RFC: New public functions to handle comparison

Peter Cao

Neil Fortner

Quincey Koziol

Vailin Choi

This RFC describes a new public function, *H5Ocompare* that compares two HDF5 objects. The comparison is performed according to the set of rules for comparing two HDF5 files or objects specified in the “HDF5 File and Object Comparison Specification”[1], which provides details and guidelines of how two objects and files should be compared.

This RFC also describes seven new public functions: *H5Fcompare\_md*, which compares two files’ file metadata, *H5Pget/set\_compare*, which manipulate properties for the comparison, *H5Pget/set\_compare\_value\_ndiffs*, which control the maximum number of differences to report when comparing values of datasets or attributes, and *H5Pset/get\_compare\_fp\_tolerance*, which sets/gets the tolerance when comparing floating-point values.

# Introduction

An HDF5 file appears to the user as a directed (multi-)graph with three higher-level objects that are exposed by the HDF5 APIs: groups, datasets, and committed datatypes. The intricate structure of an HDF5 file creates challenges in determining how to compare the content of two HDF5 files. Since the content of an HDF5 file largely consists of HDF5 objects, we tackle object-level comparison first with the proposed public function, *H5Ocompare*. The design of *H5Ocompare* incorporates lessons learned in developing and maintaining the *h5diff* tool.

# Motivation

One of the most frequently used tools, *h5diff*, compares two HDF5 files or objects and reports the differences. However, *h5diff* has major issues that cannot be easily resolved with its current implementation:

* *Maintenance*: The limited scope of *h5diff*’s original design has prevented addressing the evolving requirements of the tool.
* *Reusability*: Having the comparison operations done within the tool itself makes it difficult for other application users to use the comparison functionality.
* *Performance*: *h5diff* does not perform well especially when comparing large compressed datasets.

# Approach

With the new public function, *H5Ocompare*, we intend to address the above issues. The design is characterized by the following:

* *Completeness*: In this RFC, we provide clear and complete definitions of object characteristics to compare.
* *Reusability*: The implementation of *H5Ocompare* within the library lets everyone use the comparison functionality.
* *Maintenance*: Tools and applications built on *H5Ocompare* should be simple, specific, and have less code to maintain since this function does the main work.
* *Flexibility*: *H5Ocompare* provides callback functions, thus providing application users the choice to react to the differences found.
* *Performance*: The implementation of *H5Ocompare* within the library allows the direct comparison of compressed data. This will enhance performance when comparing large compressed dataset values having the same filters.

In this RFC, we also propose seven new auxiliary public functions as follows:

* *H5Fcompare\_md*: This function compares file-level metadata. Separating the comparison of file metadata from the object comparison done by *H5Ocompare* provides a more coherent API to developers. This allows the root group of each file to be treated in the same way as other groups.
* *H5Pset\_compare*: This function provides options that allow users to override the default comparison done by *H5Ocompare*.
* *H5Pget\_compare*: This function retrieves the properties set for the comparison.
* *H5Pset\_compare\_value\_ndiffs:* This function allows users to set the maximum number of differences to report when comparing values of datasets and attributes.
* *H5Pget\_compare\_value\_ndiffs*: This function retrieves the maximum number of differences set in the comparison property list when comparing values of datasets and attributes.
* *H5Pset\_compare\_fp\_tolerance*: This function allows users to set the tolerance in the comparison property list when comparing floating-point values.
* *H5Pget\_compare*\_*fp\_tolerance*: This function retrieves the tolerance set in the comparison property list when comparing floating-point values.

# Comparing Objects

An HDF5 file is a container for an organized collection of HDF5 objects. The objects are groups, datasets, and committed datatypes. Comparing two objects in an HDF5 file requires comparing certain characteristics of those objects. The characteristics are:

* metadata that describe the objects
* attributes attached to the objects
* specific characteristics pertaining to the objects

By default, *H5Ocompare* will compare the full set of characteristics for the objects, with options to modify this behavior.

## Groups

A group contains zero or more links. The table below lists the characteristics that *H5Ocompare* will compare by default for groups and the available options.

|  |  |
| --- | --- |
| **CHARACTERISTIC** | **AVAILABLE OPTIONS** |
| Metadata | Do not compare metadata for groups |
| Attribute | Do not compare attributes attached to the groups |
| Link | Do not compare links in the groups |

The characteristics:

* *Metadata*: See Object metadata table in Appendix A for the list of metadata for groups.
* *Attribute*: By default, attributes attached to the groups are matched by their names. See section 4.4 for details about the comparison of attributes.
* *Link*: By default, links within the groups are matched by their names. See section 4.5 for details about the comparison of links in groups.

## Datasets

A dataset is an array variable. The shape of the array is described by a dataspace, and the type of its elements by a datatype. The table below lists the characteristics that *H5Ocompare* will compare by default for datasets and the available options.

|  |  |
| --- | --- |
| **CHARACTERISTIC** | **AVAILABLE OPTIONS** |
| Metadata | Do not compare metadata for datasets |
| Dataspace | Do not compare dataspaces |
| Datatype | Do not compare datatypes |
| Dataset value | Do not compare array elements |
| Attribute | Do not compare attributes attached to the datasets |

The characteristics:

* *Metadata*: See Object metadata table in Appendix A for the list of metadata for datasets.
* *Dataspace*: See details in section 4.6.
* *Datatype*: See details in section 4.7.
* *Dataset value*: See details in section 4.8.
* *Attribute*: By default, attributes attached to the datasets are matched by their names. See section 4.4 for details about the comparison of attributes.

## Committed datatypes

A committed datatype is a datatype object stored in an HDF5 file. The table below lists the characteristics that *H5Ocompare* will compare by default for committed datatypes and the available options.

|  |  |
| --- | --- |
| **CHARACTERISTIC** | **AVAILABLE OPTIONS** |
| Metadata | Do not compare metadata for committed datatypes |
| Definition | Do not compare datatype definitions |
| Attribute | Do not compare attributes attached to the committed datatypes |

The characteristics:

* *Metadata*: See Object metadata table in Appendix A for the list of metadata for committed datatypes.
* *Definition*: See details in section 4.7.
* *Attribute*: By default, attributes attached to the committed datatypes are matched by their names. See section 4.4 for details about the comparison of attributes.

## Attributes

An attribute is similar to a dataset; it has a dataspace, a datatype, and a value. Attributes are matched by their names (by default) or creation order. The table below lists the characteristics that *H5Ocompare* will compare by default for attributes and the available options.

|  |  |
| --- | --- |
| **CHARACTERISTIC** | **AVAILABLE OPTIONS** |
| Metadata | Do not compare metadata for attributes |
| Dataspace | Do not compare dataspaces |
| Datatype | Do not compare datatypes |
| Attribute value | Do not compare array elements |
| Name | Do not compare attribute names (when compared by creation order) |

The characteristics:

* *Metadata*: See Metadata for attributes table in Appendix A for the list of metadata.
* *Dataspace*: See details in section 4.6.
* *Datatype*: See details in section 4.7.
* *Attribute value*: See details in section 4.4.
* *Name*: Compare the names of attributes (only when compared according to creation order).

By default, *H5Ocompare* will compare common attributes attached to the objects and will report attributes that exist only in one of the two objects (*extra attributes*). The table below lists the options available for users to override the default comparison.

|  |  |
| --- | --- |
| **CHARACTERISTIC** | **AVAILABLE OPTIONS** |
| Common attributes | Do not compare common attributes (by name or creation order) |
| Extra attributes | Do not report extra attributes (by name or creation order) |
| -- | Compare attributes according to creation order |

The characteristics:

* *Common attributes*: Attributes that are matched according to name or creation order.
* *Extra attributes*: Attributes that exist only in one of the two objects. They are determined based on name or creation order.

## Links

A link is contained within a group and has a name, a type, and a value. Links are matched by their names (by default) or creation order. The table below lists the characteristics that *H5Ocompare* will compare by default for links and the available options.

|  |  |
| --- | --- |
| **CHARACTERISTIC** | **AVAILABLE OPTIONS** |
| Metadata | Do not compare metadata for links |
| Link type | Do not compare link types |
| Link value | Do not compare link values (for soft, external or user-defined link) |
| Link name | Do not compare link names (when compared by creation order) |

The characteristics:

* *Metadata*: See Metadata for links table in Appendix A for the list of metadata.
* *Link type*: Different link type (hard, soft, external or user-defined) will be reported.
* *Link value*: The value of the link for soft, external or user-defined link.
* *Link name*: Compare the names of links (only when compared according to creation order).

By default, *H5Ocompare* will compare common links in the groups and will report links that exist only in one of the two groups (*extra links*). The table below lists the options available for users to override the default comparison.

|  |  |
| --- | --- |
| **CHARACTERISTIC** | **AVAILABLE OPTIONS** |
| Common links | Do not compare common links (by name or creation order) |
| Extra links | Do not report extra links (by name or creation order) |
| -- | Compare links according to creation order |

The characteristics:

* *Common links*: Links that are matched according to name or creation order.
* *Extra links*: Links that exist only in one of the two groups. They are determined based on name or creation order.

## Dataspaces

A dataspace describes the logical layout of data elements stored in a dataset or an attribute. For example, for simple dataspaces in HDF5, the layout is characterized by the number of dimensions (rank) and the size of each dimension (extent). The table below lists the characteristics that *H5Ocompare* will compare by default for dataspaces and the available options.

|  |  |
| --- | --- |
| **CHARACTERISTIC** | **AVAILABLE OPTIONS** |
| Class | None |
| Rank | None |
| Current extent | None |
| Maximum extent | Do not compare the maximum extents |

The characteristics:

* *Class:* H5S\_NULL, H5S\_SCALAR, H5S\_SIMPLE are the three classes
* *Rank*: The number of dimensions (for H5S\_SIMPLE only)
* *Current extent:* The current extent of the dataspace (for H5S\_SIMPLE only)
* *Maximum extent*: The maximum extent of the dataspace (for H5S\_SIMPLE only)

Note that when the classes and/or ranks are not the same, *H5Ocompare* will report the dataspaces as different and will not continue further comparison.

