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 by "corking" the cache, which prevents automatic flushes and evictions, and manually flushing either the entire cache or individual HDF5 objects (datasets, groups, and named datatypes).

The current corking scheme in the HDF5 library is not very dynamic and is predominantly limited to allowing the entire metadata cache to be corked 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 corking of the metadata cache and individual HDF5 objects such as datasets.

A collection of new functions will allow this dynamic, fine-grained corking 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 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.

1 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 ("corking" the cache). 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 `H5Oflush()`. This corking control leaves much to be desired, however, as the corked flag is also a part of a large struct that is passed into the function, which is less convenient than a simple `H5Xcork()` 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.

2 Normal Cache Operation

2.1 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[1,2] 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.

2.2 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).

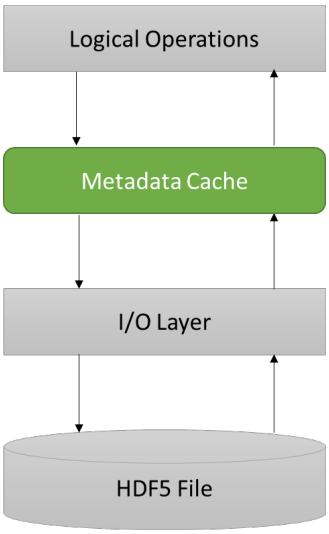

Figure 2-1: Position of the metadata cache in the HDF5 library.

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.

---

[1] Current 1.8.x format: http://www.hdfgroup.org/HDF5/doc/H5.format.html

[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).

Various metadata cache parameters can be adjusted via the public H5Pset_mdc_config() API call[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 corking section.

2.3 Corking

A cache or a particular HDF5 object in the cache (more correctly, the cache entries that are associated with an object) is considered *corked* when flushes to storage via the usual eviction algorithm passes are prevented from occurring. Instead, the programmer must manually flush entries using the `H5Fflush()` or `H5Oflush()` calls. In the current HDF5 1.8.x and future 1.10.x releases, the metadata cache can be corked by calling `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 allows fine-grained control of cache and object corking will be introduced.

Note that in our implementation of cache corking, only flushes of newly created or dirty metadata are prevented by corking 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].

3 New Functions[5]

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

The first set of functions allows corking and uncorking of the cache entries for an individual objects as well as checking to see if a particular object has been corked.

```
herr_t      H5Ocork(hid_t object_id)
herr_t      H5Ouncork(hid_t object_id)
herr_t      H5Ois_corked(hid_t object_id,
                  /*OUT*/ hbool_t *is_corked)
```

where `object_id` is an object identifier as described in section 3.1.

The second set of functions are used to cork or uncork the metadata cache for an entire file as well as checking to see if the file's cache has been corked.

[3] http://www.hdfgroup.org/HDF5/doc/RM/RM_H5P.html#Property-SetMdcConfig
[4] We may introduce a setting to prevent evictions as well in future work.
[5] As of February 1, 2014, only the H5Ocork/uncork/is_corked() functions are implemented.

```
herr_t      H5Fcork(hid_t file_id)
herr_t      H5Funcork(hid_t file_id)
herr_t      H5Fis_corked(hid_t file_id,
                  /*OUT*/ hbool_t *is_corked)
```

where `file_id` is a file identifier returned from `H5Fopen()` or `H5Fcreate()`.

