HDF5 DAOS VOL Connector User's Guide

Neil Fortner, Jordan Henderson, Jerome Soumagne

This document aims to be a helpful guide on how to use the HDF5 DAOS VOL connector to leverage the capabilities of the DAOS object storage system within an HDF5 application.

Revision History

Version Number	Date	Comments
v0.1	Mar. 29, 2019	First draft.
v0.2	Apr. 24, 2019	Second draft.
v0.3	Aug. 27, 2019	Third draft.
v0.4	Oct. 25, 2019	Fourth draft.
v0.5	Nov. 1, 2019	Fifth draft.
v0.6	Dec. 2, 2019	Sixth draft.
v1.0	Dec. 16, 2019	Initial release of DAOS VOL connector.

Contents

List of Figures

1. Overview

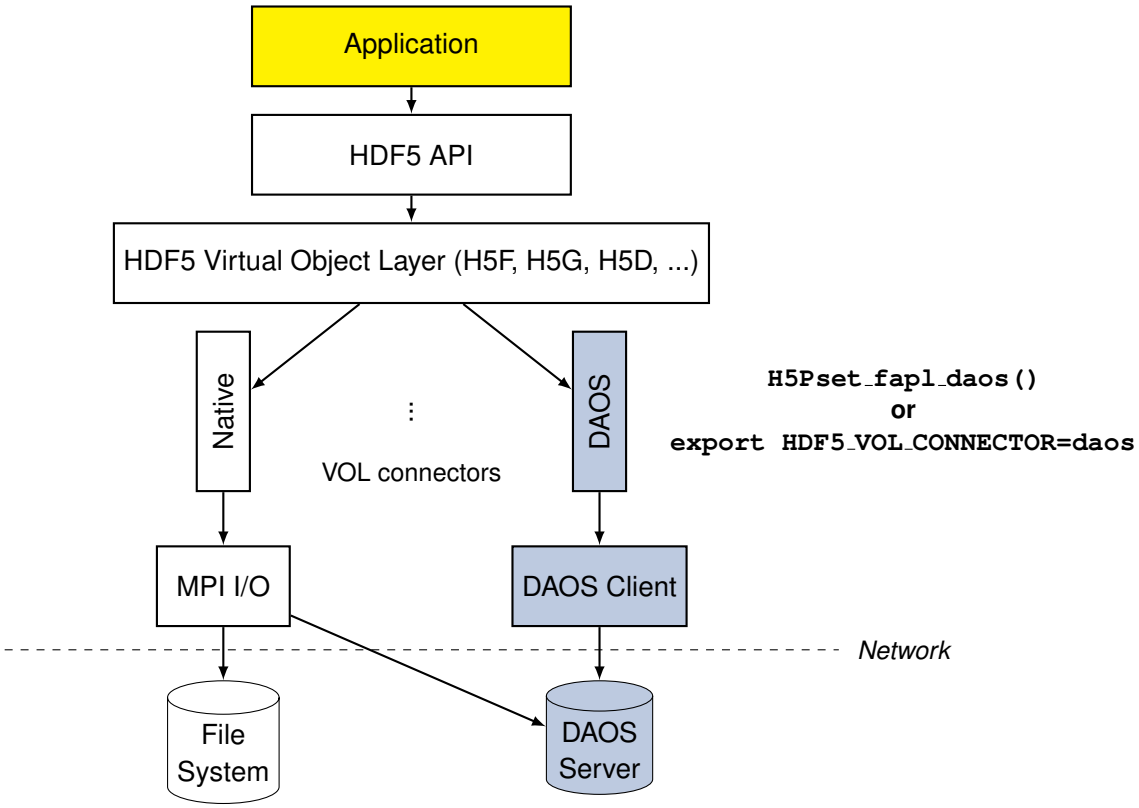

Figure 1 – DAOS within Virtual Object Layer. All of the HDF5 I/O related calls are routed to the DAOS VOL connector.

2. Using the DAOS VOL connector within an HDF5 application

This section outlines the unique aspects of writing, building and running HDF5 applications with the DAOS VOL connector.

2.1. Building the HDF5 DAOS VOL connector

The following is a quick set of instructions for building the DAOS VOL connector connector. Note that these instructions are not comprehensive and may be subject to change in future releases; please refer to the DAOS VOL connector's README file for the most up to date instructions.

The DAOS VOL connector is built using CMake. CMake version 2.8.12.2 or greater is required to build the connector itself, but version 3.1 or greater is required to build the connector's tests. To build the connector, one should create a build directory within the source tree:

```
cd daos-vol
mkdir build
cd build
```

After that, if all of the required components (DAOS, CaRT, MPI and HDF5) are located within the system path, building the connector should be as simple as running the following two commands to have CMake generate the build files for `make` to use.

```
ccmake ..
make && make install
```

Some notable CMake variables are listed below. These can be used to control the build process and can be supplied to the `cmake` command by prepending them with `-D` or turned on in `ccmake`. Some of the connector-specific options may be needed if the required components mentioned previously cannot be found within the system path.

CMake-specific options:

- `CMAKE_INSTALL_PREFIX` — This variable controls the install directory that the resulting output files are written to.
- `CMAKE_BUILD_TYPE` — This variable controls the type of build used for the VOL connector. Valid values are Release, Debug, RelWithDebInfo and MinSizeRel. (*Default: RelWithDebInfo*)

DAOS VOL connector-specific options:

- `BUILD_TESTING` — This variable is used to enable/disable building of the DAOS VOL connector's tests.
- `BUILD_EXAMPLES` — This variable is used to enable/disable building of the DAOS VOL connector's HDF5 examples.
- `CART_INCLUDE_DIR` — This variable controls the CaRT include directory used by the VOL connector build process. Used in conjunction with the `CART_LIBRARY` variable.

