RFC: Fine-Grained Control of Metadata Cache Flushes

Dana Robinson

The HDF5 library caches recently accessed or created file metadata in an internal cache. Flushing of entries from the cache is normally managed via a modified least-recently-used algorithm, though the user can manually override this, preventing automatic flushes and evictions, and manually flushing either the entire cache or individual HDF5 objects (datasets, groups, and named datatypes).

The current flush prevention scheme in the HDF5 library is not very dynamic and is predominantly limited to allowing the entire metadata cache to have flushes disabled via somewhat awkward API calls. In some cases it would be useful to allow an application to have more dynamic, fine-grained, easier-to-use control over the flushing of the metadata cache and metadata for individual HDF5 objects such as datasets.

A collection of new functions will allow this dynamic, fine-grained flush control of both the entire cache and individual HDF5 objects. This RFC makes the case for the new functions and describes their semantics. The intended audience is advanced HDF5 users who desire control over the flush behavior of the metadata cache. It is particularly intended for users of the future single-writer/multiple-readers (SWMR) feature.

This functionality will be a part of the future HDF5 1.10 release.

# Introduction

The HDF5 library caches file metadata in an internal, per-file cache that is managed via a modified least-recently-used (LRU) policy. Users can, in a limited fashion, manually control when entries are flushed or evicted from the cache. The LRU algorithm can be disabled via the H5P/H5Fset\_mdc\_cache() API calls, leaving flush control up to the programmer. The entire cache can be flushed via calls to H5Fflush() and the cache entries that represent an HDF5 object (such as a dataset) can be flushed via calls to H5F/H5D/H5G/H5T/H5Oflush(). This control leaves much to be desired, however, as the flush control flag is a part of a large struct that is passed into the function, which is less convenient than a simple dedicated function.

In some cases, users may also desire fine-grained control over when metadata cache entries for a particular object are flushed from the cache. In the case of the single-writer/multiple-readers (SWMR) access pattern, control over the flushing behavior would allow a client to defer writing out file metadata until, say, all chunks in a logical plane or volume had been filled with data. In effect, this allows for the control of when data appears in HDF5 storage since the primary data cannot be accessed until the metadata that refers to it has been flushed.

# Normal Cache Operation

## Metadata and Stored Objects

In addition to the primary data stored by the user, an HDF5 file contains *file metadata* that is used to organize, locate/index, and describe the contents of the file. It serves many purposes, including chunk index structures, symbol tables representing groups and links, and object headers that describe the stored data (modification times, number of elements, etc.). This file metadata is largely invisible to the user and should not be confused with *user metadata*, which is stored as attributes attached to HDF5 objects such as groups, datasets, and named datatypes.

An HDF5 object such as a dataset will normally be composed of multiple sub-parts that will exist as separate metadata cache entries. For example, a chunked dataset with one unlimited dimension will be composed of an object header and an extendable array chunk index. The chunk index will be itself composed of a header, index block, etc. which will exist as separate entries in the cache.

The HDF5 file format document is available on the web[[1]](#footnote-1),[[2]](#footnote-2) and describes the metadata structures used in the file. Although this is a very low-level document intended for developers, it does give a rough idea of what file metadata objects and cache entries look like.

## Normal Operations

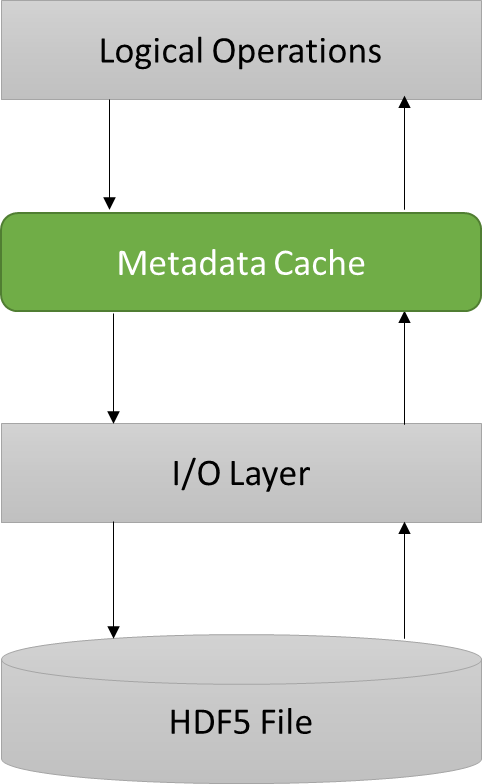
The metadata cache sits between the core object manipulation (logical) parts of the library and the I/O layer. All metadata reads and writes occur via the cache. The cache cannot be disabled; the logical library code never reads metadata directly from storage. The metadata cache is one of two key caches in the library, the other being the chunk cache which is independent and managed separately (though there are some associations under SWMR, via chunk proxies).