The last function returns a list of corked objects.

```
herr_t      H5Oget_corked_object_list(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 corked object identifiers, and `object_ids` is an array of corked object identifiers returned by the function. Like most HDF5 API calls, the output array must be allocated by the caller using a mechanism described below.

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

.

3.1 HDF5 Objects

As mentioned in the introduction, the `H5Ocork/uncork/is_corked` functions work with HDF5 objects. Hence, they will not work with all classes of `hid_t` identifiers.

3.1.1 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 `H5Ocork/uncork`.
- **Objects** (hid_t returned from `H5Oopen`)

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

3.1.2 INVALID identifiers

- **Files** (`hid_t` returned from `H5Fopen/create`)

  `H5Fcork/uncork/is_corked` are used with file identifiers instead.

- **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 H5D`get_space`)

  These are not stored in HDF5 files.

- **Property Lists** (`hid_t` obtained via H5P* functions)

  These are not stored in HDF5 files.

3.2 H5Ocork Semantics

H5O`cork(object_id)` is used to cork a specific object in the metadata cache, preventing its metadata entries from being flushed to storage. 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 "corked" in the metadata cache. Any newly created cache entries for the object will be marked as "corked" on creation.
- No cache entries for the object will be flushed to storage by the cache's LRU policy.
- Clean entries for corked 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 H5O`flush()`[6], H5D`flush()`, H5G`flush()`, H5T`flush()`, or H5F`flush()` calls.
- An object will remain corked until explicitly uncorked using the H5O`uncork()` function, except as described below.
- When a corked object is closed, all its cache entries will be marked as "uncorked" as part of the closing process.
- Calling H5O`cork()` 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 H5O`cork()` on an object that has been corked 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.

3.3 H5Ouncork Semantics

H5O`uncork(object_id)` is used to uncork specific HDF5 objects in the metadata cache, allowing the cache's normal LRU algorithm to govern the flushing of its cache entries from the cache to storage. When it is called on an object identifier:

[6] H5O`flush()` is a new function that will appear in HDF5 1.10.0.

- All cache entries for the object will be marked as "uncorked" in the metadata cache. Automatic flushing will resume on the object's entries.
- It will not necessarily result in an immediate flush of the object's entries.
- Calling H5Ouncork() 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 `H5Ouncork()` on an object that has not been corked is considered an error. This will return a negative error code.
- If the cache has been globally corked (either via `H5Pset_mdc_config()` or if `H5Fcork()`), then `H5Ouncork()` can be used to selectively uncork items.

3.4 H5Ois_corked Semantics

`H5Ois_corked(object_id, /*OUT*/ hbool_t *is_corked)` will emit the cork status of the object in the parameter is_corked: TRUE when an object is corked and FALSE when it is not. It will return a negative value on errors and a non-negative value on success.

3.5 H5Fcork Semantics

When `H5Fcork(file_id)` is called on a file identifier:

- A global "corked" flag will be set in the file's metadata cache.
- Cache entries for all entries in the metadata cache will be marked as "corked".
- No dirty or new entries for corked objects 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 been selectively uncorked with `H5Ouncork()`.
- Clean entries for corked objects can still be evicted from the cache.
- Flushing of dirty or new entries for corked objects to storage must be performed manually by the user with the `H5Oflush()` or `H5Fflush()` call.
- Individual objects can be explicitly uncorked using the `H5Ouncork()` function.
- When a corked object is closed, all its cache entries will be uncorked as part of the closing process.
- When a file using a corked cache is closed, all objects in the cache WILL be uncorked as part of the closing process.
- Calling `H5Fcork()` on an identifier that is not an HDF5 file identifier is considered an error. This will return a negative error code.

Like the `H5Ocork()` 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.

3.6 H5Funcork Semantics

When `H5Funcork(file_id)` is called on a file identifier:

- The global "corked" flag in the metadata cache will be unset.
- All entries in the metadata cache will be marked as "uncorked".
- Automatic flushing will resume on all entries in the cache.
- It will NOT necessarily result in an immediate flush of any entries in the cache.
- Calling H5Funcork() on an identifier that is not a file identifier is considered an error. This will return a negative error code.
- Calling `H5Funcork()` on a file identifier that does not have a corked cache is considered an error. This will return a negative error code.

3.7 H5Fis_corked Semantics

`H5Fis_corked(file_id, /*OUT*/ hbool_t *is_corked)` will emit the cork status of the file in the parameter is_corked: TRUE when the metadata cache for that file is corked and FALSE when it is not. It will return a negative value if `object_id` is not a valid file identifier and a non-negative value on success.

This function operates by inspecting the global cache flag set by `H5Fcork()`. Manually corking all objects in the metadata cache with `H5Ocork()` will NOT cause this function to return TRUE.

3.8 H5F_get_corked_object_list Semantics

`H5Fget_corked_object_list(hid_t file_id, /*OUT*/ int *n_objects, /*OUT*/ hid_t object_ids[])` returns an array of object identifiers that are currently corked 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))`.

3.9 Interaction with H5Pset_mdc_config

`H5Pset_mdc_config()` can also be used to cork the metadata cache, only less dynamically via the file access property list used to open or create the file. Setting `evictions_enabled` to TRUE has the same effect as calling `H5Fcork()` on the file.

4 Testing

The new functionality will be tested at two levels:

4.1 Cache Operations (test/cache.c)

The low-level cache operations of corking and uncorking 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 corking system are functioning correctly. An example might be ensuring that all cache entries for an object are listed as corked when the internal metadata cache cork function is called.

4.2 API Calls (test/cork.c – NEW)[7]

Testing of the H5Ocork/uncork API calls will take place in a new test in test/cork.c. Objects will be created or opened, corked, manipulated and then tested (via private HDF5 API calls) to see if they remain corked 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.

5 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 H5Ocork and H5Ouncork.
 *
 * 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 "cork_test.h5"
#define DSETNAME "test"
#define NELEMENTS 1048576
#define CHUNKSIZE 128
```