- CART_LIBRARY — This variable controls the CaRT library used by the VOL connector build process. It should be set to the full path to the CaRT library, including the library's name (e.g., /path/libcart.so). Used in conjunction with the CART_INCLUDE_DIR variable.
- DAOS_LIBRARY — This variable controls the DAOS library used by the VOL connector build process. It should be set to the full path to the DAOS library, including the library's name (e.g., /path/libdaos.so). Used in conjunction with the DAOS_COMMON_LIBRARY and DAOS_INCLUDE_DIR variables.
- DAOS_COMMON_LIBRARY — This variable controls the DAOS 'common' library used by the VOL connector build process. It should be set to the full path to the DAOS common library, including the library's name (e.g., /path/libdaos_common.so). Used in conjunction with the DAOS_LIBRARY and DAOS_INCLUDE_DIR variables.
- DAOS_INCLUDE_DIR — This variable controls the DAOS include directory used by the VOL connector build process. Used in conjunction with the DAOS_LIBRARY and DAOS_COMMON_LIBRARY variables.
- MPI_C_COMPILER — This variable controls the MPI C Compiler used by the VOL connector build process. It should be set to the full path to the MPI C Compiler, including the name of the executable.
- HDF5_C_COMPILER_EXECUTABLE — This variable controls the HDF5 compiler wrapper script used by the VOL connector build process. It should be set to the full path to the HDF5 compiler wrapper, including the name of the wrapper script. The following two variables may also need to be set.
- HDF5_C_LIBRARY_hdf5 — This variable controls the HDF5 library used by the VOL connector build process. It should be set to the full path to the HDF5 library, including the library's name (e.g., /path/libhdf5.so). Used in conjunction with the HDF5_C_INCLUDE_DIR variable.
- HDF5_C_INCLUDE_DIR — This variable controls the HDF5 include directory used by the VOL connector build process. Used in conjunction with the HDF5_C_LIBRARY_hdf5 variable.

2.2. Using the HDF5 DAOS VOL connector with applications

There are currently three ways to tell an HDF5 application to use the DAOS VOL connector:

1. The HDF5_VOL_CONNECTOR environment variable
2. H5Pset_vol(), an HDF5 API call
3. H5Pset_fapl_daos(), a DAOS VOL connector API call

Which option you choose will depend on things like your ability to modify the application's source code, whether dynamically loading plugins is an option, etc.

The environment variable is useful when the application either does not need to be modified to use DAOS storage (e.g., the application makes no native-specific HDF5 API calls). It is also the easiest to use as it requires very little setup and no changes to the application's source code. It is probably not going to be suitable, however, if the application needs to perform I/O with multiple connectors. For example, reading from DAOS and writing to the native VOL connector or a RESTful web service via the REST VOL connector would be a problem. In the HDF5 command-line tools, this difficulty is solved by using command-line parameters and explicitly specifying a connector via H5Pset_vol().

H5Pset_vol() is more complicated to use as it requires modification of the application's source code, which may not be possible. It's more flexible, though, and allows an application to be configured to use VOL

connectors in arbitrary ways. Since it's an HDF5 API call, you don't need to link to the DAOS VOL connector, which can be loaded as a plugin in the same way as using the environment variable, described above.

`H5Pset_fapl_daos()` will give you more control over how DAOS is initialized, but it requires linking to the DAOS VOL connector and modification of the application's source code.

Each of these ways of specifying and loading the DAOS VOL connector is fundamentally equivalent in terms of its eventual effects. There's no performance or functionality changes between them.

2.2.1. Locating and loading HDF5 plugins at runtime

HDF5 are discovered at runtime by searching in specified plugin paths. There is a library default plugin path (usually `/usr/local/hdf5/lib/plugin` on POSIX systems) but this can be overridden using the `HDF5_PLUGIN_PATH` environment variable (and/or manipulated via the various HDF5 `H5PL` API calls for more complicated situations where the application's source code can be modified).

When the HDF5 library is asked to use a plugin it doesn't currently have loaded, it searches the plugin path(s), attempting to load any plugins it finds to see if their name or connector value match.

2.2.2. Using the HDF5_VOL_CONNECTOR environment variable

Set the `HDF5_VOL_CONNECTOR` environment variable to "daos" (case-sensitive) and make sure the DAOS VOL connector plugin is discoverable, as described above.

2.2.3. Using H5Pset_vol()

The first thing you will have to do is load the DAOS VOL connector plugin via `H5VL_register_connector_by_(value|name)()`. This will load the plugin and assign it a `hid_t` connector ID. You then pass this ID as the VOL connector ID when you call `H5Pset_vol()` on the file access property list that will be used to open the file.

The DAOS VOL connector value is currently 4004. This is documented in the connector's public header. Note that the connector value is unrelated to the `hid_t` library ID obtained from the registration API call. They are completely unrelated.

2.2.4. Using H5Pset_daos_vol()

If dynamic loading of the DAOS VOL connector is not used, any HDF5 application using the connector must:

1. Include `daos_vol_public.h`, found in the `include` directory of the DAOS VOL connector installation directory.
2. Link against `libhdf5_vol_daos.so` (or similar), found in the `lib` directory of the DAOS VOL connector installation directory, and against `libuuid.so` (or similar) in order to use UUIDs. Note that dependencies can alternatively be retrieved through CMake or pkg-config.

An HDF5 DAOS VOL connector application also requires in that particular case three new function calls in addition to those for an equivalent HDF5 application (see Appendix A for more details):

- `H5daos_init()` — Initializes the DAOS VOL connector

  Called upon application startup, before any file is accessed.

- `H5Pset_fapl_daos()` — Sets DAOS VOL connector access on File Access Property List.

  Called to prepare a FAPL to open a file through the DAOS VOL connector. See HDF5 File Access Property Lists for more information about File Access Property Lists.

- `H5daos_term()` — Cleanly shutdowns the DAOS VOL connector

  Called on application shutdown, after all files have been closed.

2.2.5. Skeleton Example

Below is a no-op application that opens and closes a file using the DAOS VOL connector. For clarity, no error-checking is performed. Note that this example is meant only for the case when the DAOS VOL connector is not being dynamically loaded.

```
#include "hdf5.h"
#include "daos_vol_public.h"

int main(void)
{
    uuid_t pool_uuid;
    hid_t fapl_id, file_id;

    /* Parse the pool UUID. */
    uuid_parse("fce30f79-b34b-46c1-9b1f-bb52d99dacca", pool_uuid);

    /* Initialize DAOS VOL connector using the above parsed UUID for
     * the pool UUID, "daos_server" as the group name for the DAOS
     * servers managing the pool and simply rank 0 as the only rank
     * in the pool service list. */
    H5daos_init(pool_uuid, "daos_server", "0");

    fapl_id = H5Pcreate(H5P_FILE_ACCESS);
    H5Pset_fapl_daos(fapl_id, MPI_COMM_WORLD, MPI_INFO_NULL);

    /* Currently required for the DAOS VOL connector, set all metadata
     * operations to be collective */
    H5Pset_all_coll_metadata_ops(fapl_id, true);

    file_id = H5Fopen("my_file.h5", H5F_ACC_RDWR, fapl_id);

    /* Operate on file */
    [...]

    H5Pclose(fapl_id);
    H5Fclose(file_id);

    /* Terminate the DAOS VOL connector. */
    H5daos_term();
```

```
    return 0;
}
```

2.3. Asynchronous I/O

The DAOS VOL connector supports asynchronous HDF5 operations using the HDF5 event set (H5ES) API, released in HDF5 1.13.0. This allows I/O to proceed in the background while the application is performing other tasks.

