HDF5 File and Object Comparison Specification

Dana Robinson
Peter Cao
Frank Baker

This specification describes a set of rules for comparing two HDF5 files or two HDF5 objects. The intended purpose of the specification is to provide a definitive guide to HDF5 comparisons.

1 Introduction

To compare two objects, one must know the expected equivalence relations and the differences to be searched for. These differences and equivalence relations have not previously been clearly defined for HDF5 files or objects, causing confusion in the implementation and interpretation of applications such as h5diff.

An HDF5 file appears to the user as a directed graph with four higher-level objects that are exposed by the HDF5 APIs: groups, datasets, attributes 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. This document provides a clear definition of how two HDF5 files or objects should be compared by explicitly defining the equivalence relation and the differences to be examined.

One appropriate use of this specification would be in the development of an H5Ocompare API and/or a future version of h5diff (or a replacement tool).

2 Descriptions of HDF5 files and objects

This section briefly describes high level HDF5 objects and their metadata. For details, see the *HDF5 File Format Specification*[1] and related documents.[2]

2.1 Primary objects

HDF5 files are organized in a hierarchical structure, with four primary structures: groups, datasets, attributes and committed datatypes.

[1] http://www.hdfgroup.org/HDF5/doc/H5.format.html
[2] http://www.hdfgroup.org/HDF5/doc/index.html

- ***HDF5 group***: a grouping structure containing zero or more links to groups or datasets and supporting metadata
- ***HDF5 dataset***: a multidimensional array of data elements and supporting metadata
- ***HDF5 attribute***: a small dataset which typically contains auxiliary or descriptive data and can be linked to either a group or dataset.
- ***HDF5 committed datatype***: a committed datatype and supporting metadata

An object in HDF5 is identified either by link name within the group containing it or by its absolute path name (the path from the root group to the group containing the object plus the object's link name in that group).

2.2 Metadata

Metadata is generated by the HDF5 Library to describe the structure of the file and structure and contents of objects in the file. For example, library metadata includes information such as:

- A header block (superblock) that sets up the file, sets up the initial structures, and identifies the file as a valid HDF5 file
- B-trees that describe the location of and provide access to groups and members of groups
- Datatype, current and maximum array dimensions, and other features of a dataset
- Dataset properties such as storage layout, fill value, allocation time, or the use of filters

HDF5 natively interprets and understands library metadata. Library metadata is always present; even an otherwise-empty file must contain certain metadata to be a valid HDF5 file.

3 Definition of equivalence

Equivalence is defined for the purpose of this RFC as "identical in certain characteristics". The default is "all characteristics" though users will be able to specify a looser set of characteristics for comparison, if desired (see below).

4 Comparing HDF5 objects

This section describes how two HDF5 files or objects should be compared.

Note that, when files or objects are compared, only user-facing metadata is considered. For example, the chunking scheme can be specified by the user so that would be evalutated when two datasets are compared. However, internal data structures such as the B-Trees, which cannot normally be inspected by the user, would not be compared.

4.1 Files

4.1.1 Default comparison

Two files are considered equivalent when the following characteristics are identical:

- Creation properties such as version information.

- The user block of the file.
- All contained data objects and containers, evaluated recursively from the root group.

4.1.2 User-modified comparison

The following HDF5 file characteristics can be ignored for a less strict comparison:

- All metadata can be ignored at the file level. More granular control over metadata comparison is also available (see the data object sections below).
- File creation properties can be ignored.
- Particular data objects (groups, datasets, etc.) can be ignored. Data objects will be identified by name or path. All objects linked to an ignored data object (*e.g.* datasets contained in an ignored group) will also be ignored.
- The user can elect to only compare the common data objects between two files.
- The user block can be ignored.

4.2 Groups

4.2.1 Default comparison

Two groups are considered equivalent when all of the following are identical:

- Creation properties, such as creation order, group layout, etc.
- Attached attributes
- References to contained groups, datasets, links, etc.

Group comparison can be recursive, examining all contained data objects, or not. Both will be supported at both the API and tool levels.

4.2.2 User-modified comparison

The following HDF5 group characteristics can be ignored for a less strict comparison:

- Creation properties can be ignored.
- Attached attributes can be ignored.
- Particular data objects (groups, datasets, etc.) can be ignored. Data objects will be identified by name or path. All objects linked to an ignored data object (*e.g.* datasets contained in an ignored group) will also be ignored.
- The user can elect to ignore references and attributes which are not common between the two groups.
- Following symbolic links in the recursive case.

