HDF5 Virtual Object Layer (VOL) Connector Author Guide

-

HDF5 1.13.0

-

*** PRELIMINARY ***

The HDF Group

30th September 2021

Contents

1 Introduction

The Virtual Object Layer (VOL) is an abstraction layer in the HDF5 library which intercepts all API calls that could potentially access objects in an HDF5 container and forwards those calls to object drivers referred to as *VOL connectors*. The architecture of this feature is described in the VOL User Guide and VOL Architecture and Internals Documentation and will not be duplicated here.

This guide is for people who are interested in developing their own VOL connector for the HDF5 library. It is assumed that the reader has good knowledge of the VOL architecture obtained by reading the VOL architectural design document.

2 Creating a New Connector

2.1 Overview

Creating a new VOL connector can be a complicated process. You will need to map your storage system to the HDF5 data model through the lens of the VOL and this may involve some impedence mismatch that you will have to work around. The good news is that the HDF5 library has been re-engineered to handle arbitrary, connector-specific data structures via the VOL callbacks, so no knowledge of the library internals is necessary to write a VOL connector.

Writing a VOL connector requires these things:

1. Decide on library vs plugin vs internal.
2. Set up your build/test files (CMake, Autotools, etc.).
3. Fill in some boilerplate information in your `H5VL_class_t` struct.
4. Decide how you will perform any necessary initialization needed by your storage system.
5. Map Storage to HDF5 File Objects
6. Create implementations for the callbacks you need to support.
7. Test the connector.

Each of the steps listed above is decribed in more detail in this section of the document.

The "model then implement" steps can be performed iteratively. You might begin by only supporting files, datasets, and groups and only allowing basic operations on them. In some cases, this may be all that is needed. As your needs grow, you can repeat those steps and increase the connector's HDF5 API coverage at a pace that makes sense for your users.

Also, note that this document only covers writing VOL connectors using the C programming language. It is often possible to write connectors in other programming languages (e.g.; Python) via the language's C interop facilities, but that topic is out of scope for this document.

2.2 The HDF5 1.12.x VOL Interface Is DEPRECATED

Important changes were made to the VOL interface for HDF5 1.13.0 and, due to binary compatibility issues, these cannot be merged to HDF5 1.12.x. For this reason, VOL connector development should be shifted to target 1.13.0 as no further development of the VOL interface will take place on the 1.12.x branch. Unlike the other development branches of the library, there is no `hdf5_1_13` branch - all HDF5 1.13.0 development is taking place in the `develop` branch of the HDF5 repository and 1.13.x branches will split off from that.

Note also that HDF5 1.13.0 is considered an unstable branch, and the API and file format are subject to change ("unstable" means "not yet finalized", not "buggy"). The VOL feature is under active development and, although it is nearing its final form, may change further before the stable HDF5 1.14.0 release targeted for 2022.

2.3 VOL-Related HDF5 Header Files

Use of the VOL, including topics such as registration and loading VOL plugins, is described in the VOL User Guide.

Public header files you will need to be familiar with include:

`H5VLpublic.h`	Public VOL header.
`H5VLconnector.h`	Main header for connector authors. Contains definitions for the main VOL struct and callbacks, enum values, etc.
`H5VLconnector_passthru.h`	Helper routines for passthrough connector authors.
`H5VLnative.h`	Native VOL connector header. May be useful if your connector will attempt to implement native HDF5 API calls that are handled via the *optional* callbacks.
`H5PLextern.h`	Needed if your connector will be built as a plugin.

Many VOL connectors are listed on The HDF Group's VOL plugin registration page, located at https://portal.hdfgroup.org/display/support/Registered+VOL+Connectors. Not all of these VOL connectors are supported by The HDF Group and the level of completeness varies, but the connectors found there can serve as examples of working implementations.

2.4 Library vs Plugin vs Internal

When building a VOL connector, you have several options:

Library

The connector can be built as a normal shared or static library. Software that uses your connector will have to link to it just like any other library. This can be convenient since you don't have to deal with plugin paths and searching for the connector at runtime, but it also means that software which uses your connector will have to be built and linked against it.

Plugin

You can also build your connector as a dynamically loaded plugin. The mechanism for this is the same mechanism used to dynamically load HDF5 filter plugins. This can allow use of your connector via the VOL environment variable, without modifying the application, but requires your plugin to be discoverable at runtime. See the VOL User Guide for more information about using HDF5 plugins.

To build your connector as a plugin, you will have to include `H5PLextern.h` (a public header distributed with the library) and implement the `H5PLget_plugin_type()` and `H5PLget_plugin_info()` calls, both of which are trivial to code up. It also often requires your connector to be built with certain compile/link options. The VOL connector template does all of these things.

The HDF5 library's plugin loading code will call `H5PLget_plugin_type()` to determine the type of plugin (e.g.; filter, VOL) and `H5PLget_plugin_info()` to get the class struct, which allows the library to query the plugin for its name and value to see if it has found a requested plugin. When a match is found, the library will use the class struct to register the connector and map its callbacks.

For the HDF5 library to be able to load an external plugin dynamically, the plugin developer has to define two public routines with the following name and signature:

```
H5PL_type_t H5PLget_plugin_type(void)
const void *H5PLget_plugin_info(void)
```

To show how easy this is to accomplish, here is the complete implementation of those functions in the template VOL connector:

```
H5PL_type_t H5PLget_plugin_type(void) {return H5PL_TYPE_VOL;}
const void *H5PLget_plugin_info(void) {return &template_class_g;}
```

`H5PLget_plugin_type` should return the library type which should always be `H5PL_TYPE_VOL`. `H5PLget_plugin_info` should return a pointer to the plugin structure defining the VOL plugin with all the callbacks. For example, consider an external plugin defined as:

```
static const H5VL_class_t H5VL_foo_g = {
    2,     /* version */
    12345, /* value */
    "foo", /* name */
    ...
}
```

The plugin would implement the two routines as:

```
H5PL_type_t H5PLget_plugin_type(void) {return H5PL_TYPE_VOL;}
const void *H5PLget_plugin_info(void) {return &H5VL_foo_g;}
```

Internal

Your VOL connector can also be constructed as a part of the HDF5 library. This works in the same way as the stdio and multi virtual file drivers (VFDs) and does not require knowledge of HDF5 internals or use of non-public API calls. You simply have to add your connector's files to the `Makefile.am` and/or `CMakeLists.txt` files in the source distribution's `src` directory. This requires maintaining a private build of the library, though, and is not recommended.

2.5 Build Files / VOL Template

We have created a template terminal VOL connector that includes both Autotools and CMake build files. The constructed VOL connector includes no real functionality, but can be registered and loaded as a plugin.

The VOL template can be found here:

https://github.com/HDFGroup/vol-template

The purpose of this template is to quickly get you to the point where you can begin filling in the callback functions and writing tests. You can copy this code to your own repository to serve as the basis for your new connector.

A template passthrough VOL is also available. This will be discussed in the section on passthrough connectors.

2.6 `H5VL_class_t` Boilerplate

Several fields in the `H5VL_class_t` struct will need to be filled in.

In HDF5 1.13.0, the `version` field will be 2, indicating the connector targets version 2 of the `H5VL_class_t` struct. Version 1 of the struct was never formally released and only available in the `develop` branch of the HDF5 git repository. Version 0 is used in the deprecated HDF5 1.12.x branch.

Every connector needs a `name` and `value`. The library will use these when loading and registering the connector (as described in the VOL User Guide), so they should be unique in your ecosystem.

VOL connector values are integers, with a maximum value of 65535. Values from 0 to 255 are reserved for internal use by The HDF Group. The native VOL connector has a value of 0. Values of 256 to 511 are for connector testing and should not be found in the wild. Values of 512 to 65535 are for external connectors.

As is the case with HDF5 filters, The HDF Group can assign you an official VOL connector value. Please contact help@hdfgroup.org for help with this. We currently do not register connector names, though the name you've chosen will appear on the registered VOL connectors page.

As noted above, registered VOL connectors will be listed at:

https://portal.hdfgroup.org/display/support/Registered+VOL+Connectors

A new `conn_version` field has been added to the class struct for 1.13. This field is currently not used by the library so its use is determined by the connector author. Best practices for this field will be determined in the near future and this part of the guide will be updated.

The `cap_flags` field is used to determine the capabilities of the VOL connector. At this time, the use of this field is limited to indicating thread-safety, asynchronous capabilities, and ability to produce native HDF5 files. Supported flags can be found in `H5VLconnector.h`.

```
/* Capability flags for connector */
#define H5VL_CAP_FLAG_NONE      0    /* No special connector capabilities */
#define H5VL_CAP_FLAG_THREADSAFE 0x01 /* Connector is threadsafe */
#define H5VL_CAP_FLAG_ASYNC     0x02 /* Connector performs operations asynchronously*/
#define H5VL_CAP_FLAG_NATIVE_FILES 0x04 /* Connector produces native file format */
```

2.7 Initialization and Shutdown

You'll need to decide how to perform any initialization and shutdown tasks that are required by your connector. There are initialize and terminate callbacks in the `H5VL_class_t` struct to handle this. They are invoked when the connector is registered and unregistered, respectively. The inialize callback can take a VOL initialization property list, so any properties you need for initialization can be applied to it. The HDF5 library currently makes no use of the vipl so there are no default vipl properties.

If this is unsuitable, you may have to create custom connector-specific API calls to handle initialization and termination. It may also be useful to perform operations in a custom API call used to set the VOL connector in the fapl.

The initialization and terminate callbacks:

```
herr_t (*initialize)(hid_t vipl_id); /**< Connector initialization callback */

herr_t (*terminate)(void);       /**< Connector termination callback */
```

2.8 Map Storage to HDF5 File Objects

The most difficult part of designing a new VOL connector is going to determining how to support HDF5 file objects and operaions using your storage system. There isn't much specific advice to give here, as each connector will have unique needs, but a forthcoming "tutorial" connector will set up a simple connector and demonstrate this process.

2.9 Fill In VOL Callbacks

For each file object you support in your connector (including the file itself), you will need to create a data struct to hold whatever file object metadata that are needed by your connector. For example, a data structure for a VOL connector based on text files might have a file struct that contains a file pointer for the text file, buffers used for cacheing data, etc. Pointers to these data structures are where your connector's state is stored and are returned to the HDF5 library from the create/open/etc. callbacks such as *dataset create*.

Once you have your data structures, you'll need to create your own implementations of the callback functions and map them via your `H5VL_class_t` struct.

2.10 Handling Optional Operations

Handling optional operations has changed significantly in HDF5 1.13.0. In the past, optional operations were specified using an integer `opt_type` parameter. This proved to be a problem with pass-through connectors, though, as it was possible to have `opt_type` clash if two connectors used the same `opt_type` values.

The new scheme allows a connector to register an optional operation with the library and receive a dynamically-allocated `opt_type` value for the operation.

The following API calls can be used to manage the optional operations:

```
herr_t H5VLregister_opt_operation(H5VL_subclass_t subcls, const char *op_name, int *op_val);
herr_t H5VLfind_opt_operation(H5VL_subclass_t subcls, const char *op_name, int *op_val);
herr_t H5VLunregister_opt_operation(H5VL_subclass_t subcls, const char *op_name);
```

The `register` call is used to register an operation for a subclass (file, etc.) and the `opt_type` parameter that the library assigned to the operation will be returned via the `opt_val` parameter. This value can then be passed to one of the subclass-specific API calls (listed below). If you need to find an existing optional call's assigned `opt_type` value by name, you can use the `find` call.

One recommended way to handle optional calls is to register all the optional calls at startup, saving the values in connector state, then use these cached values in your optional calls. The assigned values should be unregistered using the `unregister` call when the connector shuts down.

Subclass-specific optional calls:

```
herr_t H5VLattr_optional_op(const char *app_file, const char *app_func, unsigned app_line,
                            hid_t attr_id, H5VL_optional_args_t *args, hid_t dxpl_id, hid_t es_id);
```

```
herr_t H5VLdataset_optional_op(const char *app_file, const char *app_func, unsigned app_line,
                               hid_t dset_id, H5VL_optional_args_t *args, hid_t dxpl_id, hid_t es_id);
herr_t H5VLdatatype_optional_op(const char *app_file, const char *app_func, unsigned app_line,
                                hid_t type_id, H5VL_optional_args_t *args, hid_t dxpl_id, hid_t
                                    es_id);
herr_t H5VLfile_optional_op(const char *app_file, const char *app_func, unsigned app_line,
                            hid_t file_id, H5VL_optional_args_t *args, hid_t dxpl_id, hid_t es_id);
herr_t H5VLgroup_optional_op(const char *app_file, const char *app_func, unsigned app_line,
                             hid_t group_id, H5VL_optional_args_t *args, hid_t dxpl_id, hid_t es_id);
herr_t H5VLlink_optional_op(const char *app_file, const char *app_func, unsigned app_line,
                            hid_t loc_id, const char *name, hid_t lapl_id, H5VL_optional_args_t *args,
                            hid_t dxpl_id, hid_t es_id);
herr_t H5VLobject_optional_op(const char *app_file, const char *app_func, unsigned app_line,
                              hid_t loc_id, const char *name, hid_t lapl_id,
                              H5VL_optional_args_t *args, hid_t dxpl_id, hid_t es_id);
herr_t H5VLrequest_optional_op(void *req, hid_t connector_id, H5VL_optional_args_t *args);
```

2.11 Testing Your Connector

At the time of writing, some of the HDF5 library tests have been abstracted out of the library with their native-file-format-only sections removed and added to a VOL test suite available here:

https://github.com/HDFGroup/vol-tests

This is an evolving set of tests, so see the documentation in that repository for instructions as to its use. You may want to clone and modify and/or extend these tests for use with your own connector.

In the future, we plan to modify the HDF5 test suite that ships with the library to use a future VOL capabilities flags scheme to selectively run tests that a particular connector supports. As this is a large task, it may be some time before that work is complete.

2.12 Passthrough Connectors

Coming Soon

2.13 Asynchronous Operations

Coming Soon

3 VOL Connector Interface Reference

Each VOL connector should be of type `H5VL_class_t`, Listing 1.

```
/* Class information for each VOL driver */
typedef struct H5VL_class_t {
    /* Overall connector fields & callbacks */
    unsigned         version;        /**< VOL connector class struct version # */
    H5VL_class_value_t value;        /**< Value to identify connector          */
    const char *     name;           /**< Connector name (MUST be unique!)     */
    unsigned         conn_version;   /**< Version # of connector               */
    unsigned         cap_flags;      /**< Capability flags for connector       */
    herr_t (*initialize)(hid_t vipl_id); /**< Connector initialization callback */
    herr_t (*terminate)(void);       /**< Connector termination callback       */

    /* VOL framework */
    H5VL_info_class_t info_cls; /**< VOL info fields & callbacks */
    H5VL_wrap_class_t wrap_cls; /**< VOL object wrap / retrieval callbacks */

    /* Data Model */
    H5VL_attr_class_t   attr_cls;    /**< Attribute (H5A*) class callbacks */
    H5VL_dataset_class_t dataset_cls; /**< Dataset (H5D*) class callbacks */
    H5VL_datatype_class_t datatype_cls; /**< Datatype (H5T*) class callbacks */
    H5VL_file_class_t   file_cls;    /**< File (H5F*) class callbacks  */
```

```
21      H5VL_group_class_t  group_cls;   /**< Group (H5G*) class callbacks */
22      H5VL_link_class_t   link_cls;    /**< Link (H5L*) class callbacks  */
23      H5VL_object_class_t object_cls;  /**< Object (H5O*) class callbacks */
24
25      /* Infrastructure / Services */
26      H5VL_introspect_class_t introspect_cls; /**< Container/connector introspection class callbacks
            */
27      H5VL_request_class_t  request_cls;   /**< Asynchronous request class callbacks */
28      H5VL_blob_class_t     blob_cls;      /**< 'Blob' class callbacks */
29      H5VL_token_class_t    token_cls;     /**< VOL connector object token class callbacks */
30
31      /* Catch-all */
32      herr_t (*optional)(void *obj, H5VL_optional_args_t *args, hid_t dxpl_id,
33                         void **req); /**< Optional callback */
34  } H5VL_class_t;
```

Listing 1: VOL connector class, H5VLpublic.h

The `version` field is the version of the `H5VL_class_t` struct. This is identical to how the `version` field is used in the `H5Z_class2_t` struct for filters.

The `value` field is a unique integer identifier that should be between 512 and 65535 for external, non-library connectors..

The `name` field is a string that uniquely identifies the VOL connector name.

The `conn_version` is the connector version. This is currently not used by the library.

