RFC: h5attribite – command line tool to handle HDF5 attributes

Albert Cheng, Jonathan Kim, Elena Pourmal

This document proposes a new command line tool for handling attributes in HDF5 file and solicits requirements for the tool's functionality and interface.

1 Introduction

The HDF5 distribution contains command line utilities for creating datasets (h5import, h5copy) and groups (h5mkgrp, h5copy). Those tools become handy, when a modification like adding new dataset or new group to an existing file or collection of files has to be done without actual programming, or when a new file is created while one works on a data layout prototype in HDF5 file. None of the utilities mentioned above, can create HDF5 attributes, and therefore one has to write and run a program to do so.

To address the lack of such functionality in HDF5, we propose to create a command line utility `h5attribute` to handle attributes in HDF5 file. This document outlines the major use cases, discusses examples of tool interface, and solicits requirements for the tool.

2 Use cases for handling attributes

This section describes several use cases that illustrate how the users may use the tool to handle attributes in an HDF5 file.

- A user would like to add an attribute to an HDF5 object (e.g., dataset or group) without writing a program. He uses the `h5attribute` tool and specifies necessary input on a command line to add the attribute.
- A user would like to use a visualization tool to display some data stored in an HDF5 dataset, but the tool requires an attribute with a special name on the dataset in order to display the data. The user uses `h5attribute` to add the required attribute and proceeds with the visualization.
- A user runs an application in a parallel environment to create a big HDF5 file with size of multiple gigabytes, containing thousands of datasets. When he runs a different application to read the data, he discovers that he has made a mistake and used the wrong name for an attribute in each of the datasets, making his file incompatible with the second application. Instead of rewriting and rerunning his first application, he uses the `h5attribute` tool to delete the incorrectly named attribute from all datasets and add a new one with the correct name to every dataset.

- A user wants to add an attribute with a complex data type such as an array of nested C-language structure. The user creates an input file with a description of the data type along with other required information for the attribute. The user uses `h5attribute` to read the input file to generate the attributes.
- A user wants to add several attributes to a dataset. The user creates an input file containing the description of the attributes. He specifies the name of the input file along with the name of the dataset as input to `h5attribute` to generate the attribute.
- A user has a collection of HDF5 files and would like to add (or remove, or copy) an attribute to a group in every file. User creates a shell script and uses `h5attribute` to modify the files.
- A user wants to duplicate an attribute in a new HDF5 file, as it is in an original file. The user runs `h5dump` on the original HDF5 file and finds HDF5 DDL (http://www.hdfgroup.org/HDF5/doc/ddl.html) representation of the desired attribute. The user copies the attribute's DDL description to an input file and makes necessary changes. The user specifies the input file as an input to `h5attribute`.
- A user would like to change an attribute value. He uses `h5attribute` and specifies the name of the attribute and its new value on the command line to change the value stored in an HDF5 file.

3 Proposal for the h5attribute tool

To address the lack of functionality in HDF5 described in the sections above, we propose to implement the `h5attribute` tool to perform certain operations on HDF5 attribute(s). We are soliciting input for the new tool's functionality, command line syntax and input parameters. Use cases above and proposal below is based on the requests and suggestions already received by The HDF Group developers.

Syntax:

`h5attribute` *<file> <op>[<op-parameters>;]…*

Description:

The `h5attribute` tool performs on the HDF5 file, named *<file>*, operations specified by the *<op>* flag on an attribute specified by *<op-parameters>*. The *<op>* flag and corresponding operations are defined as follows:

`create` : create a new attribute in an object

`delete`: delete an existing attribute

<op-parameters> specifies information required to perform the operation; syntax depends on the specified operation *<op>*. (The create operation here is equivalent to the add operation described above.)

Sections below discuss syntax of *<op-parameters>* for creating and deleting attributes since those were already given a high priority by the HDF5 users and will be implemented in the initial version of the tool.

Other operations to consider are:

`rename:` modify the name of an existing attribute

`modify:` modify the value(s) of an existing attribute

`copy:` copy an existing attribute from one object to another object

`move:` move an existing attribute from one object to another object

`exist:` check if an attribute exists

Suggestions for other operations are more than welcome.