Figure 2‑1: Position of the metadata cache in the HDF5 library.

As an example, when a chunk index node is required by the library, a request for the node is sent to the cache, which either returns the node immediately if it is contained in the cache or reads it into the cache from storage and then returns the node if it has not been previously cached. Writing is handled similarly. The metadata cache is aware of both the type of each metadata object and the higher-level object to which it belongs. This is tracked via tags attached to each metadata object. Cache entries are evicted and, if dirty, flushed using a modified least recently used (LRU) algorithm. It is important to understand that the HDF5 library and thus the cache are not asynchronous in any way so the cache does not operate on a background thread. Instead cache operations like flush passes are triggered by conditions such as the current free space in the cache on cache access. These cache operations then run to completion before processing resumes.

Various metadata cache parameters can be adjusted via the public H5Pset\_mdc\_config() API call[[3]](#footnote-3). This function takes an input H5AC\_cache\_config\_t struct that contains many members. Most of these parameters are relatively unimportant for SWMR aside from flush/eviction control, discussed below in the flush prevention section.

## Flush Prevention

In the HDF5 library, the metadata cache or a particular HDF5 object in the cache (more correctly, the cache entries that are associated with an object) can have flushes to storage (via the usual eviction algorithm passes) disabled. Instead, the programmer must manually flush entries using the H5F/H5D/H5G/H5T/H5Oflush() calls. In the current HDF5 1.8.x and future 1.10.x releases, the metadata cache can have global flushes disabled by calling H5F/H5Pset\_mdc\_config() on the file access property list with the appropriate flags set. In the future 1.0.x release, as noted in this document, additional functionality that will allow fine-grained control of cache and object flushes will be introduced.

Note that in our implementation of cache flush control, only flushes of newly created or dirty metadata are prevented since this results in potentially expensive I/O operations, which we assume the user would like to control. Evictions of clean metadata are still allowed since they do not result in I/O operations and reduce memory overhead[[4]](#footnote-4).

# New Functions

Several new functions will be introduced to allow more fine-grained control over metadata cache flushes. They are introduced here with discussions of detailed semantics following later in this section.

The first set of functions controls flushes of the cache entries for individual objects.

*herr\_t* H5Odisable\_mdc\_flushes(*hid\_t* object\_id)

*herr\_t* H5Oenable\_mdc\_flushes(*hid\_t* object\_id)

*herr\_t* H5Oare\_mdc\_flushes\_disabled(*hid\_t* object\_id,

/\*OUT\*/ hbool\_t \*are\_disabled)

where object\_id is an object identifier as described in section 3.1.

The second set of functions controls flushes of the cache entries for an entire file.

*herr\_t* H5Fdisable\_mdc\_flushes(*hid\_t* file\_id)

*herr\_t* H5Fenable\_mdc\_flushes(*hid\_t* file\_id)

*herr\_t* H5Fare\_mdc\_flushes\_disabled(*hid\_t* file\_id,

/\*OUT\*/ hbool\_t \*are\_disabled)

where file\_id is a file identifier returned from H5Fopen() or H5Fcreate().

The last function returns a list of objects for which flushing has been disabled in a particular file's metadata cache.

*herr\_t* H5Fget\_mdc\_flush\_disabled\_obj\_ids(

*hid\_t* file\_id,

/\*OUT\*/ *int* \*n\_objects,

/\*OUT\*/ *hid\_t* object\_ids[])

where file\_id is a file identifier returned from H5Fopen() or H5Fcreate(), n\_objects is the number of object identifiers being returned, and object\_ids is the array of object identifiers returned by the function. Like most HDF5 API calls, the output array must be allocated by the caller using the mechanism described below.

Tentative reference manual pages for all functions can be found in the appendices section of this document.