## Datatypes

An HDF5 datatype can be an atomic type like an integer or floating-point type, or a composite type like compound, array or variable-length sequence type. A datatype is defined by its class and class-specific properties. The table below lists the characteristics that *H5Ocompare* will compare by default for datatypes and the available options.

|  |  |
| --- | --- |
| **CHARACTERISTIC** | **AVAILABLE OPTIONS** |
| Class | None |
| Class specific properties | None |

The characteristics:

* *Class*: e.g., integer (H5T\_INTEGER), float (H5T\_FLOAT), string (H5T\_STRING), etc.
* *Class specific properties:* e.g., size, signed or unsigned, byte order, etc.
  + H5T\_INTEGER—Size, precision, offset, padding, byte order, signed/unsigned
  + H5T\_FLOAT—Size, precision, offset, padding, byte order, and field information
  + H5T\_TIME—Size, precision, byte order
  + H5T\_STRING—Size (fixed or variable), character set, pad/no pad, pad character
  + H5T\_BITFIELD --Size, precision, offset, padding, and byte order
  + H5T\_OPAQUE—Size, tag
  + H5T\_COMPOUND—Size, number of members, member names, member datatypes, member offsets
  + H5T\_REFERENCE—Reference type (object or dataset region)
  + H5T\_ENUM—Number of elements, element names, element values, base datatype
  + H5T\_VLEN—Base datatype
  + H5T\_ARRAY--Rank, extent, base datatype

Note that when the datatype classes are not the same, *H5Ocompare* will report the datatypes as different and will not continue the comparison of class-specific properties.

## Values of datasets and attributes

A value of a dataset or an attribute generally consists of multiple data elements. The comparison of such values depends on the underlying datatypes and dataspaces, and is performed elementwise.

The class, class properties and convertibility of the datatype for each of the items (datasets or attributes) determine how the data elements are compared. Values of some types can be converted to other types. This conversion might occur within the same type class or might involve a transition to another datatype class.

* Conversion within class: e.g., when comparing a signed 8-bit integer and an unsigned 16-bit integer, *H5Ocompare* might convert both datasets’ data elements to signed 32-bit integers before comparing. Conversion is not done if the resulting conversion would exceed the maximum precision allowed in HDF5 (64-bits currently). Neither character set encoding [4] nor the string length is relevant in string comparison. For example, *H5Ocompare* will compare a fixed length string and a variable-length string.
* Conversion between classes: currently the HDF5 library can convert an H5T\_FLOAT to H5T\_INTEGER and vice versa. Conversion for the remaining classes is not yet supported.

The following rules apply to comparing values of composite datatypes of the same class:

* H5T\_COMPOUND: The above conversion rules will apply recursively through the nested fields.
* H5T\_ENUM, H5T\_VLEN, H5T\_ARRAY: The above conversion rules will apply to the base datatypes.

Similarly, the class, rank and current extent of the dataspace for each of the items being compared control which data elements are compared.

The comparison will proceed as follows:

* When the dataspace classes are not the same, *H5Ocompare* will report the values as not comparable and will not continue the comparison.
* When the dataspace classes are the same:
  + H5S\_NULL: *H5Ocompare* will not perform further comparison.
  + H5S\_SIMPLE with different ranks: *H5Ocompare* will report the values as not comparable and will not continue further comparison
  + H5S\_SIMPLE with same ranks, H5S\_SCALAR:
    - If the datatypes are different and are not convertible, *H5Ocompare* will report the values as not comparable and will not continue the comparison.
    - If the datatypes are the same or convertible:
      * H5S\_SCALAR: *H5Ocompare* will perform the comparison of the two data elements.
      * H5S\_SIMPLE:
        + Same current extent: *H5Ocompare* will perform the comparison of the data elements.
        + Different current extent: *H5Ocompare* will compare the overlapping data elements starting from the origin. The shaded areas in the following examples are the compared regions. *H5Ocompare* will report the values as different for the non-common regions and will report any differences found for the common regions.

Example1: space1[6x8]; space2[3x5] Example2: space1[2x8]; space2[4x1]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |
|  |

*H5Ocompare* will compare element by element with respect to the datatype class:

* H5T\_INTEGER: Any two integer values can be directly compared regardless of their encodings.
* H5T\_FLOAT: Any two floating-point values can be directly compared regardless of their encodings. There are two aspects of floating-point value comparison that can be controlled:
  + Tolerance—To determine whether two floating-point numbers, *float1* and *float2*, are different, use the formula *|float1 – float2| >=* *tolerance*. By default, *H5Ocompare* will use the tolerance defined by the system. However, users can set the tolerance via the new public function, *H5Pset\_fp\_tolerance,* when comparing floating-point values; see details in section 6.5. See also “*Default EPSILON Values for Comparing Floating Point Data*” RFC [2].
  + Not-a-Number (NaN)— By definition, two NaNs are never equal; likewise, NaN and a finite number are always different[3]. By default, *H5Ocompare* will check NaNs via the C99 standard *isnan()*. Two options are available to users for handling NaNs:
    - Skip checking NaNs
    - Treat two NaNs as equal if their binary representations match
* H5T\_STRING: Strings are compared with the standard C *strcmp()* function [4].
* H5T\_BITFIELD: Encodings of such values will be compared byte by byte based on size, offset and precision.
* H5T\_OPAQUE: Encodings of such values will be compared byte by byte.
* H5T\_TIME: It is an unsupported datatype, but comparison will be performed byte by byte.
* H5T\_COMPOUND: Values will be compared according to matching field names based on the fields’ datatypes. For nested compound types, the comparison will recur through the nested fields. There are 4 possible sets of differences:
  + Fields having datatypes that are the same or convertible
  + Fields that exist only in object 1
  + Fields that exist only in object 2
  + Fields having datatypes that are not convertible
* H5T\_REFERENCE: Currently HDF5 has two kinds of reference datatypes—object references (H5R\_OBJECT) and dataset region references (H5R\_DATASET\_REGION). The following comparison is performed when comparing references:
  + H5R\_OBJECT: In the scope of an HDF5 file, each HDF5 object (group, dataset, committed datatye) can be referred to by a unique identifier. Such identifiers can be persisted in HDF5 object references. The default is not to perform any comparison but the following options are available:
    - Compare the object identifiers of the referenced objects
    - Compare the pathnames (if available) to the referenced objects
  + H5R\_DATASET\_REGION: In the scope of an HDF5 file, a selection in an HDF5 dataset can be persisted in an HDF5 region reference. Conceptually, such a region reference consists of an object reference to the dataset and a selection in the underlying dataspace. By default, *H5Ocompare* will compare the selections when the class and rank of the underlying dataspaces are the same; otherwise *H5Ocompare* will report them as not comparable (see previous description about dataspace class and rank in this section). The following options are available:
    - Compare the object identifiers of the referenced objects
    - Compare the pathnames (if available) to the referenced objects

For the comparison by pathnames, *H5Ocompare* proceeds with the comparison by finding common pathnames associated with the referenced objects. It also might encounter one of the following two situations:

* + For a reference to an unlinked object (no pathname to the object), *H5Ocompare* will return an empty string for the object pathname.
  + For a dangling reference (the reference cannot be resolved to an object), *H5Ocompare* will return a NULL pointer in lieu of the object pathname.

Note that *H5Ocompare* does not perform comparison of the objects being referenced.

* H5T\_ENUM: An enumerated datatype is a set of [name, value] pairs with an integer base datatype. By default, *H5Ocompare* will compare the *names* of the [name, value] pair. An option is available to compare by *values* instead.
* H5T\_VLEN: Each instance of a variable-length sequence datatype is a sequence of values of a particular base datatype. Comparison will proceed only if the base datatypes are convertible. Sequences of convertible datatypes are compared element by element. If the sequence lengths are not the same, *H5Ocompare* will report the sequences as different. If the sequence lengths match, *H5Ocompare* will return both sequences if at least one difference is found.
* H5T\_ARRAY: If the arrays’ ranks and extents are not the same, *H5Ocompare* will report the array elements as not comparable. Otherwise, *H5Ocompare* will compare element by element according to the base datatype, following the datatype conversion rules as described above. *H5Ocompare* will return both array elements if at least one difference is found.

By default, *H5Ocompare* will report all the differences found from comparing values of datasets or attributes. An option is available for users to set the maximum number of differences to report.

The table below summarizes the available options when comparing values of datasets or attributes:

|  |  |
| --- | --- |
| **CHARACTERISTIC** | **AVAILABLE OPTIONS** |
| Dataspace | Do not compare when the dataspaces are of different current extent (H5S\_SIMPLE) |
| Datatype | Do not attempt conversion of datatypes |
|  | Do not check for NaNs (H5T\_FLOAT) |
|  | Treat two NaNs as equal if their binary representations match (H5T\_FLOAT) |
|  | Use user-defined tolerance when comparing floating-point values (H5T\_FLOAT) |
|  | Compare the object identifiers of the referenced objects (H5T\_REFERENCE) |
|  | Compare the pathnames (if available) to the referenced objects (H5T\_REFERENCE) |
|  | Compare enumerated datatypes by *values* (H5T\_ENUM) |
| Difference count | Report maximum number of differences as set by the user |

# Comparing File Metadata

In this section, we describe how the new public function, *H5Fcompare\_md*, compares the file metadata of two HDF5 files. Each HDF5 file contains file metadata such as file creation properties. File metadata comparison includes comparing the following:

* version number of super block
* size of user block
* size of addresses
* size of lengths
* sizes used to control symbol tables (B-tree rank and node size)
* tree rank used to control B-trees for indexing chunked datasets
* strategy in managing file space
* file driver information
* number of shared message indexes
* configuration settings for a shared message index (type and minimum size of messages)
* threshold values for storing shared messages: maximum number of messages to store in a compact list, minimum number of messages to store in a B-tree)

# New public functions to handle comparison

In this section, we describe the following eight new public routines:

* *H5Ocompare*
* *H5Fcompare\_md*
* *H5Pset\_compare, H5Pget\_compare*
* *H5Pset\_compare\_value\_ndiffs, H5Pget\_compare\_value\_ndiffs*
* *H5Pset\_compare\_fp\_tolerance, H5Pget\_compare\_fp\_tolerance*

## New public function for comparing objects

**Name:**

H5Ocompare

**Signature:**

herr\_tH5Ocompare*( hid\_t* loc1\_id*,*

*const char \**name1*,*

*hid\_t* lapl1*,*

*hid\_t* loc2\_id*,*

*const char \**name2*,*

*hid\_t* lapl2*,*

*hid\_t* cmppl\_id*,*

*hbool\_t \**equal*,*

*H5O\_cmp\_cb\_t \**cb\_info*)*

**Purpose:**

Compares two objects in the same or different files.

**Description:**

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

name1 or name2 may be an absolute pathname in the file referenced by loc1\_id or loc2\_id respectively or a relative pathname with respect to loc1\_id or loc2\_id respectively.