3.1 Creating attributes

Syntax:

- `h5attribute` *`FILE_NAME`* `create` *`ATTRIBUTE_NAME ATTRIBUTE_VALUE`* `[`*`OBJECT_NAME`* `…]`
- `h5attribute` *`FILE_NAME`* `create` *`ATTRIBUTE_NAME ATTRIBUTE_VALUE`* `[`*`OBJECT_NAME`* `…] “;” create` *`ATTRIBUTE_NAME ATTRIBUTE_VALUE`* `[`*`OBJECT_NAME`* `…]`
- `h5attribute` *`FILE_NAME`* `--command` *`CFILE_NAME`*

Description:

According to the first syntax, h5attribute creates an attribute with a name specified by *ATTRIBUTE_NAME* and a value specified by the following parameter *ATTRIBUTE_VALUE*, in objects specified by one or more *OBJECT_NAME.*

If the ATTRIBUTE_NAME contains a slash (/), it is divided at the last slash into two parts, called *head* and *tail*; where as the *head* is everything up to the last slash and the *tail* is everything after the last slash. Therefore *head/tail* is same as the ATTRIBUTE_NAME. The *tail* is treated as the name of the attribute to be created and the *head* is the pathname of the object in which the new attribute is created.

The ATTRIBUTE_VALUE consists of three parts, *attribute_type, attribute_space,* and *attribute_data.* The *attribute_type* and *attribute_space* parts are optional.

The *attribute_type* specifies the data type[1]of the attribute. If it is missing, the default value of H5T_NATIVE_INT (C native int) is used.

The *attribute_space* specifies the data space of the attribute. If it is missing, the *attribute_space* is defined according to *attribute_data* given. If only one data point is given, *attribute_space* is defined as H5S_SCALAR (scalar dataspace). If more than one data points are given, *attribute_space* is defined as 1-D array with the dimension size equals to the number of data points given.

[1] http://www.hdfgroup.org/HDF5/doc/RM/PredefDTypes.html shows the predefined types

The second syntax shows that multiple create commands are allowed and are separated by semicolons.

The last syntax specifies that the command file, *CFILE_NAME,* contains the create commands according to the same syntax above.

If the specified attribute already exists or the target objects do not exist, the tool will fail and display an error message accordingly.

Examples:

The following three commands show how to create an integer, a pair of floating points, and a string attribute in the objects */m1 and /m2* in the file *file.h5*.

- h5attribute file.h5 create /m1/*Percentage_per_Volume 40*
- h5attribute file.h5 create /m2/*GPS_Location "{H5T_IEEE_F32LE {0.0,180.0}}"*
- h5att*ribute* file.h5 create *"Temp Scale" "{H5T_C_S1 {\"Celsius\"}" /m1 /m2*

The last command above creates the same string attribute in two objects, */m1* and */m2,* in the file *file.h5*.

The next command uses a command file to apply the above three commands to file *file2.h5* in one execution.

- h5attribute *file2.h5* --command *my_command*

The contents of the *my_command* file follows:

```
create /m1/Percentage_per_Volume 40;
create /m2/GPS_Location {
      DATATYPE H5T_IEEE_F32LE
      DATASPACE SIMPLE {(2)/(2)}
      DATA {0.0, 180.0}
      };
create ATTRIBUTE "Temp Scale" {
      DATATYPE H5T_C_S1
      DATA {"Celsius"}
      }
      /m1 /m2;
```

The keywords, DATATYPE, DATASPACE and DATA are optional. They are used for clarity. A detail explanation of the syntax is listed in Appendix 1.

3.2 Deleting attributes

Syntax:

- h5attribute *FILE_NAME* delete *ATTRIBUTE_NAME* [*OBJECT_NAME* …]
- h5attribute *FILE_NAME* delete *ATTRIBUTE_NAME* [*OBJECT_NAME* …] *";"* delete *ATTRIBUTE_NAME* [*OBJECT_NAME* …]

- h5attribute *FILE_NAME* --command *CFILE_NAME*

Description:

According to the first syntax, the h5attribute tool deletes the attribute with a name specified by *ATTRIBUTE_NAME* from objects specified by one or more *OBJECT_NAME.* The ATTRIBUTE_NAME uses the same syntax as specified in the create command above.

