

Mapping HDF5 Group to DAP

MuQun Yang, Hyo-Kyung Lee, Mike Folk

The HDF Group

James Gallagher

OPeNDAP

I. Introduction

In HDF5 an object called *group* is used to show relationships among objects. An HDF5 file may be viewed as a rooted directed-graph, with at least one group, called the *root group*. An object in HDF5 can belong to different groups. HDF5 uses a *link* to distinguish which group an object is associated with. Besides group, the most common HDF5 object is an HDF5 *dataset*. Essentially an HDF5 dataset consists of a data array with some metadata information (called an attribute) to describe the array. One can compare HDF5's group, dataset and link 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 path to a directory or to a file. A path of an object starting from the HDF5 root group is called an absolute path of the object. We will not discuss the case when there are multiple absolute paths to an HDF5 object in this document.

Defining an appropriate data mapping from the HDF5 file structure to DAP Version 2 (DAP hereafter) is not trivial. DAP does not have a concept that corresponds to HDF5's group structure. We will evaluate three options to represent HDF5 groups with auxiliary DAP records.

We will use the following example to investigate different presentations of group mapping.

An HDF5 example:

An HDF5 file (foo.h5) includes four groups, the root group (/), g1, g2 and g3. g1 includes two datasets ds1 and ds2. g2 includes two datasets d1 and d2. g3 also includes two datasets d3 and d4. d1 is an integer array of 100 elements. d2 is an floating point array of 100 elements. d3 is a scalar and the datatype is int. d4 is also a scalar and the datatype is floating point. ds1 and ds2 are scalar values and the datatype is HDF5 compound datatype. An HDF5 compound datatype is equivalent to a C struct. ds1 is equivalent to struct s1 and ds2 is equivalent to struct s2.

```
s1{
    int i;
    float f;
}
s2{
    int ai[100];
    float af[100];
}
```

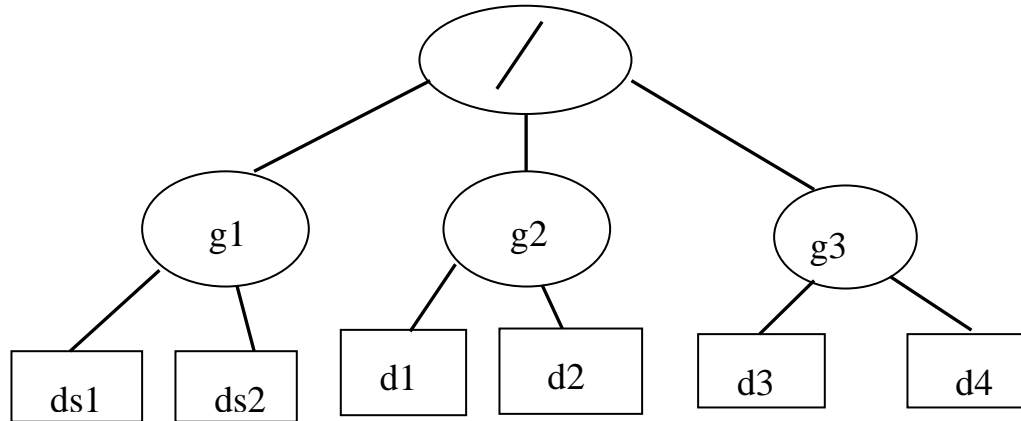


Figure 1: An illustration of an HDF5 example

II. Relevant Requirements for the HDF5 Handler

In this section we describe the requirements for the handler with respect to the handling of HDF5 groups. There are other requirements, but those are either specific to other aspects of the handler or general requirements that all DAP format handlers for use with the Hyrax data server must satisfy.

1. The handler must encode every HDF5 dataset in a way that will enable a generic client, with no special knowledge of the HDF5 handler's operation, to read those data.
2. The handler must take into account that DAP has no native concept equivalent to HDF5 groups.
3. Except some known cases 1, the handler must provide a way, for any given HDF5 input file, for a savvy client to write an HDF5 file that is indistinguishable from that input file.

III. Option #1: A DAP structure

The DAP structure can be used to represent grouping in the following way. One can define an HDF5 group as a single DAP 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. This is how the HDF4-DAP server handled HDF4 Vgroups.

The whole HDF5 file will be mapped to a DAP structure:

```

Dataset {
    Structure {

```

¹ Some HDF5 files cannot be fully recovered such as “named datatype”, “bitfield”, “variable length type” et c. Either there is no corresponding DAP datatype or it is too difficult to represent those datatypes in DAP.

```

Structure {
    Structure {
        Int32 ai[100];
        Float64 af[100];
    } ds1;
    Structure {
        Int32 i;
        Float64 f;
    } ds2;
} g1;
Structure {
    Int32 d1[100];
    Float64 d2[100];
} g2;
Structure {
    Int32 d3;
    Float64 d4;
} g3;
} /;
} foo.h5;

```

Figure 2: Mapping the example HDF5 file to a DAP structure

There are substantial limitations for this approach. First of all, the structure is a tree. Any loops or multiple links to the same object cannot be directly represented.