The parameters lapl1 and lapl2 are link access property lists associated with the links name1 and name2 respectively.

The parameter, cmppl\_id, is the comparison property list. By default, H5Ocompare will compare all the default characteristics for the objects. Users can specify a subset of the characteristics for the comparison in cmppl\_idvia the public function H5Pset\_compare and pass to H5Ocompare.

The parameter, equal, indicates the result of the comparison:

* True if the two objects are equivalent
* False if the two objects are not equivalent

Differences in the two objects are reported via callback functions, which are grouped together in a structure *H5O\_cmp\_cb\_*t as defined below. This structure is passed as the cb\_info parameter to this function along with a pointer to user-supplied data:

typedef struct H5O\_cmp\_cb\_t {

H5O\_cmp\_link\_cb\_t link;

H5O\_cmp\_obj\_md\_cb\_t obj\_md;

H5O\_cmp\_attr\_md\_cb\_t attr\_md;

H5O\_cmp\_dset\_data\_cb\_t dset\_data;

H5O\_cmp\_attr\_data\_cb\_t attr\_data;

void \*udata;

} H5O\_cmp\_cb\_t;

Details of these callbacks are described in the next sections.

On entry to H5Ocompare, the function will try to resolve name1 with respect to loc1\_id and name2 with respect to loc2\_id to objects, using lapl1 and lapl2, respectively. If not successful, H5Ocompare will return an error and exit. If successful but the object types (groups, datasets, committed datatypes) are not the same, H5Ocompare will report the two objects as different and exit. If the object types are the same, H5Ocompare will:

1. Compare the two objects’ attributes, and report any differences found in metadata via the attr\_md callback and values via the attr\_data callback.
2. Compare the metadata of the two objects, and invoke the obj\_md callback for each difference found.
3. Compare the two objects:
   1. Datasets: H5Ocompare will compare the values and report all the differences found via the dset\_data callback.
   2. Committed datatypes: H5Ocompare has already completed the comparison in steps (a) and (b) above.
   3. Groups: H5Ocompare will compare all the links in the two groups and report any differences found via the link callback. If recursive comparison is desired, applications will need to iterate links in the groups with another function and then perform object comparisons with further calls to H5Ocompare.

**Parameters:**

|  |  |
| --- | --- |
| *hid\_t* loc1\_id | IN: Location identifier of the first object to be compared |
| *const char \**name1  *hid\_t* lapl1 | IN: Pathname to the first object to be compared  IN: Link access property list associated with the first object |
| *hid\_t* loc2\_id | IN: Location identifier of the second object to be compared |
| *const char \**name2  *hid\_t* lapl2 | IN: Pathname to the second object to be compared  IN: Link access property list associated with the second object |
| *hid\_t* cmppl\_id  *hbool\_t* \*equal | IN: Comparison property list identifier  IN/OUT: Result of the comparison |
| *H5O\_cmp\_cb\_t \**cb\_info | IN/OUT: A callback structure that contains a list of callback functions and a pointer to the user’s data for reporting the comparison results. |

**Returns:**

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

### Callback functions

H5Ocompare will invoke a callback function when encountering differences from comparing:

* links
* object metadata
* attribute metadata
* values in datasets
* values in attributes

The definitions of the five callback functions—link*,* obj\_md*,* attr\_md*,* dset\_data*,* attr\_data—are described in the following sections. H5Ocompare may invoke the corresponding callback repeatedly for each type of difference found. The return value from each callback function can be:

* A zero value, which causes the callback to continue reporting the remaining differences found.
* A non-zero value, which causes the callback to stop reporting the remaining differences found.

Each callback uses an enumerated type *H5\_cmp\_status\_t* to report the comparison result—see declaration in section 9.1 in Appendix B. The four enumerated defines are:

* H5\_STATUS\_DIFFERENT
  + - The two values are different
* H5\_STATUS\_ONLY\_OBJ1
  + - The value exists only in the first object
* H5\_STATUS\_ONLY\_OBJ2
  + - The value exists only in the second object
* H5\_STATUS\_NOT\_COMPARABLE
  + - See section 4.8 for details when such cases occur.

### The link callback function

*herr\_t (\*H5O\_cmp\_link\_cb\_t)( H5O\_cmp\_index\_t* index*,*

*H5O\_cmp\_obj\_md\_type\_t* type*,*

*H5\_cmp\_status\_t* status*, const H5O\_cmp\_link\_values\_t \**values*,*

*void \**udata*)*

The parameters have the following values and meanings:

index

* Indicates which link is being compared:
  + When compared according to name, name is valid and is the link name.
  + When compared according to creation order, corder is valid and is the link’s creation order.
* A union type, *H5O\_cmp\_index\_t* is defined in section 9.2.

type

* Reports the type of difference found.
* An enumerated type, *H5O\_cmp\_link\_type\_t* is defined in section 6.1.2.1.

status

* Reports the result of the comparison for type.
* An enumerated type, *H5\_cmp\_status\_t* is defined in section 9.1.

values

* Reports the values of the difference found for type.
* A union type, *H5O\_cmp\_link\_values\_t* is defined in section 6.1.2.2.
* Each structure in the union corresponds to each value defined for type. There are two fields of the same data type in each structure:
  + If status is H5\_STATUS\_ONLY\_OBJ1, the value of the second field in the structure is undefined.
  + Ifstatus is H5\_STATUS\_ONLY\_OBJ2, the value of the first field in the structure is undefined.

udata

* Shares application-defined data between the application and the callbacks.
* Equals to the udata field in the parameter cb\_info that is passed to H5Ocompare.

#### H5O\_cmp\_link\_type\_t

The following table lists and describes the types of differences defined for *H5O\_cmp\_link\_type\_t*:

|  |  |  |
| --- | --- | --- |
| **H5O\_cmp\_link\_type\_t** | **DESCRIPTION OF THE DIFFERENCE FOUND** | **PUBLIC ROUTINE TO SET IT** |
| H5O\_LINK\_EXIST | * Indicates that the link only exists in one group * status parameter indicates which group the link exists in * values parameter is set to NULL for the callback * The only callback made for this link | -- |
| H5O\_LINK\_CSET | Character set encoding of the link name | H5Pset\_char\_encoding |
| H5O\_LINK\_CORDER | Creation order of the link | H5Pset\_link\_creation\_order |
| H5O\_LINK\_TYPE | Link type (hard, soft, external or user-defined link) | -- |
| H5O\_LINK\_VALUE | Link value (when comparing soft, external or user-defined links) | -- |
| H5O\_LINK\_NAME | Link name (when compared according to creation order) | -- |

#### H5O\_cmp\_link\_values\_t

*H5O\_cmp\_link\_values\_t* is a union of the following structures:

|  |  |  |  |
| --- | --- | --- | --- |
| **H5O\_cmp\_link\_type\_t** |  | **H5O\_cmp\_link\_type\_t** |  |
| H5O\_LINK\_CSET | struct {  H5T\_cset\_t val1;  H5T\_cset\_t val2;  } cset; | H5O\_LINK\_CORDER | struct {  int64\_t val1;  int64\_t val2;  } corder; |
| H5O\_LINK\_TYPE | struct {  H5L\_type\_t val1;  H5L\_type\_t val2;  } link\_type; | H5O\_LINK\_VALUE | struct {  \*H5O\_cmp\_link\_val\_t val1;  H5O\_cmp\_link\_val\_t val2;  } link\_val;  \*See declaration in section 9.3. |
| H5O\_LINK\_NAME | struct {  const char \*val1;  const char \*val2;  } link\_name; |  |  |

### The object metadata callback function

*herr\_t (\*H5O\_cmp\_obj\_md\_cb\_t)( H5O\_cmp\_obj\_md\_type\_t* type*,*

*H5\_cmp\_status\_t* status*,*

*const H5O\_cmp\_obj\_md\_values\_t \**values*,*

*void \**udata*)*

The parameters have the following values and meanings:

type

* Reports the type of difference found.
* An enumerated type, *H5O\_cmp\_obj\_md\_type\_t* is defined in section 6.1.3.1.

status

* Reports the result of the comparison for *type*.
* An enumerated type, *H5\_cmp\_status\_t* is defined in section 9.1.

values

* Reports the values of the difference found for type.
* A union type, *H5O\_cmp\_obj\_md\_values\_t* is defined in section 6.1.3.2.
* Each structure in the union corresponds to a value defined for type. There are two fields of the same data type in each structure:
  + If status is H5\_STATUS\_ONLY\_OBJ1, the value of the second field in the structure is undefined.
  + Ifstatus is H5\_STATUS\_ONLY\_OBJ2, the value of the first field in the structure is undefined.

*udata*

* Shares application-defined data between the application and the callbacks.
* Equals to the udata field in the parameter cb\_info that is passed to H5Ocompare.

#### H5O\_cmp\_obj\_md\_type\_t

The following table lists and describes the types of differences defined for *H5O\_cmp\_obj\_md\_type\_t*:

|  |  |  |
| --- | --- | --- |
| **H5O\_cmp\_obj\_md\_type\_t** | **DESCRIPTION OF THE DIFFERENCE FOUND** | **PUBLIC ROUTINE TO SET IT** |
| ***Groups, datasets, committed datatypes*** | | |
| H5O\_OBJ\_MD\_RC | Reference count of object | -- |
| H5O\_OBJ\_MD\_NUM\_ATTRS | Number of attributes attached to object | -- |
| H5O\_OBJ\_MD\_BTIME | Birth time | H5Pset\_obj\_track\_times |
| H5O\_OBJ\_MD\_ATIME | Access time | H5Pset\_obj\_track\_times |
| H5O\_OBJ\_MD\_CTIME | Change time | H5Pset\_obj\_track\_times |
| H5O\_OBJ\_MD\_MTIME | Modification time | H5Pset\_obj\_track\_times |
| H5O\_OBJ\_MD\_COMMENT | Object comment | H5Oset\_comment |
| H5O\_OBJ\_MD\_ATTR\_CRT\_ORDER | Creation order for attributes | H5Pset\_attr\_creation\_order |
| H5O\_OBJ\_MD\_ATTR\_MAX\_COMPACT | Max number of attributes to store in object header | H5Pset\_attr\_phase\_change |
| H5O\_OBJ\_MD\_ATTR\_MIN\_DENSE | Min number of attributes to store in dense storage | H5Pset\_attr\_phase\_change |
| ***Groups only*** | | |
| H5O\_OBJ\_MD\_GRP\_CRT\_ORDER | Creation order for links | H5Pset\_link\_creation\_order |
| H5O\_OBJ\_MD\_GRP\_MAX\_COMPACT | Max number of links to store for a compact group | H5Pset\_link\_phase\_change |
| H5O\_OBJ\_MD\_GRP\_MIN\_DENSE | Min number of links to store in a dense group | H5Pset\_link\_phase\_change |
| ***Datasets only*** | | |
| H5O\_OBJ\_MD\_DSPACE | Dataspace | -- |
| H5O\_OBJ\_MD\_LAYOUT | Layout type | H5Pset\_layout |
| H5O\_OBJ\_MD\_CHUNK | Chunked layout information | H5Pset\_chunk |
| H5O\_OBJ\_MD\_EXTERNAL\_COUNT | Number of external files for the dataset | H5Pset\_external |
| H5O\_OBJ\_MD\_EXTERNAL | External layout information (external dataset only) | H5Pset\_external |
| H5O\_OBJ\_MD\_FILL\_DTYPE | Datatype for fill value | H5Pset\_fill\_value |
| H5O\_OBJ\_MD\_FILL\_VALUE | Fill value | H5Pset\_fill\_value |
| H5O\_OBJ\_MD\_FILL\_TIME | Fill time | H5Pset\_fill\_time |
| H5O\_OBJ\_MD\_ALLOC\_TIME | Allocation time | H5Pset\_allloc\_time |
| ***Datasets and groups only*** | | |
| H5O\_OBJ\_MD\_FILTER\_COUNT | Number of filters in the pipeline | H5Pset\_filter |
| H5O\_OBJ\_MD\_FILTER\_PIPELINE | Filter pipeline | H5Pset\_filter |
| ***Datasets and committed datatypes only*** | | |
| H5O\_OBJ\_MD\_DTYPE | Datatype | -- |

#### H5O\_cmp\_obj\_md\_values\_t

*H5O\_cmp\_obj\_md\_values\_t* is a union of the following structures:

|  |  |  |  |
| --- | --- | --- | --- |
| **H5O\_cmp\_obj\_md\_type\_t** |  | **H5O\_cmp\_obj\_md\_type\_t** |  |
| H5O\_OBJ\_MD\_RC | struct {  unsigned val1;  unsigned val2;  } rc; | H5O\_OBJ\_MD\_NUM\_ATTRS | struct {  unsigned val1;  unsigned val2;  } num\_attrs; |
| H5O\_OBJ\_MD\_BTIME | struct {  time\_t val1;  time\_t val2;  } btime; | H5O\_OBJ\_MD\_ATIME | struct {  time\_t val1;  time\_t val2;  } atime; |
| H5O\_OBJ\_MD\_CTIME | struct {  time\_t val1;  time\_t val2;  } ctime; | H5O\_OBJ\_MD\_MTIME | struct {  time\_t val1;  time\_t val2;  } mtime; |
| H5O\_OBJ\_MD\_COMMENT | struct {  const char \*val1;  const char \*val2;  } comment; | H5O\_OBJ\_MD\_ATTR\_CRT\_ORDER | struct {  unsigned val1;  ussigned val2;  } attr\_crt\_order; |
| H5O\_OBJ\_MD\_ATTR\_MAX\_COMPACT | struct {  unsigned val1;  unsigned val2;  } attr\_max\_compact; | H5O\_OBJ\_MD\_ATTR\_MIN\_DENSE | struct {  unsigned val1;  unsigned val2;  } attr\_min\_dense; |
| H5O\_OBJ\_MD\_GRP\_CRT\_ORDER | struct {  unsigned val1;  ussigned val2;  } grp\_crt\_order; | H5O\_OBJ\_MD\_GRP\_MAX\_COMPACT | struct {  unsigned val1;  unsigned val2;  } grp\_max\_compact; |
| H5O\_OBJ\_MD\_GRP\_MIN\_DENSE | struct {  unsigned val1;  unsigned val2;  } grp\_min\_dense; | H5O\_OBJ\_MD\_DSPACE | struct {  H5O\_cmp\_space\_t val1;  H5O\_cmp\_space\_t val2;  } dspace1;  1See explanation below. |
| H5O\_OBJ\_MD\_LAYOUT | struct {  H5D\_layout\_t val1;  H5D\_layout\_t val2;  } layout; | H5O\_OBJ\_MD\_CHUNK | struct {  H5O\_cmp\_chunk\_t val1;  H5O\_cmp\_chunk\_t val2;  } chunk2;  2See explanation below. |
| H5O\_OBJ\_MD\_EXTERNAL\_COUNT | struct {  unsigned val1;  unsigned val2;  } external\_count; | H5O\_OBJ\_MD\_EXTERNAL | struct {  unsigned ext\_idx;  H5O\_cmp\_external\_t val1;  H5O\_cmp\_external\_t val2;  } external3;  3See explanation below. |
| H5O\_OBJ\_MD\_FILL\_DTYPE | struct {  \*H5O\_cmp\_dtype\_t val1;  H5O\_cmp\_dtype\_t val2;  } fill\_dtype;  \*See declaration in section 9.5. | H5O\_OBJ\_MD\_FILL\_VALUE | struct {  union {  struct {  hid\_t val1;  hid\_t val2;  } tids;  struct {  hid\_t tid;  const void \*val1;  const void \*val2;  } values;  } u;  } fill\_value4;  4See explanation below. |
| H5O\_OBJ\_MD\_FILL\_TIME | struct {  H5D\_fill\_time\_t val1;  H5D\_fill\_time\_t val2;  } fill\_time; | H5O\_OBJ\_MD\_ALLOC\_TIME | struct {  H5D\_alloc\_time\_t val1;  H5D\_alloc\_time\_t val2;  } alloc\_time; |
| H5O\_OBJ\_MD\_FILTER\_COUNT | struct {  unsigned val1;  unsigned val2;  } filter\_count; | H5O\_OBJ\_MD\_FILTER\_PIPELINE | struct {  unsigned pline\_idx;  H5O\_cmp\_pline\_t val1;  H5O\_cmp\_pline\_t val2;  } filter\_pline5;  5See explanation below. |
| H5O\_OBJ\_MD\_DTYPE | struct {  \*H5O\_cmp\_dtype\_t val1;  H5O\_cmp\_dtype\_t val2;  } dtype;  \*See declaration in section 9.5. |  |  |

1dspace—The fields val1 and val2 are defined as *H5O\_cmp\_space\_t*—see declaration in section 9.4. If the field class or rank in val1 is different from that in val2, the remaining fields in *H5O\_cmp\_space\_t* are undefined.

2chunk*—*The fields val1 and val2 are defined as *H5O\_cmp\_chunk\_t*—see declaration in section 9.6. If the field rank in val1 is different from that in val2, the remaining fields in *H5O\_cmp\_chunk\_t* are undefined.

3external—H5Ocompare will invoke the callback function repeatedly for the differences found for each external file’s information. The field ext\_idx is the index of the external file. The fields val1 and val2 are defined as *H5O\_cmp\_external\_t*—see declaration in section 9.7. If the external file only exists in object 1, val2 will be undefined and vice versa.

4fill\_value—Fill values are compared according to the fill value datatype, following the datatype conversion rules described previously. If status is H5\_STATUS\_NOT\_COMPARABLE due to fill values not convertible, the field u.tids will contain the two datatype identifiers that are not convertible. If status is H5\_STATUS\_DIFFERENT, the field u.values.tid will contain the native datatype identifiers for the fill values, and indicates how to interpret the values stored in u.values.val1andu.values.val2*.*

5filter\_pline—H5Ocompare will invoke the callback function repeatedly for the differences found for each filter’s information. The field pline\_idx is the index of the filter. The fields val1 and val2 are defined as *H5O\_cmp\_pline\_t*—see declaration in section 9.8. If the filter only exists in object 1, val2 will be undefined and vice versa.

### The attribute metadata callback function

*herr\_t (\*H5O\_cmp\_attr\_md\_cb\_t)( H5O\_cmp\_index\_t* index*,*

*H5O\_cmp\_attr\_md\_type\_t* type*, H5\_cmp\_status\_t* status*,*

*const H5O\_cmp\_attr\_md\_values\_t \**values*,*

*void \**udata*)*

The parameters have the following values and meanings:

index

* Indicates which attribute is being compared:
  + When compared according to name, name is valid and is the attribute name.
  + When compared according to creation order, corder is valid and is the attribute’s creation order.
* A union type, *H5O\_cmp\_index\_t* is defined in section 9.2.

type

* Reports the type of difference found.
* An enumerated type, *H5O\_cmp\_attr\_md\_type\_t* is defined in section 6.1.4.1.

status

* Reports the result of the comparison for type*.*
* An enumerated type, *H5\_cmp\_status\_t* is defined in section 9.1.

values

* Reports the values of the difference found for type.
* A union type, *H5O\_cmp\_attr\_md\_values\_t* is defined in section 6.1.4.2.
* Each structure in the union corresponds to each value defined for type. There are two fields of the same data type in each structure:
  + If status is H5\_STATUS\_ONLY\_OBJ1, the value of the second field in the structure is undefined.
  + Ifstatus is H5\_STATUS\_ONLY\_OBJ2, the value of the first field in the structure is undefined.

udata

* Shares application-defined data between the application and the callbacks.
* Equals to the udata field in the parameter cb\_info that is passed to H5Ocompare*.*

#### H5O\_cmp\_attr\_md\_type\_t

The following table lists and describes the types of differences defined for *H5O\_cmp\_attr\_md\_type\_t*:

|  |  |  |
| --- | --- | --- |
| **H5O\_cmp\_attr\_md\_type\_t** | **DESCRIPTION OF THE DIFFERENCE FOUND** | **PUBLIC ROUTINE TO SET IT** |
| H5O\_ATTR\_EXIST | * Indicates that the attribute only exists in one object * status parameter indicates which object the attribute exists in * values parameter is set to NULL for the callback * The only callback made for this attribute | -- |
| H5O\_ATTR\_CSET | Character set encoding of the attribute name | H5Pset\_char\_encoding |
| H5O\_ATTR\_CORDER | Creation order of the attribute | H5Pset\_attr\_creation\_order |
| H5O\_ATTR\_DTYPE | Datatype of the attribute | -- |
| H5O\_ATTR\_DSPACE | Dataspace of the attribute | -- |
| H5O\_ATTR\_NAME | Attribute name (when compared according to creation order) | -- |