2.3.1. Implementation and making progress

Asynchronous I/O in the DAOS VOL connector is implemented using a threadless progress engine, that checks for completion of in-flight operations any time it is entered. This means there is no background thread making progress, so you can be certain it won't interfere with computation. However, this also means that the connector needs to be entered occasionally in order to make progress, by calling `H5ESwait()` with a timeout of `0`. Otherwise, asynchronous operations will never complete until waited on, and the wait may take a long time, no matter how much time has passed since the asynchronous operation was issued.

Each HDF5 operation is split up into multiple internal tasks which may need to be executed in a certain order. For example,s a group create needs to create a link in the parent group, write the metadata to the new group, and possibly adjust the number of links in the parent group and allocate an object ID for the new group. The DAOS VOL connector uses the DAOS Thread Scheduling Engine (tse) to track dependencies between these internal tasks and make progress on them, together with custom code written to make progress on asynchronous MPI operations needed for collective parallel operations.

2.3.2. Consistency semantics

Similarly to other asynchronous I/O libraries, the application must be careful not to use, modify, or free any buffers in use by async tasks until those tasks are complete. This applies to all read and query operations, as well as raw data and attribute write operations. For non-attribute metadata write operations, the connector will make a temporary copy of any buffers passed in.

The application must also be careful not to assume write operations are visible in the file until it has verified that the operation has completed through the H5ES interface. For example, if you write to a dataset, you must wait for the write to complete before reading that data from the dataset if you wish to see the new data. Likewise, if you create a link, you must wait for the create to complete before reading that link.

It is possible in some cases, however, to issue operations before prerequisites have been complete. Any ID returned from the API can be passed back in through the HDF5 API even if the open operation for the object that that ID refers to has not completed. This allows applications to, for example, create a file, create a group in the file, create a dataset in the group, write to the dataset, and close all IDs, all in a non-blocking manner without waiting (until the dataset write buffer needs to be modified or freed). Until the open operation is complete, you can only access the object through its ID, so you cannot, for example, create a group then create an object inside that group using a path that includes the group name, without waiting for the first create to complete (or using `H5Oflush_async()` or `H5Fflush_async()`, see below).

As an example, this series of calls is legal:

```
group_id = H5Gcreate_async(file_id, "parent", ..., es_id);
dset_id = H5Dcreate_async(group_id, "dset", ..., es_id);
H5Dwrite_async(dset_id, ..., es_id);
...
H5ESwait(es_id);
```

Note that `H5Dcreate_async()` is called before we know `H5Gcreate_async()` has completed, as with `H5Dwrite_async()` and `H5Dcreate_async()`. However, this example is illegal and may result in errors:

```
group_id = H5Gcreate_async(file_id, "parent", ..., es_id);
dset_id = H5Dcreate_async(file_id, "parent/dset", ..., es_id); //may fail
```

The only difference between the traditional blocking HDF5 API calls and the async versions is that, in the blocking versions, the connector waits for that operation to complete before returning. Therefore, when mixing synchronous and asynchronous calls, the application must still obey the above rules even when calling a synchronous function - previously called asynchronous functions may not be complete.

As an example, this series of calls is illegal and may result in an error:

```
group_id = H5Gcreate_async(file_id, "parent", ..., es_id);
dset_id = H5Dcreate(file_id, "parent/dset", ...); //may fail
```

2.3.3. Operation ordering

For the most part, all asynchronous operations execute concurrently without any ordering enforced betwen them. However, there are a few exceptions. The connector enforces ordering between object open operations and operations that use that object, in order to facilitate the feature described above that allows the use of incompletely opened object IDs. In addition, `H5Dset_extent(_async)()`, link/object creates when the parent group has link creation order tracked, and attribute creates when the parent object has attribute creation order tracked are always strictly ordered, so these operations always execute after any previously issued operation related to their object, and before any subsequently issued operations.

The application can manually enforce ordering using `H5Oflush_async()` and `H5Fflush_async()`. These operations only complete when all previously issued operations for the object or file complete and, like `H5Dset_extent_async()`, all subsequently issued operations only begin after the flush is complete. Since the DAOS HDF5 connector does no caching, the flush operations have no other effect. This allows you to, for example, write to an attribute, issue an `H5Oflush_async()` on the attribute's parent object, and read the attribute back, all in a non-blocking fashion (as long as you keep both read and write buffers around and don't examine the read buffer until it is complete). See the Operation Scope section for information on which operations are in scope for `H5Oflush_async`.

As an example, here is a way to make the previously illegal example legal while remaining fully asynchronous:

```
group_id = H5Gcreate_async(file_id, "parent", ..., es_id);
H5Fflush_async(file_id, H5F_SCOPE_LOCAL, es_id);
dset_id = H5Dcreate_async(file_id, "parent/dset", ..., es_id);
```

A call to `H5Oflush_async(group_id, es_id)` would not be sufficient in this case because the group creation occurs in `file_id`'s scope, because `file_id` is what is passed to `H5Gcreate_async()`. The flush must be made on the parent object.

Close operations such as `H5Gclose()` or `H5Fclose()` will only complete once all previously issued operations on their object, file or attribute complete. Therefore, the non-async versions of these operations will block until these operations are complete. Asynchronous versions of all close operations are available for non-blocking close.

2.3.4. Parallel Considerations

Parallel collective operations add another constraint on asynchronous operations. Because asynchronous MPI operations must be strictly ordered, all collective HDF5 operations are strictly ordered with respect to each other when executed with more than one rank, and no two can execute at the same time. Non-collective operations are not affected by this, and may execute simultaneouly with collective operations.

2.3.5. Operation Scope

In order to handle some operations that need to be ordered with respect to all other operations within a certain scope, we have introduced the concept of an operation pool, which is a container for operations that can be executed simultaneously. All operations exist in an operation pool at attribute, object, file, or global scope. When the application invokes an HDF5 API routine, this creates an operation this is placed in the operation pool of the object passed to the API routine. For example, dataset creates are placed in the object scoped pool of the parent group passed as the first argument to `H5Dcreate()`, attribute creates are placed in the object scoped pool of the parent object passed as the first argument to `H5Acreate()`, and attribute writes are placed in the attribute scoped pool of the attribute. Operations that have multiple parent objects (such as `H5Ocopy()`) are placed in the file pool if the objects are in the same file, and in the global pool if they are in different files.

