RFC: H5Ocompare Function

Peter Cao

Neil Fortner

Quincey Koziol

This RFC describes the H5Ocompare function that compares two HDF5 objects. A set of rules for comparing two HDF5 files or two HDF5 objects have been specified in the “HDF5 File and Object Comparison Specification[[1]](#footnote-1)”. The purpose of this document is to provide details of how two objects should be compared and guidelines of how H5Ocompare should be implemented.

# Introduction

An HDF5 file appears to the user as a directed graph with three higher-level objects that are exposed by the HDF5 APIs: groups, datasets, and committed datatypes. The simplicity of the HDF5 model provides great flexibility with regard to what can be put in a file. At the same time, it creates challenges in determining how to compare two files or two objects.

A command line tool, h5diff, was developed to compare two HDF5 files or objects and report the differences. The tool is one of the most used tools and extensively tested over the years. However, three major issues cannot be easily (or impossible) resolved from the current implementation of h5diff:

* The tool has poor performance in some cases. Although the performance of h5diff has been improved a lot recently, some issues cannot be resolved from the current implementation. For example, uncompressing the data is a major bottleneck for compressed datasets; it takes over 80% of the total time. There is no better solution other than performing the comparison within the library, on the compressed data.
* h5diff was implemented with insufficient design and definition. When two objects are compared, h5diff does not provide sufficient details of what should be examined. For example, when two datasets are compared, creation properties such as storage layout are not compared. There is no clear definition of what should be compared and what should be not. One of the main goals of this document is to provide a clear and complete set of definitions for the H5Ocompare function on what to compare.
* The code is very complex. Features and options have been added to h5diff in an *ad hoc* way, which makes the code very complex and prone to errors. Over the years, a great deal of effort has been spent fixing problems and errors.

The purpose of H5Ocompare is to address the problems above. Other advantages of having H5Ocompare function include the following:

* Any tool built on H5Ocompare should have less code to maintain since the main work will be done by the function. Library functions are designed to be simple and specific and any tool and application built on the function should be simple and specific.
* The function can be used in other applications such as HDFView.
* The function can be used in other high level languages such as Python. Users can program their own applications to accomplish their own goals.

H5Ocompare is not targeted to replace the current h5diff; it is intended for building new tools (or used in applications) that will address the problems of the current h5diff. H5Ocompare does not try to mimic the output format of the current h5diff; however, some useful elements of h5diff output format will be adopted by H5Ocompare. Not all options of H5Ocompare discussed later in this RFC will be implemented at the first stage but the design of the function should allow options to be added incrementally.

# New API function

**Name:**

*H5Ocompare*

**Signature:**

*herr\_t H5Ocompare(hid\_t loc1\_id, const char \*name1, hid\_t loc2\_id, const char \*name2, hid\_t ocmppl\_id, H5O\_cmp\_cb\_t \*cb\_info)*

**Purpose:**

Compare two objects in the same or different files.

**Description:**

H5Ocompare compares the object specified by name1 from the file or group specified by loc1\_id to the object specified by name2 from the file or group specified by loc2\_id .

Several properties are available to govern the behavior of H5Ocompare. These properties are set in the access property lists, ocmppl\_id with H5Pset\_compare\_object.

The differences of the two objects are reported to a compare callback struct that is passed in as a parameter. The callback struct contains a list callback functions and a pointer of user’s data.

Each callback function is intended for reporting results for a specific object or data such as:

* one for file metadata,
* one for object metadata,
* one for group differences (like links that are different, etc),
* one for raw data differences in datasets,
* one for raw data differences of attributes, and etc.

The structure of the callback functions will be added to this RFC as more details come out from our further discussions.