NOTE: As of February 2014, only the H5O functions are implemented.

## HDF5 Objects

As mentioned in the introduction, the object-level flush control functions work with HDF5 objects. Hence, they will not work with all classes of hid\_t identifiers.

### Valid HDF5 object identifiers

* **Datasets** (hid\_t returned from H5Dopen/create)
* **Groups** (hid\_t returned from H5Gopen/create)
* **Named Datatypes** (hid\_t obtained from H5Topen/commit)

Only identifiers for named datatypes will work with these functions.

* **Objects** (hid\_t returned from H5Oopen)

An identifier returned from H5Oopen actually resolves to a dataset, group, or named datatype and is not really a separate category.

### INVALID identifiers

* **Files** (hid\_t returned from H5Fopen/create)

The H5F versions of the functions are used with file identifiers.

* **Attributes** (hid\_t returned from H5Aopen/create, etc.)

These are considered a part of the object to which they are attached.

* **Dataspaces** (hid\_t obtained from H5S\* functions or H5Dget\_space)

These are not stored in HDF5 files.

* **Property Lists** (hid\_t obtained via H5P\* functions)

These are not stored in HDF5 files.

## H5Odisable\_mdc\_flushes Semantics

H5Odisable\_mdc\_flushes(object\_id) is used to disable flushes for a specific object in the metadata cache. When it is called on an object identifier:

* Only identifiers that refer to HDF5 objects (datasets, groups, named datatypes) can be passed to the function.
* All cache entries for the object will be marked as "flushes disabled" in the metadata cache. Any newly created cache entries for the object will be marked on creation.
* No cache entries for the object will be flushed to storage by the cache's LRU policy.
* Clean entries for flush disabled objects can still be evicted from the cache.
* Flushing of the object's cache entries to storage must be performed manually by the user with the H5Oflush()[[5]](#footnote-5), H5Dflush(), H5Gflush(), H5Tflush(), or H5Fflush() calls.
* Flushes will be disabled for an object until explicitly changed using the H5Oenable\_mdc\_flushes() function, except as described below.
* When an object with disabled flushes is closed, the "flushes disabled" flag will be removed from all its cache entries as a part of the closing process.
* Calling the function on an identifier that does not refer to an object (e.g., a property list or file identifier) is considered an error. Like any other HDF5 error, this will return a negative error code.
* Calling the function on an object that has already been marked as "flushes disabled" is considered an error. This will return a negative error code.

The call must be used carefully to avoid running out of memory. Neglecting to flush large amounts of metadata could cause the cache to become large enough to consume all memory.

## H5Oenable\_mdc\_flushes Semantics

H5Oenable\_mdc\_flushes(object\_id) is used to re-enable metadata flushes for a specific HDF5 object in the metadata cache, allowing the cache's normal LRU algorithm to govern the flushing of its cache entries from the cache to storage (i.e., the default state). When it is called on an object identifier:

* All cache entries for the object will have the "flushes disabled" flag removed.
* Automatic flushing will resume on the object's entries. It will not necessarily result in an immediate flush of the object's entries.
* Calling the function on an identifier that does not refer to an object (e.g., a property list identifier or file identifier) is considered an error. This will return a negative error code.
* Calling the function on an object that has not been marked as "flushes disabled" is considered an error. This will return a negative error code.
* If the cache has been globally marked as having flushes disabled (either via H5Pset\_mdc\_config() or if H5Fdisable\_mdc\_flushes()), then H5Oenable\_mdc\_flushes() can be used to selectively enable flushes on a per-object basis.

## H5Oare\_mdc\_flushes\_disabled Semantics

H5Oare\_mdc\_flushes\_disabled(object\_id, /\*OUT\*/ hbool\_t \*are\_disabled) will emit the status of the object in the parameter are\_disabled: TRUE when an object has had flushes disabled and FALSE when it has not. It will return a negative value on errors and a non-negative value on success.