All operations placed in a pool will be executed after all previously issued operations at a different scope in a location that contains, or is contained in, the pool for the original operation. For example, an operation in an attribute's pool will execute after all previously issued operations in the pools for the attribute's parent object, its file, or the global pool. It is not affected by operations in a different object or file. Operations in the global pool will execute after all previously issued operations not in the global pool.

For the most part, this operation pool scheme is transparent to the application and can be ignored. However, for maximum concurrency, it is worth considering that certain access patterns will result operations being serialized, so only one may execute at a time (though they will remain asynchronous). This will happen if, for example, the app alternates between attribute and object operations.

2.3.6. Asynchronous Example Program

Below is an example of an asynchronous application that writes to a dataset:

```
#include "hdf5.h"
#include "daos_vol_public.h"

int main(void)
```

```
{
    hid_t file_id, group_id, dset_id, es_id;
    ... // Buffers, dataspaces, etc.
    herr_t status;

    es_id = H5EScreate();       // Create event set for tracking async
    ↪  operations

    file_id = H5Fopen_async("file.h5", H5F_ACC_RDWR, H5P_DEFAULT, es_id);
    group_id = H5Gopen_async(file_id, "parent", H5P_DEFAULT, es_id);
                                        // Starts when H5Fopen completes
    dset_id = H5Dopen_async(group_id, "dataset", H5P_DEFAULT, es_id);
                                        // Starts when H5Gopen completes

    status = H5Dwrite_async(dset_id, ..., es_id); // Asynchronous, starts when
                                                  // H5Dopen completes, may run
                                                  // concurrently with other
                                                  // H5Dwrite in event set
    status = H5Dwrite_async(dset_id, ..., es_id); // Asynchronous, starts when
                                                  // H5Dopen completes, may run
                                                  // concurrently with other
                                                  // H5Dwrite in event set

    status = H5Dclose_async(dset_id); // Asynchronous, will complete when both
                                      // writes above complete

    ...
    <other user code>
    ...
    H5ESwait(es_id); // Wait for operations in event set to complete, buffers
                     // used for H5Dwrite must not be changed until after wait
                     // returns
    ...

    status = H5Gclose(group_id); // Blocking call, will return as soon as all
                                 // operations on group_id complete
    status = H5Fclose(file_id);  // Blocking call, will return as soon as all
                                 // operations on file_id complete

    return 0;
}
```

2.4. Building HDF5 DAOS VOL connector applications

Assuming an HDF5 application has been written following the instructions in the previous section, the application should be built as normal for any other HDF5 application. However, if the DAOS VOL connector is not being dynamically loaded, the steps in the following section are required to build the application.

2.4.1. Without the DAOS VOL connector as a dynamically-loaded plugin

To link in the required libraries, the compiler will likely require the additional linker flags:

```
-lhdf5_vol_daos -luuid
```

However, these flags may vary depending on platform, compiler and installation location of the DAOS VOL connector. It is highly recommended that compilation of HDF5 DAOS VOL connector applications be done using either the `h5cc/h5pcc` script included with HDF5 distributions, or CMake, pkg-config, as these will manage linking with the HDF5 library.

If HDF5 was built using autotools, this script will be called `h5pcc` and may be found in the `bin` directory of the HDF5 installation. If HDF5 was built with CMake, this script will simply be called `h5cc` and can be found in the same location. The above notice about additional library linking applies to usage of `h5cc/h5pcc`. For example:

```
h5cc/h5pcc -lhdf5_vol_daos -luuid my_application.c -o my_application
```

2.5. Running HDF5 DAOS VOL connector applications

Running applications that use the DAOS VOL connector connector requires access to a DAOS server. Refer to DAOS Software Installation for more information on the setup process for this. For the DAOS VOL connector to correctly interact with a DAOS server instance, the server must be running and it must be passed the UUID of the DAOS pool to use and a list of DAOS pool service list ranks, as detailed in the following sections.

2.5.1. Starting the DAOS Server

Instructions for starting a DAOS Server can be found in the DAOS Documentation.

2.5.2. With the DAOS VOL connector as a dynamically-loaded plugin

If the DAOS VOL connector is dynamically loaded by HDF5, the DAOS pool UUID and DAOS pool service rank list are passed via the two environment variables below.

```
DAOS_POOL - The UUID of the DAOS pool to use.

DAOS_SVCL - A comma-separated list of server ranks used for
            daos_pool_connect(). Generated from daos_pool_create().
```

2.5.3. Without the connector as a dynamically-loaded plugin

If the DAOS VOL connector is not being dynamically loaded, the DAOS pool UUID and DAOS pool service rank list should be passed via the call to H5daos_init() within the application.

2.5.4. Example Applications

Some of the example C applications which are included with HDF5 distributions have been adapted to work with the DAOS VOL connector and are included under the top-level `examples` directory in the DAOS VOL connector source root directory. The built example applications can be run from the `bin` directory inside the build directory.

In addition to these examples, the `test/vol` directory contains several test files, each containing test functions that are examples of HDF5 applications in miniature, focused on a particular behavior. These mini-application tests cover a moderate amount of HDF5's public API functionality and should be a good indicator of whether the DAOS VOL connector is working correctly in conjunction with a running DAOS API-aware instance. Note that these tests currently rely on HDF5's dynamically-loaded VOL connector capabilities in order to run with the DAOS VOL connector.

3. HDF5 API Support

3.1. Feature Specific Support

The following sections serve to illustrate the DAOS VOL connector's support for features in HDF5, as well as to highlight any differences between the expected behavior of an HDF5 feature versus the actual behavior as implemented by the VOL connector.

3.1.1. Attribute Features

Feature			Supported?	Notes
Dataspace	Dimensionality	H5S_NULL	Yes	
		H5S_SCALAR	Yes	
		SIMPLE	Yes	
Datatype		Atomic	Yes	
		Compound	Yes	
		Variable-length	Yes	
		Array	Yes	
		Opaque	Yes	
		Reference	Yes	
Properties	Name Encoding	ASCII	Yes	
		UTF-8	No	

3.1.2. Dataset Features

Feature			Supported?	Notes
Dataspace	Dimensionality	H5S_NULL	Yes	
		H5S_SCALAR	Yes	
		SIMPLE	Yes	
	Selection Type	NONE	Yes	
		H5S_ALL	Yes	
		Hyperslab Selection	Yes	
		Point Selection	Yes	
Datatype		Atomic	Yes	
		Compound	Yes	
		Variable-length	Yes	
		Array	Yes	
		Opaque	Yes	
		Reference	Yes	

Feature			Supported?	Notes
Properties	Storage Properties (creation)	Compact	No	Setting is ignored; stored as contiguous.
		External	No	Setting is ignored; stored as contiguous.
		Contiguous	Yes	Default storage type.
		Chunked	Yes	
		VDS	No	The VDS feature is not currently planned to be supported.
	Other Properties (creation)	Attribute Creation Order	Yes	In order to work correctly, the attribute creation order feature requires that the dataset is touched collectively or that the application otherwise ensures no concurrent access to the dataset[1].
		Fill Value	Yes	
		Filters	No	HDF5 does not expose any public APIs for working with the filter pipeline; however, this feature may be supported in the future.
		Storage Allocation Time	N/A	

[1] This restriction may be removed in the future.