4.3 Datatypes

There are two fundamental classes of datatypes: simple datatypes such as integer and floating point types and complex datatypes, which are built up from the simple types.

4.3.1 Simple datatypes: Default comparison

- Two simple datatypes are equivalent when they are exactly identical.

4.3.2 Simple datatypes: User-modified comparison

The following simple datatype characteristics can be ignored for a less strict comparison:

- Big- and little-endian differences can be ignored.
- Floating point and integer data widths can be ignored (*e.g.* 32- vs. 64-bit integers).
- Floating point encoding schemes can be ignored (*e.g.* IEEE vs. Cray).
- Signed/unsigned differences can be ignored.

4.3.3 Complex datatypes: Default comparison

Two complex datatypes are equivalent when the following characteristics are identical:

- Type class (array, compound, variable-length, enumeration)
- Underlying simple type (array, variable length, enumeration)
- Length (array)
- Data members, recursive (compound)
- Data member order (compound)
- Enumeration names and values (enumeration)

4.3.4 Complex datatypes: User-modified comparison

The following complex datatype characteristics can be ignored for a less strict comparison:

- The same ignorable differences listed for the simple datatypes.
- Array length
- Compound types with different member order
- Enumerations where one is a proper subset of the other.
- Enumerations where the member names are identical but their values are not.

4.4 Datasets and Attributes

4.4.1 Default comparison

Two datasets or attributes are considered equivalent when the following characteristics are identical (Some of these characteristics only apply to datasets):

- Creation properties, such as storage layout, chunking, compression, fill value, etc.
- Attached attributes
- Datatype
- Rank

- Current and maximum dimension sizes
- Raw data values

Unlike the current implementation of h5diff, empty datasets will be compared normally.

4.4.2 User-modified comparison

The following dataset and/or attribute characteristics can be ignored for a less strict comparison:

- Creation properties
- Attached attributes
- Datatype, as described in the datatype section
- Current and maximum dimension sizes
- Raw data values
- The user can elect to ignore attributes that are not found in both datasets.

4.5 Links

4.5.1 Default comparison

Two links are considered equivalent when the following characteristics are identical:

- Name
- Class (hard link, soft link, or external link).
- Value. For soft links, this is the path to which the link points; for external and user-defined links, it is the link buffer.
- Creation properties, such as link order.

Note that dangling symbolic links are handled like any other link. Checking for equivalency does not imply checking for consistency, though the h5compare tool may optionally flag such links as errors.

4.5.2 User-modified comparison

The following link characteristics can be ignored for a less strict comparison:

- Creation properties
- Class
- Value

4.6 Raw data values

When two datasets or attributes are compared, their values will be examined and compared. Comparing data values can be straightforward or complicated depending on their datatype. Special values and data structures must be handled case by case.

4.6.1 Default comparison

Two data values are considered equivalent when they are identical. However, keep in mind the following special cases:

- 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. Clearly, these precision limits will not apply to non-numeric values such as NaN or infinity. For details, see the "Default EPSILON values for comparing floating point data" RFC.[3]
- Not-a-Number (NaN): NaN is treated as a regular number and different NaN values are considered different. Identical NaN values are considered identical. IEEE 754-2008, for example, allows for signaling and quiet values of NaN, which are considered different by default.
- Infinity: Infinity is treated as a regular number and different infinity values are considered different (*e.g.* positive and negative infinity). Identical values of infinity are considered identical.
- Special datatypes: The comparison values of special datatypes, such as opaque and variable-length datatypes, can be complicated. Such comparisons must be handled on a case-by-case basis.

4.6.2 User-modified comparison

The following data value characteristics can be ignored for a less strict comparison:

- The user can specify a delta for any numerical data type. This delta can be absolute or relative. As noted above, these deltas are not applicable when non-numeric values like NaN or infinity are considered – those will always be compared using simple equality.
- Differences between different values of NaN can be ignored. In this scheme, two NaNs are always equal. A NaN and a regular number are always different.
- Enumeration data can be compared by name only.
- Enumeration data can be compared by value only.
- The presence or absence of ASCII NUL (\0) character(s) at the end of a string can be ignored.