The `cap_flags` holds bitwise capability/feature flags that determine which operations and capabilities are supported by a the VOL connector. These fields were enumerated in the previous section.

The `initialize` field is a function pointer to a routine that a connector implements to set up or initialize access to the connector. Implementing this function by the connector is not required since some connectors do not require any set up to start accessing the connector. In that case, the value of the function pointer should be set to NULL. Connector specific variables that are required to be passed from users should be passed through the VOL initialize property list. Generic properties can be added to this property class for user-defined connectors that cannot modify the HDF5 library to add internal properties. For more information consult the property list reference manual pages.

The `terminate` field is a function pointer to a routine that a connector implements to terminate or finalize access to the connector. Implementing this function by the connector is not required since some connectors do not require any termination phase to the connector. In that case, the value of the function pointer should be set to NULL.

The rest of the fields in the `H5VL_class_t` struct are "subclasses" that define all the VOL function callbacks that are mapped to from the HDF5 API layer. Those subclasses are categorized into three categories, VOL Framework, Data Model, and Infrastructure / Services.

VOL Framework classes provide functionality for working with the VOL connectors themselves (e.g., working with connector strings) and with wrapping and unwrapping objects for passthrough connectors.

Data Model classes are those that provide functionality for accessing an HDF5 container and objects in that container as defined by the HDF5 data model.

Infrastructure / Service classes are those that provide services for users that are not related to the data model specifically. Asynchronous operations, for example, are a service that most connectors can implement, so we add a class for it in the VOL structure.

If a service becomes generic enough and common among many connectors, a class for it should be added to the VOL structure. However, many connectors can/will provide services that are not shared by other connectors. A good way to support these services is through an optional callback in the VOL structure which can be a hook from the API to the connector that provides those services, passing any necessary arguments needed without the HDF5 library having to worry about supporting that service. A similar API operation to allow users to use that service will be added. This API call would be similar to an "ioctl" call where any kind of operation can be supported and passed down to the connector that has enough knowledge from the user to interpret the type of the operation. All classes and their defined callbacks will be detailed in the following sub-sections.

To handle that large set of API routines, each class in the Data Model category has three generic callbacks, `get`, `specific`, and `optional` to handle the three set of API operations outline above respectively. To handle the varying parameters that can be passed to the callback, each callback will take a struct parameter that includes an enum `get/specific` or integer `optional` field indicating the operation and a union of the possible parameters `get/specific` or void pointer to the parameters `optional`.

The optional args struct used for all optional operations:

```
/* Struct for all 'optional' callbacks */
typedef struct H5VL_optional_args_t {
    int  op_type; /* Operation to perform */
    void *args;   /* Pointer to operation's argument struct */
} H5VL_optional_args_t;
```

The `opt_type` member is the value assigned by the library when the optional operation was registered (or `#defined` in the case of the native VOL connector) and the `args` member is a pointer to the optional operation's parameters (usually passed in as a struct).

Note that this differs from the HDF5 1.12.x scheme, which used `va_lists`.

The `optional` callback is a free for all callback where anything from the API layer is passed in directly. This callback is used to support connector specific operations in the API that other connectors should or would not know about. More information about types and the arguments for each type will be detailed in the corresponding class arguments.

3.1 Mapping the API to the Callbacks

The callback interface defined for the VOL has to be general enough to handle all the HDF5 API operations that would access the file. Furthermore, it has to capture future additions to the HDF5 library with little to no changes to the callback interface. Changing the interface often whenever new features are added would be discouraging to connector developers since that would mean reworking their VOL connector structure. To remedy this issue, every callback will contain two parameters:

- A data transfer property list (DXPL) which allows that API to put some properties on for the connectors to retrieve if they have to for particular operations, without having to add arguments to the VOL callback function.
- A pointer to a request (`void **req`) to handle asynchronous operations if the HDF5 library adds support for them in future releases. That pointer is set by the VOL connector to a request object it creates to manage progress on that asynchronous operation. If the `req` is `NULL`, that means that the API operation is blocking and so the connector would not execute the operation asynchronously. If the connector does not support asynchronous operations, it needs not to worry about this field and leaves it unset.

In order to keep the number of the VOL object classes and callbacks concise and readable, it was decided not to have a one-to-one mapping between API operation and callbacks. The parameter names and types will be detailed when describing each callback in their respective sections.

The HDF5 library provides several routines to access an object in the container. For example, to open an attribute on a group object, the user could use `H5Aopen()` and pass the group identifier directly where the attribute needs to be opened. Alternatively, the user could use `H5Aopen_by_name()` or `H5Aopen_by_idx()` to open the attribute, which provides a more flexible way of locating the attribute, whether by a starting object location and a path or an index type and traversal order. All those types of accesses usually map to one VOL callback with a parameter that indicates the access type. In the example of opening an attribute, the three API open routine will map to the same VOL open callback but with a different location parameter. The same applies to all types of routines that have multiple types of accesses. The location parameter is a structure defined in Listing 2.

```
/*
 * Structure to hold parameters for object locations.
 * either: BY_SELF, BY_NAME, BY_IDX, BY_TOKEN
 */

typedef struct H5VL_loc_params_t {
    H5I_type_t obj_type; /* The object type of the location object */
```

```
    H5VL_loc_type_t type; /* The location type */
    union { /* parameters of the location */
        H5VL_loc_by_token_t   loc_by_token;
        H5VL_loc_by_name_t    loc_by_name;
        H5VL_loc_by_idx_t     loc_by_idx;
    }loc_data;
} H5VL_loc_params_t

/*
 * Types for different ways that objects are located in an
 * HDF5 container.
 */
typedef enum H5VL_loc_type_t {
    /* starting location is the target object */
    H5VL_OBJECT_BY_SELF,

    /* location defined by object and path in H5VL_loc_by_name_t */
    H5VL_OBJECT_BY_NAME,

    /* location defined by object, path, and index in H5VL_loc_by_idx_t */
    H5VL_OBJECT_BY_IDX,

    /* location defined by token (e.g. physical address) in H5VL_loc_by_token_t */
    H5VL_OBJECT_BY_TOKEN,
} H5VL_loc_type_t;

typedef struct H5VL_loc_by_name {
    const char *name; /* The path relative to the starting location */
    hid_t lapl_id; /* The link access property list */
}H5VL_loc_by_name_t;

typedef struct H5VL_loc_by_idx {
    const char *name; /* The path relative to the starting location */
    H5_index_t idx_type; /* Type of index */
    H5_iter_order_t order; /* Index traversal order */
    hsize_t n; /* Position in index */
    hid_t lapl_id; /* The link access property list */
}H5VL_loc_by_idx_t;

typedef struct H5VL_loc_by_token {
    void *token; /* arbitrary token (physical address of location in native VOL) */
}H5VL_loc_by_token_t;
```

Listing 2: Structure to hold parameters for object locations, H5VLconnector.h

3.2 Connector Information Callbacks

This section's callbacks involve the connector-specific information that will be associated with the VOL in the fapl via `H5Pset_fapl()` et al. This data is copied into the fapl so the library needs these functions to manage this in a way that prevents resource leaks.

The `to_str` and `from_str` callbacks are used to convert the connector-specific data to and from a configuration string. There is no official way to construct VOL configuration strings, so the format used (JSON, XML, getopt-style processing, etc.) is up to the connector author. These connector configuration strings can be used to set up a VOL connector via mechanisms like command-line parameters and environment variables.

```
typedef struct H5VL_info_class_t {
    size_t size;
    void * (*copy)(const void *info);
    herr_t (*cmp)(int *cmp_value, const void *info1, const void *info2);
    herr_t (*free)(void *info);
    herr_t (*to_str)(const void *info, char **str);
    herr_t (*from_str)(const char *str, void **info);
} H5VL_info_class_t;
```

Listing 3: Info class for connector information routines, H5VLconnector.h

3.2.1 info: size

The `size` field indicates the size required to store any special information that the connector needs.

If the connector requires no special information, set this field to zero.

Signature:

```
1    size_t size;
```

3.2.2 info: copy

The `copy` callback is invoked when the connector is selected for use with `H5Pset_fapl()`, the connector-specific set call, etc. Where possible, the information should be deep copied in such a way that the original data can be freed.

Signature:

```
1    void * (*copy)(const void *info);
```

Arguments:

`info`	(IN): The connector-specific info to copy.

3.2.3 info: cmp

The `cmp` callback is uesd to determine if two connector-specific data structs are identical and helps the library manage connector resources.

Signature:

```
1    herr_t (*cmp)(int *cmp_value, const void *info1, const void *info2);
```

Arguments:

`cmp_value`	(OUT): A strcmp-like compare value.
`info1`	(IN): The 1st connector-specific info to copy.
`info2`	(IN): The 2nd connector-specific info to copy.

3.2.4 info: free

The `free` callback is used to clean up the connector-specific information that was copied when set in the fapl via the `copy` callback.

Signature:

```
1    herr_t (*free)(void *info);
```

Arguments:

`info`	(IN): The connector-specific info to free.

3.2.5 info: to_str

The `to_str` callback converts a connector-specific information structure to a connector-specific configuration string. It is the opposite of the `from_str` callback.

Signature:

```
1    herr_t (*to_str)(const void *info, char **str);
```

Arguments:

`info` (IN): The connector-specific info to convert to a configuration string.
`str` (OUT): The constructed configuration string.

3.2.6 info: from_str

The `to_str` callback converts a connector-specific configuration string to a connector-specific information structure. It is the opposite of the `to_str` callback.

Signature:

```
herr_t (*from_str)(const char *str, void **info);
```

Arguments:
`str` (IN): The connector-specific configuration string.
`info` (OUT): The connector-specific info generated from the configuration string.

3.3 Object Wrap Callbacks

The object wrap callbacks are used by passthrough connectors to wrap/unwrap objects and contexts when passing them up and down the VOL chain.

```
typedef struct H5VL_wrap_class_t {
    void * (*get_object)(const void *obj);
    herr_t (*get_wrap_ctx)(const void *obj, void **wrap_ctx);
    void * (*wrap_object)(void *obj, H5I_type_t obj_type, void *wrap_ctx);
    void * (*unwrap_object)(void *obj);
    herr_t (*free_wrap_ctx)(void *wrap_ctx);
} H5VL_wrap_class_t;
```

Listing 4: Wrap class for object wrapping routines, H5VLconnector.h

3.3.1 wrap: get_object

Retrieves an underlying object.

Signature:

```
void * (*get_object)(const void *obj);
```

Arguments:
`obj` (IN): Object being unwrapped.

3.3.2 wrap: get_wrap_ctx

Get a VOL connector's object wrapping context.

Signature:

```
herr_t (*get_wrap_ctx)(const void *obj, void **wrap_ctx);
```

Arguments:
`obj` (IN): Object for which we need a context.
`wrap_ctx` (OUT): Context.

3.3.3 wrap: wrap_object

Asks a connector to wrap an underlying object.

Signature:

```
void * (*wrap_object)(void *obj, H5I_type_t obj_type, void *wrap_ctx);
```

Arguments:

`obj` (IN): Object being wrapped.
`obj_type` (IN): Object type (see `H5Ipublic.h`).
`wrap_ctx` (IN): Context.

3.3.4 wrap: unwrap_object

Unwrap an object from connector.

Signature:

```
1    void * (*unwrap_object)(void *obj);
```

Arguments:
`obj` (IN): Object being unwrapped.

3.3.5 wrap: free_wrap_ctx

Release a VOL connector's object wrapping context.

Signature:

```
1    herr_t (*free_wrap_ctx)(void *wrap_ctx);
```

Arguments:
`wrap_ctx` (IN): Context to be freed.

3.4 The Attribute Function Callbacks

The attribute API routines (H5A) allow HDF5 users to create and manage HDF5 attributes. All the H5A API routines that modify the HDF5 container map to one of the attribute callback routines in this class that the connector needs to implement.

```
1  typedef struct H5VL_attr_class_t {
2      void *(*create)(void *obj, const H5VL_loc_params_t *loc_params, const char *attr_name, hid_t
           type_id, hid_t space_id, hid_t acpl_id, hid_t aapl_id, hid_t dxpl_id, void **req);
3      void *(*open)(void *obj, const H5VL_loc_params_t *loc_params, const char *attr_name, hid_t
           aapl_id, hid_t dxpl_id, void **req);
4      herr_t (*read)(void *attr, hid_t mem_type_id, void *buf, hid_t dxpl_id, void **req);
5      herr_t (*write)(void *attr, hid_t mem_type_id, const void *buf, hid_t dxpl_id, void **req);
6      herr_t (*get)(void *obj, H5VL_attr_get_args_t *args, hid_t dxpl_id, void **req);
7      herr_t (*specific)(void *obj, const H5VL_loc_params_t *loc_params, H5VL_attr_specific_args_t
           *args, hid_t dxpl_id, void **req);
8      herr_t (*optional)(void *obj, H5VL_optional_args_t *args, hid_t dxpl_id, void **req);
9      herr_t (*close) (void *attr, hid_t dxpl_id, void **req);
10 } H5VL_attr_class_t;
```

Listing 5: Structure for attribute callback routines, H5VLconnector.h

3.4.1 attr: create

The `create` callback in the attribute class creates an attribute object in the container of the location object and returns a pointer to the attribute structure containing information to access the attribute in future calls.

Signature:

```
1    void *(*create)(void *obj, H5VL_loc_params_t *loc_params,
2        const char *attr_name, hid_t type_id, hid_t space_id,
3        hid_t acpl_id, hid_t aapl_id,
4        hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): Pointer to an object where the attribute needs to be created or where the look-up of the target object needs to start.
loc_params	(IN): Pointer to the location parameters as explained in Section 3.1.
attr_name	(IN): The name of the attribute to be created.
type_id	(IN): The datatype of the attribute.
space_id	(IN): The dataspace of the attribute.
acpl_id	(IN): The attribute creation property list.
aapl_id	(IN): The attribute access property list.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.4.2 attr: open

The `open` callback in the attribute class opens an attribute object in the container of the location object and returns a pointer to the attribute structure containing information to access the attribute in future calls.

Signature:

```
1  void *(*open)(void *obj, H5VL_loc_params_t *loc_params,
2      const char *attr_name, hid_t aapl_id, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): Pointer to an object where the attribute needs to be opened or where the look-up of the target object needs to start.
loc_params	(IN): Pointer to the location parameters as explained in Section 3.1.
attr_name	(IN): The name of the attribute to be opened.
aapl_id	(IN): The attribute access property list.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.4.3 attr: read

The `read` callback in the attribute class reads data from the attribute object and returns an `herr_t` indicating success or failure.

Signature:

```
1  herr_t (*read)(void *attr, hid_t mem_type_id, void *buf,
2      hid_t dxpl_id, void **req);
```

Arguments:

attr	(IN): Pointer to the attribute object.
mem_type_id	(IN): The memory datatype of the attribute.
buf	(OUT): Data buffer to be read into.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.4.4 attr: write

The `write` callback in the attribute class writes data to the attribute object and returns an `herr_t` indicating success or failure.