#### H5O\_cmp\_attr\_md\_values\_t

*H5O\_cmp\_attr\_md\_values\_t* is a union of the following structures:

|  |  |  |  |
| --- | --- | --- | --- |
| **H5O\_cmp\_attr\_md\_type\_t** |  | **H5O\_cmp\_obj\_md\_type\_t** |  |
| H5O\_ATTR\_CSET | struct {  H5T\_cset\_t val1;  H5T\_cset\_t val2;  } cset; | H5O\_ATTR\_CORDER | struct {  H5O\_msg\_crt\_idx\_t val1;  H5O\_msg\_crt\_idx\_t val2;  } corder; |
| H5O\_ATTR\_DTYPE | struct {  \*H5O\_cmp\_dtype\_t val1;  H5O\_cmp\_dtype\_t val2;  } dtype;  \*See declaration in section 9.5. | H5O\_ATTR\_DSPACE | struct {  \*H5O\_cmp\_space\_t val1;  H5O\_cmp\_space\_t val2;  } dspace;    \*See declaration in section 9.4. |
| H5O\_ATTR\_NAME | struct {  const char \*val1;  const char \*val2;  } name; |  |  |

### The dataset value callback function

*herr\_t (\*H5O\_cmp\_dset\_data\_cb\_t) (H5\_cmp\_status\_t* status*,*

*const H5O\_cmp\_data\_ctx\_t \**ctx*,*

*void \**udata*)*

The parameters have the following values and meanings:

status

* Reports the result of the comparison.
* An enumerated type, *H5\_cmp\_status\_t* is defined in section 9.1.

ctx

* Provides the context for the differences found.
* A structure, *H5O\_cmp\_data\_ctx\_t* is defined in section 9.9. It is a union of two structures, *H5O\_cmp\_data\_tids\_t* and *H5O\_cmp\_data\_values\_t* defined in sections 9.10 and 9.11 respectively.
* It will have the following values depending on status:
  + If status is H5\_STATUS\_DIFFERENT, ctx->values will describe and contain the differences found from comparing the values of the datasets (or attributes).
  + If status is H5\_STATUS\_ONLY\_OBJ1*,* ctx->values.diffs.val2 will be NULL.
  + IfstatusisH5\_STATUS\_ONLY\_OBJ2, ctx->values.diffs.val1 will be NULL.
  + If status is H5\_STATUS\_NOT\_COMPARABLE*:*
    - * when not comparable due to different datatypes that are not convertible, ctx->tids will contain the two datatype identifiers and ctx->values will be NULL.
      * ctx will be NULL for all other cases.

udata

* Shares application-defined data between the application and the callbacks.
* Equals to the udata field in the parameter cb\_info that is passed to H5Ocompare.

### The attribute data callback function

*herr\_t (\*H5O\_cmp\_attr\_data\_cb\_t)( H5O\_cmp\_index\_t* index*,*

*H5\_cmp\_status\_t* status*,*

*const H5O\_cmp\_data\_ctx\_t \**ctx*,*

*void \**udata*)*

The parameters have the following values and meanings:

index

* Indicates which attribute is being compared:
  + When compared according to name, name is valid and is the attribute’s name.
  + When compared according to creation order, corder is valid and is the attribute’s creation order.
* A union type, *H5O\_cmp\_index\_t* is defined in section 9.2.

status

* Reports the result of the comparison.
* An enumerated type, *H5\_cmp\_status\_t* is defined in section 9.1.

ctx

* Provides the context for the differences found.
* See the description of ctx in section 6.1.5.

udata

* Shares application-defined data between the application and the callbacks.
* Equals to the udata field in the parameter cb\_info that is passed to H5Ocompare.

## New public function for comparing file metadata

**Name:**

H5Fcompare\_md

**Signature:**

herr\_tH5Fcompare\_md *( hid\_t* loc1\_id*,*

*hid\_t* loc2\_id*,*

*hid\_t* cmppl\_id*,*

*hbool\_t \**equal*,*

*H5F\_cmp\_file\_md\_cb\_t \**file\_md*,*

*void \**udata*)*

**Purpose:**

Compares the file metadata of the two files.

**Description:**

H5Fcompare\_md compares the file metadata of the file specified by loc1\_id with the file metadata of the file specified by loc2\_id. File metadata is information the library uses to describe the HDF5 file and to identify its associated objects.

The parameter cmppl\_id is the comparison property list (and is currently unused).

The parameter, equal, indicates the result of the comparison:

* True if all the file metadata of the two files are equivalent
* False if at least one difference is found from comparing the file metadata

Differences in the metadata are reported via the callback function, file\_md. This is passed as a parameter to this routine and is described below.

The parameter udata points to the user data and is passed as a parameter to the callback function.

**Parameters:**

|  |  |
| --- | --- |
| *hid\_t* loc1\_id  *hid\_t* loc2\_id  *hid\_t* cmppl\_id  *hbool\_t \**equal  *H5F\_cmp\_file\_md\_cb\_t* \*file\_md  *void \**udata | IN: Location identifier of the first file to be compared  IN: Location identifier of the second file to be compared  IN: The comparison property list  IN/OUT: Result of the comparison  IN/OUT: A callback function  IN/OUT: Pointer to the user data |
|  |  |

**Returns:**

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

### The file metadata callback function

*herr\_t (\*H5F\_cmp\_file\_md\_cb\_t)( H5F\_cmp\_file\_md\_type\_t* type*,*

*H5\_cmp\_status\_t* status*,*

*const H5F\_cmp\_file\_md\_values\_t \**values*,*

*void \**udata*)*

The callback function is invoked repeatedly for each difference found while comparing the two file’s metadata. The return values from the callback are the same as described previously in H5Ocompare.

The parameters of this callback function have the following values and meanings:

type

* Reports the type of difference found.
* An enumerated type, *H5F\_cmp\_file\_md\_type\_t* is defined in section 6.2.1.1.

status

* Reports the result of the comparison for *type*.
* An enumerated type, *H5\_cmp\_status\_t* is defined in section 9.1.

values

* Reports the values of the difference found for type.
* A union type, *H5F\_cmp\_file\_md\_values\_t* is defined in section 6.2.1.2.
* Each structure in the union corresponds to each value defined for type. There are two fields of the same data type in each structure:
  + If status is H5\_STATUS\_ONLY\_OBJ1, the value of the second field in the structure is undefined.
  + Ifstatus is H5\_STATUS\_ONLY\_OBJ2, the value of the first field in the structure is undefined.

udata

* Shares any application-defined data between the application and the callbacks.
* Equals to the udata field in the parameter cb\_info that is passed to H5Fcompare\_md.

#### H5F\_cmp\_file\_md\_type\_t

The following table lists and describes the types of differences defined for *H5O\_cmp\_file\_md\_type\_t*:

|  |  |  |
| --- | --- | --- |
| **H5O\_cmp\_file\_md\_type\_t** | **DESCRIPTION OF THE DIFFERENCE FOUND** | **PUBLIC ROUTINE TO SET IT** |
| H5F\_FILE\_MD\_USERBLOCK\_SIZE | Size of the user block | H5Pset\_userblock |
| H5F\_FILE\_MD\_SIZEOF\_ADDR | Size of addresses stored in the file | H5Pset\_sizes |
| H5F\_FILE\_MD\_SIZEOF\_SIZE | Size of lengths stored in the file | H5Pset\_sizes |
| H5F\_FILE\_MD\_SYM\_IK | “K” value of group B-tree internal nodes | H5set\_sym\_k |
| H5F\_FILE\_MD\_SYM\_LK | “K” value of group B-tree leaf nodes | H5Pset\_sym\_k |
| H5F\_FILE\_MD\_ISTORE\_K | “K” value of data chunk B-trees | H5Pset\_istore\_k |
| H5F\_FILE\_MD\_FILE\_SPACE | Strategy in managing file space | H5Pset\_file\_space |
| H5F\_FILE\_MD\_DRIVER\_INFO | File driver information | H5Pset\_fapl\_sec2, H5Pset\_fapl\_stdio  H5Pset\_fapl\_core, H5Pset\_fapl\_direct  H5Pset\_fapl\_family, H5Pset\_fapl\_log  H5Pset\_fapl\_multi, H5Pset\_fapl\_split |
| H5F\_FILE\_MD\_SH\_MSG\_COUNT | # of shared message indexes | H5Pset\_shared\_mesg\_nindexes |
| H5F\_FILE\_MD\_SH\_MSG\_IDX | Type of shared message indexes | H5Pset\_shared\_mesg\_index |
| H5F\_FILE\_MD\_SHARED\_MSG\_MAX | Max # of shared messages to store in a list | H5Pset\_shared\_mesg\_phase\_change |
| H5F\_FILE\_MD\_SHARED\_MSG\_MIN | Min # of shared messages to store in a B-tree | H5Pset\_shared\_mesg\_phase\_change |

#### H5F\_cmp\_file\_md\_values\_t

*H5F\_cmp\_file\_md\_values\_t* is a union of the following structures:

|  |  |  |  |
| --- | --- | --- | --- |
| **H5F\_cmp\_file\_md\_type\_t** |  | **H5F\_cmp\_file\_md\_type\_t** |  |
| H5F\_FILE\_MD\_USERBLOCK\_SIZE | struct {  hsize\_t val1;  hsize\_t val2;  } userblock\_size; | H5F\_FILE\_MD\_SIZEOF\_ADDR | struct {  size\_t val1;  size\_t val2;  } sizeof\_addr; |
| H5F\_FILE\_MD\_SIZEOF\_SIZE | struct {  size\_t val1;  size\_t val2;  } sizeof\_size; | H5F\_FILE\_MD\_SYM\_IK | struct {  unsigned val1;  unsigned val2;  } sym\_ik; |
| H5F\_FILE\_MD\_SYM\_LK | struct {  unsigned val1;  unsigned val2;  } sym\_lk; | H5F\_FILE\_MD\_ISTORE\_K | struct {  unsigned val1;  unsigned val2;  } istore\_k; |
| H5F\_FILE\_MD\_FILE\_SPACE | struct {  H5F\_file\_space\_type\_t val1;  H5F\_file\_space\_type\_t val2;  } file\_space\_type; | H5F\_FILE\_MD\_DRIVER\_INFO | struct {  \*H5F\_cmp\_driver\_t val1;  H5F\_cmp\_driver\_t val2;  } driver\_info;  \*See declaration in section 9.12. |
| H5F\_FILE\_MD\_SH\_MSG\_COUNT | struct {  unsigned val1;  unsigned val2;  } sh\_msg\_count; | H5F\_FILE\_MD\_SH\_MSG\_IDX | struct {  unsigned idx;  \*const H5F\_cmp\_sh\_msg\_idx\_t \*val1;  const H5F\_cmp\_sh\_msg\_idx\_t \*val2;  } sh\_msg\_idx1;  1See explanation below. |
| H5F\_FILE\_MD\_SH\_MSG\_MAX | struct {  unsigned val1;  unsigned val2;  } sh\_msg\_max\_list; | H5F\_FILE\_MD\_SH\_MSG\_MIN | struct {  unsigned val1;  unsigned val2;  } sh\_msg\_min\_btree; |

1sh\_msg\_idx—*H5Ocompare* will invoke the callback function repeatedly for the differences found for each shared message. The field idx is the index of the shared messages. The fields val1 and val2 are defined as *H5F\_cmp\_sh\_msg\_idx\_t*—see declaration in section 9.13. If the shared message only exists in object 1, val2 will be undefined and vice versa.

