HDF5 File and Object Differences Specification

Peter Cao

Frank Baker

This document (or specification) describes the differences or equivalence relation of two HDF5 files or two HDF5 objects. The specification only handles HDF5 files and HDF5 objects. It does not apply to non-HDF5 files and non-HDF5 objects, which include a user block in an HDF5 file. The intended purpose of the document is to provide guidelines or information for developing tools or APIs for comparing HDF5 files or objects.

# Introduction

The differences or equivalence relation of two HDF5 files or objects have not been defined. Lack of such a specification caused confusion in applications such as h5diff. The purpose of this document is to provide a clear definition on how two HDF5 files or objects should be compared.

An HDF5 file appears to the user as a directed graph. There are three higher-level objects that are exposed by the HDF5 APIs: groups, datasets, and named datatypes. The simplicity of the HDF5 model provides a great flexibility on what things can be put in a file. In the mean time, it also creates challenges on how to compare two files or two objects. The intention of this specification is to clarify the confusion and provide information for developing tools or APIs for comparing HDF5 files or objects.

The specification described in this document is intended for HDF5 files or objects only. It does not apply to non-HDF5 files and objects, which include a user block in an HDF5 file.

# Descriptions of HDF5 files and objects

This section briefly describes high level HDF5 objects and their metadata. For details, please read the HDF5 File Format Specification at <http://www.hdfgroup.org/HDF5/doc/H5.format.html> and other related documents at <http://www.hdfgroup.org/HDF5/doc/index.html>.

## Primary objects

HDF5 files are organized in a hierarchical structure, with two primary structures: groups and datasets.

* HDF5 group: a grouping structure containing zero or more links to groups or datasets, together with supporting metadata.
* HDF5 dataset: a multidimensional array of data elements, together with supporting metadata.

An object in HDF5 is identified by its link name (path + name) when two objects are compared. Although each object has a unique identification number, which is used to identify objects under the lower level, the object is recognized by its link name not the ID as to users.

## Metadata

HDF5 files generally contain two types of metadata: structural metadata and application metadata.

*Structural 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, structural 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 structural metadata.  Structural metadata is always present; even an otherwise-empty file must contain certain metadata to be a valid HDF5 file.

*Application metadata* is defined and provided by a user application is often stored in an HDF5 attribute, and may describe virtually anything.  For example:

* Minimum and maximum valid values in a dataset
* Conditions under which data was collected
* Data history and/or provenance
* Relationships among datasets
* Scales or other interpretive information

HDF5 does not natively understand application metadata; it must be understood and interpreted by the application.  Application metadata is technically optional but commonly used.

In both cases, these are just a few examples illustrating the range of possibilities.

# Definition of equivalence

In this section, we will start a general definition of equivalence relation and its properties. Later, HDF5 equivalence relation will be introduced. Two files or objects are different if they are not equivalent.

## Mathematic equivalence

“A given binary relation “~” on a set A is said to be an equivalence relation if and only if it is reflexive, symmetric and transitive.” based on the definition of mathematical equivalence relation from Wikipedia. Equivalence relation has three basic properties:

* Reflexivity: “a” ~ “a”
* Symmetry: if “a” ~ “b” then “b” ~ “a”
* Transitivity: if “a” ~ “b” and “b” ~ “c” then “a” ~ “c”

## HDF5 equivalence

A single definition of equivalence or differences of two HDF5 files or two HDF5 objects is not adequate for all situations. Two definitions of equivalence are proposed here: strictly equivalent and loosely equivalent.

### Strictly equivalent

Two objects or files are strictly equivalent if their content (all values or members) and metadata are the same. The three properties of the equivalence relation should be applied under strictly equivalent.

Under strict equivalence, two files are equal if their file structures, objects (all groups and dataset), and metadata are the same. Two groups are equal if structures, objects (all groups and dataset), and metadata are the same. Two datasets are equal if their data values and metadata are the same.

### Loosely equivalent

Two objects or files are loosely equivalent if they are equivalent under certain given conditions or options. The three properties of the equivalence relation may not be applied to loosely equivalent.

Below are some examples of comparison options for loosely equivalent. Other comparison options can be specified by applications.

1. Excluding certain objects when comparing two files or two groups
2. Comparing only common objects
3. Ignoring attributes or other metadata when comparing two objects
4. Comparing only attributes or other metadata

# Comparing HDF5 objects

This section describes how two HDF5 files or objects should be compared. Objects are identified by their link names when they are compared. Options, such as ignoring certain metadata, can be applied to comparisons (loosely equivalent).

## Files

A valid HDF5 file contains a root group and metadata such as file creation properties. Comparing two files means to compare the root groups and the metadata of the two files. The group comparison will be discussed next. File metadata to be compared includes:

* File creation properties such as version information.