Signature:

```
1  herr_t (*write)(void *attr, hid_t mem_type_id, const void *buf,
2      hid_t dxpl_id, void **req);
```

Arguments:

attr	(IN): Pointer to the attribute object.
mem_type_id	(IN): The memory datatype of the attribute.
buf	(IN): Data buffer to be written.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.4.5 attr: get

The `get` callback in the attribute class retrieves information about the attribute as specified in the `get_type` parameter.It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*get)(void *obj, H5VL_attr_get_args_t *args, hid_t dxpl_id,
    void **req);
```

Arguments:

obj	(IN): An attribute or location object where information needs to be retrieved from.
args	(IN/OUT): A pointer to the arguments struct.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
/* Values for attribute 'get' operations */
typedef enum H5VL_attr_get_t {
    H5VL_ATTR_GET_ACPL,       /* creation property list          */
    H5VL_ATTR_GET_INFO,       /* info                            */
    H5VL_ATTR_GET_NAME,       /* access property list            */
    H5VL_ATTR_GET_SPACE,      /* dataspace                       */
    H5VL_ATTR_GET_STORAGE_SIZE, /* storage size                  */
    H5VL_ATTR_GET_TYPE        /* datatype                        */
} H5VL_attr_get_t;

/* Parameters for attribute 'get_name' operation */
typedef struct H5VL_attr_get_name_args_t {
    H5VL_loc_params_t loc_params; /* Location parameters for object access */
    size_t           buf_size;    /* Size of attribute name buffer */
    char *           buf;         /* Buffer for attribute name (OUT) */
    size_t *         attr_name_len; /* Actual length of attribute name (OUT) */
} H5VL_attr_get_name_args_t;

/* Parameters for attribute 'get_info' operation */
typedef struct H5VL_attr_get_info_args_t {
    H5VL_loc_params_t loc_params; /* Location parameters for object access */
    const char *     attr_name; /* Attribute name (for get_info_by_name) */
    H5A_info_t *     ainfo;     /* Attribute info (OUT) */
} H5VL_attr_get_info_args_t;

/* Parameters for attribute 'get' operations */
typedef struct H5VL_attr_get_args_t {
    H5VL_attr_get_t op_type; /* Operation to perform */

    /* Parameters for each operation */
    union {
        /* H5VL_ATTR_GET_ACPL */
        struct {
            hid_t acpl_id; /* Attribute creation property list ID (OUT) */
        } get_acpl;

        /* H5VL_ATTR_GET_INFO */
        H5VL_attr_get_info_args_t get_info; /* Attribute info */

```

```
        /* H5VL_ATTR_GET_NAME */
        H5VL_attr_get_name_args_t get_name; /* Attribute name */

        /* H5VL_ATTR_GET_SPACE */
        struct {
            hid_t space_id; /* Dataspace ID (OUT) */
        } get_space;

        /* H5VL_ATTR_GET_STORAGE_SIZE */
        struct {
            hsize_t *data_size; /* Size of attribute in file (OUT) */
        } get_storage_size;

        /* H5VL_ATTR_GET_TYPE */
        struct {
            hid_t type_id; /* Datatype ID (OUT) */
        } get_type;
    } args;
} H5VL_attr_get_args_t;
```

3.4.6 attr: specific

The `specific` callback in the attribute class implements specific operations on HDF5 attributes as specified in the `specific_type` parameter. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*specific)(void *obj, H5VL_loc_params_t *loc_params, H5VL_attr_specific_args_t
    *args, hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): The location object where the operation needs to happen.
`loc_params`	(IN): A pointer to the location parameters as explained in Section 3.1.
`args`	(IN/OUT): A pointer to the arguments struct.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
/* Values for attribute 'specific' operation */
typedef enum H5VL_attr_specific_t {
    H5VL_ATTR_DELETE,      /* H5Adelete(_by_name) */
    H5VL_ATTR_DELETE_BY_IDX, /* H5Adelete_by_idx */
    H5VL_ATTR_EXISTS,      /* H5Aexists(_by_name) */
    H5VL_ATTR_ITER,        /* H5Aiterate(_by_name) */
    H5VL_ATTR_RENAME       /* H5Arename(_by_name) */
} H5VL_attr_specific_t;

/* Parameters for attribute 'iterate' operation */
typedef struct H5VL_attr_iterate_args_t {
    H5_index_t     idx_type; /* Type of index to iterate over */
    H5_iter_order_t order;  /* Order of index iteration */
    hsize_t *      idx;     /* Start/stop iteration index (IN/OUT) */
    H5A_operator2_t op;     /* Iteration callback function */
    void *         op_data; /* Iteration callback context */
} H5VL_attr_iterate_args_t;

/* Parameters for attribute 'delete_by_idx' operation */
typedef struct H5VL_attr_delete_by_idx_args_t {
    H5_index_t     idx_type; /* Type of index to iterate over */
    H5_iter_order_t order;  /* Order of index iteration */
    hsize_t        n;       /* Iteration index */
} H5VL_attr_delete_by_idx_args_t;

/* Parameters for attribute 'specific' operations */
```

```c
typedef struct H5VL_attr_specific_args_t {
    H5VL_attr_specific_t op_type; /* Operation to perform */

    /* Parameters for each operation */
    union {
        /* H5VL_ATTR_DELETE */
        struct {
            const char *name; /* Name of attribute to delete */
        } del;

        /* H5VL_ATTR_DELETE_BY_IDX */
        H5VL_attr_delete_by_idx_args_t delete_by_idx;

        /* H5VL_ATTR_EXISTS */
        struct {
            const char *name; /* Name of attribute to check */
            hbool_t *  exists; /* Whether attribute exists (OUT) */
        } exists;

        /* H5VL_ATTR_ITER */
        H5VL_attr_iterate_args_t iterate;

        /* H5VL_ATTR_RENAME */
        struct {
            const char *old_name; /* Name of attribute to rename */
            const char *new_name; /* New attribute name */
        } rename;
    } args;
} H5VL_attr_specific_args_t;
```

3.4.7 attr: optional

The `optional` callback in the attribute class implements connector specific operations on an HDF5 attribute. It returns an `herr_t` indicating success or failure.

Signature:

```c
herr_t (*optional)(void *obj, H5VL_optional_args_t *args, hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): The container or object where the operation needs to happen.
`args`	(IN/OUT): A pointer to the arguments struct.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

Each connector that requires connector-specific operations should compare the value of the `op_type` field of the `H5VL_optional_args_t` struct with the values returned from calling `H5VLregister_opt_operation` to determine how to handle the optional call and interpret the arguments passed in the struct.

3.4.8 attr: close

The `close` callback in the attribute class terminates access to the attribute object and free all resources it was consuming, and returns an `herr_t` indicating success or failure.

Signature:

```c
herr_t (*close)(void *attr, hid_t dxpl_id, void **req);
```

Arguments:

attr	(IN): Pointer to the attribute object.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.5 Dataset Callbacks

The dataset API routines (H5D) allow HDF5 users to create and manage HDF5 datasets. All the H5D API routines that modify the HDF5 container map to one of the dataset callback routines in this class that the connector needs to implement.

```
typedef struct H5VL_dataset_class_t {
    void *(*create)(void *obj, const H5VL_loc_params_t *loc_params, const char *name, hid_t
        lcpl_id, hid_t type_id, hid_t space_id, hid_t dcpl_id, hid_t dapl_id, hid_t dxpl_id, void
        **req);
    void *(*open)(void *obj, const H5VL_loc_params_t *loc_params, const char *name, hid_t dapl_id,
        hid_t dxpl_id, void **req);
    herr_t (*read)(void *dset, hid_t mem_type_id, hid_t mem_space_id, hid_t file_space_id, hid_t
        dxpl_id, void * buf, void **req);
    herr_t (*write)(void *dset, hid_t mem_type_id, hid_t mem_space_id, hid_t file_space_id, hid_t
        dxpl_id, const void * buf, void **req);
    herr_t (*get)(void *obj, H5VL_dataset_get_args_t *args, hid_t dxpl_id, void **req);
    herr_t (*specific)(void *obj, H5VL_dataset_specific_args_t *args, hid_t dxpl_id, void **req);
    herr_t (*optional)(void *obj, H5VL_optional_args_t *args, hid_t dxpl_id, void **req);
    herr_t (*close) (void *dset, hid_t dxpl_id, void **req);
} H5VL_dataset_class_t;
```

Listing 6: Structure for dataset callback routines, H5VLconnector.h

3.5.1 dataset: create

The `create` callback in the dataset class creates a dataset object in the container of the location object and returns a pointer to the dataset structure containing information to access the dataset in future calls.

Signature:

```
    void *(*create)(void *obj, H5VL_loc_params_t *loc_params, const char *name, hid_t lcpl_id,
        hid_t type_id, hid_t space_id, hid_t dcpl_id, hid_t dapl_id, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): Pointer to an object where the dataset needs to be created or where the look-up of the target object needs to start.
loc_params	(IN): Pointer to the location parameters as explained in Section 3.1. The type can be only `H5VL_OBJECT_BY_SELF` in this callback.
name	(IN): The name of the dataset to be created.
lcpl_id	(IN): The link creation property list.
type_id	(IN): The datatype of the dataset.
space_id	(IN): The dataspace of the dataset.
dcpl_id	(IN): The dataset creation property list.
dapl_id	(IN): The dataset access property list.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.5.2 dataset: open

The `open` callback in the dataset class opens a dataset object in the container of the location object and returns a pointer to the dataset structure containing information to access the dataset in future calls.

Signature:

```
    void *(*open)(void *obj, H5VL_loc_params_t *loc_params, const char *name, hid_t dapl_id,
```

```
        hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): Pointer to an object where the dataset needs to be opened or where the look-up of the target object needs to start.
loc_params	(IN): Pointer to the location parameters as explained in Section 3.1. The type can be only H5VL_OBJECT_BY_SELF in this callback.
name	(IN): The name of the dataset to be opened.
dapl_id	(IN): The dataset access property list.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.5.3 dataset: read

The read callback in the dataset class reads data from the dataset object and returns an herr_t indicating success or failure.

Signature:

```
1    herr_t (*read)(void *dset, hid_t mem_type_id, hid_t mem_space_id,
2        hid_t file_space_id, hid_t dxpl_id, void *buf, void **req);
```

Arguments:

dset	(IN): Pointer to the dataset object.
mem_type_id	(IN): The memory datatype of the data.
mem_space_id	(IN): The memory dataspace selection.
file_space_id	(IN): The file dataspace selection.
dxpl_id	(IN): The data transfer property list.
buf	(OUT): Data buffer to be read into.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.5.4 dataset: write

The write callback in the dataset class writes data to the dataset object and returns an herr_t indicating success or failure.

Signature:

```
1    herr_t (*write)(void *dset, hid_t mem_type_id, hid_t mem_space_id,
2        hid_t file_space_id, hid_t dxpl_id, const void *buf, void **req);
```

Arguments:

dset	(IN): Pointer to the dataset object.
mem_type_id	(IN): The memory datatype of the data.
mem_space_id	(IN): The memory dataspace selection.
file_space_id	(IN): The file dataspace selection.
dxpl_id	(IN): The data transfer property list.
buf	(IN): Data buffer to be written from.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.5.5 dataset: get

The get callback in the dataset class retrieves information about the dataset as specified in the get_type parameter.It returns an herr_t indicating success or failure.

Signature:

```
herr_t (*get)(void *dset, H5VL_dataset_get_args_t *args,
    hid_t dxpl_id, void **req);
```

Arguments:

dset	(IN): The dataset object where information needs to be retrieved from.
args	(IN/OUT): A pointer to the arguments struct.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
/* Values for dataset 'get' operation */
typedef enum H5VL_dataset_get_t {
    H5VL_DATASET_GET_DAPL,      /* access property list           */
    H5VL_DATASET_GET_DCPL,      /* creation property list         */
    H5VL_DATASET_GET_SPACE,     /* dataspace                      */
    H5VL_DATASET_GET_SPACE_STATUS, /* space status                */
    H5VL_DATASET_GET_STORAGE_SIZE, /* storage size                */
    H5VL_DATASET_GET_TYPE       /* datatype                       */
} H5VL_dataset_get_t;

/* Parameters for dataset 'get' operations */
typedef struct H5VL_dataset_get_args_t {
    H5VL_dataset_get_t op_type; /* Operation to perform */

    /* Parameters for each operation */
    union {
        /* H5VL_DATASET_GET_DAPL */
        struct {
            hid_t dapl_id; /* Dataset access property list ID (OUT) */
        } get_dapl;

        /* H5VL_DATASET_GET_DCPL */
        struct {
            hid_t dcpl_id; /* Dataset creation property list ID (OUT) */
        } get_dcpl;

        /* H5VL_DATASET_GET_SPACE */
        struct {
            hid_t space_id; /* Dataspace ID (OUT) */
        } get_space;

        /* H5VL_DATASET_GET_SPACE_STATUS */
        struct {
            H5D_space_status_t *status; /* Storage space allocation status (OUT) */
        } get_space_status;

        /* H5VL_DATASET_GET_STORAGE_SIZE */
        struct {
            hsize_t *storage_size; /* Size of dataset's storage (OUT) */
        } get_storage_size;

        /* H5VL_DATASET_GET_TYPE */
        struct {
            hid_t type_id; /* Datatype ID (OUT) */
        } get_type;
    } args;
} H5VL_dataset_get_args_t;
```

3.5.6 dataset: specific

The `specific` callback in the dataset class implements specific operations on HDF5 datasets as specified in the `specific_type` parameter. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*specific)(void *obj, H5VL_file_specific_args_t *args, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): The dset where the operation needs to happen.
args	(IN/OUT): A pointer to the arguments struct.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
/* Values for dataset 'specific' operation */
typedef enum H5VL_dataset_specific_t {
    H5VL_DATASET_SET_EXTENT, /* H5Dset_extent                */
    H5VL_DATASET_FLUSH,    /* H5Dflush                       */
    H5VL_DATASET_REFRESH   /* H5Drefresh                     */
} H5VL_dataset_specific_t;

/* Parameters for dataset 'specific' operations */
typedef struct H5VL_dataset_specific_args_t {
    H5VL_dataset_specific_t op_type; /* Operation to perform */

    /* Parameters for each operation */
    union {
        /* H5VL_DATASET_SET_EXTENT */
        struct {
            const hsize_t *size; /* New dataspace extent */
        } set_extent;

        /* H5VL_DATASET_FLUSH */
        struct {
            hid_t dset_id; /* Dataset ID (IN) */
        } flush;

        /* H5VL_DATASET_REFRESH */
        struct {
            hid_t dset_id; /* Dataset ID (IN) */
        } refresh;
    } args;
} H5VL_dataset_specific_args_t;
```

3.5.7 dataset: optional

The `optional` callback in the dataset class implements connector specific operations on an HDF5 dataset. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*optional)(void *obj, H5VL_optional_args_t *args, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): The container or object where the operation needs to happen.
args	(IN/OUT): A pointer to the arguments struct.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

Each connector that requires connector-specific operations should compare the value of the `op_type` field of the `H5VL_optional_args_t` struct with the values returned from calling `H5VLregister_opt_operation` to determine how to handle the optional call and interpret the arguments passed in the struct.

3.5.8 dataset: close

The `close` callback in the dataset class terminates access to the dataset object and free all resources it was consuming and returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*close)(void *dset, hid_t dxpl_id, void **req);
```

Arguments:

`dset`	(IN): Pointer to the dataset object.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.6 Datatype Callbacks

The HDF5 datatype routines (H5T) allow users to create and manage HDF5 datatypes. Those routines are divided into two categories. One that operates on all types of datatypes but do not modify the contents of the container (all in memory), and others that operate on named datatypes by accessing the container. When a user creates an HDF5 datatype, it is still an object in memory space (transient datatype) that has not been added to the HDF5 containers. Only when a user commits the HDF5 datatype, it becomes persistent in the container. Those are called named/committed datatypes. The transient H5T routines should work on named datatypes nevertheless.

All the H5T API routines that modify the HDF5 container map to one of the named datatype callback routines in this class that the connector needs to implement.

```
typedef struct H5VL_datatype_class_t {
    void *(*commit)(void *obj, const H5VL_loc_params_t *loc_params, const char *name, hid_t
        type_id, hid_t lcpl_id, hid_t tcpl_id, hid_t tapl_id, hid_t dxpl_id, void **req);
    void *(*open)(void *obj, const H5VL_loc_params_t *loc_params, const char * name, hid_t
        tapl_id, hid_t dxpl_id, void **req);
    herr_t (*get)  (void *obj, H5VL_datatype_get_args_t *args, hid_t dxpl_id, void **req);
    herr_t (*specific)(void *obj, H5VL_datatype_specific_args_t *args, hid_t dxpl_id, void **req);
    herr_t (*optional)(void *obj, H5VL_optional_args_t *args, hid_t dxpl_id, void **req);
    herr_t (*close) (void *dt, hid_t dxpl_id, void **req);
} H5VL_datatype_class_t;
```

Listing 7: Structure for datatype callback routines, H5VLconnector.h

3.6.1 datatype: commit

The `commit` callback in the named datatype class creates a datatype object in the container of the location object and returns a pointer to the datatype structure containing information to access the datatype in future calls.

Signature:

```
void *(*commit)(void *obj, H5VL_loc_params_t *loc_params,
    const char *name, hid_t type_id, hid_t lcpl_id, hid_t tcpl_id,
    hid_t tapl_id, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): Pointer to an object where the datatype needs to be committed or where the look-up of the target object needs to start.
loc_params	(IN): Pointer to location parameters as explained in Section 3.1. In this call, the location type is always H5VL_OBJECT_BY_SELF.
name	(IN): The name of the datatype to be created.
type_id	(IN): The transient datatype identifier to be committed.
lcpl_id	(IN): The link creation property list.
tcpl_id	(IN): The datatype creation property list.
tapl_id	(IN): The datatype access property list.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.6.2 datatype: open

The open callback in the named datatype class opens a previously committed datatype object in the container of the location object and returns a pointer to the datatype structure containing information to access the datatype in future calls.

