg/rfc/ese-rfc-004/ESE-RFC-004v1.1.pdf>
- [7] *HDF5 User's Guide*: <http://www.hdfgroup.org/HDF5/doc/UG/index.html>
- [8] *OPeNDAP User Guide*: <http://www.opendap.org/user/guide-html/guide.html>
- [9] Integrated Data Viewer (IDV): <http://www.unidata.ucar.edu/software/idv/>
- [10] Grid Analysis and Display System (GrADS): <http://www.iges.org/grads/>
- [11] Classic NetCDF Data Model:
<http://www.unidata.ucar.edu/software/netcdf/workshops/2008/datamodel/NcClassicModel.html>
- [12] ESDS-RFC-011 NetCDF Classic:
<http://www.esdswg.com/spg/rfc/esds-rfc-011/ESDS-RFC-011v1.00.pdf>
- [13] NetCDF Climate and Forecast (CF) Metadata Conventions:
<http://cf-pcmdi.llnl.gov/documents/cf-conventions/1.4/>

Authors

The HDF5-OPeNDAP Project Team:
MuQun Yang, The HDF Group, myang6@hdfgroup.org
James Gallagher, OPeNDAP, jgallagher@opendap.org
Hyo-Kyung Lee, The HDF Group, hyoklee@hdfgroup.org