The second syntax shows that multiple delete commands are allowed and are separated by semicolons.

The last syntax specifies that the command file, *CFILE_NAME,* contains the delete commands according to the same syntax above.

Examples:

The following commands show how to delete attributes.

- h5attribute *file.h5* delete /m1/*Percentage_per_Volume ";"* delete /m2/*GPS_Location*
- h5attribute *file.h5* delete *"Temp Scale" /m1 /m2*
- h5attribute *file2.h5* --command *my_command*

The first command deletes *"Percentage_per_Volume" and "GPS_Location"* attributes from the objects */m1 and /m2 respectively,* in file *file.h5.* The second command deletes *"Temp Scale"* attribute from the objects */m1* and */m2* in file *file.h5.* The third command reads the delete commands in file *my_command* and apply them to file *file2.h5.*

4 Discussion of input information to the h5attribute tool

[Note that this section is moot since we have decided to use the DDL format for command syntax. The previous text have been moved to the appendix section as a reference for now and will likely be deleted from the final version of this design.]

5 Atomicity of h5attribute

The operation effect of h5attribute should be atomic, that is, the changes must be done as all or nothing. Consider the following example which tries to add the attribute "Temp_Scale" to objects /m1, /m2, …, /m100.

- h5attribute file.h5 create *"Temp Scale" "{H5T_C_S1 {\"Celsius\"}" /m1 /m2 … /m100*

It happens that all specified objects, except /m50, do not have the attribute "Temp_Scale". Therefore, the command will fail halfway through, with /m1 to /m49 changed. Not all users like to have the data file partially changed. They would rather have no changes applied to the original data file, find out the cause of the failure, correct the command and apply it again without errors.

It is also crucial that h5attribute should avoid exiting the program with the HDF5 data file in an unstable state. Consider the example above, when h5attribute detects that the new attribute cannot

be created in /m50, it should print an error message explaining the cause of the failure, close all open objects, close the data file, then exit with a failure code.

5.1 No Atomicity Option

A naïve way to support Atomicity operation is to make a backup copy of the HDF5 data file at the very beginning of h5attribute and then process the commands requested. If it succeeds, discard the backup copy and quite. If it fails somewhere, discard the new file and restore original file from the backup copy. This is expensive and some users may not care. Therefore, h5attribute will support atomic change as the default but provide an option, say, --noatomic, to allow partial changes to occur. Again, the partially changed data file must be a properly closed HDF5 file.

5.2 Incremental Atomicity Option

Some users may also prefer an incremental atomicity which means atomicity at individual command level is needed, not the entire execution. A simple but not necessarily most efficient way to implement the *incremental atomicity* is to close the data file after every successful execution of one command, save a backup copy of the data file, reopen the file for the next command execution. If the next command fails, the data file is restored from the backup copy.

6 Exit code from the h5attribute

- 0: when operation succeeds.
- 1: when operation fails.

However more error conditions can be added as work progresses. If added, the exit code values will be adjusted accordingly. In such case, the exit code values will be bigger than 0.

7 Conclusion

Currently a user has to write a program to add or remove attribute(s) from object(s) in HDF5 file.

It's very inconvenient to deal with programming repeatedly when the attribute is simple or the operation over the attribute involves simple actions.

The `h5attribute` tool will provide a convenient way to manipulate attribute(s) on object(s) in HDF5 file.

For the initial implementation, essential operations like adding and deleting attribute(s) will be provided. Along with that this RFC also presented extensible framework for additional operations that can be added in the future.

Appendix 1: Detail explanation of h5attribute syntax
[will be provided in the next revision.]

Appendix 2: List of Configuration Keywords of h5import

- RANK
- DIMENSION-SIZES
- PATH
- INPUT-CLASS
- INPUT-SIZE
- OUTPUT-CLASS
- OUTPUT-SIZE
- OUTPUT-ARCHITECTURE
- OUTPUT-BYTE-ORDER
- CHUNKED-DIMENSION-SIZES
- COMPRESSION-TYPE
- COMPRESSION-PARAM
- EXTERNAL-STORAGE
- MAXIMU-DIMENSIONS

8 Discussion of input information to the h5attribute tool

[Note that this chapter is moot since we have decided to use the DDL format for command syntax. This is kept here as a reference for now and will likely be deleted from the final version of this design.]