Signature:

```
1    void *(*open) (void *obj, H5VL_loc_params_t *loc_params,
2        const char * name, hid_t tapl_id, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): Pointer to an object where the datatype needs to be opened or where the look-up of the target object needs to start.
loc_params	(IN): Pointer to location parameters as explained in Section 3.1. In this call, the location type is always H5VL_OBJECT_BY_SELF.
name	(IN): The name of the datatype to be opened.
tapl_id	(IN): The datatype access property list.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.6.3 datatype: get

The get callback in the named datatype class retrieves information about the named datatype as specified in the get_type parameter.It returns an herr_t indicating success or failure.

Signature:

```
1    herr_t (*get) (void *obj, H5VL_datatype_get_args_t *args, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): The named datatype to retrieve information from.
args	(IN/OUT): A pointer to the arguments struct.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
1  /* Values for datatype 'get' operation */
2  typedef enum H5VL_datatype_get_t {
3      H5VL_DATATYPE_GET_BINARY_SIZE, /* Get size of serialized form of transient type */
4      H5VL_DATATYPE_GET_BINARY,   /* Get serialized form of transient type    */
5      H5VL_DATATYPE_GET_TCPL      /* Datatype creation property list          */
6  } H5VL_datatype_get_t;
7
8  /* Parameters for datatype 'get' operations */
9  typedef struct H5VL_datatype_get_args_t {
10     H5VL_datatype_get_t op_type; /* Operation to perform */
```

```
11
12      /* Parameters for each operation */
13      union {
14          /* H5VL_DATATYPE_GET_BINARY_SIZE */
15          struct {
16              size_t *size; /* Size of serialized form of datatype (OUT) */
17          } get_binary_size;
18
19          /* H5VL_DATATYPE_GET_BINARY */
20          struct {
21              void * buf;     /* Buffer to store serialized form of datatype (OUT) */
22              size_t buf_size; /* Size of serialized datatype buffer */
23          } get_binary;
24
25          /* H5VL_DATATYPE_GET_TCPL */
26          struct {
27              hid_t tcpl_id; /* Named datatype creation property list ID (OUT) */
28          } get_tcpl;
29      } args;
30  } H5VL_datatype_get_args_t;
```

3.6.4 datatype: specific

The `specific` callback in the datatype class implements specific operations on HDF5 named datatypes as specified in the `specific_type` parameter. It returns an `herr_t` indicating success or failure.

Signature:

```
1       herr_t (*specific)(void *obj, H5VL_loc_params_t *loc_params, H5VL_object_specific_args_t
            *args, hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): The container or object where the operation needs to happen.
`loc_params`	(IN): Pointer to location parameters as explained in Section 3.1.
`args`	(IN/OUT): A pointer to the arguments struct.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
1   /* Values for datatype 'specific' operation */
2   typedef enum H5VL_datatype_specific_t {
3       H5VL_DATATYPE_FLUSH, /* H5Tflush */
4       H5VL_DATATYPE_REFRESH /* H5Trefresh */
5   } H5VL_datatype_specific_t;
6
7   /* Parameters for datatype 'specific' operations */
8   typedef struct H5VL_datatype_specific_args_t {
9       H5VL_datatype_specific_t op_type; /* Operation to perform */
10
11      /* Parameters for each operation */
12      union {
13          /* H5VL_DATATYPE_FLUSH */
14          struct {
15              hid_t type_id; /* Named datatype ID (IN) */
16          } flush;
17
18          /* H5VL_DATATYPE_REFRESH */
19          struct {
20              hid_t type_id; /* Named datatype ID (IN) */
21          } refresh;
22      } args;
23  } H5VL_datatype_specific_args_t;
```

3.6.5 datatype: optional

The `optional` callback in the datatype class implements connector specific operations on an HDF5 datatype. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*optional)(void *obj, H5VL_optional_args_t *args, hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): The container or object where the operation needs to happen.
`args`	(IN/OUT): A pointer to the arguments struct.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

Each connector that requires connector-specific operations should compare the value of the `op_type` field of the `H5VL_optional_args_t` struct with the values returned from calling `H5VLregister_opt_operation` to determine how to handle the optional call and interpret the arguments passed in the struct.

3.6.6 datatype: close

The `close` callback in the named datatype class terminates access to the datatype object and free all resources it was consuming and returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*close) (void *dt, hid_t dxpl_id, void **req);
```

Arguments:

`dt`	(IN): Pointer to the datatype object.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.7 File Callbacks

The file API routines (H5F) allow HDF5 users to create and manage HDF5 containers. All the H5F API routines that modify the HDF5 container map to one of the file callback routines in this class that the connector needs to implement.

```
typedef struct H5VL_file_class_t {
    void *(*create)(const char *name, unsigned flags, hid_t fcpl_id, hid_t fapl_id, hid_t dxpl_id,
        void **req);
    void *(*open)(const char *name, unsigned flags, hid_t fapl_id, hid_t dxpl_id, void **req);
    herr_t (*get)(void *obj, H5VL_file_get_args_t *args, hid_t dxpl_id, void **req);
    herr_t (*specific)(void *obj, H5VL_file_specific_args_t *args, hid_t dxpl_id, void **req);
    herr_t (*optional)(void *obj, H5VL_optional_args_t *args, hid_t dxpl_id, void **req);
    herr_t (*close) (void *file, hid_t dxpl_id, void **req);
} H5VL_file_class_t;
```

Listing 8: File class for file API routines, H5VLconnector.h

3.7.1 file: create

The `create` callback in the file class should create a container and returns a pointer to the file structure created by the connector containing information to access the container in future calls.

Signature:

```
void *(*create)(const char *name, unsigned flags, hid_t fcpl_id, hid_t fapl_id, hid_t
    dxpl_id, void **req);
```

Arguments:

name	(IN): The name of the container to be created.
flags	(IN): The creation flags of the container.
fcpl_id	(IN): The file creation property list.
fapl_id	(IN): The file access property list.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.7.2 file: open

The open callback in the file class should open a container and returns a pointer to the file structure created by the connector containing information to access the container in future calls.

Signature:

```
1    void *(*open)(const char *name, unsigned flags, hid_t fapl_id, hid_t dxpl_id, void **req);
```

Arguments:

name	(IN): The name of the container to open.
flags	(IN): The open flags of the container.
fapl_id	(IN): The file access property list.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.7.3 file: get

The get callback in the file class should retrieve information about the container as specified in the get_type parameter. It returns an herr_t indicating success or failure.

Signature:

```
1    herr_t (*get)(void *obj, H5VL_file_get_args_t *args, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): The container or object where information needs to be retrieved from.
args	(IN/OUT): A pointer to the arguments struct.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
1  /* Info for H5VL_FILE_GET_CONT_INFO */
2  typedef struct H5VL_file_cont_info_t {
3      unsigned version;     /* version information (keep first) */
4      uint64_t feature_flags; /* Container feature flags    */
5                            /* (none currently defined)     */
6      size_t token_size;    /* Size of tokens               */
7      size_t blob_id_size;  /* Size of blob IDs             */
8  } H5VL_file_cont_info_t;
9
10 /* Values for file 'get' operation */
11 typedef enum H5VL_file_get_t {
12     H5VL_FILE_GET_CONT_INFO, /* file get container info       */
13     H5VL_FILE_GET_FAPL,    /* file access property list       */
14     H5VL_FILE_GET_FCPL,    /* file creation property list     */
15     H5VL_FILE_GET_FILENO,  /* file number                     */
16     H5VL_FILE_GET_INTENT,  /* file intent                     */
17     H5VL_FILE_GET_NAME,    /* file name                       */
18     H5VL_FILE_GET_OBJ_COUNT, /* object count in file          */
19     H5VL_FILE_GET_OBJ_IDS  /* object ids in file              */
```

```
} H5VL_file_get_t;

/* Parameters for file 'get_name' operation */
typedef struct H5VL_file_get_name_args_t {
    H5I_type_t type;        /* ID type of object pointer */
    size_t    buf_size;     /* Size of file name buffer (IN) */
    char *    buf;          /* Buffer for file name (OUT) */
    size_t *  file_name_len; /* Actual length of file name (OUT) */
} H5VL_file_get_name_args_t;

/* Parameters for file 'get_obj_ids' operation */
typedef struct H5VL_file_get_obj_ids_args_t {
    unsigned types;  /* Type of objects to count */
    size_t  max_objs; /* Size of array of object IDs */
    hid_t * oid_list; /* Array of object IDs (OUT) */
    size_t * count;  /* # of objects (OUT) */
} H5VL_file_get_obj_ids_args_t;

/* Parameters for file 'get' operations */
typedef struct H5VL_file_get_args_t {
    H5VL_file_get_t op_type; /* Operation to perform */

    /* Parameters for each operation */
    union {
        /* H5VL_FILE_GET_CONT_INFO */
        struct {
            H5VL_file_cont_info_t *info; /* Container info (OUT) */
        } get_cont_info;

        /* H5VL_FILE_GET_FAPL */
        struct {
            hid_t fapl_id; /* File access property list (OUT) */
        } get_fapl;

        /* H5VL_FILE_GET_FCPL */
        struct {
            hid_t fcpl_id; /* File creation property list (OUT) */
        } get_fcpl;

        /* H5VL_FILE_GET_FILENO */
        struct {
            unsigned long *fileno; /* File "number" (OUT) */
        } get_fileno;

        /* H5VL_FILE_GET_INTENT */
        struct {
            unsigned *flags; /* File open/create intent flags (OUT) */
        } get_intent;

        /* H5VL_FILE_GET_NAME */
        H5VL_file_get_name_args_t get_name;

        /* H5VL_FILE_GET_OBJ_COUNT */
        struct {
            unsigned types; /* Type of objects to count */
            size_t * count; /* # of objects (OUT) */
        } get_obj_count;

        /* H5VL_FILE_GET_OBJ_IDS */
        H5VL_file_get_obj_ids_args_t get_obj_ids;
    } args;
} H5VL_file_get_args_t;
```

3.7.4 file: specific

The `specific` callback in the file class implements specific operations on HDF5 files as specified in the `specific_type` parameter. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*specific)(void *obj, H5VL_file_specific_args_t *args, hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): The container or object where the operation needs to happen.
`args`	(IN/OUT): A pointer to the arguments struct.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
/* Values for file 'specific' operation */
typedef enum H5VL_file_specific_t {
    H5VL_FILE_FLUSH,       /* Flush file                    */
    H5VL_FILE_REOPEN,      /* Reopen the file               */
    H5VL_FILE_IS_ACCESSIBLE, /* Check if a file is accessible */
    H5VL_FILE_DELETE,      /* Delete a file                 */
    H5VL_FILE_IS_EQUAL     /* Check if two files are the same */
} H5VL_file_specific_t;

/* Parameters for file 'specific' operations */
typedef struct H5VL_file_specific_args_t {
    H5VL_file_specific_t op_type; /* Operation to perform */

    /* Parameters for each operation */
    union {
        /* H5VL_FILE_FLUSH */
        struct {
            H5I_type_t obj_type; /* Type of object to use */
            H5F_scope_t scope;  /* Scope of flush operation */
        } flush;

        /* H5VL_FILE_REOPEN */
        struct {
            void **file; /* File object for new file (OUT) */
        } reopen;

        /* H5VL_FILE_IS_ACCESSIBLE */
        struct {
            const char *filename; /* Name of file to check */
            hid_t      fapl_id;   /* File access property list to use */
            hbool_t *  accessible; /* Whether file is accessible with FAPL settings (OUT) */
        } is_accessible;

        /* H5VL_FILE_DELETE */
        struct {
            const char *filename; /* Name of file to delete */
            hid_t      fapl_id; /* File access property list to use */
        } del;

        /* H5VL_FILE_IS_EQUAL */
        struct {
            void *  obj2;     /* Second file object to compare against */
            hbool_t *same_file; /* Whether files are the same (OUT) */
        } is_equal;
    } args;
} H5VL_file_specific_args_t;
```

3.7.5 file: optional

The `optional` callback in the file class implements connector specific operations on an HDF5 container. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*optional)(void *obj, H5VL_optional_args_t *args, hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): The container or object where the operation needs to happen.
`args`	(IN/OUT): A pointer to the arguments struct.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

Each connector that requires connector-specific operations should compare the value of the `op_type` field of the `H5VL_optional_args_t` struct with the values returned from calling `H5VLregister_opt_operation` to determine how to handle the optional call and interpret the arguments passed in the struct.

3.7.6 file: close

The `close` callback in the file class should terminate access to the file object and free all resources it was consuming, and returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*close)(void *file, hid_t dxpl_id, void **req);
```

Arguments:

`file`	(IN): Pointer to the file.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.8 Group Callbacks

The group API routines (H5G) allow HDF5 users to create and manage HDF5 groups. All the H5G API routines that modify the HDF5 container map to one of the group callback routines in this class that the connector needs to implement.

```
typedef struct H5VL_group_class_t {
    void *(*create)(void *obj, const H5VL_loc_params_t *loc_params, const char *name, hid_t
        lcpl_id, hid_t gcpl_id, hid_t gapl_id, hid_t dxpl_id, void **req);
    void *(*open)(void *obj, const H5VL_loc_params_t *loc_params, const char *name, hid_t gapl_id,
        hid_t dxpl_id, void **req);
    herr_t (*get)(void *obj, H5VL_group_get_args_t *args, hid_t dxpl_id, void **req);
    herr_t (*specific)(void *obj, H5VL_group_specific_args_t *args, hid_t dxpl_id, void **req);
    herr_t (*optional)(void *obj, H5VL_optional_args_t *args, hid_t dxpl_id, void **req);
    herr_t (*close) (void *grp, hid_t dxpl_id, void **req);
} H5VL_group_class_t;
```

Listing 9: Structure for group callback routines, H5VLconnector.h

3.8.1 group: create

The `create` callback in the group class creates a group object in the container of the location object and returns a pointer to the group structure containing information to access the group in future calls.

Signature:

```
1    void *(*create)(void *obj, H5VL_loc_params_t *loc_params, const char *name, hid_t gcpl_id,
         hid_t gapl_id, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): Pointer to an object where the group needs to be created or where the look-up of the target object needs to start.
loc_params	(IN): Pointer to the location parameters as explained in Section 3.1. The type can be only H5VL_OBJECT_BY_SELF in this callback.
name	(IN): The name of the group to be created.
dcpl_id	(IN): The group creation property list. It contains all the group creation properties in addition to the link creation property list of the create operation (an hid_t) that can be retrieved with the property H5VL_GRP_LCPL_ID.
gapl_id	(IN): The group access property list.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.8.2 group: open

The open callback in the group class opens a group object in the container of the location object and returns a pointer to the group structure containing information to access the group in future calls.

Signature:

```
1    void *(*open)(void *obj, H5VL_loc_params_t *loc_params,
2        const char*name, hid_t gapl_id, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): Pointer to an object where the group needs to be opened or where the look-up of the target object needs to start.
loc_params	(IN): Pointer to the location parameters as explained in Section 3.1. The type can be only H5VL_OBJECT_BY_SELF in this callback.
name	(IN): The name of the group to be opened.
dapl_id	(IN): The group access property list.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.8.3 group: get

The get callback in the group class retrieves information about the group as specified in the get_type parameter. It returns an herr_t indicating success or failure.

Signature:

```
1    herr_t (*get)(void *obj, H5VL_group_get_args_t *args, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): The group object where information needs to be retrieved from.
args	(IN/OUT): A pointer to the arguments struct.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
1  /* Values for group 'get' operation */
2  typedef enum H5VL_group_get_t {
3      H5VL_GROUP_GET_GCPL, /* group creation property list */
4      H5VL_GROUP_GET_INFO /* group info                    */
5  } H5VL_group_get_t;
```

```
6
7  /* Parameters for group 'get_info' operation */
8  typedef struct H5VL_group_get_info_args_t {
9      H5VL_loc_params_t loc_params; /* Location parameters for object access */
10     H5G_info_t *    ginfo;     /* Group info (OUT) */
11 } H5VL_group_get_info_args_t;
12
13 /* Parameters for group 'get' operations */
14 typedef struct H5VL_group_get_args_t {
15     H5VL_group_get_t op_type; /* Operation to perform */
16
17     /* Parameters for each operation */
18     union {
19         /* H5VL_GROUP_GET_GCPL */
20         struct {
21             hid_t gcpl_id; /* Group creation property list (OUT) */
22         } get_gcpl;
23
24         /* H5VL_GROUP_GET_INFO */
25         H5VL_group_get_info_args_t get_info; /* Group info */
26     } args;
27 } H5VL_group_get_args_t;
```

3.8.4 group: specific

The `specific` callback in the group class implements specific operations on HDF5 groups as specified in the `specific_type` parameter. It returns an `herr_t` indicating success or failure.