Feature			Supported?	Notes
Properties (cont.)	Access Properties	Chunk cache	No	HDF5 does not expose any public APIs for implementing a chunk cache for arbitrary VOL connectors; however, this feature may be supported in the future.
		VDS views and printf	No	The VDS feature is not currently planned to be supported.
		MPI-I/O Collective Metadata Ops	Yes	By default, all metadata operations are collective for writes and independent for reads. Also see H5P(set/get)_all_ind_metadata_ops().
	Transfer Properties	MPI-I/O Independent or Collective I/O mode	N/A	

3.1.3. File Features

Feature			Supported?	Notes
File creation flags		H5F_ACC_TRUNC	Yes	The file creation flags behave as for native HDF5.
		H5F_ACC_EXCL	Yes	
File opening flags		H5F_ACC_RDWR	Yes	
		H5F_ACC_RDONLY	Yes	
Properties	Creation Properties	Attribute Creation Order	Yes	In order to work correctly, the attribute creation order feature requires that the dataset is touched collectively or that the application otherwise ensures no concurrent access to the dataset[1]. The rest of the file creation properties are related to the native HDF5-specific file format.
	Access Properties (Drivers)	SEC2 Driver	N/A	These drivers are applicable to native HDF5 only.
		Family Driver	N/A	
		Split Driver	N/A	
		Multi Driver	N/A	
		Core Driver	N/A	
		Log Driver	N/A	
		MPI-I/O	Yes	This property just indicates parallel access to the file; it doesn't use HDF5 MPI I/O driver underneath.

[1] This restriction may be removed in the future.

Feature			Supported?	Notes
Properties (cont.)	Access Properties (Other)	MPI-I/O Collective Metadata Ops	Yes	By default, all metadata operations are collective for writes and independent for reads. Also see H5P(set/get)_all_ind_metadata_ops().
		User block	N/A	
		Chunk Cache	No	HDF5 does not expose any public APIs for implementing a chunk cache for arbitrary VOL connectors; however, this feature may be supported in the future.
		Object flushing callbacks	N/A	
		File closing degree	N/A	
		Evict on close	N/A	
		Sieve buffer size for partial I/O	No	HDF5 does not expose any public APIs for implementing a partial I/O sieve buffer for arbitrary VOL connectors; however, this feature may be supported in the future.
		File Image	N/A	

3.1.4. Group Features

Feature			Supported?	Notes
Properties	Creation Properties	Link Creation Order	Yes	In order to work correctly, the link creation order feature requires that the parent group is touched collectively or that the application otherwise ensures no concurrent access to the group.[2]
		Attribute Creation Order	Yes	In order to work correctly, the attribute creation order feature requires that the dataset is touched collectively or that the application otherwise ensures no concurrent access to the dataset.[1]
		Other Properties	N/A	These properties are related to the native HDF5-specific file format.
	Access Properties	MPI-I/O Collective Metadata Ops	Yes	By default, all metadata operations are collective for writes and independent for reads. Also see H5P(set/get)_all_ind_metadata_ops().

[1] This restriction may be removed in the future.

3.2. API Specific Support

The following sections serve to illustrate the DAOS VOL connector's support for the HDF5 API, as well as to highlight any differences between the expected behavior of an HDF5 API call versus the actual behavior as implemented by the VOL connector. If a particular HDF5 API call does not appear among these tables, it is most likely a native HDF5-specific API call which cannot be implemented by non-native HDF5 VOL connectors. These types of API calls are listed among the tables in Appendix B.

3.2.1. H5A interface

Supported API calls

API call	Notes
H5Acreate(1/2)	
H5Acreate_async	
H5Acreate_by_name	
H5Acreate_by_name_async	
H5Aopen(_by_name/_by_idx)	For H5Aopen_by_idx, `H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Aopen(_by_name)_async	
H5Aopen_by_idx_async	Currently executes in a blocking fashion
H5Aopen_idx	Deprecated in favor of H5A_open_by_idx
H5Aopen_name	Deprecated in favor of H5A_open_by_name
H5Awrite	
H5Awrite_async	
H5Aread	
H5Aread_async	
H5Aclose	Blocks until all in-flight async tasks on the attribute are complete
H5Aclose_async	Completes when all in-flight async tasks on the attribute are complete
H5Aiterate(2)	■ Restarting iteration from an index value is currently unsupported[1] ■ `H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type

[1] Will be supported in the future.

API call	Notes
H5Aiterate_by_name	■ Restarting iteration from an index value is currently unsupported[1] ■ `H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Aexists(_by_name)	
H5Aexists_async	
H5Aexists_by_name_async	Currently untested
H5Arename(_by_name)	
H5Arename(_by_name)_async	Currently executes in a blocking fashion
H5Adelete(_by_name/_by_idx)	For H5Adelete_by_idx, `H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Aget_name(_by_idx)	For H5Aget_name_by_idx, `H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Aget_space	
H5Aget_type	
H5Aget_info(_by_name/_by_idx)	Of the four fields in the `H5A_info_t` struct: ■ `corder_valid` is set to TRUE only if attribute creation order tracking is enabled for the object containing the attribute; it is set to FALSE otherwise ■ `corder` is set appropriately if attribute creation order tracking is enabled for the object containing the attribute; it is set to 0 otherwise ■ `cset` is currently always set to `H5T_CSET_ASCII` ■ `data_size` is set appropriately For H5Aget_info_by_idx, `H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Aget_create_plist	

Currently unsupported API calls

API call	Notes
H5Aget_storage_size	`H5Aget_storage_size` is not currently planned to be supported

3.2.2. H5D interface

Supported API calls

API call	Notes
H5Dcreate(1/2)	
H5Dcreate_async	
H5Dcreate_anon	
H5Dopen(1/2)	
H5Dopen_async	
H5Dwrite	
H5Dwrite_async	
H5Dread	
H5Dread_async	
H5Dclose	Blocks until all in-flight async tasks on the dataset are complete
H5Dclose_async	Completes when all in-flight async tasks on the dataset are complete
H5Dextend	Upon dataset shrinking, data is currently not cleared.[1]
H5Dset_extent	Upon dataset shrinking, data is currently not cleared.[1]
H5Dset_extent_async	
H5Dget_space	
H5Dget_space_async	
H5Dget_type	
H5Dget_create_plist	
H5Dget_access_plist	
H5Dget_space_status	Space status is currently always set to `H5D_SPACE_STATUS_NOT_ALLOCATED`
H5Dflush	Blocks until all in-flight async tasks on the dataset are complete
H5Drefresh	

Currently unsupported API calls

API call	Notes
H5Dget_storage_size	`H5Dget_storage_size` is not currently planned to be supported

[1] Will be supported in the future.