This section discusses required information to create an attribute on a HDF5 object and different approaches to obtain the information.

When HDF5 library creates an attribute, the following information has to be provided:

1. Attribute name
2. Attribute value
3. Object to which the attribute will be attached
4. Data type used to store attribute value in the file
5. Dimensionality (rank and sizes of dimensions)

8.1 Providing attribute information on a command line

- For adding an attribute, the following flags and information should be provided:
    1. `--add` flag followed by the name of the attribute .
    2. `--value` flag followed by the attribute value(s); the value types are limited to integer, floating point, , and string. Multiple values, separated by commas, can be specified.
    3. `--type` followed by the type specification TYPE. Current implementation is restricted to the predefined HDF5 types, for example, H5T_IEEE_F64BE.
       (See http://www.hdfgroup.org/HDF5/doc/RM/PredefDTypes.html for the predefined types)
    4. `--object` followed by the path to the object to which the attribute will be attached to. The flag may be repeated to specify multiple objects.
    5. An HDF5 file name after all flags and values.

Since command line should be used for adding simple attributes only, it is not required to provide a dimensionality of the attribute. It is assumed to be a scalar (H5S_SCALAR) or a one-dimensional array depending on the attribute value. Please note that "," serves as a separator for the attribute values and it cannot be used as a part of the string value, e.g., "one, two, three" is considered as three strings instead of one string with two commas in it.

- For deleting an attribute, the following flags and information should be provided:
    1. `--del` flag followed by the name of the attribute .
    2. `--object` followed by the path to the object from which the attribute is deleted. The flag may be repeated to specify multiple objects.
    3. An HDF5 file name after all flags and values.

Future consideration of the 'type' flag:

Flag '--type' may be an optional feature when datatype of the attribute can be "recognized" from the specified value by parsing the value string.

For example,

- h5attribute --add *"Ignition_Temp"* –value *"451.0"* --object */m2 file.h5*

Noticed that no '--type' is specified. The attribute value "*451.0*" will be written as a floating-point number scalar (i.e., using H5T_NATIVE_FLOAT(DOUBLE) datatype and H5S_SCALAR dataspace).

The h5attribute will parse the given value string to figure out the appropriate datatype. The implementation would be limited to integer, floating point and string data types.

8.2 Providing attribute information in configuration file

Using a configuration file as an input is designed to work for multiple attributes of any complexity on the multiple objects. HDF5 tools have already defined several different data model formats that h5attribute may adopt. In the following subsections, we will describe these formats and other additional requirements of the configuration file that h5attribute needs.

8.2.1 Analysis of possible file formats

In this subsection, we describe three possible file formats that h5attribute may adopt. We also present the pros and cons of each format according to these criteria:

- Completeness of the format definition
- Extensibility of the format definition
- User friendliness of the format
- Reuse of existing code

8.2.1.1 Using configuration file format similar to the h5import tool

Currently, h5import obtains information to generate a dataset from a configuration file. (Appendix 1 shows a list of Configuration Keywords of the h5import tool. For more details, read the h5import tool reference page.) h5import requires information about datasets, and since HDF5 datasets and attributes have a lot in common, we can reuse the syntax (and the corresponding parsing code) to provide information about the attribute. Here is an example of how *my_attr_conf* file from the section 3.1 may look like:

```
%cat  my_attr_conf
      PATH /m1
      PATH /m2
      ATTRIBUTE-NAME "Speed Vector"
      INPUT-CLASS TEXTIN
      INPUT-SIZE 64
```

```
OUTPUT-BYTE-ORDER BE
OUTPUT-ARCHITECTURE STD
RANK 1
DIMENSION-SIZES 3
ATTRIBUTE-DATA
1,2,3
```

Three new keywords PATH, ATTRIBUTE-NAME and ATTRIBUTE-DATA are introduced to the syntax of the attribute configuration file.