Signature:

```
1        herr_t (*specific)(void *obj, H5VL_loc_params_t *loc_params, H5VL_object_specific_args_t
             *args, hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): The container or object where the operation needs to happen.
`loc_params`	(IN): Pointer to the location parameters as explained in Section 3.1.
`args`	(IN/OUT): A pointer to the arguments struct.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
1  /* Values for group 'specific' operation */
2  typedef enum H5VL_group_specific_t {
3      H5VL_GROUP_MOUNT, /* Mount a file on a group      */
4      H5VL_GROUP_UNMOUNT, /* Unmount a file on a group  */
5      H5VL_GROUP_FLUSH, /* H5Gflush                     */
6      H5VL_GROUP_REFRESH /* H5Grefresh                  */
7  } H5VL_group_specific_t;
8
9  /* Parameters for group 'mount' operation */
10 typedef struct H5VL_group_spec_mount_args_t {
11     const char *name;     /* Name of location to mount child file */
12     void *     child_file; /* Pointer to child file object */
13     hid_t      fmpl_id;   /* File mount property list to use */
14 } H5VL_group_spec_mount_args_t;
15
16 /* Parameters for group 'specific' operations */
17 typedef struct H5VL_group_specific_args_t {
18     H5VL_group_specific_t op_type; /* Operation to perform */
19
20     /* Parameters for each operation */
21     union {
22         /* H5VL_GROUP_MOUNT */
23         H5VL_group_spec_mount_args_t mount;
```

```
24
25          /* H5VL_GROUP_UNMOUNT */
26          struct {
27              const char *name; /* Name of location to unmount child file */
28          } unmount;
29
30          /* H5VL_GROUP_FLUSH */
31          struct {
32              hid_t grp_id; /* Group ID (IN) */
33          } flush;
34
35          /* H5VL_GROUP_REFRESH */
36          struct {
37              hid_t grp_id; /* Group ID (IN) */
38          } refresh;
39      } args;
40  } H5VL_group_specific_args_t;
```

3.8.5 group: optional

The `optional` callback in the group class implements connector specific operations on an HDF5 group. It returns an `herr_t` indicating success or failure.

Signature:

```
1      herr_t (*optional)(void *obj, H5VL_optional_args_t *args, hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): The container or object where the operation needs to happen.
`args`	(IN/OUT): A pointer to the arguments struct.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

Each connector that requires connector-specific operations should compare the value of the `op_type` field of the `H5VL_optional_args_t` struct with the values returned from calling `H5VLregister_opt_operation` to determine how to handle the optional call and interpret the arguments passed in the struct.

3.8.6 group: close

The `close` callback in the group class terminates access to the group object and frees all resources it was consuming, and returns an `herr_t` indicating success or failure.

Signature:

```
1      herr_t (*close)(void *group, hid_t dxpl_id, void **req);
```

Arguments:

`group`	(IN): Pointer to the group object.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.9 Link Callbacks

The link API routines (H5L) allow HDF5 users to create and manage HDF5 links. All the H5L API routines that modify the HDF5 container map to one of the link callback routines in this class that the connector needs to implement.

```
1   typedef struct H5VL_link_class_t {
2       herr_t (*create)(H5VL_link_create_args_t *args, void *obj, const H5VL_loc_params_t
            *loc_params, hid_t lcpl_id, hid_t lapl_id, hid_t dxpl_id, void **req);
```

```
    herr_t (*copy)(void *src_obj, const H5VL_loc_params_t *loc_params1, void *dst_obj, const
        H5VL_loc_params_t *loc_params2, hid_t lcpl_id, hid_t lapl_id, hid_t dxpl_id, void **req);
    herr_t (*move)(void *src_obj, const H5VL_loc_params_t *loc_params1, void *dst_obj, const
        H5VL_loc_params_t *loc_params2, hid_t lcpl_id, hid_t lapl_id, hid_t dxpl_id, void **req);
    herr_t (*get)(void *obj, const H5VL_loc_params_t *loc_params, H5VL_link_get_args_t *args,
        hid_t dxpl_id, void **req);
    herr_t (*specific)(void *obj, const H5VL_loc_params_t *loc_params, H5VL_link_specific_args_t
        *args, hid_t dxpl_id, void **req);
    herr_t (*optional)(void *obj, const H5VL_loc_params_t *loc_params, H5VL_optional_args_t *args,
        hid_t dxpl_id, void **req);
} H5VL_link_class_t;
```

Listing 10: Structure for link callback routines, H5VLconnector.h

3.9.1 link: create

The `create` callback in the group class creates a hard, soft, external, or user-defined link in the container. It returns an `herr_t` indicating success or failure.

Signature:

```
    herr_t (*create)(H5VL_link_create_args_t *args, void *obj,
        H5VL_loc_params_t *loc_params, hid_t lcpl_id,
        hid_t lapl_id, hid_t dxpl_id, void **req);
```

Arguments:

`args`	(IN/OUT): A pointer to the arguments struct.
`obj`	(IN): Pointer to an object where the link needs to be created from.
`loc_params`	(IN): Pointer to the location parameters as explained in Section 3.1 for the source object.
`lcpl_id`	(IN): The link creation property list. It contains all the link creation properties in addition to other API parameters depending on the creation type, which will be detailed next.
`lapl_id`	(IN): The link access property list.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
/* Link create types for VOL */
typedef enum H5VL_link_create_t {
    H5VL_LINK_CREATE_HARD,
    H5VL_LINK_CREATE_SOFT,
    H5VL_LINK_CREATE_UD
} H5VL_link_create_t;

/* Parameters for link 'create' operations */
typedef struct H5VL_link_create_args_t {
    H5VL_link_create_t op_type; /* Operation to perform */

    /* Parameters for each operation */
    union {
        /* H5VL_LINK_CREATE_HARD */
        struct {
            void *          curr_obj;       /* Current object */
            H5VL_loc_params_t curr_loc_params; /* Location parameters for current object */
        } hard;

        /* H5VL_LINK_CREATE_SOFT */
        struct {
            const char *target; /* Target of soft link */
        } soft;

        /* H5VL_LINK_CREATE_UD */
        struct {
            H5L_type_t type;    /* Type of link to create */
```

```
            const void *buf;    /* Buffer that contains link info */
            size_t     buf_size; /* Size of link info buffer */
        } ud;
    } args;
} H5VL_link_create_args_t;
```

3.9.2 link: copy

The `copy` callback in the link class copies a link within the HDF5 container. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*copy)(void *src_obj, H5VL_loc_params_t *loc_params1, void *dst_obj,
    H5VL_loc_params_t *loc_params2, hid_t lcpl_id, hid_t lapl_id, hid_t dxpl_id, void
    **req);
```

Arguments:

`src_obj`	(IN): original/source object or file.
`loc_params1`	(IN): Pointer to the location parameters for the source object as explained in Section 3.1. The type can be only `H5VL_OBJECT_BY_NAME` in this callback.
`dst_obj`	(IN): destination object or file.
`loc_params1`	(IN): Pointer to the location parameters for the destination object as explained in Section 3.1. The type can be only `H5VL_OBJECT_BY_NAME` in this callback.
`lcpl_id`	(IN): The link creation property list.
`lapl_id`	(IN): The link access property list.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.9.3 link: move

The `move` callback in the link class moves a link within the HDF5 container. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*move)(void *src_obj, H5VL_loc_params_t *loc_params1, void *dst_obj,
    H5VL_loc_params_t *loc_params2, hid_t lcpl_id, hid_t lapl_id, hid_t dxpl_id, void
    **req);
```

Arguments:

`src_obj`	(IN): original/source object or file.
`loc_params1`	(IN): Pointer to the location parameters for the source object as explained in Section 3.1. The type can be only `H5VL_OBJECT_BY_NAME` in this callback.
`dst_obj`	(IN): destination object or file.
`loc_params1`	(IN): Pointer to the location parameters for the destination object as explained in Section 3.1. The type can be only `H5VL_OBJECT_BY_NAME` in this callback.
`lcpl_id`	(IN): The link creation property list.
`lapl_id`	(IN): The link access property list.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.9.4 link: get

The `get` callback in the link class retrieves information about links as specified in the `get_type` parameter. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*get)(void *obj, H5VL_loc_params_t *loc_params,
    H5VL_link_get_args_t *args, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): The file or group object where information needs to be retrieved from.
loc_params	(IN): Pointer to the location parameters for the source object as explained in Section 3.1. The type can be only H5VL_OBJECT_BY_NAME or H5VL_OBJECT_BY_IDX in this callback.
args	(IN/OUT): A pointer to the arguments struct.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
/* Values for link 'get' operation */
typedef enum H5VL_link_get_t {
    H5VL_LINK_GET_INFO, /* link info                    */
    H5VL_LINK_GET_NAME, /* link name                    */
    H5VL_LINK_GET_VAL /* link value                     */
} H5VL_link_get_t;

/* Parameters for link 'get' operations */
typedef struct H5VL_link_get_args_t {
    H5VL_link_get_t op_type; /* Operation to perform */

    /* Parameters for each operation */
    union {
        /* H5VL_LINK_GET_INFO */
        struct {
            H5L_info2_t *linfo; /* Pointer to link's info (OUT) */
        } get_info;

        /* H5VL_LINK_GET_NAME */
        struct {
            size_t name_size; /* Size of link name buffer (IN) */
            char * name;     /* Buffer for link name (OUT) */
            size_t *name_len; /* Actual length of link name (OUT) */
        } get_name;

        /* H5VL_LINK_GET_VAL */
        struct {
            size_t buf_size; /* Size of link value buffer (IN) */
            void * buf;     /* Buffer for link value (OUT) */
        } get_val;
    } args;
} H5VL_link_get_args_t;
```

3.9.5 link: specific

The specific callback in the link class implements specific operations on HDF5 links as specified in the specific_type parameter. It returns an herr_t indicating success or failure.

Signature:

```
herr_t (*specific)(void *obj, H5VL_loc_params_t *loc_params, H5VL_link_specific_args_t
    *args, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): The location object where the operation needs to happen.
loc_params	(IN): Pointer to the location parameters as explained in Section 3.1.
args	(IN/OUT): A pointer to the arguments struct.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
/* Values for link 'specific' operation */
typedef enum H5VL_link_specific_t {
    H5VL_LINK_DELETE, /* H5Ldelete(_by_idx)          */
    H5VL_LINK_EXISTS, /* link existence              */
    H5VL_LINK_ITER  /* H5Literate/visit(_by_name)         */
} H5VL_link_specific_t;

/* Parameters for link 'iterate' operation */
typedef struct H5VL_link_iterate_args_t {
    hbool_t       recursive; /* Whether iteration is recursive */
    H5_index_t    idx_type; /* Type of index to iterate over */
    H5_iter_order_t order;  /* Order of index iteration */
    hsize_t *     idx_p;    /* Start/stop iteration index (OUT) */
    H5L_iterate2_t op;      /* Iteration callback function */
    void *        op_data;  /* Iteration callback context */
} H5VL_link_iterate_args_t;

/* Parameters for link 'specific' operations */
typedef struct H5VL_link_specific_args_t {
    H5VL_link_specific_t op_type; /* Operation to perform */

    /* Parameters for each operation */
    union {
        /* H5VL_LINK_DELETE */
        /* No args */

        /* H5VL_LINK_EXISTS */
        struct {
            hbool_t *exists; /* Whether link exists (OUT) */
        } exists;

        /* H5VL_LINK_ITER */
        H5VL_link_iterate_args_t iterate;
    } args;
} H5VL_link_specific_args_t;
```

3.9.6 link: optional

The `optional` callback in the link class implements connector specific operations on an HDF5 link. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*optional)(void *obj, const H5VL_loc_params_t *loc_params, H5VL_optional_args_t
    *args, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): The container or object where the operation needs to happen.
args	(IN/OUT): A pointer to the arguments struct.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

Each connector that requires connector-specific operations should compare the value of the `op_type` field of the `H5VL_optional_args_t` struct with the values returned from calling `H5VLregister_opt_operation` to determine how to handle the optional call and interpret the arguments passed in the struct.

3.10 Object Callbacks

The object API routines (H5O) allow HDF5 users to manage HDF5 group, dataset, and named datatype objects. All the H5O API routines that modify the HDF5 container map to one of the object callback routines in this class that the connector needs to implement.

```
typedef struct H5VL_object_class_t {
    void *(*open)(void *obj, const H5VL_loc_params_t *loc_params, H5I_type_t *opened_type, hid_t
        dxpl_id, void **req);
    herr_t (*copy)(void *src_obj, const H5VL_loc_params_t *loc_params1, const char *src_name, void
        *dst_obj, const H5VL_loc_params_t *loc_params2, const char *dst_name, hid_t ocpypl_id,
        hid_t lcpl_id, hid_t dxpl_id, void **req);
    herr_t (*get)(void *obj, const H5VL_loc_params_t *loc_params, H5VL_object_get_args_t *args,
        hid_t dxpl_id, void **req);
    herr_t (*specific)(void *obj, const H5VL_loc_params_t *loc_params, H5VL_object_specific_args_t
        *args, hid_t dxpl_id, void **req);
    herr_t (*optional)(void *obj, const H5VL_loc_params_t *loc_params, H5VL_optional_args_t *args,
        hid_t dxpl_id, void **req);
} H5VL_object_class_t;
```

Listing 11: Structure for object callback routines, H5VLconnector.h

3.10.1 object: open

The `open` callback in the object class opens the object in the container of the location object and returns a pointer to the object structure containing information to access the object in future calls.

Signature:

```
    void *(*open)(void *obj, H5VL_loc_params_t *loc_params,
        H5I_type_t *opened_type, hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): Pointer to a file or group where the objecy needs to be opened or where the look-up of the target object needs to start.
`loc_params`	(IN): Pointer to location parameters as explained in Section 3.1.
`opened_type`	(OUT): buffer to return the type of the object opened (`H5I_GROUP` or `H5I_DATASET` or `H5I_DATATYPE`).
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.10.2 object: copy

The `copy` callback in the object class copies the object from the source object to the destination object. It returns an `herr_t` indicating success or failure.

Signature:

```
    herr_t (*copy)(void *src_obj, H5VL_loc_params_t *loc_params1,
        const char *src_name, void *dst_obj,
        H5VL_loc_params_t *loc_params2, const char *dst_name,
        hid_t ocpypl_id, hid_t lcpl_id, hid_t dxpl_id, void **req);
```

Arguments:

src_obj	(IN): Pointer to location of the source object to be copied.
loc_params1	(IN): Pointer to location parameters as explained in Section 3.1. The type should only be H5VL_OBJECT_BY_SELF for this callback.
src_name	(IN): Name of the source object to be copied.
dst_obj	(IN): Pointer to location of the destination object.
loc_params2	(IN): Pointer to location parameters as explained in Section 3.1. The type should only be H5VL_OBJECT_BY_SELF for this callback.
dst_name	(IN): Name to be assigned to the new copy.
ocpypl_id	(IN): The object copy property list.
lcpl_id	(IN): The link creation property list.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

3.10.3 object: get

The get callback in the object class retrieves information about the object as specified in the get_type parameter.It returns an herr_t indicating success or failure.