3.2.3. H5F interface

Supported API calls

API call	Notes
H5Fcreate	
H5Fcreate_async	
H5Fopen	
H5Fopen_async	
H5Freopen	
H5Freopen_async	
H5Fis_accessible	
H5Fget_create_plist	
H5Fget_access_plist	
H5Fget_intent	
H5Fget_name	
H5Fget_obj_count	
H5Fget_obj_ids	
H5Fdelete	
H5Fflush	Blocks until all in-flight async tasks on the file are complete
H5Fflush_async	Completes when all in-flight async tasks on the file are complete
H5Fclose	Blocks until all in-flight async tasks on the file are complete
H5Fclose_async	Completes when all in-flight async tasks on the file are complete

Currently unsupported API calls

API call	Notes
H5Fmount	`H5Fmount` is not currently planned to be supported
H5Funmount	`H5Funmount` is not currently planned to be supported

3.2.4. H5G interface

Supported API calls

API call	Notes
H5Gcreate(1/2)	
H5Gcreate_async	
H5Gcreate_anon	
H5Gopen(1/2)	
H5Gopen_async	
H5Gclose	Blocks until all in-flight async tasks on the group are complete
H5Gclose_async	Completes when all in-flight async tasks on the group are complete
H5Gunlink	
H5Gget_create_plist	
H5Gget_info(_by_name/_by_idx)	Of the four fields in the `H5G_info_t` struct: ■ `storage_type` is always set to `H5G_STORAGE_TYPE_UNKNOWN` ■ `nlinks` is set appropriately ■ `max_corder` is set appropriately if link creation order is tracked for the group ■ `mounted` is currently always set to `FALSE` For `H5Gget_info_by_idx`, `H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Gget_info(_by_name/_by_idx)_async	
H5Gget_linkval	
H5Gget_num_objs	
H5Gget_objname_by_idx	`H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Glink(2)	Currently only hard and soft link creation are supported[1]
H5Gmove(2)	Refer to Notes for `H5Lmove`

[1] External links are not currently planned to be supported.

API call	Notes
H5Gflush	Blocks until all in-flight async tasks on the file are complete
H5Grefresh	`H5Grefresh` is currently implemented as a no-op

Currently unsupported API calls

API call	Notes

3.2.5. H5L interface

Supported API calls

API call	Notes
H5Lcreate_hard	
H5Lcreate_hard_async	Not currently tested due to a bug in HDF5
H5Lcreate_soft	
H5Lcreate_soft_async	Currently H5P_DEFAULT cannot be passed as the lapl_id due to a bug in HDF5
H5Lexists	
H5Lexists_async	
H5Literate(_by_name)	■ Restarting iteration from an index value is currently unsupported[1] ■ `H5_ITER_DEC` is currently unsupported for the iteration order when `H5_INDEX_NAME` is used for the index type
H5Literate_async	Currently executes in a blocking fashion
H5Lvisit(_by_name)	■ Restarting iteration from an index value is currently unsupported[1] ■ `H5_ITER_DEC` is currently unsupported for the iteration order when `H5_INDEX_NAME` is used for the index type
H5Ldelete	
H5Ldelete_async	
H5Ldelete_by_idx	`H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Ldelete_by_idx_async	

[1] Will be supported in the future.

API call	Notes
H5Lget_info	Of the five fields in the `H5L_info_t` struct: ■ `type` is set appropriately ■ `corder_valid` is set to TRUE only if link creation order tracking is enabled for the group containing the link; it is set to FALSE otherwise ■ `corder` is set appropriately if link creation order tracking is enabled for the group containing the link; it is set to 0 otherwise ■ `cset` is currently always set to `H5T_CSET_ASCII` ■ `u` has member `address` or `val_size` set appropriately based on whether the link is a hard link or not
H5Lget_info_by_idx	`H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Lget_val	
H5Lget_val_by_idx	`H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Lget_name_by_idx	`H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Lcopy	Currently no support for the following properties: ■ LAPL – H5Pset_nlinks – H5Pset_elink_prefix[1]
H5Lmove	Currently no support for the following properties: ■ LAPL – H5Pset_nlinks – H5Pset_elink_prefix[1]

Currently unsupported API calls

API call	Notes
H5Lcreate_external	`H5Lcreate_external` is not currently planned to be supported. As DAOS containers can contain large amounts of objects, the necessity for external links is lessened as compared to a traditional storage system
H5Lcreate_ud	`H5Lcreate_ud` is not currently planned to be supported

[1] External links are not currently planned to be supported.

3.2.6. H5M interface

Supported API calls

API call	Notes
H5Mcreate	
H5Mcreate_async	
H5Mcreate_anon	
H5Mopen	
H5Mopen_async	
H5Mclose	Blocks until all in-flight async tasks on the group are complete
H5Mclose_async	Completes when all in-flight async tasks on the group are complete
H5Mget_key_type	
H5Mget_val_type	
H5Mget_create_plist	
H5Mget_access_plist	
H5Mget_count	
H5Mput	
H5Mput_async	Currently executes in a blocking fashion
H5Mget	
H5Mget_async	Currently executes in a blocking fashion
H5Mexists	
H5Miterate	
H5Miterate_by_name	
H5Mdelete	

Currently unsupported API calls

API call	Notes

3.2.7. H5O interface

Supported API calls

API call	Notes
H5Oopen	
H5Oopen_async	
H5Oopen_by_token	
H5Oopen_by_idx	`H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Oopen_by_idx_async	
H5Oclose	Blocks until all in-flight async tasks on the object are complete
H5Oclose_async	Completes when all in-flight async tasks on the object are complete
H5Oincr_refcount	
H5Odecr_refcount	
H5Olink	
H5Oexists_by_name	
H5Oget_info(_by_name)3	
H5Oget_info_by_name_async	Currently executes in a blocking fashion
H5Ovisit(1/2)	`H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Ovisit_by_name(1/2)	`H5_ITER_DEC` is currently unsupported for the index ordering when `H5_INDEX_NAME` is used for the index type
H5Ocopy	Currently no support for the following properties: ■ OCpyPL – `H5O_COPY_EXPAND_EXT_LINK_FLAG`[1] – `H5O_COPY_EXPAND_REFERENCE_FLAG`[2] – `H5O_COPY_MERGE_COMMITTED_DTYPE_FLAG`
H5Ocopy_async	

[1] External links are not currently planned to be supported.
[2] Will be supported in the future.