## H5Fdisable\_mdc\_flushes Semantics

When H5Fdisable\_mdc\_flushes(file\_id) is called on a file identifier:

* A global "flushes disabled" flag will be set in the file's metadata cache.
* Cache entries for all entries in the metadata cache will be marked as "flushes disabled".
* No "flush disabled" entries in the cache will be flushed to storage by the cache's LRU policy. This does not turn off the LRU algorithm, which can still flush entries for objects that have selectively had flushes enabled via H5Oenable\_mdc\_flushes().
* Clean entries can still be evicted from the cache.
* While flushes are disabled, flushing of dirty objects to storage must be performed manually by the user with the H5Oflush() or H5Fflush() call.
* Flushing for individual objects can be explicitly enabled using the H5Oenable\_mdc\_flushes() function.
* When an object that has flushes disabled is closed, all its cache entries will have the "flushes disabled" flag removed as a part of the closing process.
* When a file using a cache bearing the "flushes disabled" flag is closed, all objects in the cache will also have the "flushes disabled" flag removed as part of the closing process so they can be flushed.
* Calling the function on an identifier that is not an HDF5 file identifier is considered an error. This will return a negative error code.

Like the H5Odisable\_mdc\_flushes() function, the call must be used carefully to avoid running out of memory. Neglecting to flush large amounts of metadata could cause the cache to become large enough to consume all memory.

## H5Fenable\_mdc\_flushes Semantics

When H5enable\_mdc\_flushes(file\_id) is called on a file identifier:

* The global "flushes disabled" flag in the metadata cache will be unset.
* All entries in the metadata cache will have the "flushes disabled" flag removed.
* Automatic flushing will resume on all entries in the cache. This will not necessarily result in an immediate flush of any entries in the cache.
* Calling the function on an identifier that is not a file identifier is considered an error. This will return a negative error code.
* Calling the function on a file identifier that does not have the cache's "flushes disabled" flag set is considered an error. This will return a negative error code.

## H5Fare\_mdc\_flushes\_disabled Semantics

H5Fare\_mdc\_flushes\_enabled(file\_id, /\*OUT\*/ hbool\_t \*are\_disabled) will emit the status of the cache in the parameter are\_disabled: TRUE when the metadata cache for that file has flushes globally disabled and FALSE when it does not. It will return a negative value on errors and a non-negative value on success.

This function operates by inspecting the global cache flag set by H5Fdisable\_mdc\_flushes(). Manually disabling flushes for all objects in the metadata cache with H5Odisable\_mdc\_flushes() will NOT cause this function to return TRUE.

## H5Fget\_mdc\_flush\_disabled\_obj\_ids Semantics

H5Fget\_mdc\_flush\_disabled\_obj\_ids(*hid\_t* file\_id, /\*OUT\*/ *int* \*n\_objects, /\*OUT\*/ *hid\_t* object\_ids[]) returns an array of object identifiers that are currently marked as "flushes disabled" as well as the number of objects in the returned array. This function works like other HDF5 API calls that return arrays of things: The user must allocate the array that will be filled by the API call. This can be done by calling the function with a NULL object\_ids array, which will return the number of IDs in the n\_objects pointer. The correct size for the array will then be (\*n\_objects \* sizeof(hid\_t)).

## Interaction with H5Pset\_mdc\_config

H5Pset\_mdc\_config() can also be used to globally set the "flushes disabled" flag in the metadata cache, only less dynamically via the file access property list used to open or create the file. Setting the evictions\_enabled struct member to TRUE has the same effect as calling H5Fdisable\_mdc\_flushes() on the file.

# Testing

The new functionality will be tested at two levels:

## Cache Operations (test/cache.c)

The low-level cache operations of enabling and disabling cache flushes for objects will be tested in one or more functions added to the existing metadata cache tests in test/cache.c. These functions will use private HDF5 library functions to ensure that the internal mechanics of the flushing/eviction system are functioning correctly. An example might be ensuring that all cache entries for an object are listed as having flushes disabled when the internal object-level flush disable function is called.

## API Calls (test/flush\_disable.c – NEW)

Testing of the new API calls will take place in a new test in test/flush\_disable.c. Objects will be created or opened, have flushes disabled, be manipulated (e.g. datasets might be extended), and then be tested (via private HDF5 API calls) to see if they are correctly marked as having flushes disabled and have not been written to storage.

