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.

1 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.

2 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.

3 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.

4 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.

4.1 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.

4.2 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.

4.3 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.

4.4 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.

4.5 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.

4.6 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.

4.7 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.

4.8 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:

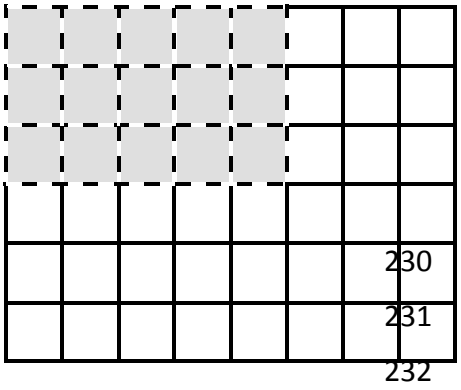
- H5T_COMPOUND: The above conversion rules will apply recursively through the nested fields.
- H5T_ENUM, H5T_VLEN, H5T_ARRAY: The above conversion rules will apply to the base datatypes.

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

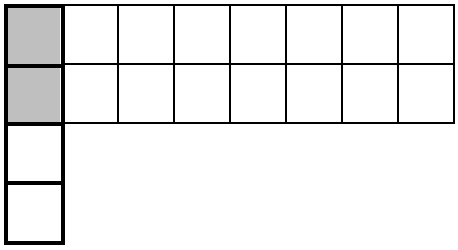
The comparison will proceed as follows:

- 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

5 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)

6 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*

6.1 New public function for comparing objects

Name:

`H5Ocompare`

Signature:

herr_t `H5Ocompare(`

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_id` via 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:

a) Compare the two objects' attributes, and report any differences found in metadata via the `attr_md` callback and values via the `attr_data` callback.
b) Compare the metadata of the two objects, and invoke the `obj_md` callback for each difference found.
c) Compare the two objects:
   i. Datasets: `H5Ocompare` will compare the values and report all the differences found via the `dset_data` callback.
   ii. Committed datatypes: `H5Ocompare` has already completed the comparison in steps (a) and (b) above.
   iii. 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`	IN: Pathname to the first object to be compared
hid_t `lapl1`	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`	IN: Pathname to the second object to be compared
hid_t `lapl2`	IN: Link access property list associated with the second object
hid_t `cmppl_id`	IN: Comparison property list identifier
hbool_t *`equal`	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.

6.1.1 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.

6.1.2 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.
  - If `status` 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`.

6.1.2.1 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)	--

458

459 *6.1.2.2 H5O_cmp_link_values_t*

460 *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;`		

461

462

6.1.3 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.
  - If `status` 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`.

6.1.3.1 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	--

491

492 *6.1.3.2 H5O_cmp_obj_md_values_t*

493 *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; } dspace[1]; [1]See 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; } chunk[2]; [2]See 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; } external[3]; [3]See 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_value[4];
			
			[4]See 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_pline[5]; [5]See explanation below.
H5O_OBJ_MD_DTYPE	struct { *H5O_cmp_dtype_t val1; H5O_cmp_dtype_t val2; } dtype; *See declaration in section 9.5.		

[1]`dspace`—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.

[2]`chunk`—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.

[3]`external`—`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.

516 [4]`fill_value`—Fill values are compared according to the fill value datatype, following the datatype
517 conversion rules described previously. If `status` is `H5_STATUS_NOT_COMPARABLE` due to fill values not
518 convertible, the field `u.tids` will contain the two datatype identifiers that are not convertible. If
519 `status` is `H5_STATUS_DIFFERENT`, the field `u.values.tid` will contain the native datatype identifiers for
520 the fill values, and indicates how to interpret the values stored in `u.values.val1` and `u.values.val2`.

521 [5]`filter_pline`—`H5Ocompare` will invoke the callback function repeatedly for the differences found for
522 each filter's information. The field `pline_idx` is the index of the filter. The fields `val1` and `val2` are
523 defined as *H5O_cmp_pline_t*—see declaration in section 9.8. If the filter only exists in object 1, `val2`
524 will be undefined and vice versa.

525

6.1.4 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.
  - If `status` 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`.

6.1.4.1 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)	--

562

563 *6.1.4.2 H5O_cmp_attr_md_values_t*

564 *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;		

565

566

6.1.5 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.
  - If `status` is `H5_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`.

6.1.6 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`.

6.2 New public function for comparing file metadata

Name:

H5Fcompare_md

Signature:

```
herr_t H5Fcompare_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	IN: Location identifier of the first file to be compared
hid_t loc2_id	IN: Location identifier of the second file to be compared
hid_t cmppl_id	IN: The comparison property list
hbool_t *equal	IN/OUT: Result of the comparison
H5F_cmp_file_md_cb_t *file_md	IN/OUT: A callback function
void *udata	IN/OUT: Pointer to the user data

Returns:

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

6.2.1 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.
  - If `status` 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`.

6.2.1.1 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

6.2.1.2 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_idx`[1]`;`
			
			[1]See 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;`

[1]`sh_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.

6.3 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.

6.3.1 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` IN: The comparison property list
H5_flags_t `compare_option` IN: Flag(s) to be set for the comparison

Returns:

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

6.3.2 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`	IN: The comparison property list
H5_flags_t *`compare_options`	OUT: Flag(s) set in the comparison property list

Returns:

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

6.4 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.

6.4.1 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`	IN: The comparison property list
size_t `dset_ndiffs`	IN: The number of differences to report for datasets
size_t `attr_ndiffs`	IN: The number of differences to report for attributes

Returns:

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

6.4.2 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`	IN: The comparison property list
size_t *`dset_ndiffs`	OUT: The number of differences to report for datasets that is set in the comparison property list
size_t *`attr_ndiffs`	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.

6.5 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
```

6.5.1 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	IN: The comparison property list
H5_cmp_tolerance_t tolerance	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.

6.5.2 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	IN: The comparison property list
H5_cmp_tolerance_t *tolerance	OUT: The tolerance that is set in the comparison property list

Returns:

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

7 Examples

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

7.1 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");

     :
     :
}
```

7.2 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");

     :
     :
}
```

7.3 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);
    :
}
```

7.4 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;
}
```

8 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

1118

1119

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

1120

1121

Appendix B

9.1 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;
```

9.2 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;
```

9.3 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;
```

9.4 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;
```

9.5 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;
```

9.6 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;
```

9.7 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;
```

9.8 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;
```

9.9 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;
}
```

9.10 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;
```

9.11 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`.

9.12 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;
```

9.13 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;
```