Pros:

- The format is human readable, and can be easily edited by hand.
- The format is similar to the input of the h5import utility. Users familiar with h5import can easily learn the new tool.
- h5import parser code can be reused for h5attrubute reducing implementation time.

Cons:

- Some key words are not intuitive (for example, TEXTIN means an integer number represented by a string and doesn't mean a text input). We can introduce new keywords and obsolete the current ones in the future.
- The format is of the fixed format type as the keyword and value must be on the same line of input. This disallows either a more compact view of multiple keyword-value on one line or the free format of continuation of long list of values onto multiple lines of input.
- The format definition covers only the basic types. If h5attribute is extended to cover other types (e.g., bit-field, compound, …), it needs to be expanded if possible.
- If more complex input structure is needed, the one keyword-value per line restriction will take many lines to define one structure. That will become harder for humans to follow.

8.2.1.2 Using DDL format used by the h5dump tool

The h5dump tool has a defined Data Description Language (DDL) as the output format. For details of the DDL, refere to http://www.hdfgroup.org/HDF5/doc/ddl.html. This format is also used by the H5LTtext_to_dtype() and H5LTdtype_to_text() functions.

```
%cat  my_attr_conf
```

```
PATH "/m1"
PATH "/m2"
ATTRIBUTE "Speed Vector" {
    DATATYPE  H5T_STD_I64BE
    DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
    DATA {
    (0): 1, 2, 3
    }
}
```

The PATH keyword specifies a full path to an object to which the given attribute is added.

It is currently not a part of the DDL format; however it can either be added to DDL or it can be considered as a section identifier in the configuration file for each attribute specified there.

Pros:

- The format is similar to the output of the `h5dump tool`. Users familiar with `h5dump` can easily learn the new tool.
- User can easily create a configuration file especially for the complex attributes by using the output of h5dump on an existing file.
- The format is human readable and can be easily edited by hand.
- The H5LTtext_to_dtype() function code can be reused by h5attribute to parse the input.

Cons:

- The format depends on DDL, so if DDL changes, the format also needs to be changed as well as the parsing code. With careful design, any new changes to DDL should be backward compatible so that the new version of DDL format is a superset of the previous version. Therefore, h5attribute can choose to accept those new changes at its own pace.
- Implementing parser for this format will require more work comparing to reusing `h5import` syntax discussed in 4.2.1.1. However, the H5LTtext_to_dtype() function code can be reused.

8.2.1.3 Other considerations for possible input

1. Configuration file in the XML format

   Pros:

   - Since XML is a widely used standard format a user can take advantage of many XML tools to manipulate input configuration file in this format.

   Cons:

   - Currently XML is not well supported with HDF5. We will need to create an XML schema for HDF5, update h5dump, create documentation, etc. However, providing a better support for XML with HDF5 will be a big plus for the HDF5 users.
   - Creating a configuration file in the XML format will require a substantial development effort for the h5attribute tool

2. DDL description for complex datatypes on a command line or in a configuration file.

   This option will allow extending proposed specification of an attribute's datatype TYPE to a more complex datatypes.

   - Examples:
     - H5T_STRING { STRSIZE H5T_VARIABLE; STRPAD H5T_STR_NULLPAD; CSET H5T_CSET_ASCII; CTYPE H5T_C_S1; }
     - H5T_ENUM { H5T_STD_I32LE; \"RED\" 5; \"GREEN\" 6; \"BLUE\" 7; }
     - H5T_VLEN { H5T_VLEN { H5T_STD_I32BE } }
     - H5T_COMPOUND { H5T_STD_I16BE \"field1\" : 2; H5T_STD_U8LE \"field2\" : 6; }

Pros:

- DDL input will allow us to use the H5LTtext_to_datatype function to parse datatype specification for creating a datatype for an attribute. Implementation of the parser will be easy and straightforward.

Cons:

- While this approach can handle creation of the complex types, it requires a user to figure out how to express such complex types in this format, which is not trivial to do without a proper tool that doesn't exist at this point.