Situations that will be tested:

* File
* Dataset (unchunked)
* Dataset (version 1 B-tree chunk indexing)
* Dataset (fixed array chunk indexing)
* Dataset (extensible array chunk indexing)
* Dataset (version 2 B-tree chunk indexing)
* Group (old style)
* Group (new style)
* Named Datatype
* Attributes (new style that uses the fractal heap; small-, medium-, and large-size entries)
* Variable-length dataset data (due to interactions with the global heap)
* Region references as dataset data (due to interactions with the global heap)

Each dataset configuration will be tested with both SWMR on and off. All other tests will be performed with SWMR off since SWMR is only supported in the context of dataset extension at this time.

# Example Code

The following example shows an example of how the feature can be used to control the flushing of a particular object.

/\* Simple example of object-level metadata flush control.

\*

\* In this example, a dataset is created and filled with data.

\*

\* The dataset’s metadata will only be flushed after a chunk has been filled.

\*/

#define FILENAME "flush\_disable\_test.h5"

#define DSETNAME "test"

#define NELEMENTS 1048576

#define CHUNKSIZE 128

int main(int argc, char \*argv[])

{

hid\_t fid, pid, dsid, msid, fsid, did;

hsize\_t chunk\_dims;

hsize\_t cur\_dims, max\_dims;

hsize\_t start, count;

int i;

/\* create the file \*/

fid = H5Fcreate(FILENAME, H5F\_ACC\_TRUNC, H5P\_DEFAULT, H5P\_DEFAULT);

/\* create the dataset

\* 1D integer dataset, unlimited in size, chunk size = CHUNKSIZE

\*/

chunk\_dims = CHUNKSIZE;

pid = H5Pcreate(H5P\_DATASET\_CREATE)

H5Pset\_chunk(pid, 1, &chunk\_dims);

cur\_dims = 0;

max\_dims = H5S\_UNLIMITED;

dsid = H5Screate\_simple(1, &cur\_dims, &max\_dims);

did = H5Dcreate2(fid, DSETNAME, H5T\_NATIVE\_INT, dsid, H5P\_DEFAULT, pid, H5P\_DEFAULT);

H5Pclose(pid);

H5Sclose(dsid);

/\* disable metadata flushes for the dataset \*/

H5Odisable\_mdc\_flushes(did);

/\* store some data \*/

max\_dims = NELEMENTS;

H5Dset\_extent(did, &max\_dims);

cur\_dims = 1;

max\_dims = 1;

msid = H5Screate\_simple(1, &cur\_dims, &max\_dims);

for(i = 0; i < NELEMENTS; i++) {

/\* write the data (in an inefficient manner) \*/

fsid = H5Dget\_space(did);

start = i;

count = 1;

H5Sselect\_hyperslab(fsid, H5S\_SELECT\_SET, &start, NULL, &count, NULL);

H5Dwrite(did, H5T\_NATIVE\_INT, msid, fsid, H5P\_DEFAULT, &i);

H5Sclose(fsid);

/\* flush the dataset after a chunk has been filled \*/

if(i % CHUNKSIZE == (CHUNKSIZE - 1)) {

H5Oflush(did);

}

}

H5Sclose(msid);

/\* re-enable metadata flushes for the dataset \*/

H5Oflush(did);

H5Oenable\_mdc\_flushes(did);

/\* close everything \*/

H5Dclose(did);

H5Fclose(fid);

return 0;

}

# Acknowledgements

This work is being funded by Dectris, a customer of The HDF Group.

# Revision History

|  |  |
| --- | --- |
| *December 11, 2013:* | Version 1 circulated for comment to HDF5 SWMR team. |
| *January 7, 2014:* | Version 2 incorporates changes suggested by Quincey and Elena. Circulated for comment to HDF5 SWMR team. |
| *January 21, 2014:* | Version 3 incorporates Quincey's comments on version 2 and suggestions from the meeting on Jan 13. Circulated for comment to HDF5 SWMR team. |
| *February 2, 2014:* | Version 4 incorporates some comments from Vailin after implementing H5Ocork. Circulated for comment to HDF5 SWMR team. |
| *February 24, 2014:* | Version 5 changes the new function names (e.g. H5Ocork => H5Odisable\_mdc\_flushes) after conversations with the customer. Circulated for comment to HDF5 SWMR team. |

# [Glossary, Terminology]

**cache entry** An item that is stored in the metadata cache. An HDF5 object will often be represented by multiple cache entries. As an example, each node in a B-tree index is represented as a separate cache entry.

**file metadata** Metadata that describes the internal structure of the file. Created by the HDF5 library and largely invisible to users.