## New public functions for handling comparison properties

There will be a new property list class (H5P\_OBJ\_COMPARE) for comparing objects. Two new public functions are available to set and get properties when comparing objects.

### H5Pset\_compare

**Name:**

H5Pset\_compare

**Signature:**

herr\_t H5Pset\_compare *( hid\_t* cmppl\_id*,*

*H5\_flags\_t* compare\_options*)*

**Purpose:**

Sets the properties to use when comparing two objects.

**Description:**

H5Pset\_compare sets the properties in the comparison property list cmppl\_id that will be invoked when comparing two objects.

The parameter cmppl\_id is the comparison property list and specifies the properties governing the comparison of the two objects.

The parameter compare\_options is of type H5\_flags\_t with the following values:

|  |  |
| --- | --- |
| *Groups, datasets, committed datatypes* | |
| H5O\_COMPARE\_SKIP\_OBJ\_MD | Do not compare object metadata |
| H5O\_COMPARE\_SKIP\_OBJ\_ATTRS | Do not compare attributes attached to the objects |
| *Groups only* | |
| H5O\_COMPARE\_SKIP\_LINKS | Do not compare links in the groups |
| *Datasets only* | |
| H5O\_COMPARE\_SKIP\_DSPACES | Do not compare dataspaces |
| H5O\_COMPARE\_SKIP\_DVALUES | Do not compare dataset values |
| *Datasets and committed datatypes only* | |
| H5O\_COMPARE\_SKIP\_DTYPES | Do not compare datatypes |
| *Attributes attached to objects (groups, datasets, committed datatypes)* | |
| H5O\_COMPARE\_SKIP\_ATTR\_MD | Do not compare metadata |
| H5O\_COMPARE\_SKIP\_ATTR\_DTYPES | Do not compare datatypes |
| H5O\_COMPARE\_SKIP\_ATTR\_DSPACES | Do not compare dataspaces |
| H5O\_COMPARE\_SKIP\_ATTR\_DVALUES | Do not compare attribute values |
| H5O\_COMPARE\_SKIP\_ATTR\_NAMES | Do not compare attribute names (when compared by creation order) |
| H5O\_COMPARE\_SKIP\_COMMON\_ATTRS | Do not compare common attributes (name or creation order) |
| H5O\_COMPARE\_SKIP\_EXTRA\_ATTRS | Do not report attributes that exist only in one of the two objects |
| H5O\_COMPARE\_ATTRS\_BY\_CRT\_ORDER | Compare attributes according to creation order |
| *Links* | |
| H5O\_COMPARE\_SKIP\_LINK\_MD | Do not compare metadata |
| H5O\_COMPARE\_SKIP\_LINK\_TYPES | Do not compare link types |
| H5O\_COMPARE\_SKIP\_LINK\_VALUES | Do not compare link values (for soft, external or user-defined links) |
| H5O\_COMPARE\_SKIP\_LINK\_NAMES | Do not compare link names (when compared by creation order) |
| H5O\_COMPARE\_SKIP\_COMMON\_LINKS | Do not compare common links (name or creation order) |
| H5O\_COMPARE\_SKIP\_EXTRA\_LINKS | Do not report links that exist only in one of the two groups |
| H5O\_COMPARE\_LINKS\_BY\_CRT\_ORDER | Compare links according to creation order |
| *Dataspaces* | |
| H5O\_COMPARE\_SKIP\_MAX\_EXTENTS | Do not compare the maximum extents |
| *Values of datasets and attributes* | |
| H5O\_COMPARE\_SKIP\_DIFF\_DSPACES | Do not compare when the dataspaces are of different current extents (H5S\_SIMPLE) |
| H5O\_COMPARE\_SKIP\_DTYPES\_CONV | Do not attempt the conversion of datatypes |
| H5O\_COMPARE\_SKIP\_NANS | Do not check for NaNs (H5T\_FLOAT) |
| H5O\_COMPARE\_NANS\_ARE\_EQUAL | Treat two NaNs as equal if their binary representations match (H5T\_FLOAT) |
| H5O\_COMPARE\_REF\_IDS | Compare the object identifiers of the referenced objects (H5T\_REFERENCE) |
| H5O\_COMPARE\_REF\_PATHS | Compare the pathnames to the referenced objects (H5T\_REFERENCE) |
| H5O\_COMPARE\_ENUM\_VALUES | Compare enumerated datatypes by *values* (H5T\_ENUM) |

**Parameters:**

|  |  |
| --- | --- |
| *hid\_t* cmppl*\_id*  *H5\_flags\_t* compare\_option | IN: The comparison property list  IN: Flag(s) to be set for the comparison |

**Returns:**

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

### H5Pget\_compare

**Name:**

H5Pget\_compare

**Signature:**

herr\_t H5Pget\_compare *( hid\_t* cmppl\_id*,*

*H5\_flags\_t \**compare\_options*)*

**Purpose:**

Retrieves properties to be used when comparing two objects.

**Description:**

H5Pget\_compare retrieves the properties currently specified in the comparison property list cmppl\_id, which will be invoked when comparing two objects.

The parameter compare\_options is a bit map indicating the flags which govern the comparison of the two objects that are set in the comparison property list cmppl\_id.

**Parameters:**

|  |  |
| --- | --- |
| *hid\_t* cmppl\_id  *H5\_flags\_t*  \*compare\_options | IN: The comparison property list  OUT: Flag(s) set in the comparison property list |

**Returns:**

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

## New public functions to control the reporting of differences found for values

Two new public functions are available to control the reporting of differences when comparing the values of datasets or attributes. By default, *H5Ocompare* will report all the differences found.

### H5Pset\_compare\_value\_ndiffs

**Name:**

H5Pset\_compare\_value\_ndiffs

**Signature:**

herr\_t H5Pset\_compare\_value\_ndiffs *( hid\_t* cmppl\_id*,*

*size\_t* dset\_ndiffs*,*

*size\_t* attr\_ndiffs*)*

**Purpose:**

Sets the maximum number of differences to report when comparing the values of datasets and attributes.

**Description:**

H5Pset\_compare\_value\_ndiffs sets the maximum number of differences to report when comparing the values of datasets or attributes.

The parameter cmppl\_id is the comparison property list. The parameter dset\_ndiffs is the maximum number of differences to report when comparing the values of datasets, while the parameter attr\_ndiffs is the maximum number of differences to report when comparing the values of attributes. Passing in a value of 0 for dset\_ndiffs or attr\_ndiffs will retain the default setting—reporting all the differences found.

**Parameters:**

|  |  |
| --- | --- |
| *hid\_t* cmppl\_id  *size\_t* dset\_ndiffs  *size\_t* attr\_ndiffs | IN: The comparison property list  IN: The number of differences to report for datasets  IN: The number of differences to report for attributes |

**Returns:**

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

### H5Pget\_compare\_value\_ndiffs

**Name:**

H5Pget\_compare\_value\_ndiffs

**Signature:**

herr\_t H5Pget\_compare\_value\_ndiffs *( hid\_t* cmppl\_id*,*

*size\_t \**dset\_ndiffs*,*

*size\_t \**attr\_ndiffs*)*

**Purpose:**

Retrieves the maximum number of differences to report when comparing the values of datasets or attributes.

**Description:**

H5Pget\_compare\_value\_ndiffs retrieves the maximum number of differences to report that is set in the parameter cmppl\_id, which is the comparison property list.

The parameters dset\_ndiffs and attr\_ndiffs will contain the maximum number of differences to report when comparing values of datasets and attributes respectively. A return value of 0 in dset\_ndiffs or attr\_ndiffs indicates that the default setting (report all differences) is used.

**Parameters:**

|  |  |
| --- | --- |
| *hid\_t* cmppl\_id  *size\_t* \*dset\_ndiffs  *size\_t* \*attr\_ndiffs | IN: The comparison property list  OUT: The number of differences to report for datasets that is set in the comparison property list  OUT: The number of differences to report for attributes that is set in the comparison property list |

**Returns:**

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

## New public functions for handling tolerance

Two new public functions are available to set and get the tolerance when comparing floating-point values. The default to use will be the tolerance defined by the system, FLT\_EPSILON, DBL\_EPSILON, and LDBL\_EPSILON. If the system values for tolerance are not defined, use constants that are close to most tolerance values as:

#define FLT\_EPSILON 1.19209E-07

#define DBL\_EPSILON 2.22045E-16

#define LDBL\_EPSILON 1.0842E-19

### H5Pset\_compare\_fp\_tolerance

**Name:**

H5Pset\_compare\_fp\_tolerance

**Signature:**

herr\_t H5Pset\_compare\_fp\_tolerance *( hid\_t* cmppl\_id*,*

*H5\_cmp\_tolerance\_t* tolerance*)*

**Purpose:**

Sets the tolerance to use when comparing the two objects’ floating-point values.

**Description:**

H5Pset\_compare\_fp\_tolerance sets the tolerance, tolerance, in the comparison property list cmppl\_id that will be used when comparing floating-point values.

**Parameters:**

|  |  |
| --- | --- |
| *hid\_t* cmppl\_id  *H5\_cmp\_tolerance\_t* tolerance | IN: The comparison property list  IN: The tolerance value to be set  *H5\_cmp\_tolerance\_t* is defined as:  typedef union H5\_cmp\_tolerance\_t {  float f\_tolerance; /\* float \*/  double d\_tolerance; /\* double \*/  long double l\_tolerance; /\* long double \*/  } H5\_cmp\_tolerance\_t; |

**Returns:**

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

### H5Pget\_compare\_fp\_tolerance

**Name:**

H5Pget\_compare\_fp\_tolerance

**Signature:**

herr\_t H5Pget\_compare\_fp\_tolerance *( hid\_t* cmppl\_id*,*

*H5\_cmp\_tolerance\_t \**tolerance*)*

**Purpose:**

Retrieves the tolerance used when comparing the two objects’ floating-point values.