Signature:

```
herr_t (*get)(void *obj, H5VL_loc_params_t *loc_params,
    H5VL_object_get_args_t *args, hid_t dxpl_id,
    void **req);
```

Arguments:

obj	(IN): A location object where information needs to be retrieved from.
loc_params	(IN): Pointer to location parameters as explained in Section 3.1.
args	(IN/OUT): A pointer to the arguments struct.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
/* Values for object 'get' operation */
typedef enum H5VL_object_get_t {
    H5VL_OBJECT_GET_FILE, /* object file                 */
    H5VL_OBJECT_GET_NAME, /* object name                 */
    H5VL_OBJECT_GET_TYPE, /* object type                 */
    H5VL_OBJECT_GET_INFO /* H5Oget_info(_by_idx|name)    */
} H5VL_object_get_t;

/* Parameters for object 'get' operations */
typedef struct H5VL_object_get_args_t {
    H5VL_object_get_t op_type; /* Operation to perform */

    /* Parameters for each operation */
    union {
        /* H5VL_OBJECT_GET_FILE */
        struct {
            void **file; /* File object (OUT) */
        } get_file;

        /* H5VL_OBJECT_GET_NAME */
        struct {
            size_t buf_size; /* Size of name buffer (IN) */
            char * buf;     /* Buffer for name (OUT) */
            size_t *name_len; /* Actual length of name (OUT) */
        } get_name;

        /* H5VL_OBJECT_GET_TYPE */
        struct {
            H5O_type_t *obj_type; /* Type of object (OUT) */
        } get_type;
```

```
31
32          /* H5VL_OBJECT_GET_INFO */
33          struct {
34              unsigned    fields; /* Flags for fields to retrieve */
35              H5O_info2_t *oinfo; /* Pointer to object info (OUT) */
36          } get_info;
37      } args;
38  } H5VL_object_get_args_t;
```

3.10.4 object: specific

The `specific` callback in the object class implements specific operations on HDF5 objects as specified in the `specific_type` parameter. It returns an `herr_t` indicating success or failure.

Signature:

```
1       herr_t (*specific)(void *obj, H5VL_loc_params_t *loc_params, H5VL_object_specific_args_t
            *args, hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): The location object where the operation needs to happen.
`loc_params`	(IN): Pointer to location parameters as explained in Section 3.1.
`args`	(IN/OUT): A pointer to the arguments struct.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

```
1   /* Values for object 'specific' operation */
2   typedef enum H5VL_object_specific_t {
3       H5VL_OBJECT_CHANGE_REF_COUNT, /* H5Oincr/decr_refcount      */
4       H5VL_OBJECT_EXISTS,           /* H5Oexists_by_name          */
5       H5VL_OBJECT_LOOKUP,           /* Lookup object              */
6       H5VL_OBJECT_VISIT,            /* H5Ovisit(_by_name)         */
7       H5VL_OBJECT_FLUSH,            /* H5{D|G|O|T}flush           */
8       H5VL_OBJECT_REFRESH           /* H5{D|G|O|T}refresh         */
9   } H5VL_object_specific_t;
10
11  /* Parameters for object 'visit' operation */
12  typedef struct H5VL_object_visit_args_t {
13      H5_index_t     idx_type; /* Type of index to iterate over */
14      H5_iter_order_t order;  /* Order of index iteration */
15      unsigned       fields;  /* Flags for fields to provide in 'info' object for 'op' callback */
16      H5O_iterate2_t op;      /* Iteration callback function */
17      void *         op_data; /* Iteration callback context */
18  } H5VL_object_visit_args_t;
19
20  /* Parameters for object 'specific' operations */
21  typedef struct H5VL_object_specific_args_t {
22      H5VL_object_specific_t op_type; /* Operation to perform */
23
24      /* Parameters for each operation */
25      union {
26          /* H5VL_OBJECT_CHANGE_REF_COUNT */
27          struct {
28              int delta; /* Amount to modify object's refcount */
29          } change_rc;
30
31          /* H5VL_OBJECT_EXISTS */
32          struct {
33              hbool_t *exists; /* Whether object exists (OUT) */
34          } exists;
35
36          /* H5VL_OBJECT_LOOKUP */
37          struct {
```

```
            H5O_token_t *token_ptr; /* Pointer to token for lookup (OUT) */
        } lookup;

        /* H5VL_OBJECT_VISIT */
        H5VL_object_visit_args_t visit;

        /* H5VL_OBJECT_FLUSH */
        struct {
            hid_t obj_id; /* Object ID (IN) */
        } flush;

        /* H5VL_OBJECT_REFRESH */
        struct {
            hid_t obj_id; /* Object ID (IN) */
        } refresh;
    } args;
} H5VL_object_specific_args_t;
```

3.10.5 object: optional

The `optional` callback in the object class implements connector specific operations on an HDF5 object. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*optional)(void *obj, const H5VL_loc_params_t *loc_params, H5VL_optional_args_t
    *args, hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): The container or object where the operation needs to happen.
`loc_params`	(IN): Pointer to location parameters as explained in Section 3.1.
`args`	(IN/OUT): A pointer to the arguments struct.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

Each connector that requires connector-specific operations should compare the value of the `op_type` field of the `H5VL_optional_args_t` struct with the values returned from calling `H5VLregister_opt_operation` to determine how to handle the optional call and interpret the arguments passed in the struct.

3.11 Introspection Callbacks

```
typedef struct H5VL_introspect_class_t {
    herr_t (*get_conn_cls)(void *obj, H5VL_get_conn_lvl_t lvl, const struct H5VL_class_t
        **conn_cls);
    herr_t (*get_cap_flags)(const void *info, unsigned *cap_flags);
    herr_t (*opt_query)(void *obj, H5VL_subclass_t cls, int opt_type, hbool_t *supported);
} H5VL_introspect_class_t;
```

Listing 12: Structure for VOL connector introspection callback routines, H5VLconnector.h

3.11.1 introspect: get_conn_cls

Get a connector's H5VL_class_t struct.

Signature:

```
herr_t (*get_conn_cls)(void *obj, H5VL_get_conn_lvl_t lvl, const struct H5VL_class_t
    **conn_cls);
```

The `lvl` argument is an `enum`:

```
/* "Levels" for 'get connector class' introspection callback */
typedef enum H5VL_get_conn_lvl_t {
    H5VL_GET_CONN_LVL_CURR,      /* Get "current" connector (for this object) */
    H5VL_GET_CONN_LVL_TERM       /* Get "terminal" connector (for this object) */
                                 /* (Recursively called, for pass-through connectors) */
                                 /* (Connectors that "split" must choose which connector to
                                     return) */
} H5VL_get_conn_lvl_t;
```

Arguments:

`obj`	(IN): The VOL object.
`lvl`	(IN): Current or terminal connector.
`cls`	(OUT): A const pointer to the connector.

3.11.2 introspect: get_cap_flags

Get a connector's capability flags.

Signature:

```
herr_t (*get_cap_flags)(const void *info, unsigned *cap_flags);
```

Arguments:

`info`	(IN): A const pointer to pertinent VOL info.
`cap_flags`	(OUT): A pointer to capability flags.

3.11.3 introspect: opt_query

Query a class for a capability or functionality.

Signature:

```
herr_t (*opt_query)(void *obj, H5VL_subclass_t cls, int opt_type, hbool_t *supported);
```

The `lvl` argument is an `enum`:

```
/* Enum type for each VOL subclass */
/* (Used for various queries, etc) */
typedef enum H5VL_subclass_t {
    H5VL_SUBCLS_NONE,               /* Operations outside of a subclass */
    H5VL_SUBCLS_INFO,               /* 'Info' subclass */
    H5VL_SUBCLS_WRAP,               /* 'Wrap' subclass */
    H5VL_SUBCLS_ATTR,               /* 'Attribute' subclass */
    H5VL_SUBCLS_DATASET,            /* 'Dataset' subclass */
    H5VL_SUBCLS_DATATYPE,           /* 'Named datatype' subclass */
    H5VL_SUBCLS_FILE,               /* 'File' subclass */
    H5VL_SUBCLS_GROUP,              /* 'Group' subclass */
    H5VL_SUBCLS_LINK,               /* 'Link' subclass */
    H5VL_SUBCLS_OBJECT,             /* 'Object' subclass */
    H5VL_SUBCLS_REQUEST,            /* 'Request' subclass */
    H5VL_SUBCLS_BLOB,               /* 'Blob' subclass */
    H5VL_SUBCLS_TOKEN               /* 'Token' subclass */
} H5VL_subclass_t;
```

Arguments:

`obj`	(IN): The VOL object.
`cls`	(IN): The VOL 'class' to query.
`opt_type`	(IN): The specific option to query.
`supported`	(OUT): Whether the operation is supported or not.

3.12 Request (Async) Callbacks

```
typedef struct H5VL_request_class_t {
```

```
    herr_t (*wait)(void *req, uint64_t timeout, H5VL_request_status_t *status);
    herr_t (*notify)(void *req, H5VL_request_notify_t cb, void *ctx);
    herr_t (*cancel)(void *req, H5VL_request_status_t *status);
    herr_t (*specific)(void *req, H5VL_request_specific_args_t *args);
    herr_t (*optional)(void *req, H5VL_optional_args_t *args);
    herr_t (*free)(void *req);
} H5VL_request_class_t;
```

Listing 13: Structure for async request callback routines, H5VLconnector.h

3.12.1 request: wait

Wait (with a timeout) for an async operation to complete. Releases the request if the operation has completed and the connector callback succeeds.

Signature:

```
    herr_t (*wait)(void *req, uint64_t timeout, H5ES_status_t *status);
```

The `status` argument is an `enum` (from `H5ESpublic.h`):

```
/* Asynchronous operation status */
typedef enum H5ES_status_t {
    H5ES_STATUS_IN_PROGRESS, /* Operation has not yet completed                */
    H5ES_STATUS_SUCCEED,     /* Operation has completed, successfully          */
    H5ES_STATUS_FAIL,        /* Operation has completed, but failed            */
    H5ES_STATUS_CANCELED     /* Operation has not completed and was canceled   */
} H5ES_status_t;
```

Arguments:

`req`	(IN): The async request on which to wait.
`timeout`	(IN): The timeout value.
`status`	(IN): The status.

3.12.2 request: notify

Registers a user callback to be invoked when an asynchronous operation completes. Releases the request if connector callback succeeds.

Signature:

```
    herr_t (*notify)(void *req, H5VL_request_notify_t cb, void *ctx);
```

The `cb` argument is a function pointer:

```
    typedef herr_t (*H5VL_request_notify_t)(void *ctx, H5ES_status_t status);
```

Arguments:

`req`	(IN): The async request that will receive the notify callback.
`cb`	(IN): The notify callback for the request.
`ctx`	(IN): The request's context.

3.12.3 request: cancel

Cancels an asynchronous operation. Releases the request if connector callback succeeds.

Signature:

```
    herr_t (*cancel)(void *req);
```

Arguments:

`req`	(IN): The async request to be cancelled.

3.12.4 request: specific

Perform a specific operation on an asynchronous request.

Signature:

```
herr_t (*specific)(void *req, H5VL_request_specific_args_t *args);
```

Arguments:

req	(IN): The async request on which to perform the operation.
args	(IN/OUT): A pointer to the arguments struct.

```
/* Values for async request 'specific' operation */
typedef enum H5VL_request_specific_t {
    H5VL_REQUEST_GET_ERR_STACK, /* Retrieve error stack for failed operation */
    H5VL_REQUEST_GET_EXEC_TIME /* Retrieve execution time for operation */
} H5VL_request_specific_t;

/* Parameters for request 'specific' operations */
typedef struct H5VL_request_specific_args_t {
    H5VL_request_specific_t op_type; /* Operation to perform */

    /* Parameters for each operation */
    union {
        /* H5VL_REQUEST_GET_ERR_STACK */
        struct {
            hid_t err_stack_id; /* Error stack ID for operation (OUT) */
        } get_err_stack;

        /* H5VL_REQUEST_GET_EXEC_TIME */
        struct {
            uint64_t *exec_ts; /* Timestamp for start of task execution (OUT) */
            uint64_t *exec_time; /* Duration of task execution (in ns) (OUT) */
        } get_exec_time;
    } args;
} H5VL_request_specific_args_t;
```

3.12.5 request: optional

Perform a connector-specific operation for a request.

Signature:

```
herr_t (*optional)(void *req, H5VL_optional_args_t *args);
```

Arguments:

req	(IN): The async request on which to perform the operation.
args	(IN/OUT): A pointer to the arguments struct.

Each connector that requires connector-specific operations should compare the value of the `op_type` field of the `H5VL_optional_args_t` struct with the values returned from calling `H5VLregister_opt_operation` to determine how to handle the optional call and interpret the arguments passed in the struct.

3.12.6 request: free

Frees an asynchronous request.

Signature:

```
herr_t (*free)(void *req);
```

Arguments:

req	(IN): The async request to be freed.

3.13 Blob Callbacks

```
typedef struct H5VL_blob_class_t {
    herr_t (*put)(void *obj, const void *buf, size_t size, void *blob_id, void *ctx);
    herr_t (*get)(void *obj, const void *blob_id, void *buf, size_t size, void *ctx);
    herr_t (*specific)(void *obj, void *blob_id, H5VL_blob_specific_args_t *args);
    herr_t (*optional)(void *obj, void *blob_id, H5VL_optional_args_t *args);
} H5VL_blob_class_t;
```

Listing 14: Structure for blob callback routines, H5VLconnector.h

3.13.1 blob: put

Put a blob through the VOL.

Signature:

```
    herr_t (*put)(void *obj, const void *buf, size_t size, void *blob_id, void *ctx);
```

Arguments:

obj	(IN): Pointer to the blob container.
buf	(IN): Pointer to the blob.
size	(IN): Size of the blob.
blob_id	(OUT): Pointer to the blob's connector-specific ID.
ctx	(IN): Connector-specific blob context.

3.13.2 blob: get

Get a blob through the VOL.

Signature:

```
    herr_t (*get)(void *obj, const void *blob_id, void *buf, size_t size, void *ctx);
```

Arguments:

obj	(IN): Pointer to the blob container.
blob_id	(IN): Pointer to the blob's connector-specific ID.
buf	(IN/OUT): Pointer to the blob.
size	(IN): Size of the blob.
ctx	(IN): Connector-specific blob context.

3.13.3 blob: specific

Perform a defined operation on a blob via the VOL.

Signature:

```
    herr_t (*specific)(void *obj, void *blob_id, H5VL_blob_specific_args_t *args);
```

Arguments:

obj	(IN): Pointer to the blob container.
blob_id	(IN): Pointer to the blob's connector-specific ID.
args	(IN/OUT): A pointer to the arguments struct.

```
/* Values for 'blob' 'specific' operation */
typedef enum H5VL_blob_specific_t {
    H5VL_BLOB_DELETE, /* Delete a blob (by ID) */
    H5VL_BLOB_ISNULL, /* Check if a blob ID is "null" */
    H5VL_BLOB_SETNULL /* Set a blob ID to the connector's "null" blob ID value */
} H5VL_blob_specific_t;

/* Parameters for blob 'specific' operations */
typedef struct H5VL_blob_specific_args_t {
    H5VL_blob_specific_t op_type; /* Operation to perform */
```

```
11
12      /* Parameters for each operation */
13      union {
14          /* H5VL_BLOB_DELETE */
15          /* No args */
16
17          /* H5VL_BLOB_ISNULL */
18          struct {
19              hbool_t *isnull; /* Whether blob ID is "null" (OUT) */
20          } is_null;
21
22          /* H5VL_BLOB_SETNULL */
23          /* No args */
24      } args;
25  } H5VL_blob_specific_args_t;
```

3.13.4 blob: optional

Perform a connector-specific operation on a blob via the VOL.

Signature:

```
1       herr_t (*optional)(void *obj, void *blob_id, H5VL_optional_args_t *args);
```

Arguments:

obj	(IN): Pointer to the blob container.
blob_id	(IN): Pointer to the blob's connector-specific ID.
args	(IN/OUT): A pointer to the arguments struct.

Each connector that requires connector-specific operations should compare the value of the `op_type` field of the `H5VL_optional_args_t` struct with the values returned from calling `H5VLregister_opt_operation` to determine how to handle the optional call and interpret the arguments passed in the struct.

3.14 Token Callbacks

```
1   typedef struct H5VL_token_class_t {
2       herr_t (*cmp)(void *obj, const H5O_token_t *token1, const H5O_token_t *token2, int *cmp_value);
3       herr_t (*to_str)(void *obj, H5I_type_t obj_type, const H5O_token_t *token, char **token_str);
4       herr_t (*from_str)(void *obj, H5I_type_t obj_type, const char *token_str, H5O_token_t *token);
5   } H5VL_token_class_t;
```

Listing 15: Structure for token callback routines, H5VLconnector.h

3.14.1 token: cmp

Compares two tokens and outputs a value like strcmp.

Signature:

```
1       herr_t (*cmp)(void *obj, const H5O_token_t *token1, const H5O_token_t *token2, int
            *cmp_value);
```

Arguments:

obj	(IN): The underlying VOL object.
token1	(IN): The first token to compare.
token2	(IN): The second token to compare.
cmp_value	(OUT): A value like strcmp.

3.14.2 token: to_str

Converts a token to a string representation.