Another disadvantage of this approach is that 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 DAP clients, which expect a list of data arrays, not one big, complicated structure. It will also make it difficult for DAP clients to send a request to obtain a subset of an HDF5 dataset with expression constraints.

The fatal flaw for this approach is that it will cause ambiguities for an HDF5 dataset using an HDF5 compound datatype. An HDF5 compound datatype is equivalent to a C struct, which should naturally be mapped to a DAP structure. However, since an HDF5 group is also mapped to a DAP structure, it will make the DAP clients treat both the group and dataset having compound datatype as DAP structures. This will cause confusion. For example, as illustrated in Figure 2, compound dataset ds1 cannot be distinguished from group g2 and compound dataset ds2 cannot be distinguished from group g3. If an HDF5 file is mapped in this way, it is impossible for an HDF5 client or a DAP to HDF5 conversion tool to properly understand the structure of the HDF5 object.

IV. Option#2: DAP attributes

All HDF5 datasets will be mapped to DAP variables, with information about the group structure mapped as DAP attributes. This will be natural for most DAP applications; the file will be presented as a set of DAP 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. This has been implemented inside the current HDF5-DAP handler.

Illustration with the example file:

When mapping to DAP, in the DAP DAS table,

The following attributes can be found:

DAS:

```
Attributes {
    HDF5_Root_Group {
        /g1 {
            String Datasets “/ds1” “/ds2”;
        }
        /g2 {
            String Datasets “/d1” “/d2”;
        }
        /g3 {
            String Datasets “/d3” “/d4”;
        }
    }
}
```

Note that the ‘Datasets’ String attributes in the two containers ‘/g1’ and ‘/g2’ can be vectors. So they can list a vector of values of HDF5 dataset names.

DDS:

```
Dataset {
    Structure {
        Int32 I;
        Float64 f;
    } /g1/ds1;
    Structure {
        Int32 ai[100];
        Float64 af[100];
    }
```

```

    } /g1/ds2;
    Int32 /g2/d1[100];
    Float64 /g2/d2[100];
    Int32 /g3/d3;
    Float64 /g3/d4;
} foo.h5;

```

We can also represent this using the DDX as:

```

<Dataset name="foo.h5" >
  <Attribute name="HDF5_Group" type="Container">
    <Attribute name="/g1" type="Container">
      <Attribute name="Datasets" type="String">
        <value>"/ds1 /ds2"</value>
      </Attribute>
    </Attribute>
    <Attribute name="/g2" type="Container">
      <Attribute name="Datasets" type="String">
        <value>"/d1 /d2"</value>
      </Attribute>
    </Attribute>
    <Attribute name="/g3" type="Container">
      <Attribute name="Datasets" type="String">
        <value>"/d3 /d4"</value>
      </Attribute>
    </Attribute>
  </Attribute>

  <Structure name="/g1/ds1">
    <Int32 name="I"/>
    <Float64 name="f"/>
  </Structure>
  <Structure name="/g1/ds2">
    <Array name="ai">
      <Int32/>
      <dimension size="100"/>
    </Array>
    <Array name="af">
      <Float64/>
      <dimension size="100"/>
    </Array>
  </Structure>

  <Array name="/g2/d1">
    <Attribute name="long_name" type="String">
      <value>U_Wind_Vector</value> <!--just an example attribute -->
    </Attribute>
    <Int32/>
    <dimension size="100"/>
  </Array>

```

```

</Array>
<Array name="/g2/d2">
  <Int32/>
  <dimension size="100"/>
</Array>

<Int32 name="/g3/d3"/>
<Float64 name="/g3/d4"/>
</Dataset>

```

V. Option #3: Represent group in DAP structure

One can map the whole HDF5 file structure to a DAP variable of which the data type is a DAP structure that contains only metadata information about the HDF5 file structure. The group name and dataset name will be stored as individual field names of the structure. We may choose to use different strings to separate group from dataset. For example, 'G' can be used to represent a Group. 'D' can be used to represent a Dataset.

Illustration with the example HDF5 file

The following is the DDS representation of HDF5 file structure via OPeNDAP:

```

Dataset {
  Structure {
    /g1 'G';
  /g2 'G';
    /g3 'G';
    /g1/ds1 'D';
    /g1/ds2 'D';
    /g2/d1 'D';
    /g2/d2 'D';
    /g3/d3 'D';
    /g3/d4 'D';
  } /;
} foo.h5;

```

VI. Recommendations

We chose option two because it provides a way to represent the HDF5 group information in a way that can be used by DAP clients that understand it. Clients that do not need to use the group information can simply ignore it. Option one provides a lossy representation of the group structure, so savvy applications will not be able to reproduce the original data source's group and compound dataset hierarchy. Option three is very similar to option two except that a DAP variable is used to store the metadata for the group; this seems unwise since applications will expect that type of metadata in DAP attributes.