**Description:**

H5Pget\_compare\_fp\_tolerance retrieves the tolerance currently specified in the comparison property list cmppl\_id when comparing the two objects’ floating-point values.

**Parameters:**

|  |  |
| --- | --- |
| *hid\_t* cmppl\_id  *H5\_cmp\_tolerance\_t* \*tolerance | IN: The comparison property list  OUT: The tolerance that is set in the comparison property list |

**Returns:**

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

# Examples

In this section, we present a few examples for *H5Ocompare* and some of the auxiliary public functions proposed in this RFC.

## Example 1: Compare two groups

In this example, we will compare two groups, group1 and group2, in file1 and file2 respectively. group1 contains three datasets—dset1, dset2, dset3 while group2 contains two datasets—dset4, dset5. *H5Ocompare* will report the two groups as different since the link names in the two groups do not match. The link callback function will print the names of the three datasets as existing in group1 only and the names of the two datasets as existing in group2 only.

/\* The link callback function \*/

herr\_t link\_cb(H5O\_cmp\_index\_t index, H5O\_cmp\_obj\_md\_type\_t type, H5\_cmp\_status\_t status,

const H5O\_cmp\_link\_values\_t \*values, void \*udata)

{

herr\_t ret\_value = H5\_ITER\_CONT;

switch(type) {

case H5O\_LINK\_EXIST:

assert(values == NULL);

switch(status) {

case H5\_STATUS\_ONLY\_OBJ1:

printf(“%s exists only in the first group\n”, index.name);

break;

case H5\_STATUS\_ONLY\_OBJ2:

printf(“%s exists only in the second group\n”, index.name);

break;

default:

break;

} /\* end switch of status \*/

case H5O\_LINK\_CSET:

:

:

default:

break;

} /\* end switch of type \*/

return(ret\_value);

}

main()

{

hid\_t fid1, fid2;

H5O\_cmp\_cb\_t cb\_info;

hbool\_t equal;

fid1 = H5Fopen(“file1.h5”, H5F\_ACC\_RDONLY, H5P\_DEFAULT);

fid2 = H5Fopen(“file2.h5”, H5F\_ACC\_RDONLY, H5P\_DEFAULT);

cb\_info.link = link\_cb;

/\* Create group1 in file1.h5 with datasets “dset1”, “dset2”, “dset3” \*/

/\* Create group2 in file2.h5 with datasets “dset4”, “dset5” \*/

:

:

/\* Compare the two groups \*/

H5Ocompare(fid1, “group1”, H5P\_DEFAULT, fid2, “group2”, H5P\_DEFAULT, H5P\_DEFAULT, &equal, &cb\_info);

If(!equal)

printf(“group1 in file1.h5 is different from group2 in file2.h5\n”);

else

printf(“group1 in file1.h5 and group2 in file2.h5 are the same\n”);

:

:

}

## Example 2: Compare two datasets

In this example, we will compare two datasets, dset1 and dset2, in file1 and file2 respectively. The dataspace for dset1 is H5S\_SIMPLE with rank 1, while the dataspace for dset2 is H5S\_SIMPLE with rank 2. *H5Ocompare* will report the two datasets as not equal. The object metadata callback function will print the ranks of the two datasets, and the dataset value callback function will report the values of the two datasets as not comparable since the dataspace ranks are different.

/\* The object metadata callback function \*/

herr\_t obj\_md\_cb(H5O\_cmp\_obj\_md\_type\_t type, H5\_cmp\_status\_t status, const H5O\_cmp\_obj\_md\_values\_t \*values, void \*udata)

{

herr\_t ret\_value = H5\_ITER\_CONT;

switch(type) {

case H5O\_OBJ\_MD\_DSPACE:

switch(status) {

case H5\_STATUS\_DIFFERENT:

:

:

if(values->dspace.val1.rank != values->dspace.val2.rank)

printf(“rank for object 1 is %d, rank for object 2 is %d\n”,

values->dspace.val1.rank, values->dspace.val2.rank);

:

:

break;

:

:

default:

break;

} /\* end switch of status \*/

:

:

case default:

break;

} /\* end switch of type \*/

return(ret\_value);

} /\* obj\_md\_cb \*/

/\* The dataset value callback function \*/

herr\_t dset\_value\_cb(H5\_cmp\_status\_t status, const H5O\_cmp\_data\_ctx\_t \*ctx, void \*udata)

{

herr\_t ret\_value = H5\_ITER\_CONT;

switch(status) {

case H5\_STATUS\_NOT\_COMPARABLE:

if(ctx == NULL)

printf(“The values of the two datasets cannot be compared\n”);

else

:

:

case default:

break;

} /\* end switch of type \*/

return(ret\_value);

} /\* dset\_value\_cb \*/

main()

{

hid\_t fid1, fid2;

H5O\_cmp\_cb\_t cb\_info;

hbool\_t equal;

fid1 = H5Fopen(“file1.h5”, H5F\_ACC\_RDONLY, H5P\_DEFAULT);

fid2 = H5Fopen(“file2.h5”, H5F\_ACC\_RDONLY, H5P\_DEFAULT);

cb\_info.obj\_md = obj\_md\_cb;

cb\_info.dset\_data = dset\_value\_cb;

/\* Create dataset “dset1” with dataspace H5S\_SIMPLE and rank 1 in “file1.h5” \*/

:

:

/\* Create dataset “dset2” with dataspace H5S\_SIMPLE and rank 2 in “file2.h5 ” \*/

:

:

/\* Compare the two datasets \*/

H5Ocompare (fid1, “dset1”, H5P\_DEFAULT, fid2, “dset2”, H5P\_DEFAULT, H5P\_DEFAULT, &equal, &cb\_info);

If(!equal)

printf(“dset1 in file1.h5 is different from dset2 in file2.h5\n”);

else

printf(“dset1 in file1.h5 and dset2 in file2.h5 are the same\n”);

:

:

}

## Example 3: Compare values of two datasets

This example shows how to compare the values of two datasets. We use the public function *H5Pset\_compare* to skip the comparison of object metadata in the comparison property list. *H5Ocompare* will report the two datasets as not equal and the dataset value callback function will return the different dataset values to the caller.

typedef struct dset\_udata\_t {

hsize\_t \*offset\_dset; /\* OUT \*/

void \*value\_dset1; /\* OUT \*/

void \*value\_dset2; /\* OUT \*/

} dset\_udata\_t;

/\* The dataset value callback function \*/

herr\_t dset\_value\_cb(H5\_cmp\_status\_t status, const H5O\_cmp\_data\_ctx\_t \*ctx, void \*\_udata)

{

dset\_udata\_t \*udata = (dset\_udata\_t \*)\_udata;

herr\_t ret\_value = H5\_ITER\_CONT;

if(status == H5\_STATUS\_DIFFERENT && ctx && ctx->values) {

udata->offset\_dset = (hsize\_t \*) calloc(ctx->values.ndiffs \* ctx->values.rank \* sizeof(hsize\_t));

udata->value\_dset1 = calloc (ctx->values.ndiffs \* H5Tget\_size(ctx->values.tid));

udata->value\_dset2 = calloc (ctx->values.ndiffs \* H5Tget\_size(ctx->values.tid));

memcpy(udata->offset\_dset, ctx->values.offset, ctx->values.ndiff \* ctx->values.rank \* sizeof(hsize\_t));

memcpy(udata->value\_dset1, ctx->values.diffs.val1, ctx->values.ndiffs \* H5Tget\_size(ctx->values.tid));

memcpy(udata->value\_dset2, ctx->values.diffs.val2, ctx->values.ndiffs \* H5Tget\_size(ctx->values.tid));

}

return(ret\_value);

};

main()

{

hid\_t fid1, fid2, cmpl\_id;

H5O\_cmp\_cb\_t cb\_info;

H5\_flags\_t compare\_options = 0;

dset\_udata\_t udata;

hbool\_t equal;

fid1 = H5Fopen(“file1.h5”, H5F\_ACC\_RDONLY, H5P\_DEFAULT);

fid2 = H5Fopen(“file2.h5”, H5F\_ACC\_RDONLY, H5P\_DEFAULT);

cb\_info.dset\_data = dset\_value\_cb;

cb\_info.udata = &udata;

/\* Create dataset “dset1” and “dset2” with same dataspace class and rank in “file1.h5” and “file2.h5” \*/

/\* Write to the two datasets with different values \*/

:

/\* Do not compare dataset metadata \*/

cmpl\_id = H5Pcreate(H5P\_OBJ\_COMPARE);

compare\_options |= H5O\_COMPARE\_SKIP\_OBJ\_MD;

H5Pset\_compare(cmpl\_id, compare\_option);

/\* Compare the two datasets \*/

H5Ocompare (fid1, “dset1”, H5P\_DEFAULT, fid2, “dset2”, H5P\_DEFAULT, cmpl\_id, &equal, &cb\_info);

:

}

## Example 4: Compare file metadata

In this example, we will compare the file metadata of the two files, file1 and file2. The file metadata callback function will print out the metadata that are different between the two files.