Signature:

```
herr_t (*to_str)(void *obj, H5I_type_t obj_type, const H5O_token_t *token, char **token_str);
```

The obj_type argument is an enum (from H5Ipublic.h):

```
typedef enum H5I_type_t {
    H5I_UNINIT     = (-2),    /* uninitialized type                      */
    H5I_BADID      = (-1),    /* invalid Type                            */
    H5I_FILE       = 1,       /* type ID for File objects                */
    H5I_GROUP,                /* type ID for Group objects               */
    H5I_DATATYPE,             /* type ID for Datatype objects            */
    H5I_DATASPACE,            /* type ID for Dataspace objects           */
    H5I_DATASET,              /* type ID for Dataset objects             */
    H5I_MAP,                  /* type ID for Map objects                 */
    H5I_ATTR,                 /* type ID for Attribute objects           */
    H5I_VFL,                  /* type ID for virtual file layer          */
    H5I_VOL,                  /* type ID for virtual object layer        */
    H5I_GENPROP_CLS,          /* type ID for generic property list classes */
    H5I_GENPROP_LST,          /* type ID for generic property lists      */
    H5I_ERROR_CLASS,          /* type ID for error classes               */
    H5I_ERROR_MSG,            /* type ID for error messages              */
    H5I_ERROR_STACK,          /* type ID for error stacks                */
    H5I_SPACE_SEL_ITER,       /* type ID for dataspace selection iterator */
    H5I_NTYPES                /* number of library types, MUST BE LAST!  */
} H5I_type_t;
```

The only values which should be used for this call are:

- H5I_GROUP
- H5I_DATATYPE
- H5I_DATASET
- H5I_MAP

as these are the only objects for which tokens are valid.

Arguments:

obj	(IN): The underlying VOL object.
obj_type	(IN): The type of the object.
token	(IN): The token to turn into a string representation.
token_str	(OUT): The string representation of the token.

3.14.3 token: from_str

Converts a string representation of a token to a token.

Signature:

```
herr_t (*from_str)(void *obj, H5I_type_t obj_type, const char *token_str, H5O_token_t
    *token);
```

The obj_type argument is an enum (from H5Ipublic.h):

```
typedef enum H5I_type_t {
    H5I_UNINIT     = (-2),    /* uninitialized type                      */
    H5I_BADID      = (-1),    /* invalid Type                            */
    H5I_FILE       = 1,       /* type ID for File objects                */
    H5I_GROUP,                /* type ID for Group objects               */
    H5I_DATATYPE,             /* type ID for Datatype objects            */
    H5I_DATASPACE,            /* type ID for Dataspace objects           */
    H5I_DATASET,              /* type ID for Dataset objects             */
    H5I_MAP,                  /* type ID for Map objects                 */
```

```
    H5I_ATTR,                /* type ID for Attribute objects           */
    H5I_VFL,                 /* type ID for virtual file layer          */
    H5I_VOL,                 /* type ID for virtual object layer        */
    H5I_GENPROP_CLS,         /* type ID for generic property list classes */
    H5I_GENPROP_LST,         /* type ID for generic property lists      */
    H5I_ERROR_CLASS,         /* type ID for error classes               */
    H5I_ERROR_MSG,           /* type ID for error messages              */
    H5I_ERROR_STACK,         /* type ID for error stacks                */
    H5I_SPACE_SEL_ITER,      /* type ID for dataspace selection iterator */
    H5I_NTYPES               /* number of library types, MUST BE LAST!  */
} H5I_type_t;
```

The only values which should be used for this call are:

- H5I_GROUP
- H5I_DATATYPE
- H5I_DATASET
- H5I_MAP

as these are the only objects for which tokens are valid.

Arguments:

obj	(IN): The underlying VOL object.
obj_type	(IN): The type of the object.
token_str	(IN): The string representation of the token.
token	(OUT): The token created from the string representation.

3.15 Optional Generic Callback

A generic optional callback is provided for services that are specific to a connector.

The `optional` callback has the following definition. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*optional)(void *obj, H5VL_optional_args_t *args, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): The container or object where the operation needs to happen.
args	(IN/OUT): A pointer to the arguments struct.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the connector.

4 New VOL API Routines

API routines have been added to the HDF5 library to manage VOL connectors. This section details each new API call and explains its intended usage. Additionally, a set of API calls that map directly to the VOL callbacks themselves have been added to aid in the development of passthrough connectors which can be stacked and/or split. A list of these API calls is given in an appendix.

4.1 H5VLpublic.h

The API calls in this header are for VOL management and general use (i.e., not limited to VOL connector authors).

4.1.1 H5VLregister_connector_by_name

Signature:

```
hid_t H5VLregister_by_name(const char *connector_name, hid_t vipl_id);
```

Arguments:

name	(IN): The connector name to search for and register.
vipl_id	(IN): An ID for a VOL initialization property list (vipl).

Registers a VOL connector with the HDF5 library given the name of the connector and returns an identifier for it (H5I_INVALID_HID on errors). If the connector is already registered, the library will create a new identifier for it and returns it to the user; otherwise the library will search the plugin path for a connector of that name, loading and registering it, returning an ID for it, if found. See the VOL User Guide for more information on loading plugins and the search paths.

4.1.2 H5VLregister_connector_by_value

Signature:

```
hid_t H5VLregister_by_value(H5VL_class_value_t connector_value, hid_t vipl_id);
```

Arguments:

connector_value	(IN): The connector value to search for and register.
vipl_id	(IN): An ID for a VOL initialization property list (vipl).

Registers a VOL connector with the HDF5 library given a value for the connector and returns an identifier for it (H5I_INVALID_HID on errors). If the connector is already registered, the library will create a new identifier for it and returns it to the user; otherwise the library will search the plugin path for a connector of that name, loading and registering it, returning an ID for it, if found. See the VOL User Guide for more information on loading plugins and the search paths.

4.1.3 H5VLis_connector_registered_by_name

Signature:

```
htri_t H5VLis_connector_registered_by_name(const char *name);
```

Arguments:

name	(IN): The connector name to check for.

Checks if a VOL connector is registered with the library given the connector name and returns TRUE/FALSE on success, otherwise it returns a negative value.

4.1.4 H5VLis_connector_registered_by_value

Signature:

```
htri_t H5VLis_connector_registered_by_value(H5VL_class_value_t connector_value);
```

Arguments:

value	(IN): The connector value to check for.

Checks if a VOL connector is registered with the library given the connector value and returns TRUE/FALSE on success, otherwise it returns a negative value.

4.1.5 H5VLget_connector_id

Signature:

```
hid_t H5VLget_connector_id(hid_t obj_id);
```

Arguments:
`id` (IN): An ID for an HDF5 VOL object.

Given a VOL object such as a dataset or an attribute, this function returns an identifier for its associated connector. If the ID is not a VOL object (such as a dataspace or uncommitted datatype), `H5I_INVALID_HID` is returned. The identifier must be released with a call to `H5VLclose()`.

4.1.6 H5VLget_connector_id_by_name

Signature:

```
hid_t H5VLget_connector_id_by_name(const char *name);
```

Arguments:
`name` (IN): The connector name to check for.

Given a connector name that is registered with the library, this function returns an identifier for the connector. If the connector is not registered with the library, `H5I_INVALID_HID` is returned. The identifier must be released with a call to `H5VLclose()`.

4.1.7 H5VLget_connector_id_by_value

Signature:

```
hid_t H5VLget_connector_id_by_value(H5VL_class_value_t connector_value);
```

Arguments:
`value` (IN): The connector value to check for.

Given a connector value that is registered with the library, this function returns an identifier for the connector. If the connector is not registered with the library, `H5I_INVALID_HID` is returned. The identifier must be released with a call to `H5VLclose()`.

4.1.8 H5VLget_connector_name

Signature:

```
ssize_t H5VLget_connector_name(hid_t id, char *name/*out*/, size_t size);
```

Arguments:
`id` (IN): The object identifier to check.
`name` (OUT): Buffer pointer to put the connector name. If NULL, the library just returns the size required to store the connector name.
`size` (IN): the size of the passed in buffer.

Retrieves the name of a VOL connector given an object identifier that was created/opened with it. On success, the name length is returned.

4.1.9 H5VLclose

Signature:

```
herr_t H5VLclose(hid_t connector_id);
```

Arguments:
`connector_id` (IN): A valid identifier of the connector to close.

Shuts down access to the connector that the identifier points to and release resources associated with it.

4.1.10 H5VLunregister_connector

Signature:

```
herr_t H5VLunregister(hid_t connector_id);
```

Arguments:

connector_id	(IN): A valid identifier of the connector to unregister.

Unregisters a connector from the library and return a positive value on success otherwise return a negative value. The native VOL connector cannot be unregistered (this will return a negative herr_t value).

4.1.11 H5VLquery_optional

Signature:

```
herr_t H5VLquery_optional(hid_t obj_id, H5VL_subclass_t subcls, int opt_type, uint64_t *flags);
```

Arguments:

obj_id	(IN): A valid identifier of a VOL-managed object.
subcls	(IN): The subclass of the optional operation.
opt_type	(IN): The optional operation. The native VOL connector uses hard-coded values. Other VOL connectors get this value when the optional operations are registered.
flags	(OUT): Bitwise flags indicating support and behavior.

Determines if a connector or connector stack (determined from the passed-in object) supports an optional operation. The returned flags (listed below) not only indicate whether the operation is supported or not, but also give a sense of the option's behavior (useful for pass-through connectors).

Bitwise query flag values:

```
#define H5VL_OPT_QUERY_SUPPORTED   0x0001 /* VOL connector supports this operation */
#define H5VL_OPT_QUERY_READ_DATA   0x0002 /* Operation reads data for object */
#define H5VL_OPT_QUERY_WRITE_DATA  0x0004 /* Operation writes data for object */
#define H5VL_OPT_QUERY_QUERY_METADATA 0x0008 /* Operation reads metadata for object */
#define H5VL_OPT_QUERY_MODIFY_METADATA 0x0010 /* Operation modifies metadata for object */
#define H5VL_OPT_QUERY_COLLECTIVE                                                    \
    0x0020 /* Operation is collective (operations without this flag are assumed to be independent)
        */
#define H5VL_OPT_QUERY_NO_ASYNC 0x0040 /* Operation may NOT be executed asynchronously */
#define H5VL_OPT_QUERY_MULTI_OBJ 0x0080 /* Operation involves multiple objects */
```

4.2 H5VLconnector.h

This functionality is intended for VOL connector authors and includes helper functions that are useful for writing connectors.

API calls to manage optional operations are also found in this header file. These are discussed in the section on optional operations, above.

4.2.1 H5VLregister_connector

Signature:

```
hid_t H5VLregister_connector(const H5VL_class_t *cls, hid_t vipl_id);
```

Arguments:

cls	(IN): A pointer to the connector structure to register.
vipl_id	(IN): An ID for a VOL initialization property list (vipl).

Registers a user-defined VOL connector with the HDF5 library and returns an identifier for that connector (H5I_INVALID_HID on errors). This function is used when the application has direct access to the connector

it wants to use and is able to obtain a pointer for the connector structure to pass to the HDF5 library.

4.2.2 H5VLobject

Signature:

```
void *H5VLobject(hid_t obj_id);
```

Arguments:

`obj_id` (IN): identifier of the object to dereference.

Retrieves a pointer to the VOL object from an HDF5 file or object identifier.

4.2.3 H5VLget_file_type

Signature:

```
hid_t H5VLget_file_type(void *file_obj, hid_t connector_id, hid_t dtype_id);
```

Arguments:

`file_obj` (IN): pointer to a file or file object's connector-specific data.
`connector_id` (IN): A valid identifier of the connector to use.
`dtype_id` (IN): A valid identifier for the type.

Returns a copy of the `dtype_id` parameter but with the location set to be in the file. Returns a negative value (`H5I_INVALID_HID`) on errors.

4.2.4 H5VLpeek_connector_id_by_name

Signature:

```
hid_t H5VLpeek_connector_id_by_name(const char *name);
```

Arguments:

`name` (IN): name of the connector to query.

Retrieves the ID for a registered VOL connector based on a connector name. This is done without duplicating the ID and transferring ownership to the caller (as it normally the case in the HDF5 library). The ID returned from this operation should not be closed. This is intended for use by VOL connectors to find their own ID. Returns a negative value (`H5I_INVALID_HID`) on errors.

4.2.5 H5VLpeek_connector_id_by_value

Signature:

```
hid_t H5VLpeek_connector_id_by_value(H5VL_class_value_t value);
```

Arguments:

`value` (IN): value of the connector to query.

Retrieves the ID for a registered VOL connector based on a connector value. This is done without duplicating the ID and transferring ownership to the caller (as it normally the case in the HDF5 library). The ID returned from this operation should not be closed. This is intended for use by VOL connectors to find their own ID. Returns a negative value (`H5I_INVALID_HID`) on errors.

4.3 H5VLconnector_passthru.h

This functionality is intended for VOL connector authors who are writing pass-through connectors and includes helper functions that are useful for writing such connectors. Callback equivalent functions can be found in this header as well. A list of these functions is included as an appendix to this document.

4.3.1 H5VLcmp_connector_cls

Signature:

```
herr_t H5VLcmp_connector_cls(int *cmp, hid_t connector_id1, hid_t connector_id2);
```

Arguments:

`cmp`	(OUT): a value like strcmp.
`connector_id1`	(IN): the ID of the first connector to compare.
`connector_id2`	(IN): the ID of the second connector to compare.

Compares two connectors (given by the connector IDs) to see if they refer to the same connector underneath. Returns a positive value on success and a negative value on errors.

4.3.2 H5VLwrap_register

Signature:

```
hid_t H5VLwrap_register(void *obj, H5I_type_t type);
```

Arguments:

`cmp`	(IN): an object to wrap.
`type`	(IN): the type of the object (see below).

Wrap an internal object with a "wrap context" and register and return an hid_t for the resulting object. This routine is mainly targeted toward wrapping objects for iteration routine callbacks (i.e. the callbacks from H5Aiterate*, H5Literate* / H5Lvisit*, and H5Ovisit*). The type must be a VOL-managed object class (H5I_FILE, H5I_GROUP, H5I_DATATYPE, H5I_DATASET, H5I_MAP, or H5I_ATTR). Returns a negative value (`H5I_INVALID_HID`) on errors.

4.3.3 H5VLretrieve_lib_state

Signature:

```
herr_t H5VLretrieve_lib_state(void **state);
```

Arguments:

`state`	(OUT): the library state.

Retrieves a copy of the internal state of the HDF5 library, so that it can be restored later. Returns a positive value on success and a negative value on errors.

4.3.4 H5VLstart_lib_state

Signature:

```
herr_t H5VLstart_lib_state(void);
```

Opens a new internal state for the HDF5 library. Returns a positive value on success and a negative value on errors.

4.3.5 H5VLrestore_lib_state

Signature:

```
herr_t H5VLrestore_lib_state(const void *state);
```

Arguments:
`state` (IN): the library state.

Restores the internal state of the HDF5 library. Returns a positive value on success and a negative value on errors.

4.3.6 H5VLfinish_lib_state

Signature:

```
herr_t H5VLfinish_lib_state(void);
```

Closes the state of the library, undoing the effects of `H5VLstart_lib_state`. Returns a positive value on success and a negative value on errors.

4.3.7 H5VLfree_lib_state

Signature:

```
herr_t H5VLfree_lib_state(const void *state);
```

Arguments:
`state` (IN): the library state.

Free a retrieved library state. Returns a positive value on success and a negative value on errors.