8.2.2 Specifying multiple attributes

Configuration file contains the specifications of the objects and the attributes that will be attached to the objects by `h5attribute`. We need to impose some rules on a structure of the configuration file in order for the parser to understand which attribute(s) goes with which object(s). In this section we propose two different ways of constructing the configuration file.

8.2.2.1 Using keywords order

Parser will rely on the order of the PATH and ATTRIBUTE keywords to figure out which attribute(s) belongs to which objects as illustrated in the example below. Each attribute will be attached to the objects appeared on the lines containing the PATH keyword that immediately precede the line with the ATTRIBUTE keyword.

For example, in the example below the attribute "Heights" will be attached to the objects /m1 and /m2, while attributes the "Weights" and "Speed" will be attached to /m3.

```
% cat my_attr_conf
      PATH "/m1"
      PATH "/m2"
      ATTRIBUTE "Heights" {
          DATATYPE  H5T_STD_I64BE
          DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
          DATA {
          (0): 10, 20, 30
          }
      }


      PATH "/m3"
      ATTRIBUTE "Weights" {
          DATATYPE  H5T_STD_I64BE
          DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
          DATA {
          (0): 10, 20, 30
```

```
        }
    }
    ATTRIBUTE "Speed" {
        DATATYPE  H5T_STD_I32BE
        DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
        DATA {
        (0): 1.5, 3.6
        }
    }
```

The grammar for the configuration file can be summarized as following: Every line with an ATTRIBUTE keyword has to be preceded by a line or several lines with a PATH keyword. Line with a PATH keyword cannot be the last in the file.

8.2.2.2 Using special keyword

One can use a special keyword to indicate the beginning of the section in the configuration file that will contain paths to all objects and attributes definitions for those objects. The order in which the PATH and ATTRIBUTE keywords will appear doesn't matter.

The example below illustrates the concept. The "BEGIN" keyword is chosen as a section separator.

```
% cat  my_attr_conf
    BEGIN
    ATTRIBUTE "Heights" {
        DATATYPE  H5T_STD_I64BE
        DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
        DATA {
        (0): 10, 20, 30
        }
    }
    PATH /m1
    PATH /m2


    BEGIN
    ATTRIBUTE "Speed" {
        DATATYPE  H5T_STD_I32BE
        DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
        DATA {
        (0): 1.5, 3.6
        }
    }

    PATH "/m3"
    ATTRIBUTE "Weights" {
        DATATYPE  H5T_STD_I64BE
        DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
```

```
        DATA {
        (0): 10, 20, 30
        }
    }
```

8.2.3 Handling configuration files of different formats

One idea is to code h5attribute to accept multiple formats for a configuration file such that users may use the h5import style format for simple attributes but use the DDL format for more complex types. The h5attribute tool needs to distinguish which format is used.

One way to address the issue is to add a FORMAT specification at the beginning of the configuration file. Here is an example.

```
% cat  my_attr_conf

    [ FORMAT=DDL ]

    PATH "/m2"
    ATTRIBUTE "Heights" {
        DATATYPE  H5T_STD_I64BE
        DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
        DATA {
        (0): 10, 20, 30
        }
    }


% cat  my_attr_conf
    [ FORMAT=TXT1 ]

    PATH "/m1"
    ATTRIBUTE-NAME "Speed Vector"
    INPUT-CLASS TEXTIN
    INPUT-SIZE 64
    OUTPUT-BYTE-ORDER BE
    OUTPUT-ARCHITECTURE STD
    RANK 1
    DIMENSION-SIZES 3
    ATTRIBUTE-DATA
    "1,2,3"
```

Another way to handle different file formats is to specify it with an argument on the command line.

For example, use `--control-file <OPT>=my_attr_conf,` where `<OPT>` indicates the format.

- `h5attribute –add –control-file DDL=my_attr_conf file.h5`

Revision History

May 27, 2010:	Version 1 draft for initial review to send to NPOESS developers
July 22, 2010:	Version 2 sent to Jonathan and Albert
July 27, 2010:	Version 3 atomicity section added. Sent to HDF5 developers
August 25, 2010:	Version 4 revision to use DDL syntax in both command line and command files. Added incremental atomcity option. Sent to HDF5 developers