API call	Notes
H5Oclose	Blocks until all in-flight async tasks on the object are complete
H5Oclose_async	Completes when all in-flight async tasks on the object are complete
H5Oflush	Blocks until all in-flight async tasks on the object are complete
H5Oflush_async	Completes when all in-flight async tasks on the object are complete
H5Orefresh	`H5Orefresh` delegates to the appropriate `H5Xrefresh` routine based upon the given object's type, but that particular object's refresh routine may be implemented as a no-op

Currently unsupported API calls

API call	Notes
H5Oopen_by_addr	

3.2.8. H5R interface

Supported API calls

API call	Notes
H5Rcreate_object	Object lookup and retrieval of a container's info (which are needed for `H5Rcreate_object`) are supported
H5Rcreate_region	Object lookup and retrieval of a container's info (which are needed for `H5Rcreate_region`) are supported
H5Rcreate_attr	Object lookup and retrieval of a container's info (which are needed for `H5Rcreate_attr`) are supported
H5Ropen_object	Object opening (which is needed for `H5Ropen_object`) is supported
H5Ropen_region	Object opening and retrieval of a dataset's dataspace (which are needed for `H5Ropen_region`) are supported
H5Ropen_attr	Object opening and attribute opening (which are needed for `H5Ropen_attr`) are supported
H5Ropen_object_async	Currently untested
H5Ropen_region_async	Currently untested
H5Ropen_attr_async	Currently untested
H5Rget_obj_type3	
H5Rget_file_name	Retrieval of a file's name (which is needed for `H5Rget_file_name`) is supported

Currently unsupported API calls

API call	Notes
H5Rget_obj_name	

3.2.9. H5T interface

Supported API calls

API call	Notes
H5Tcommit(1/2)	
H5Tcommit_async	Currently executes in a blocking fashion
H5Tcommit_anon	
H5Topen(1/2)	
H5Topen_async	Currently untested
H5Tclose	Blocks until all in-flight async tasks on the datatype are complete
H5Tclose_async	Completes when all in-flight async tasks on the datatype are complete, currently untested
H5Tget_create_plist	
H5Tflush	Blocks until all in-flight async tasks on the datatype are complete
H5Trefresh	`H5Trefresh` is currently implemented as a no-op

Currently unsupported API calls

API call	Notes

3.3. Known Limitations

The following sections outline the known current limitations of the DAOS VOL connector.

3.3.1. Limitations in regards to the HDF5 API

- If an application abnormally exits, the DAOS VOL connector currently leaves the file in an unusable state. Currently, the only way to re-use the same filename after an application interruption is to use a new DAOS pool. This issue will be resolved when rollback to a previous snapshot is supported.

3.3.2. Limitations in regards to DAOS

- Following the previous point about application abnormal exits, as DAOS does not currently support forced container deletion, trying to overwrite an existing HDF5 file using the `H5F_ACC_TRUNC` flag when the file was left in an unusable state will fail; the error *"can't destroy container: generic I/O error (DER_IO)"* will be returned.
- No support for conditional key insert/remove.
- There is currently no support for distributed transactions.

4. Testing the DAOS VOL connector

The following sections cover how to test the DAOS VOL connector, as well as the individual components of the DAOS VOL connector's overall testing infrastructure.

4.1. With CTest

Once the DAOS VOL connector has been built, running the connector's tests should be as simple as running

```
ctest .
```

from the build directory. This will run each of the VOL connector's test components in turn. For more information on using CTest's options to control testing behavior, refer to the CTest Documentation.

4.2. Manually

If testing the DAOS VOL connector without CTest, refer to Starting the DAOS Server and Running HDF5 DAOS VOL connector applications to make sure that the DAOS Server is up and running and your environment is setup correctly. Once that is done, the DAOS VOL connector's tests can be run directly from the `bin` directory inside the build directory. For a listing of the different test executables and the functionality they test, refer to the following sections.

4.3. DAOS VOL connector's testing components

4.3.1. Generic HDF5 VOL connector test suite

In order to test HDF5 VOL connectors to make sure that they are functioning as expected, a suite of tests which only use the public HDF5 API has been written. This suite of tests is available under the path:

`test/vol`

and when built, will appear as the `h5vl_test` and `h5vl_test_parallel` executables in the `bin` directory inside the build directory.

Note that running this test suite requires that your environment is setup to have HDF5 dynamically load the DAOS VOL connector. Also, this test suite currently does not have the capability to query what kind of functionality an HDF5 VOL connector supports and therefore certain tests will be skipped if they use an HDF5 API call which is not implemented, or which is specifically unsupported, by the DAOS VOL connector.

4.3.2. DAOS VOL connector-specific test suite

In addition to the generic VOL connector testing suite, the DAOS VOL connector also includes the following test suites, which test features specific to the connector:

- DAOS VOL connector Map test suite

This test suite tests the DAOS VOL connector against the 'Map' functionality in HDF5, which concerns map objects that store key-value pairs. When built, this test suite will appear as the `h5daos_test_map` and `h5daos_test_map_parallel` executables in the `bin` directory inside the build directory.

- DAOS VOL connector Recovery testing suite

  This test suite tests the DAOS VOL connector's ability to recover from a fault that causes the DAOS server or the HDF5 application to stop functioning. It also ensures the integrity of both metadata and raw data in a file after such a fault. When built, this test suite will appear as the `h5daos_test_recovery` executable in the `bin` directory inside the build directory.

A. Reference Manual

A.1. H5daos_init

Synopsis:

```
herr_t H5daos_init(uuid_t pool_uuid, const char *pool_grp, const char
↪  *pool_svcl);
```

Purpose:

Initialize the DAOS VOL connector.

Description:

`H5daos_init` initializes the VOL connector by registering the connector with the library.