#include “hdf5.h”

/\* The file metadata callback function \*/

herr\_t file\_md\_cb(H5F\_cmp\_file\_md\_type\_t type, H5\_cmp\_status\_t status, const H5F\_cmp\_file\_md\_values\_t \*cmp\_info, UNUSED void \*udata)

{

herr\_t ret\_value = H5\_ITER\_CONT;

if(status == H5\_STATUS\_DIFFERENT && cmp\_info)

{

switch(type) {

case H5F\_FILE\_MD\_USERBLOCK\_SIZE:

printf("Userblock size(file1, file2): \t%d\t\%d\n",

cmp\_info->userblock\_size.val1, cmp\_info->userblock\_size.val2);

break;

case H5F\_FILE\_MD\_SIZEOF\_ADDR:

printf("Size of addresses(file1, file2):: \t%d\t%d\n",

cmp\_info->sizeof\_addr.val1, cmp\_info->sizeof\_addr.val2);

break;

:

:

Default:

break;

}

}

return(ret\_value);

}

main()

{

hid\_t fid1, fid2;

hbool\_t equal;

fid1 = H5Fopen(“file1.h5”, H5F\_ACC\_RDONLY, H5P\_DEFAULT);

fid2 = H5Fopen(“file2.h5”, H5F\_ACC\_RDONLY, H5P\_DEFAULT);

H5Fcompare\_md(fid1, fid2, H5P\_DEFAULT, &equal, file\_md\_cb, NULL);

H5Fclose(fid1);

H5Fclose(fid2);

return 0;

}

# Future Extensions

* Allow user to specify the maximum number of differences reported per callback.
* Options to strengthen compatibility requirements for datatypes (for example, to require all fields in a compound datatype be present in both datatypes) or relax compatibility requirements for dataspaces (for example, to allow comparison as long as the total number of data elements is the same).
* Public routine *H5Scompare* for users to compare dataspaces.
* Allow user to specify a comparison function, which *H5Ocompare* will call when comparing values of datasets. (see HDFFV-7637)

# 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. |
| *February 4, 2011:* | Version 3 added more details on how to compare objects. |
| *March 16, 2011:* | Version 4 added details for *H5Ocompare* function and examples. |
| *January 18, 2012:* | Version 5 completely revised, removing recursive operation and revamping interface. |
| *October 2, 2018* | Version 6 updated version # and moved the section “Future Extensions” to the proper place. |

# References

[1] “*HDF5 File and Object Comparison Specification*” <http://www.hdfgroup.uiuc.edu/RFC/HDF5/tools/h5diff/Compare_spec/HDF5-comparisons_v3-RFC-2011-08-03.pdf>

[2] “RFC*: Default EPSILON values for comparing floating point data*” <http://www.hdfgroup.uiuc.edu/RFC/HDF5/tools/h5diff/h5diff_default_epsilon/RFC_h5diff_default_epsilon.pdf>

[3] “*Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic.*” <http://www.cs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF>

[4] HDF5 currently supports two types of character set encoding: US ASCII and UTF-8 Unicode encoding. UTF-8 is a superset of US ASCII. See the document “*UTF-8, a transformation format of ISO 10646*” <http://tools.ietf.org/html/rfc3629>

# Appendix A

|  |  |
| --- | --- |
| **Object metadata** | **Public routine to set it** |
| *Groups, datasets, committed datatype* | |
| Type of object | -- |
| Reference count of object | -- |
| Number of attributes attached to object | -- |
| Birth time | H5Pset\_obj\_track\_times |
| Access time | H5Pset\_obj\_track\_times |
| Change time | H5Pset\_obj\_track\_times |
| Modification time | H5Pset\_obj\_track\_times |
| Object comment | H5Oset\_comment |
| Creation order for attributes | H5Pset\_attr\_creation\_order (?RM only G, D) |
| Maximum number of attributes to store in object header | H5Pset\_attr\_phase\_change (?RM only G, D) |
| Minimum number of attributes to store in dense storage | H5Pset\_attr\_phase\_change (?RM only G, D) |
| *Groups only* | |
| Creation order for links | H5Pset\_link\_creation\_order |
| Maximum number of links to store for a compact group (new format only) | H5Pset\_link\_phase\_change |
| Minimum number of links to store in a dense group (new format only) | H5Pset\_link\_phase\_change |
| *Datasets only* | |
| Dataspace | -- |
| Layout type | H5Pset\_layout |
| Chunked layout information | H5Pset\_chunk |
| External layout information (external dataset only) | H5Pset\_external |
| Datatype for fill value | H5Pset\_fill\_value |
| Fill value | H5Pset\_fill\_value |
| Fill time | H5Pset\_fill\_time |
| Allocation time | H5Pset\_alloc\_time |
| *Datasets and groups only* | |
| Filter pipeline | H5Pset\_filter |
| *Datasets and committed datatypes only* | |
| Datatype | -- |

|  |  |
| --- | --- |
| **Metadata for links** | **Public routine to set it** |
| Character set encoding of the link name | H5Pset\_char\_encoding |
| Creation order of the link | H5Pset\_link\_creation\_order |

|  |  |
| --- | --- |
| **Metadata for attributes** | **Public routine to set it** |
| Character set encoding of attribute name | H5Pset\_char\_encoding |
| Creation order of the attribute | H5Pset\_attr\_creation\_order |

# Appendix B

## H5\_cmp\_status\_t

*H5\_cmp\_status\_t* is used by all callback functions and is defined as:

typedef enum H5\_cmp\_status\_t {

H5\_STATUS\_DIFFERENT,

H5\_STATUS\_ONLY\_OBJ1,

H5\_STATUS\_ONLY\_OBJ2,

H5\_STATUS\_NOT\_COMPARABLE

} H5\_cmp\_status\_t;

## H5O\_cmp\_index\_t

*H5O\_cmp\_index\_t* is used by the link and attribute metadata callback functions and is defined as:

typedef union H5O\_cmp\_index\_t {

const char \*name;

int64\_t corder;

} H5O\_cmp\_index\_t;

## H5O\_cmp\_link\_val\_t

*H5O\_cmp\_link\_val\_t* is used by the link callback function and is defined as:

typedef struct H5O\_cmp\_link\_val\_t {

H5L\_type\_t ltype;

union {

const char \*soft\_link;

struct {

const char \*filename;

const char \*obj\_path;

} ext\_link;

} lval;

} H5O\_cmp\_link\_val\_t;

## H5O\_cmp\_space\_t

*H5O\_cmp\_space\_t* is used by the object metadata and attribute metadata callback functions and is defined as:

typedef struct H5O\_cmp\_space\_t {

H5S\_class\_t class;

unsigned rank;

const hsize\_t size[H5S\_MAX\_RANK];

const hsize\_t max[H5S\_MAX\_RANK];

} H5O\_cmp\_space\_t;

## H5O\_cmp\_dtype\_t

*H5O\_cmp\_dtype\_t* is used by the object metadata and attribute metadata callback functions and is defined as:

typedef union H5O\_cmp\_dtype\_t {

H5T\_class\_t tclass;

size\_t size;

struct atomic {

H5T\_order\_t order;

size\_t prec;

size\_t offset;

H5T\_pad\_t lsb\_pad;

H5T\_pad\_t msb\_pad;

} atomic;

struct cmpd {

hid\_t dtype;

unsigned nmembs;

} cmpd;

struct enumer {

hid\_t base\_dtype;

unsigned nmembs;

} enumer;

struct vlen {

hid\_t base\_dtype;

} vlen;

struct opaque {

const char \*tag;

} opaque;

struct array {

hid\_t base\_dtype;

unsigned ndims;

const size\_t dim[H5S\_MAX\_RANK];

} array;

} H5O\_cmp\_dtype\_t;

## H5O\_cmp\_chunk\_t

*H5O\_cmp\_chunk\_t* is used by the object metadata callback function and is defined as:

typedef struct H5O\_cmp\_chunk\_t {

unsigned rank;

const hsize\_t dims[H5S\_MAX\_RANK];

} H5O\_cmp\_chunk\_t;

## H5O\_cmp\_external\_t

*H5O\_cmp\_external\_t* is used by the object metadata callback function and is defined as:

typedef struct H5O\_cmp\_external\_t {

const char \*name;

off\_t offset;

hsize\_t size;

} H5O\_cmp\_external\_t;

## H5O\_cmp\_pline\_t

*H5O\_cmp\_pline\_t* is used by the object metadata callback function and is defined as:

typedef struct H5O\_cmp\_pline\_t {

H5Z\_filter\_t id; /\* filter identification # \*/

unsigned int flags; /\* general properties of the filter \*/

const unsigned int \*cd\_values; /\* auxiliary data \*/

} H5O\_cmp\_pline\_t;

## H5O\_cmp\_data\_ctx\_t

*H5O\_cmp\_data\_ctx\_t* is used by the dataset value and attribute value callback functions and is defined as:

typedef union H5O\_cmp\_data\_ctx\_t {

H5O\_cmp\_data\_tids\_t tids;

H5O\_cmp\_data\_values\_t values;

}

## H5O\_cmp\_data\_tids\_t

*H5O\_cmp\_data\_tids\_t* is used by the dataset value and attribute value callback functions and is defined as:

typedef struct H5O\_cmp\_data\_tids\_t {

hid\_t tid1;

hid\_t tid2;

} H5O\_cmp\_data\_tids\_t;

## H5O\_cmp\_data\_values\_t

*H5O\_cmp\_data\_values\_t* is used by the dataset value and attribute value callback functions and is defined as:

typedef struct H5O\_cmp\_data\_values\_t {

unsigned rank;

unsigned ndiffs;

hid\_t tid;

const hsize\_t \*offset;

struct {

const void \*val1;

const void \*val2;

} diffs;

} H5O\_cmp\_data\_values\_t;

The five fields in *H5O\_cmp\_data\_values\_t* have the following values and meanings:

rank

* The number of dimensions for the dataspaces.

ndiffs

* The number of differences reported by this call which may be one of the following:
  + The number of differences specified by the user via H5Pset\_compare\_value\_ndiffs.
  + The total number of differences .
  + The maximum number of differences based on the library default buffer size.

tid

* The datatype identifiers (native or native datatype after conversion) of the two datasets (or attributes).

offset

* The array of coordinate tuples where the differences were found. The size of the offset array is rank \* ndiffs.

diffs

* A structure of two arrays containing elements that were found to different.
* Data element types are described by tid.

## H5F\_cmp\_driver\_t

*H5F\_cmp\_driver \_t* is used by the file metadata callback function and is defined as:

typedef struct H5F\_cmp\_driver\_t {

hid\_t driver\_id;

union {/\* nothing for sec2 and stdio drivers \*/

struct {

size\_t mboundary;

size\_t fbsize;

size\_t cbuf\_size;

hbool\_t must\_align;

} direct;

struct {

const char \*logfile;

unsigned long long flags;

size\_t buf\_size;

} log;

struct {

size\_t increment;

hbool\_t backing\_store;

} core;

struct {

hsize\_t memb\_size;

hid\_t memb\_fapl\_id;

} family;

struct {

H5FD\_mem\_t memb\_map[H5FD\_MEM\_NTYPES];

const hid\_t memb\_fapl[H5FD\_MEM\_NTYPES];

const char \*memb\_name[H5FD\_MEM\_NTYPES]

const haddr\_t memb\_addr[H5FD\_MEM\_NTYPES];

hbool\_t relax;

} multi;

} u;

} H5F\_cmp\_driver\_t;

## H5F\_cmp\_sh\_msg\_idx\_t

*H5F\_cmp\_sh\_msg\_idx\_t* is used by the file metadata callback function and is defined as:

typedef struct H5F\_cmp\_sh\_msg\_idx\_t {

unsigned msg\_type\_flags;

unsigned min\_msg\_size;

} H5F\_cmp\_sh\_msg\_idx\_t;