**HDF5 object** A "thing" stored in HDF5 storage. Includes datasets, groups, and named datatypes. Note that attributes are not considered HDF5 objects in their own right, but instead are considered a part of the object to which they are attached.

**user metadata** Attributes created by the user that are attached to datasets, groups, or named datatypes.

# Appendix: H5Odisable\_mdc\_flushes Reference Manual Page

**Name:** H5Odisable\_mdc\_flushes

**Signature:**

*herr\_t* H5Odisable\_mdc\_flushes(*hid\_t* object\_id)

**Purpose:**

Prevents metadata entries for an HDF5 object from being flushed from the metadata cache to storage.

**Description:**

The H5O/H5Fenable/disable\_mdc\_flushes() and associated H5Xflush() functions can be used to control the flushing of entries from a file's metadata cache. Metadata cache entries can be controlled at both the individual HDF5 object level (datasets, groups, named datatypes) and the entire metadata cache level. The function prevents an object or cache's dirty metadata entries from being flushed from the cache by the usual cache eviction/flush policy. Instead, users must manually flush the cache or entries for individual objects via H5F/H5D/H5G/H5T/H5Oflush() calls.

**Note:**

HDF5 objects include datasets, groups, and named datatypes. Only *hid\_t* identifiers that represent these objects can be passed to the function.

Passing in a hid\_t identifier that represents any other HDF5 entity is considered an error.

It is an error to pass an HDF5 file identifier (obtained from H5Fopen() or H5Fcreate()) to this function. Use H5F version instead.

Misuse of this function can cause the cache to exhaust available memory.

Objects can be returned to the default automatic flush behavior with H5Oenable\_mdc\_flushes().

Flush prevention only pertains to new or dirty metadata entries. Clean entries can be evicted from the cache.

Calling this function on an object that has already had flushes disabled will return an error.

**Parameters:**

*hid\_t* object\_id IN: ID of object that will have flushes disabled.

(See the above notes for restrictions)

**Returns:**

Returns a non-negative value if successful. Otherwise returns a negative value.

# Appendix: H5Oenable\_mdc\_flushes Reference Manual Page

**Name:** H5Oenable\_mdc\_flushes

**Signature:**

*herr\_t* H5Oenable\_mdc\_flushes(*hid\_t* object\_id)

**Purpose:**

Returns the cache entries associated with an HDF5 object to the default metadata flush and eviction algorithm.

**Description:**

The H5O/H5Fenable/disable\_mdc\_flushes() and associated H5Xflush() functions can be used to control the flushing of entries from a file's metadata cache. Metadata cache entries can be controlled at both the individual HDF5 object level (datasets, groups, named datatypes) and the entire metadata cache level. The function prevents an object or cache's dirty metadata entries from being flushed from the cache by the usual cache eviction/flush policy. Instead, users must manually flush the cache or entries for individual objects via H5F/H5D/H5G/H5T/H5Oflush() calls.

**Note:**

HDF5 objects include datasets, groups, and named datatypes. Only *hid\_t* identifiers that represent these objects can be passed to the function.

Passing in a hid\_t identifier that represents any other HDF5 entity is considered an error.

It is an error to pass an HDF5 file identifier (obtained from H5Fopen() or H5Fcreate()) to this function. Use the H5F version of the function instead.

Using this function on an object that has not had flushes disabled is considered an error. The state of an object can be determined with H5Oare\_flushes\_disabled().

Individual objects can be returned to the default flush algorithm with this function after H5Fdisable\_mdc\_flushes() has been used to globally prevent flushes.

An object will be returned to the default flush algorithm when it is closed.

All objects will be returned to the default flush algorithm when the file is closed.

An object's entries will not necessarily be flushed as a result of calling this function.

**Parameters:**

*hid\_t* object\_id IN: ID of object that will have flushes re-enabled.

(See the above notes for restrictions)

**Returns:**

Returns a non-negative value if successful. Otherwise returns a negative value.

# Appendix: H5Oare\_mdc\_flushes\_disabled Reference Manual Page

**Name:** H5Oare\_mdc\_flushes\_disabled

**Signature:**

*herr\_t* H5Oare\_mdc\_flushes\_enabled(*hid\_t* object\_id,

/\*OUT\*/ hbool\_t \*are\_disabled)

**Purpose:**

Determines if an HDF5 object (dataset, group, named datatype) has had flushes of metadata entries disabled.