5 Justification

5.1 Inclusion of the user block

Earlier versions of this document did not include the user block of an HDF5 file as a comparable object as it is, technically, outside of the HDF5 data model. The user block *should* be included in the specification, however, as h5compare will have to consider it in order to not exhibit surprising

[3] https://www.hdfgroup.uiuc.edu/RFC/HDF5/tools/h5diff/RFC_h5diff_default_epsilon.pdf

behavior. If h5compare were blind to the user block, two otherwise identical HDF5 files, one with a user block and one without, would be evaluated as identical by h5compare, which would be probably be considered surprising behavior by most users. Although the user block is not a part of the data model, *per se*, this subtlety would likely be lost on most users who will see the user block as a valid part of an HDF5 file (it's in the file format specification, after all!).

5.2 Ignoring non-common data objects, especially attributes

In some cases, users will add their own data objects to a canonical data file. Allowing users to ignore the added data objects makes it easier to compare their annotated file with the original data file.

5.3 Not comparing data

This allows users to compare the overall data storage schema of two files, without regard for the underlying data stored in the file. This can be useful for ensuring that an HDF5 file complies with a user-defined schema, such as HDF-EOS.

5.4 NaN and infinity

By default, identical bit patterns in a numerical type will evaluate as identical. Not only is this the most straightforward method of comparison, but it allows for very fast bitwise comparisons between numerical data. Furthermore, it is not up to the HDF5 library to decide which particular numerical values are "unimportant". The API and tools will have options that will allow users to modify this behavior.

Revision History

October 8, 2010:	Version 1 circulated for comment within The HDF Group.
November 4, 2010	Version 2 revised after h5diff specification meeting with HDF Group staff (participants: Peter Cao, Quincey Koziol, Elena Pourmal, Jonathan Kim, and Neil Fortner).
August 1, 2011	Version 3 revised by Dana Robinson.

Appendix: Background Material

6 Major issues of h5diff

h5diff is a command line tool that compares two HDF5 files or objects and reports the differences. The tool provides options on what to compare and how to compare. For details, see the online h5diff description.[4]

This section briefly describes some major issues regarding h5diff. It is not meant to be comprehensive.

6.1 Performance

A performance problem has been a known issue for h5diff and is under investigation. One performance issue is in the handling of NaN datatypes. The current version of h5diff checks NaN values by default and the operation of checking and comparing special values is very time consuming. For datasets without special values, the time was totally wasted. A "-N" option is provided so that users can skip checking for NaN values.

Another performance issue in h5diff arises in retrieving and checking datatype information for each data point. This can be a major problem for datasets with large number of data points since checking datatype information, such as H5Tequal() and H5Tget_member_type(), is very expensive.

6.2 File structure

The current version of h5diff does not provide options for strictly equivalent and loosely equivalent. h5diff compares objects ONLY if it finds common objects in both files. This h5diff behavior will also hide potential errors in when testing the h5copy or h5repack tools; for example, if h5copy generates an empty HDF5 file by mistake, h5diff, as is, will report that the erroroneous empty file is not different from the original file. Therefore, h5diff should report that an empty HDF5 file is different from a non-empty HDF5 file.

A proposal has been made to add a new option "-c" as "Contents mode. Objects in both files must match." In view of the common practice of other comparison tools (e.g., the cmp and diff tools on Unix systems), it is better to fix h5diff as described in the above paragraph rather than introduce a new flag.

6.3 Non-Comparable datasets

For a variety of reasons, h5diff does not compare some dataset objects. When this happens, h5diff prints "Some objects are not comparable" at the end of the program execution. In verbose mode, h5diff also prints the reason(s) why it did not perform the comparison. The following are examples of non-comparable datasets listed in the RFC on non-comparable objects:

- Empty datasets

[4] http://www.hdfgroup.org/HDF5/doc/RM/Tools.html#Tools-Diff

- Different datatype classes or H5T_TIME class or H5T_COMPOUND class
- Different dataspace ranks
- Different dataspace dimensions
- Different order properties
- Different sign properties
- Invalid numeric operation in relative error calculation

For details, see the "Reporting of Non-Comparable Datasets by h5diff" RFC.[5]

[5] https://www.hdfgroup.uiuc.edu/RFC/HDF5/tools/h5diff/RFC_h5diff_NonComparable.pdf