**Parameters:**

|  |  |
| --- | --- |
| *hid\_t* loc1\_id | IN: Location identifier of the first object to be compared |
| *const char \**name1 | IN: Name of the first object to be compared |
| *hid\_t* loc2\_id | IN: Location identifier of the second object to be compared |
| *const char \**name2 | IN: Name of the second object to be compared |
| *hid\_t* ocmppl\_id | IN: Object compare property list identifier |
| *H5O\_cmp\_cb\_t \*cb\_info* | IN/OUT: A callback struct that contains a list of callback functions and a ponter of user’s data for reporting the results of comparsion. |

**Returns:**

Returns a non-negative value if successful; otherwise returns a negative value.

# Comparing objects

This section describes how two HDF5 files or objects will be compared by H5Ocompare. Options, such as excluding certain metadata, can be applied to the comparison.

## Files

A valid HDF5 file contains a root group and file metadata such as file creation properties. Comparing two files means comparing the root groups and the metadata of the two files. The appropriate way to call the function is to use the “/” as the object name, e.g.

*herr\_t retval = H5Ocompare(file1\_id, “/”, file2\_id, “/”, H5P\_DEFAULT, cb\_info);*

### Things to be compared

When two files are compared, things to be compared include:

* file creation properties such as version information and
* the root groups of the two files.

The result of the comparison is determined by the file metadata and the contents of the root groups of the two files. Group comparison is discussed in the next section.

### Options

The following options can be applied when two files are compared:

* Creation properties are excluded. By default, creation properties will be checked.
* Options of comparing groups discussed in the next section will be applied to file comparison.

### Special cases

H5Ocompare should handle the following special cases of file comparison:

1. A file is compared to itself. There should be no difference if a file is compared to itself. H5Ocompare should quickly confirm that it is the same file -- without going through all the objects in the file.
2. Two identical files are compared. There should be no difference if two identical files are compared. This case is the same as (A) except that the two files are two separate physical files with exactly the same contents.
3. Two empty files are compared. A file is empty if it contains only an empty root group – a group does not have any member or link. If two empty files are compared, the result varies according to the comparison options. If file creation properties are ignored in the comparison, there should be no difference between two empty files. Otherwise, the result is determined by the file creation properties of the two files.
4. An empty file is compared with a non-empty file. The result varies according the comparison options. By default, an empty file and a non-empty file should be different. If unique objects file properties are excluded, there will be no difference.

## Groups

When two groups are compared, the groups’ creation properties and attributes, as well as the links under the two groups and objects contained in the groups and having the same link names will be compared.

### Things to be compared

Things to be compared for groups include:

* group creation properties (creation order, group layout, etc),
* attributes attached to the group,
* links under the group, and
* objects having the same link names.

### Options

Options of comparing groups are given below:

* Creation properties are excluded. By default, creation properties will be checked.
* Attributes are excluded. By default, attributes will be compared (see Attribute section below).
* Unique links, links with different names, are excluded. The comparison results will be determined by the links with common names and objects reached by the links. By default, unique links will be reported as different.
* Comparing only immediate links and objects reached by the links (shallow hierarchy). By default, recursively comparing all objects below the groups.

## Datasets

An HDF5 dataset is an object that contains raw data and metadata that describes the data elements. For details, see the *HDF5 User’s Guide*.[[2]](#footnote-2)

### Things to be compared

Comparing datasets means comparing both the data values and the datasets’ metadata, unless an option excluding a specific comparison is given. Things to be compared include:

* data values,
* dataset creation properties (storage layout, chunking, compression, fill value, etc.),
* attributes attached to the dataset,
* datatype, and
* dataspace

### Options

Options of comparing datasets are listed below:

* Creation properties are excluded. By default, creation properties will be checked.
* Attributes are excluded. By default, attributes will be compared (see Attribute section below).
* Datatypes are excluded. By default, datatype will be compared. Two datasets with different datatypes will be considered as different and their data values will be not compared. If datatypes are ignored and the data values can be converted from one datatype to another, the data values will be compared.
* Dataspaces are excluded –By default, dataspace will be compared. Two datasets with different dataspaces will be reported as different. If dataspace is ignored, data values will be compared up to the smaller datasets. The rest of the larger dataset will be reported as different.
* EPSILON is needed for floating-point values. To determine whether two floating-point values, float1 and float2, are different, one cannot use the simple comparison of (float1 == float2). Two floating-point values can be the same while (float1 == float2) may appear to differ because of floating-point precision. In HDF5, a precision limit can be set when comparing floating point values; for details, see the “Default EPSILON values for comparing floating point data[[3]](#footnote-3)” RFC. An additional API routine for the compare property list will be needed to set the EPSILON value.
* An option is needed to skip checking Not-a-Number (NaN). By definition, two NaNs are always equal. A NaN and a regular number are always different. Checking for NaN is very expensive. By default, H5Ocompare will check NaNs. If we know there is no NaN in the datasets, we skip checking NaN for better performance.
* Datasets of special datatypes must be handled on a case-by-case. For example, we can compare the raw byte for opaque data values.

Dataset data is only compared by raw bytes on a conditional basis. If they are different, then it will compare the data logically, following the rules for special values, epsilon, etc.

## Links

A link is owned by a group and points to an existing object or a non-existing object (symbolic links only). Each link has a name, type, and value.

### Things to be compared

Link characteristics to be compared include:

* creation properties (e.g. link order),
* link name,
* link type (hard, soft, external, or user-defined), and
* link value.

### Options

Options of comparing links include the following:

* Creation properties are excluded. By default, creation properties will be checked. Currently the only link "creation property" is character set encoding.
* Link type is excluded. By default, link types will be compared. if links of different types different, e.g. one is external link and the other is soft link, will be reported as different.
* Objects pointed symbolic links (soft links or external links) are excluded. By default, objects pointed by symbolic links will be compared. If the option is given, objects pointed by the links will not be compared.

## Attributes

An attribute is a small dataset; it has a name, a datatype, a dataspace, and raw data like a dataset. Attributes are compared by name and comparing two attributes is the same as comparing two datasets except that some data properties such as storage layout are not applied to attributes.

## Datatypes

A datatype can describe an atomic type like a fixed- or floating-point type or more complex types like a C struct (compound datatype), array (array datatype) or C++ vector (variable-length datatype). A datatype is defined by its class and class-specific properties.

### Things to be compared

Link characteristics to be compared include:

* datatype class (e.g. Integer, Float, String, etc.),
* class-specific properties (e.g. size, signed or unsigned, byte order, etc.),
* names (for committed datatypes), and
* attributes (for committed datatypes).

### Options

Options of comparing committed datatypes are listed below:

* Attributes are excluded. By default, attributes will be compared (see Attribute section below).

## Dataspace

A dataspace describes the rank, that is, the number of dimensions, and the size of each dimension in the data object array.

### Things to be compared

Dataspace characteristics to be compared include:

* rank – number of dimensions,
* current dimension sizes, and
* max dimension sizes.

### Options

Options of comparing links include:

* excluding max dim size – max dim sizes will not be compared. By default, max dim sizes will be compared.

# Revision History

|  |  |
| --- | --- |
| *January 12, 2011:* | Version 1 circulated for comment within The HDF Group. |
| *January 20, 2011:* | Version 2 revised with Quincey’s and Neil’s feedback. |
|  |  |

1. <https://www.hdfgroup.uiuc.edu/RFC/HDF5/tools/h5diff/h5diff_spec.pdf> [↑](#footnote-ref-1)
2. [http://www.hdfgroup.org/HDF5/doc/UG/](http://www.hdfgroup.org/HDF5/doc/UG/UG_frame10Datasets.html) [↑](#footnote-ref-2)
3. https://www.hdfgroup.uiuc.edu/RFC/HDF5/tools/h5diff/RFC\_h5diff\_default\_epsilon.pdf [↑](#footnote-ref-3)