Special cases of file comparison include:

1. Itself: there should be no difference if a file is compared to itself.
2. Identical files: there should be no difference if two identical files are compared. This case is the same as case A) except that the two file are two separate physical files in the system.
3. Empty files: a file is empty if it contains only a root group.
   * If two empty files are compared, the result varies according the comparison options. If metadata is ignored in the comparison, there should be no difference between two empty files. Otherwise, the result is determined by the metadata of the two files.
   * If an empty file is compared with a non-empty file, the result varies according the comparison options. Under strictly equivalent, an empty file and a non-empty file should be different. Under certain loosely equivalent, such as comparing only common objects, there will be no difference.

## Groups

When two groups are compared, the links under the two groups will be compared. Objects with the same link names will be followed by the comparison. Things to be compared between two groups include:

* Group creation properties, such as creation order, group layout, etc
* Attributes attached to the Group
* Links

## Datasets

An HDF5 dataset is an object that contains raw data and metadata that describes the data elements, data layout, and all other information necessary to write and read the stored data. For details, read the HDF5 user’s guide at [http://www.hdfgroup.org/HDF5/doc/UG/](http://www.hdfgroup.org/HDF5/doc/UG/UG_frame10Datasets.html).

Comparing datasets means comparing both the data values and metadata unless a certain option is given. Things to be compared in datasets include:

* Data creation properties, such as storage layout, chunking, compression, fill value, etc
* Attributes attached to the dataset
* Datatype
* Dataspace
* Data values

## Links

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

* Name: a link name may be any string of ASCII characters not containing a slash or a dot. The link name must be unique within each group
* Type: a type specifies the link class. Valid types are hard link, soft link, and 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

## Attributes

Attribute are "metadata" about the current object. An attribute is a small dataset; it has a name, a datatype, a dataspace, and raw data. Comparing two attributes is similar to comparing two datasets. Attributes are compared by names.

* Datatype
* Dataspace
* Value(s)

## Datatypes

Datatype defines the datatype for each element of a dataset or a named datatype for sharing between multiple datasets. 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.

* Dataype class, e.g. Integer, Float, String, etc.
* Class-specific properties, e.g. size, signed or unsigned, byte order, etc.

## Dataspace

Dataspace describes the number of dimensions (i.e. "rank") and size of each dimension that the data object has.

* Rank
* Current dimension sizes
* Max dimension sizes

## Data values (or raw data)

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. Below are some examples:

* Floating points: to determine if 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 show different because of the precision of floating points. Limit of precision needs to be set when comparing floating point values. For details, see RFC at <https://www.hdfgroup.uiuc.edu/RFC/HDF5/tools/h5diff/RFC_h5diff_default_epsilon.pdf>.
* Not-a-Number (NaN): two NaNs are equal. A NaN and a regular number are different.
* Infinity: infinity should be treated as a regular number. Two infinity numbers with the same sign (+/-) are equal. Two infinity numbers with different signs (+/-) are different. An infinity number and a regular number are different.
* Special datatypes: comparing values of special datatypes such as opaque and variable length can be complicated and need to be handled case by case.

# Revision History

|  |  |
| --- | --- |
| *October 8, 2010:* | Version 1 circulated for comment within The HDF Group. |
| *November 4, 2010* | Version 2 revised after h5diff spec meeting with Peter, Quincey, Elena, Jonathan, and Neil. |

# Appendix: Background Material

# Major issues of h5diff

h5diff is a command line tool that compare two HDF5 files or objects and report the differences. The tool provides options on what to compare and how to compare. For details, see the online guide at <http://www.hdfgroup.org/HDF5/doc/RM/Tools.html#Tools-Diff>.

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

## Performance

Performance problem has been a known issue for h5diff. It is under investigation. One of the causes is from handling NaN. Current h5diff checks NaN values by default. The operation of checking and comparing special values is very time consuming. For datasets without special values, the time was totally wasted. “-N” option is provided so that users can skip checking NaN.

Another performance issues in h5diff is retrieving and checking information of datatypes for each data point. This can be a major problem for datasets with large number of data points since checking datatypes information, such as H5Tequal() and H5Tget\_member\_type() are very expensive.

## File structure

Current h5diff does not provide options for strictly equivalent and loosely equivalent. h5diff compares objects ONLY if it finds common objects in both files. If one of the files is an empty hdf5 file, h5diff concludes there is nothing to compare and since no difference has been found, it exits with 0. This is against common practice. E.g., the Unix tool, diff, considers the two files are different if ONLY one of them is an empty file. The h5diff behavior will also hide potential error in the testing of the h5copy or h5repack tools since if h5copy generates an empty hdf5 file by mistake, h5diff, as is, will report the error empty file no different from the original file. Therefore, h5diff should report an empty hdf5 file is different from a non-empty hdf5 file.

A proposal was 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 in Unix systems), it is better to fix h5diff as described in the above paragraph than introducing a new flag.

## Non-Comparable datasets

h5diff does not compare some dataset objects, for a variety of reasons. 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. For details, see the RFC at <https://www.hdfgroup.uiuc.edu/RFC/HDF5/tools/h5diff/RFC_h5diff_NonComparable.pdf>.

* Empty datasets
* 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