[7] The tests will be implemented by February 28, 2014

```
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);

    /* cork the dataset */
    H5Ocork(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);
```

```
    /* uncork the dataset */
    H5Oflush(did);
    H5Ouncork(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.

[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: H5Ocork Reference Manual Page

Name: H5Ocork

Signature:

`herr_t H5Ocork(hid_t object_id)`

Purpose:

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

Description:

The `H5Ocork/uncork/flush()` and `H5Fcork/uncork/flush()` 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. Corking 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 `H5Fis_corked()` instead.

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

Objects can be uncorked with `H5Ouncork()` or `H5Funcork()`.

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

Corking an object that is already corked will return an error.

Parameters:

`hid_t` `object_id`	IN: ID of object to be corked in the cache. (See the above notes for restrictions)

Returns:

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

Appendix: H5Ouncork Reference Manual Page

Name: H5Ouncork

Signature:

```
herr_t H5Ouncork(hid_t object_id)
```

Purpose:

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

Description:

The H5Ocork/uncork/flush() and H5Fcork/uncork/flush() 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. Corking 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/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 H5Fis_corked() instead.

Uncorking an object that is not corked is considered an error. The corked/uncorked state of an objected can be determined with H5Ois_corked().

Individual objects can be uncorked with this function after H5Fcork() has been used to globally cork the cache.

An object will be uncorked when it is closed.

All objects will be uncorked when the file is closed.

An object's entries will not necessarily be flushed as a part of the uncork process.

Parameters:

hid_t object_id	IN: ID of object to be uncorked in the cache.
	(See the above notes for restrictions)

Returns:

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

Appendix: H5Ois_corked Reference Manual Page

Name: H5Ois_corked

Signature:

```
herr_t H5Ois_corked(hid_t object_id, /*OUT*/ hbool_t *is_corked)
```

Purpose:

Determines if the entries for an HDF5 object (dataset, group, named datatype) have been corked in the metadata cache.

Description:

The H5Ocork/uncork/flush() and H5Fcork/uncork/flush() 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. Corking 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/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 H5Fis_corked() instead.

Parameters:

hid_t object_id	IN: ID of an object in the cache.
	(See the above notes for restrictions)
hbool_t *is_corked	OUT: Corked status.

Returns:

is_corked will be set to TRUE if an object is corked, FALSE if it is not.

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

Appendix: H5Fcork Reference Manual Page

Name: H5Fcork

Signature:

```
herr_t H5Fcork(hid_t file_id)
```

Purpose:

Corks a file's metadata cache, preventing dirty metadata entries from being flushed from the cache to storage.

Description:

The H5Ocork/uncork/flush() and H5Fcork/uncork/flush() 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. Corking 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/H5Oflush() calls.

Note:

Only HDF5 file identifiers (obtained from H5Fopen() or H5Fcreate()) may be passed to this function. To cork individual HDF5 objects, use H5Ocork() 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.

Corking only pertains to new or dirty metadata entries. Clean entries can 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: H5Funcork Reference Manual Page

Name: H5Funcork

Signature:

```
herr_t H5Funcork(hid_t file_id)
```

Purpose:

Uncorks a file's metadata cache, returning it to the standard eviction and flushing algorithm.

Description:

The H5Ocork/uncork/flush() and H5Fcork/uncork/flush() 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. Corking 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/H5Oflush() calls.

Note:

Only HDF5 file identifiers (obtained from H5Fopen() or H5Fcreate()) may be passed to this function. To uncork individual HDF5 objects, use H5Ouncork() instead.

Passing in a hid_t identifier that represents any other HDF5 entity is considered an error.

A file will be uncorked when closed.

A file's cache entries will not necessarily be flushed as a part of the uncork process.

Parameters:

hid_t file_id	IN: An HDF5 file identifier.

Returns:

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

Appendix: H5Fis_corked Reference Manual Page

Name: H5Fis_corked

Signature:

```
htri_t H5Fis_corked(hid_t file_id, /*OUT*/ hbool_t *is_corked)
```

Purpose:

Determines if a file's metadata cache has been globally corked.

Description:

The H5Ocork/uncork/flush() and H5Fcork/uncork/flush() 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. Corking 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/H5Oflush() calls.

Note:

Only HDF5 file identifiers (obtained from H5Fopen() or H5Fcreate()) may be passed to this function. To determine the corked state of individual HDF5 object identifiers, use H5Ois_corked() 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 *is_corked	OUT: Corked status.

Returns:

is_corked will be set to TRUE if the file's metadata cache is globally corked, FALSE if it is not.

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

Appendix: H5Fget_corked_object_list Reference Manual Page

Name: H5Fget_corked_object_list

Signature:

```
herr_t      H5Fget_corked_object_list(hid_t file_id,
                /*OUT*/ int *n_objects,
                /*OUT*/ hid_t object_ids[])
```

Purpose:

Returns a list of all corked object identifiers in an open file's metadata cache.

Description:

The H5Ocork/uncork/flush() and H5Fcork/uncork/flush() 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. Corking 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/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 corked object IDs (allocated by caller)

Returns:

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