Parameters:

`uuid_t pool_uuid`	IN: UUID to identify a pool
`const char *pool_grp`	IN: Process set name of the DAOS servers managing the pool
`const char *pool_svcl`	IN: Comma-separated list of pool service replica ranks

Returns:

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

A.2. H5daos_term

Synopsis:

```
herr_t H5daos_term(void);
```

Purpose:

Terminate the DAOS VOL connector.

Description:

`H5daos_term` terminates the DAOS VOL connector.

Parameters:

None.

Returns:

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

A.3. H5Pset_fapl_daos

Synopsis:

```
herr_t H5Pset_fapl_daos(hid_t fapl_id, MPI_Comm comm, MPI_Info info);
```

Purpose:

Set the file access property list to use the DAOS VOL connector.

Description:

`H5Pset_fapl_daos` modifies the file access property list to use the DAOS VOL connector. `file_comm` and `file_info` identify the communicator and info object used to coordinate actions on file create, open, flush, and close.

Parameters:

`hid_t fapl_id`	IN: File access property list ID
`MPI_Comm file_comm`	IN: MPI Communicator
`MPI_Info file_info`	IN: MPI Info

Returns:

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

A.4. H5daos_set_all_ind_metadata_ops

Synopsis:

```
herr_t H5daos_set_all_ind_metadata_ops(hid_t accpl_id, hbool_t
    is_independent);
```

Purpose:

Sets the I/O mode for metadata read/write operations in the access property list `accpl_id`.

When engaging in parallel I/O with the DAOS VOL connector, all metadata read operations are independent and all metadata write operations are collective by default. If `is_independent` is specified as `TRUE`, this property indicates that the DAOS VOL connector will perform all metadata read and write operations independently.

If this property is set to `TRUE` on a file access property list that is used in creating or opening a file, the DAOS VOL connector will assume that all metadata read and write operations issued on that file identifier should be issued independently from all ranks, irrespective of the individual setting for a particular operation.

Alternatively, a user may wish to avoid setting this property globally on the file access property list and individually set it on particular object access property lists (dataset, group, link, datatype, attribute access property lists) for certain operations instead. This will indicate that only the operations issued with such an access property list will perform metadata I/O independently, whereas other operations may perform metadata I/O collectively.

Description:

`H5daos_set_all_ind_metadata_ops` modifies the access property list to indicate that metadata I/O operations should be performed independently.

Parameters:

`hid_t accpl_id`	IN: File, group, dataset, datatype, link or attribute access property list ID
`hbool_t is_independent`	IN: Boolean value indicating whether metadata I/O operations should be performed independently (`TRUE`) or should be allowed to be performed collectively (`FALSE`).

Returns:

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

A.5. H5daos_get_all_ind_metadata_ops

Synopsis:

```
herr_t H5daos_get_all_ind_metadata_ops(hid_t accpl_id, hbool_t
    *is_independent);
```

Purpose:

Retrieves the independent metadata I/O setting from the access property list `accpl_id`.

Description:

`H5daos_get_all_ind_metadata_ops` retrieves the independent metadata I/O setting from the access property list `accpl_id`.

Parameters:

`hid_t accpl_id`	IN: File, group, dataset, datatype, link or attribute access property list ID
`hbool_t *is_independent`	OUT: Pointer to a Boolean value to be set, indicating whether metadata I/O is performed independently or is allowed to be performed collectively.

Returns:

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

B. Native HDF5 VOL connector-specific API calls

The following HDF5 API calls are either specific to the native HDF5 VOL connector or are not routed through the VOL and thus are not able to be implemented by the DAOS VOL connector (or other VOL connectors):

B.1. H5A interface

API call	Notes
H5Aiterate1	Deprecated in favor of H5Aiterate2
H5Aget_num_attrs	Deprecated in favor of H5Oget_info

B.2. H5D interface

API call	Notes
H5Dformat_convert	
H5Dget_offset	
H5Dget_chunk_index_type	
H5Dget_chunk_storage_size	
H5Dvlen_reclaim	
H5Dvlen_get_buf_size	
H5Diterate	
H5Dscatter	
H5Dgather	
H5Dfill	
H5Dread_chunk	
H5Dwrite_chunk	

B.3. H5F interface

API call	Notes
H5Fis_hdf5	Uses a default FAPL so can only ever be routed through the native HDF5 VOL connector
H5Fget_vfd_handle	
H5Fget_freespace	
H5Fget_filesize	
H5Fget_file_image	
H5Fget_mdc_config	
H5Fset_mdc_config	
H5Fget_mdc_hit_rate	
H5Fget_mdc_size	
H5Freset_mdc_hit_rate_stats	
H5Fget_info(1/2)	
H5Fget_metadata_read_retry_info	
H5Fget_free_sections	
H5Fclear_elink_file_cache	
H5Fstart_swmr_write	
H5Fstart_mdc_logging	
H5Fstop_mdc_logging	
H5Fget_mdc_logging_status	
H5Fset_libver_bounds	
H5Fformat_convert	
H5Freset_page_buffering_stats	
H5Fget_page_buffering_stats	
H5Fget_mdc_image_info	
H5Fget_eoa	
H5Fincrement_filesize	
H5Fget_dset_no_attrs_hint	
H5Fset_dset_no_attrs_hint	
H5Fset_latest_format	
H5Fset_mpi_atomicity	
H5Fget_mpi_atomicity	

B.4. H5G interface

API call	Notes
H5Gset_comment	Deprecated in favor of H5Oset_comment/H5Oset_comment_by_name
H5Gget_comment	Deprecated in favor of H5Oget_comment/H5Oget_comment_by_name
H5Giterate	Deprecated in favor of H5Literate
H5Gget_objinfo	Deprecated in favor of H5Lget_info/H5Oget_info
H5Gget_objtype_by_idx	Deprecated in favor of H5Lget_info/H5Oget_info

B.5. H5L interface

API call	Notes
H5Lregister	
H5Lunregister	
H5Lis_registered	
H5Lunpack_elink_val	

B.6. H5O interface

API call	Notes
H5Oget_info(1/2)	
H5Oget_info_by_name(1/2)	
H5Oget_info_by_idx(1/2)	
H5Oset_comment(_by_name)	Deprecated in favor of using attributes on objects
H5Oget_comment(_by_name)	Deprecated in favor of using attributes on objects
H5Oare_mdc_flushes_disabled	
H5Oenable_mdc_flushes	
H5Odisable_mdc_flushes	

B.7. H5R interface

API call	Notes
H5Rdestroy	
H5Rget_type	
H5Requal	
H5Rcopy	
H5Rget_attr_name	

B.8. H5T interface

API call	Notes