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 objects 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.

The current corking scheme in the HDF5 library is not very dynamic, only allowing the entire metadata cache to be corked a part of opening or creating a file. In some cases it would be useful to allow an application to have more dynamic, fine-grained control over the corking and flushing 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 and implementation. 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. Eviction control of this cache by the user is limited, primarily via the H5Pset_mdc_cache() API call that can be used to modify the file access property list used to open or create a file. In some cases, however, users may desire more fine-grained control over when metadata for an object is flushed from the cache. This extra level of control would allow a programmer to restrict expensive I/O-intensive flushes to periods of relative inactivity. 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 file objects such as groups and datasets.

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 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 file object reads and writes occur via the cache. The cache cannot be disabled; the logical library code never reads metadata directly from the disk. 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).

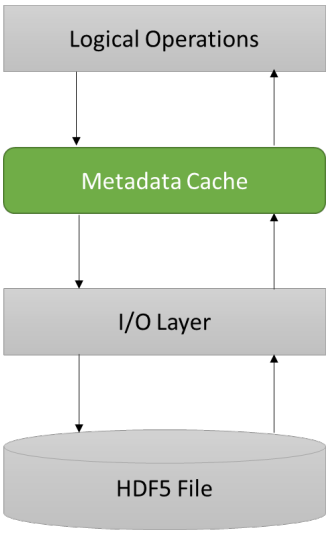

Figure 2-1: Position of the metadata cache in the HDF5 library.

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 disk 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 objects are evicted and, if dirty, flushed using a modified least recently used (LRU) algorithm. It is very important to understand that the HDF5 library and thus the cache are not asynchronous in any way. 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. 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 eviction control, discussed below in the corking section.

2.3 Corking

A cache or individual object in the cache is considered *corked* when evictions and flushes are prevented from occurring via the usual eviction algorithm passes. Instead, the programmer must manually flush objects using the `H5Fflush()` or `H5Oflush()` calls. The metadata cache can be

[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).

corked by calling H5Pset_mdc_config() on the file access property list with the appropriate flags set.

2.4 A Note on Flushing Datasets

The metadata cache (obviously) only manages metadata, and not raw data. In the case of chunked datasets, a separate, per-dataset cache (the *chunk cache*) manages the raw data. These two caches do not normally communicate. The implication of this is that a call to H5Fflush() or H5Oflush() will not result in the raw data being flushed to disk. The exception to this is the SWMR case. Under SWMR semantics, stubs that link to the raw data in the chunk cache are stored in the metadata cache. These stubs allow H5Fflush() or H5Oflush() calls to also flush raw data.

An option for future work would be to use the chunk proxies at all times so that H5Fflush() or H5Oflush() calls would also flush raw data chunks under non-SWMR conditions, but that is not in scope at this time.

3 New Functions

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 later in this section.

The first set of functions allows corking and uncorking of individual persistent 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)
htri_t      H5Ois_corked(hid_t object_id)
```

where object_id is a persistent 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.