**Description:**

The H5O/H5Fenable/disable\_mdc\_flushes() and associated H5Xflush() functions can be used to control the flushing of entries from a file's metadata cache. Metadata cache entries can be controlled at both the individual HDF5 object level (datasets, groups, named datatypes) and the entire metadata cache level. The function prevents an object or cache's dirty metadata entries from being flushed from the cache by the usual cache eviction/flush policy. Instead, users must manually flush the cache or entries for individual objects via H5F/H5D/H5G/H5T/H5Oflush() calls.

**Note:**

HDF5 objects include datasets, groups, and named datatypes. Only *hid\_t* identifiers that represent these objects can be passed to the function.

Passing in a hid\_t identifier that represents any other HDF5 entity is considered an error.

It is an error to pass an HDF5 file identifier (obtained from H5Fopen() or H5Fcreate()) to this function. Use the H5F version of the function instead.

**Parameters:**

*hid\_t* object\_id IN: ID of an object in the cache.

(See the above notes for restrictions)

*hbool\_t* \*are\_disabled OUT: Flushes enabled/disabled.

**Returns:**

are\_disabled will be set to TRUE if an object is has had flushes disabled, FALSE if it has not.

Returns a non-negative value if successful, a negative value on errors.

# Appendix: H5Fdisable\_mdc\_flushes Reference Manual Page

**Name:** H5Fdisable\_mdc\_flushes

**Signature:**

*herr\_t* H5Fdisable\_mdc\_flushes(*hid\_t* file\_id)

**Purpose:**

Globally prevents dirty metadata entries from being flushed from the metadata cache to storage.

**Description:**

The H5O/H5Fenable/disable\_mdc\_flushes() and associated H5Xflush() functions can be used to control the flushing of entries from a file's metadata cache. Metadata cache entries can be controlled at both the individual HDF5 object level (datasets, groups, named datatypes) and the entire metadata cache level. The function prevents an object or cache's dirty metadata entries from being flushed from the cache by the usual cache eviction/flush policy. Instead, users must manually flush the cache or entries for individual objects via H5F/H5D/H5G/H5T/H5Oflush() calls.

**Note:**

Only HDF5 file identifiers (obtained from H5Fopen() or H5Fcreate()) may be passed to this function. To restore flushes on individual HDF5 objects, use H5O version of this function instead.

Passing in a hid\_t identifier that represents any other HDF5 entity is considered an error.

Misuse of this function can cause the cache to exhaust available memory.

Prevention only pertains to new or dirty metadata entries. Clean entries can still be evicted from the cache.

**Parameters:**

*hid\_t* file\_id IN: An HDF5 file identifier.

**Returns:**

Returns a non-negative value if successful. Otherwise returns a negative value.

# Appendix: H5Fenable\_mdc\_flushes Reference Manual Page

**Name:** H5Fenable\_mdc\_flushes

**Signature:**

*herr\_t* H5Fenable\_mdc\_flushes(*hid\_t* file\_id)

**Purpose:**

Returns a file's metadata cache to the standard eviction and flushing algorithm.

**Description:**

The H5O/H5Fenable/disable\_mdc\_flushes() and associated H5Xflush() functions can be used to control the flushing of entries from a file's metadata cache. Metadata cache entries can be controlled at both the individual HDF5 object level (datasets, groups, named datatypes) and the entire metadata cache level. The function prevents an object or cache's dirty metadata entries from being flushed from the cache by the usual cache eviction/flush policy. Instead, users must manually flush the cache or entries for individual objects via H5F/H5D/H5G/H5T/H5Oflush() calls.

**Note:**

Only HDF5 file identifiers (obtained from H5Fopen() or H5Fcreate()) may be passed to this function. To restore flushes on individual HDF5 objects, use H5O version of this function instead.

Passing in a hid\_t identifier that represents any other HDF5 entity is considered an error.

A file will be returned to the default flushing algorithm when closed.

A file's cache entries will not necessarily be flushed when the cache is returned to the default algorithm.

**Parameters:**

*hid\_t* file\_id IN: An HDF5 file identifier.

**Returns:**

Returns a non-negative value if successful. Otherwise returns a negative value.