```
herr_t      H5Fcork(hid_t file_id)
herr_t      H5Funcork(hid_t file_id)
htri_t      H5Fis_corked(hid_t file_id)
```

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 persistent object identifiers returned by the function.

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

3.1 Persistent Objects

As mentioned in the introduction, the `H5Ocork/uncork/is_corked` functions will be designed for use with HDF5 objects that are persisted to storage. Hence, they will not work with all classes of `hid_t` identifiers.

3.1.1 Valid persistent objects

-
- **Datasets** (`hid_t` returned from `H5Dopen/create`)
- **Groups** (`hid_t` returned from `H5Gopen/create`)
- **Attributes** (`hid_t` obtained via `H5Aopen/create`)
- **Datatypes** (`hid_t` obtained from `H5T*` functions)
- **Objects** (hid_t returned from `H5Oopen`)

3.1.2 INVALID objects

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

  `H5Fcork/uncork/is_corked` are used with file identifiers instead.
- **Dataspaces** (`hid_t` obtained from `H5S*` functions or `H5Dget_space`)

  These are not stored on disk.
- **Property Lists** (`hid_t` obtained via `H5P*` functions)

  These are not stored on disk.

3.2 H5Ocork Semantics

`H5Ocork(object_id)` is used to cork specific persistent objects in the metadata cache, preventing them from being flushed to storage. When it is called on a persistent object identifier:

- The object will be marked as "corked" in the metadata cache.
- No components of the object will be evicted or flushed to storage by the cache's LRU policy.

- Flushing/eviction must be performed manually by the user with the `H5Oflush()`[4] or `H5Fflush()` call.
- An object will remain corked until explicitly uncorked using the `H5Ouncork()` function, except as described below.
- When a corked object is closed, it will be uncorked as part of the closing process.
- Calling `H5Ocork()` on an identifier that does not refer to a persistent 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.

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

`H5Ouncork(object_id)` is used to uncork specific persistent objects in the metadata cache, allowing the cache's normal LRU algorithm to govern their flushing from the cache to storage. When it is called on a persistent object identifier:

- The object will be marked as "uncorked" in the metadata cache.
- Automatic flushing will resume on the object.
- It will NOT result in an immediate flush of the object.
- Calling H5Ouncork() on an identifier that does not refer to a persistent object (e.g., a property list identifier or file identifier) is considered an error. This will return a negative error code.
- Calling `H5Ouncork()` on a persistent 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)` will return TRUE when an object is corked and FALSE when it is not. It will return a negative value if `object_id` is not a valid persistent object.

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[6].
- All objects in the metadata cache will be marked as "corked".
- All objects added to the metadata cache will automatically be marked as "corked".

---

[4] `H5Oflush()` is a new function that will appear in HDF5 1.10.0.
[6] Recall that each open file has its own metadata cache.

- No corked objects will be evicted or flushed to storage by the cache's LRU policy. This does not turn off the LRU algorithm, which can still flush objects that have been selectively uncorked with H5Ouncork().
- Flushing/eviction 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 in the corked cache is closed, it will NOT be uncorked as part of the closing process.
- When a file using a corked cache is closed, the cache and all objects in it WILL be uncorked as part of the closing process.
- Calling H5Fcork() on an identifier that does not refer to a 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 objects in the metadata cache will be marked as "uncorked".
- Automatic flushing will resume on all objects in the cache.
- It will NOT result in an immediate flush of any objects 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 has not been corked is considered an error. This will return a negative error code.

3.7 H5Fis_corked Semantics

H5Fis_corked(file_id) will return 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.

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 H5O_get_corked_object_list Semantics

H5Oget_corked_object_list(*hid_t* file_id, /*OUT*/ *int* *n_objects, /*OUT*/ *hid_t* *object_ids[]) returns an array of persistent object identifiers that are currently corked

as well as the number of elements in the array. The array of object identifiers must be freed by the caller.

Alternatively, this function could use the scheme where the caller passes in a buffer of appropriate size (determined by calling the function with a NULL pointer for the array), although this introduces potential concurrency issues if we intend to ever introduce an internally threaded library. An `H5free_memory()` function will be added to the library in HDF5 1.8.13 (JIRA issue HDFFV-8551).

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 Flushing Corked Objects

The flushing behavior of a corked object follows a single important principle:

> *While an object is corked, flushing, and thus the appearance of an object in the file, is entirely at the programmer's discretion.*

The normal metadata cache operations will *never* flush a corked object, even if the system runs out of memory. A corked object must be flushed by the application with either `H5Fflush()`, which will flush the entire file's cache, or `H5Oflush()`, which will flush a particular object.

When SWMR is enabled, flushing semantics are modified to handle flush dependencies. This can be boiled down to two rules:

1. *If a flush dependency child is corked, any parents will not be flushed by normal cache operations.*

   Flushing the child in this case would be out of the programmer's control, which would violate the corked flushing principle.

   Consider a group containing a corked dataset. If the cache wanted to flush and evict the group to make space, it would have to also flush the dataset in case the object header had moved. This would violate programmer control over the appearance of the dataset, so the flush would not occur.

2. *If a flush dependency child is corked, and the parent is manually flushed by the user, the child will be flushed.*

   In this case, we assume that the programmer is aware of the parent/child relationship of the group and dataset, making a dataset flush implicit. Since the flush has been initiated by the programmer, this does not violate the corked flushing principle and would be allowed.

5 Testing

The new functionality will be tested at two levels:

5.1 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 create specific data structures, cork them, manipulate the structures and/or the cache, and ensure that all components are flagged as corked and that they are not flushed to disk.

As an example, these tests would ensure that a corked B-tree would have all its nodes corked.

5.2 API Calls (test/cork.c – NEW)

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 the file.

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)
- Attribute (small)
- Attribute (large)
- Datatype

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.

6 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 will only be flushed after a chunk has been filled.
```

```
 */

#define FILENAME "cork_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);

    /* 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 the Diamond Light Source.

Revision History

December 11, 2013:	Version 1 circulated for comment to HDF5 SWMR team.
January 7, 2014:	Version 2 circulated for comment to HDF5 SWMR team.

[Glossary, Terminology]

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

persistent object An HDF5 object that is persisted to storage. Includes datasets, groups, attributes, and stored data types.

transient object An HDF5 object that is not persisted to storage. Includes datasets and property lists.

user metadata Attributes created by the user that are attached to datasets, groups, or stored data types.

Appendix: H5Ocork Reference Manual Page

Name: H5Ocork

Signature:

```
herr_t H5Ocork(hid_t object_id)
```

Purpose:

Prevents a persistent HDF5 object from being flushed from the metadata cache to storage.

Description:

This function is used in cases where a programmer would like to control when particular persistent HDF5 objects are flushed from the file's metadata cache. A corked object will never be flushed or evicted from the metadata cache. Instead, the programmer must manually perform flushes with `H5Fflush()` or `H5Oflush()`.

Note:

HDF5 persistent objects include datasets, attributes, stored datatypes, and groups. Only *`hid_t`* identifiers that represent these objects can be passed to the function.

This function does not apply to *`hid_t`* identifiers that represent property lists or dataspaces since those are not stored in the file. Attempting to cork either of these is considered an error.

It is an error to pass an HDF5 file identifier (obtained from `H5Fopen()` or `H5Fcreate()`) to this function. Use `H5Fcork()` instead.

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

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

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 a corked persistent HDF5 object to the default metadata flush and eviction algorithm.

Description:

This function is used in cases where a programmer would like to control when particular persistent HDF5 objects are flushed from the file's metadata cache. A corked cache or object will never be flushed or evicted from the metadata cache. Instead, the programmer must manually perform flushes with `H5Fflush()` or `H5Oflush()`.

Note:

HDF5 persistent objects include datasets, attributes, stored datatypes, and groups. Only *`hid_t`* identifiers that represent these objects can be passed to the function.

This function does not apply to *`hid_t`* identifiers that represent property lists or dataspaces since those are not stored in the file. Attempting to cork either of these is considered an error.

It is an error to pass an HDF5 file identifier (obtained from `H5Fopen()` or `H5Fcreate()`) to this function. Use `H5Funcork()` 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 the cache has been globally corked with `H5Fcork()`.

An object will be uncorked when closed by the user.

All objects will be uncorked when the file is closed.

An object 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:

```
htri_t H5Ois_corked(hid_t object_id)
```

Purpose:

Determines if a persistent HDF5 object has been corked in the metadata cache.

Description:

The H5Ocork() and H5Ouncork() functions can be used to control the flushing of a persistent HDF5 object such as a dataset from the metadata cache. This function reports whether a particular object has been corked.

Note:

HDF5 persistent objects include datasets, attributes, stored datatypes, and groups. Only *hid_t* identifiers that represent these objects can be passed to the function.

This function does not apply to *hid_t* identifiers that represent property lists or dataspaces since those are not stored in the file. Attempting to cork either of these 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)

Returns:

Returns TRUE if an object tis corked, returns FALSE if it is not. Returns 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 all metadata from being evicted or flushed from the to storage.

Description:

This function is used in cases where a programmer would like to control when metadata is flushed from the file's metadata cache. Metadata in a corked cache will never be flushed or evicted from the metadata cache. Instead, the programmer must manually perform flushes with `H5Fflush()` or `H5Oflush()`.

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 object is considered an error.

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

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:

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

Description:

This function is used in cases where a programmer would like to control when metadata is flushed from the file's metadata cache. Metadata in a corked cache will never be flushed or evicted from the metadata cache. Instead, the programmer must manually perform flushes with `H5Fflush()` or `H5Oflush()`.

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 object is considered an error.

A file will be uncorked when closed.

A file 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)
```

Purpose:

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

Description:

The H5Fcork(), H5Funcork(), and H5Pset_mdc_config() functions can be used to control the flushing behavior of persistent HDF5 objects from a file's metadata cache. This function reports whether a file's metadata cache has been globally corked and must be manually flushed by the user.

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 object is considered an error.

Parameters:

hid_t file_id	IN: An HDF5 file identifier.

Returns:

Returns TRUE if the file's metadata cache is globally corked, returns FALSE if it is not. Returns a negative value on errors.

Appendix: H5Oget_corked_object_list Reference Manual Page

Name: H5Oget_corked_object_list

Signature:

```
herr_t      H5Oget_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() and H5Fcork/uncork() functions can be used to control the flushing of persistent HDF5 file objects from the metadata cache. This function returns a list of all corked objects in a particular file's cache to the user.

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 object is considered an error.

This function does not apply to hid_t identifiers that represent property lists or dataspaces since those are not stored in the file. Attempting to cork either of these is considered an error.

The object_ids pointer will be NULL when the number of corked objects is zero.

The array returned from this function must be freed by the caller.

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 identifiers

Returns:

Returns a non-negative value if successful. On errors, a negative value will be returned and object_ids will be set to NULL.