# Appendix: H5Fare\_mdc\_flushes\_disabled Reference Manual Page

**Name:** H5Fare\_mdc\_flushes\_disabled

**Signature:**

*htri\_t* H5Fare\_mdc\_flushes\_enabled(*hid\_t* file\_id,

/\*OUT\*/ hbool\_t \*are\_disabled)

**Purpose:**

Determines if flushes have been globally disabled for a file's metadata cache.

**Description:**

The H5O/H5Fenable/disable\_mdc\_flushes() and associated H5Xflush() functions can be used to control the flushing of entries from a file's metadata cache. Metadata cache entries can be controlled at both the individual HDF5 object level (datasets, groups, named datatypes) and the entire metadata cache level. The function prevents an object or cache's dirty metadata entries from being flushed from the cache by the usual cache eviction/flush policy. Instead, users must manually flush the cache or entries for individual objects via H5F/H5D/H5G/H5T/H5Oflush() calls.

**Note:**

Only HDF5 file identifiers (obtained from H5Fopen() or H5Fcreate()) may be passed to this function. To determine the enabled/disabled state of metadata flushes for individual HDF5 objects, use H5O version of this function instead.

Passing in a hid\_t identifier that represents any other HDF5 entity is considered an error.

**Parameters:**

*hid\_t* file\_id IN: An HDF5 file identifier.

*hbool\_t* \*are\_disabled OUT: Flushes enabled/disabled.

**Returns:**

are\_disabled will be set to TRUE if the file's metadata cache has been set to globally prevent flushes via H5Fdisable\_mdc\_flushes, FALSE if it is not.

Returns a non-negative value if successful, a negative value on errors.

# Appendix: H5Fget\_mdc\_flush\_disabled\_obj\_ids Reference Manual Page

**Name:** H5Fget\_mdc\_flush\_disabled\_obj\_ids

**Signature:**

*herr\_t* H5Fget\_mdc\_flush\_disabled\_obj\_ids(

*hid\_t* file\_id,

/\*OUT\*/ *int* \*n\_objects,

/\*OUT\*/ *hid\_t* object\_ids[])

**Purpose:**

Returns a list of all object identifiers for which flushes have been disabled in a file's metadata cache.

**Description:**

The H5O/H5Fenable/disable\_mdc\_flushes() and associated H5Xflush() functions can be used to control the flushing of entries from a file's metadata cache. Metadata cache entries can be controlled at both the individual HDF5 object level (datasets, groups, named datatypes) and the entire metadata cache level. The function prevents an object or cache's dirty metadata entries from being flushed from the cache by the usual cache eviction/flush policy. Instead, users must manually flush the cache or entries for individual objects via H5F/H5D/H5G/H5T/H5Oflush() calls.

**Note:**

The object\_ids array must be allocated by the caller. The appropriate size can be determined by calling the function with object\_ids set to NULL, which will return the number of objects via the n\_objects pointer. The correct size of the array will then be (\*n\_objects \* sizeof(hid\_t)).

Only HDF5 file identifiers (obtained from H5Fopen() or H5Fcreate()) may be passed to this function.

Passing in a hid\_t identifier that represents any other HDF5 entity is considered an error.

**Parameters:**

*hid\_t* file\_id IN: File identifier

*int* \*n\_objects OUT: Number of object identifiers being returned

*hid\_t* object\_ids[] OUT: Array of object IDs (allocated by caller)

**Returns:**

Returns a non-negative value if successful, a negative value on errors.

1. Current 1.8.x format: <http://www.hdfgroup.org/HDF5/doc/H5.format.html> [↑](#footnote-ref-1)
2. Future 1.10.x format (supported under SWMR): <http://www.hdfgroup.org/HDF5/doc_test/revise_chunks/H5.format.html> (this is a temporary location). [↑](#footnote-ref-2)
3. [http://www.hdfgroup.org/HDF5/doc/RM/RM\_H5P.html#Property-SetMdcConfig](http://www.hdfgroup.org/HDF5/doc/RM/RM_H5P.html%23Property-SetMdcConfig) [↑](#footnote-ref-3)
4. We may introduce a setting to prevent evictions as well in future work. [↑](#footnote-ref-4)
5. H5Oflush() is a new function that will appear in HDF5 1.10.0. [↑](#footnote-ref-5)