Appendix A Mapping of VOL Callbacks to HDF5 API Calls

VOL Callback	HDF5 API Call
FILE	
create	H5Fcreate
open	H5Fopen
get	H5Fget_access_plist H5Fget_create_plist H5Fget_fileno H5Fget_intent H5Fget_name H5Fget_obj_count H5Fget_obj_ids
specific	H5Fdelete H5Fflush H5Fis_accessible H5Fis_hdf5 (deprecated, hard-coded to use native connector) H5Freopen
close	H5Fclose
GROUP	
create	H5Gcreate1 (deprecated) H5Gcreate2 H5Gcreate_anon
open	H5Gopen1 (deprecated) H5Gopen2
get	H5Gget_create_plist H5Gget_info H5Gget_info_by_idx H5Gget_info_by_name H5Gget_num_objs (deprecated)
specific	H5Fmount H5Funmount H5Gflush H5Grefresh
close	H5Gclose
DATASET	
create	H5Dcreate1 (deprecated) H5Dcreate2
open	H5Dopen1 (deprecated) H5Dopen2
read	H5Dread
write	H5Dwrite
get	H5Dget_access_plist H5Dget_create_plist H5Dextend H5Dget_space H5Dget_space_status H5Dget_storage_size H5Dget_type
specific	H5Dextend (deprecated) H5Dflush H5Drefresh H5Dset_extent
close	H5Dclose
OBJECT	
open	H5Oopen H5Oopen_by_addr (deprecated) H5Oopen_by_idx H5Oopen_by_name H5Oopen_by_token

copy	H5Ocopy
get	H5Oget_info1 (deprecated) H5Oget_info2 (deprecated) H5Oget_info3
specific	H5Odecr_refcount H5Oexists_by_name H5Oflush H5O_incr_refcount H5Orefresh H5Ovisit_by_name1 (deprecated) H5Ovisit_by_name2 (deprecated) H5Ovisit_by_name3 H5Ovisit1 (deprecated) H5Ovisit2 (deprecated) H5Ovisit3
close	H5Oclose
LINK	
create	H5Glink (deprecated) H5Glink2 (deprecated) H5Lcreate_hard H5Lcreate_soft H5Lcreate_ud H5Olink
copy	H5Lcopy
move	H5Gmove (deprecated) H5Gmove2 (deprecated) H5Lmove
get	H5Gget_linkval (deprecated) H5Lget_info1 (deprecated) H5Lget_info2 H5Lget_info_by_idx H5Lget_name_by_idx H5Lget_val H5Lget_val_by_idx
specific	H5Gunlink (deprecated) H5Ldelete H5Ldelete_by_idx H5Lexists H5Literate1 (deprecated) H5Literate2 H5Literate_by_name1 (deprecated) H5Literate_by_name2 H5Lvisit1 (deprecated) H5Lvisit2 H5Lvisit_by_name1 (deprecated) H5Lvisit_by_name2
DATATYPE	
commit	H5Tcommit1 (deprecated) H5Tcommit2 H5Tcommit_anon
open	H5Topen1 (deprecated) H5Topen2
get	H5Tget_create_plist
specific	H5Tflush H5Trefresh
close	H5Tclose
ATTRIBUTE	
create	H5Acreate1 (deprecated) H5Acreate2

	H5Acreate_by_name
open	H5Aopen H5Aopen_by_idx H5Aopen_by_name H5Aopen_idx (deprecated) H5Aopen_name (deprecated)
read	H5Aread
write	H5Awrite
get	H5Aget_get_create_plist H5Aget_info H5Aget_info_by_idx H5Aget_info_by_name H5Aget_name H5Aget_name_by_idx H5Aget_space H5Aget_storage_size H5Aget_type
specific	H5Adelete H5Adelete_by_idx H5Adelete_by_name H5Aexists H5Aexists_by_name H5Aiterate1 (deprecated) H5Aiterate2 H5Aiterate_by_name H5Arename H5Arename_by_name
close	H5Aclose

Table 1: Breakdown of HDF5 API calls by VOL callback

Appendix B Callback Wrapper API Calls for Passthrough Connector Authors

From H5VL_connector_passthru.h

```
1
2
3  /* Pass-through callbacks */
4  void * H5VLget_object(void *obj, hid_t connector_id);
5  herr_t H5VLget_wrap_ctx(void *obj, hid_t connector_id, void **wrap_ctx);
6  void * H5VLwrap_object(void *obj, H5I_type_t obj_type, hid_t connector_id, void *wrap_ctx);
7  void * H5VLunwrap_object(void *obj, hid_t connector_id);
8  herr_t H5VLfree_wrap_ctx(void *wrap_ctx, hid_t connector_id);
9
10 /* Public wrappers for generic callbacks */
11 herr_t H5VLinitialize(hid_t connector_id, hid_t vipl_id);
12 herr_t H5VLterminate(hid_t connector_id);
13 herr_t H5VLget_cap_flags(hid_t connector_id, unsigned *cap_flags);
14 herr_t H5VLget_value(hid_t connector_id, H5VL_class_value_t *conn_value);
15
16 /* Public wrappers for info fields and callbacks */
17 herr_t H5VLcopy_connector_info(hid_t connector_id, void **dst_vol_info, void *src_vol_info);
18 herr_t H5VLcmp_connector_info(int *cmp, hid_t connector_id, const void *info1, const void *info2);
19 herr_t H5VLfree_connector_info(hid_t connector_id, void *vol_info);
20 herr_t H5VLconnector_info_to_str(const void *info, hid_t connector_id, char **str);
21 herr_t H5VLconnector_str_to_info(const char *str, hid_t connector_id, void **info);
22
23 /* Public wrappers for attribute callbacks */
24 void * H5VLattr_create(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
25                        const char *attr_name, hid_t type_id, hid_t space_id, hid_t acpl_id,
26                        hid_t aapl_id, hid_t dxpl_id, void **req);
27 void * H5VLattr_open(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
28                      const char *name, hid_t aapl_id, hid_t dxpl_id, void **req);
29 herr_t H5VLattr_read(void *attr, hid_t connector_id, hid_t dtype_id, void *buf, hid_t dxpl_id,
30                      void **req);
31 herr_t H5VLattr_write(void *attr, hid_t connector_id, hid_t dtype_id, const void *buf, hid_t
       dxpl_id,
32                       void **req);
33 herr_t H5VLattr_get(void *obj, hid_t connector_id, H5VL_attr_get_args_t *args, hid_t dxpl_id,
34                     void **req);
35 herr_t H5VLattr_specific(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
36                          H5VL_attr_specific_args_t *args, hid_t dxpl_id, void **req);
37 herr_t H5VLattr_optional(void *obj, hid_t connector_id, H5VL_optional_args_t *args, hid_t dxpl_id,
38                          void **req);
39 herr_t H5VLattr_close(void *attr, hid_t connector_id, hid_t dxpl_id, void **req);
40
41 /* Public wrappers for dataset callbacks */
42 void * H5VLdataset_create(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
43                           const char *name, hid_t lcpl_id, hid_t type_id, hid_t space_id,
                              hid_t dcpl_id,
44                           hid_t dapl_id, hid_t dxpl_id, void **req);
45 void * H5VLdataset_open(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
46                         const char *name, hid_t dapl_id, hid_t dxpl_id, void **req);
47 herr_t H5VLdataset_read(void *dset, hid_t connector_id, hid_t mem_type_id, hid_t mem_space_id,
48                         hid_t file_space_id, hid_t plist_id, void *buf, void **req);
49 herr_t H5VLdataset_write(void *dset, hid_t connector_id, hid_t mem_type_id, hid_t mem_space_id,
50                          hid_t file_space_id, hid_t plist_id, const void *buf, void **req);
51 herr_t H5VLdataset_get(void *dset, hid_t connector_id, H5VL_dataset_get_args_t *args, hid_t
       dxpl_id,
52                        void **req);
53 herr_t H5VLdataset_specific(void *obj, hid_t connector_id, H5VL_dataset_specific_args_t *args,
54                             hid_t dxpl_id, void **req);
55 herr_t H5VLdataset_optional(void *obj, hid_t connector_id, H5VL_optional_args_t *args, hid_t
       dxpl_id,
56                             void **req);
57 herr_t H5VLdataset_close(void *dset, hid_t connector_id, hid_t dxpl_id, void **req);
```

```
58
59  /* Public wrappers for named datatype callbacks */
60  void * H5VLdatatype_commit(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
61                             const char *name, hid_t type_id, hid_t lcpl_id, hid_t tcpl_id,
                               hid_t tapl_id,
62                             hid_t dxpl_id, void **req);
63  void * H5VLdatatype_open(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
64                           const char *name, hid_t tapl_id, hid_t dxpl_id, void **req);
65  herr_t H5VLdatatype_get(void *dt, hid_t connector_id, H5VL_datatype_get_args_t *args, hid_t
        dxpl_id,
66                          void **req);
67  herr_t H5VLdatatype_specific(void *obj, hid_t connector_id, H5VL_datatype_specific_args_t *args,
68                               hid_t dxpl_id, void **req);
69  herr_t H5VLdatatype_optional(void *obj, hid_t connector_id, H5VL_optional_args_t *args, hid_t
        dxpl_id,
70                               void **req);
71  herr_t H5VLdatatype_close(void *dt, hid_t connector_id, hid_t dxpl_id, void **req);
72
73  /* Public wrappers for file callbacks */
74  void * H5VLfile_create(const char *name, unsigned flags, hid_t fcpl_id, hid_t fapl_id, hid_t
        dxpl_id,
75                         void **req);
76  void * H5VLfile_open(const char *name, unsigned flags, hid_t fapl_id, hid_t dxpl_id, void **req);
77  herr_t H5VLfile_get(void *file, hid_t connector_id, H5VL_file_get_args_t *args, hid_t dxpl_id,
78                      void **req);
79  herr_t H5VLfile_specific(void *obj, hid_t connector_id, H5VL_file_specific_args_t *args, hid_t
        dxpl_id,
80                           void **req);
81  herr_t H5VLfile_optional(void *obj, hid_t connector_id, H5VL_optional_args_t *args, hid_t dxpl_id,
82                           void **req);
83  herr_t H5VLfile_close(void *file, hid_t connector_id, hid_t dxpl_id, void **req);
84
85  /* Public wrappers for group callbacks */
86  void * H5VLgroup_create(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
87                          const char *name, hid_t lcpl_id, hid_t gcpl_id, hid_t gapl_id, hid_t
                            dxpl_id,
88                          void **req);
89  void * H5VLgroup_open(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
90                        const char *name, hid_t gapl_id, hid_t dxpl_id, void **req);
91  herr_t H5VLgroup_get(void *obj, hid_t connector_id, H5VL_group_get_args_t *args, hid_t dxpl_id,
92                       void **req);
93  herr_t H5VLgroup_specific(void *obj, hid_t connector_id, H5VL_group_specific_args_t *args,
94                            hid_t dxpl_id, void **req);
95  herr_t H5VLgroup_optional(void *obj, hid_t connector_id, H5VL_optional_args_t *args, hid_t dxpl_id,
96                            void **req);
97  herr_t H5VLgroup_close(void *grp, hid_t connector_id, hid_t dxpl_id, void **req);
98
99  /* Public wrappers for link callbacks */
100 herr_t H5VLlink_create(H5VL_link_create_args_t *args, void *obj, const H5VL_loc_params_t
        *loc_params,
101                        hid_t connector_id, hid_t lcpl_id, hid_t lapl_id, hid_t dxpl_id, void
                           **req);
102 herr_t H5VLlink_copy(void *src_obj, const H5VL_loc_params_t *loc_params1, void *dst_obj,
103                      const H5VL_loc_params_t *loc_params2, hid_t connector_id, hid_t lcpl_id,
104                      hid_t lapl_id, hid_t dxpl_id, void **req);
105 herr_t H5VLlink_move(void *src_obj, const H5VL_loc_params_t *loc_params1, void *dst_obj,
106                      const H5VL_loc_params_t *loc_params2, hid_t connector_id, hid_t lcpl_id,
107                      hid_t lapl_id, hid_t dxpl_id, void **req);
108 herr_t H5VLlink_get(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
109                     H5VL_link_get_args_t *args, hid_t dxpl_id, void **req);
110 herr_t H5VLlink_specific(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
111                          H5VL_link_specific_args_t *args, hid_t dxpl_id, void **req);
112 herr_t H5VLlink_optional(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
113                          H5VL_optional_args_t *args, hid_t dxpl_id, void **req);
114
115 /* Public wrappers for object callbacks */
```

```
116 void * H5VLobject_open(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
117                          H5I_type_t *opened_type, hid_t dxpl_id, void **req);
118 herr_t H5VLobject_copy(void *src_obj, const H5VL_loc_params_t *loc_params1, const char *src_name,
119                         void *dst_obj, const H5VL_loc_params_t *loc_params2, const char
                                *dst_name,
120                         hid_t connector_id, hid_t ocpypl_id, hid_t lcpl_id, hid_t dxpl_id, void
                                **req);
121 herr_t H5VLobject_get(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
122                        H5VL_object_get_args_t *args, hid_t dxpl_id, void **req);
123 herr_t H5VLobject_specific(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
124                            H5VL_object_specific_args_t *args, hid_t dxpl_id, void **req);
125 herr_t H5VLobject_optional(void *obj, const H5VL_loc_params_t *loc_params, hid_t connector_id,
126                            H5VL_optional_args_t *args, hid_t dxpl_id, void **req);
127
128 /* Public wrappers for connector/container introspection callbacks */
129 herr_t H5VLintrospect_get_conn_cls(void *obj, hid_t connector_id, H5VL_get_conn_lvl_t lvl,
130                                    const H5VL_class_t **conn_cls);
131 herr_t H5VLintrospect_get_cap_flags(const void *info, hid_t connector_id, unsigned *cap_flags);
132 herr_t H5VLintrospect_opt_query(void *obj, hid_t connector_id, H5VL_subclass_t subcls, int
        opt_type,
133                                 uint64_t *flags);
134
135 /* Public wrappers for asynchronous request callbacks */
136 herr_t H5VLrequest_wait(void *req, hid_t connector_id, uint64_t timeout,
137                         H5VL_request_status_t *status);
138 herr_t H5VLrequest_notify(void *req, hid_t connector_id, H5VL_request_notify_t cb, void *ctx);
139 herr_t H5VLrequest_cancel(void *req, hid_t connector_id, H5VL_request_status_t *status);
140 herr_t H5VLrequest_specific(void *req, hid_t connector_id, H5VL_request_specific_args_t *args);
141 herr_t H5VLrequest_optional(void *req, hid_t connector_id, H5VL_optional_args_t *args);
142 herr_t H5VLrequest_free(void *req, hid_t connector_id);
143
144 /* Public wrappers for blob callbacks */
145 herr_t H5VLblob_put(void *obj, hid_t connector_id, const void *buf, size_t size, void *blob_id,
146                     void *ctx);
147 herr_t H5VLblob_get(void *obj, hid_t connector_id, const void *blob_id, void *buf, size_t size,
148                     void *ctx);
149 herr_t H5VLblob_specific(void *obj, hid_t connector_id, void *blob_id,
150                          H5VL_blob_specific_args_t *args);
151 herr_t H5VLblob_optional(void *obj, hid_t connector_id, void *blob_id, H5VL_optional_args_t *args);
152
153 /* Public wrappers for token callbacks */
154 herr_t H5VLtoken_cmp(void *obj, hid_t connector_id, const H5O_token_t *token1,
155                      const H5O_token_t *token2, int *cmp_value);
156 herr_t H5VLtoken_to_str(void *obj, H5I_type_t obj_type, hid_t connector_id, const H5O_token_t
        *token,
157                         char **token_str);
158 herr_t H5VLtoken_from_str(void *obj, H5I_type_t obj_type, hid_t connector_id, const char
        *token_str,
159                           H5O_token_t *token);
160
161 /* Public wrappers for generic 'optional' callback */
162 herr_t H5VLoptional(void *obj, hid_t connector_id, H5VL_optional_args_t *args, hid_t dxpl_id,
163                     void **req);
```

Appendix C Native VOL Connector Optional Values By Subclass

```
/* H5VL_SUBCLS_ATTR */
/* H5Aiterate (deprecated routine) */
#define H5VL_NATIVE_ATTR_ITERATE_OLD 0

/* H5VL_SUBCLS_DATASET */
/* H5Dformat_convert (internal) */
#define H5VL_NATIVE_DATASET_FORMAT_CONVERT     0
/* H5Dget_chunk_index_type */
#define H5VL_NATIVE_DATASET_GET_CHUNK_INDEX_TYPE 1
/* H5Dget_chunk_storage_size */
#define H5VL_NATIVE_DATASET_GET_CHUNK_STORAGE_SIZE 2
/* H5Dget_num_chunks */
#define H5VL_NATIVE_DATASET_GET_NUM_CHUNKS     3
/* H5Dget_chunk_info */
#define H5VL_NATIVE_DATASET_GET_CHUNK_INFO_BY_IDX 4
/* H5Dget_chunk_info_by_coord */
#define H5VL_NATIVE_DATASET_GET_CHUNK_INFO_BY_COORD 5
/* H5Dchunk_read */
#define H5VL_NATIVE_DATASET_CHUNK_READ         6
/* H5Dchunk_write */
#define H5VL_NATIVE_DATASET_CHUNK_WRITE        7
/* H5Dvlen_get_buf_size */
#define H5VL_NATIVE_DATASET_GET_VLEN_BUF_SIZE  8
/* H5Dget_offset */
#define H5VL_NATIVE_DATASET_GET_OFFSET         9
/* H5Dget_offset */
#define H5VL_NATIVE_DATASET_CHUNK_ITER         10

/* H5VL_SUBCLS_FILE */
/* H5Fclear_elink_file_cache */
#define H5VL_NATIVE_FILE_CLEAR_ELINK_CACHE      0
/* H5Fget_file_image */
#define H5VL_NATIVE_FILE_GET_FILE_IMAGE         1
/* H5Fget_free_sections */
#define H5VL_NATIVE_FILE_GET_FREE_SECTIONS      2
/* H5Fget_freespace */
#define H5VL_NATIVE_FILE_GET_FREE_SPACE         3
/* H5Fget_info1/2 */
#define H5VL_NATIVE_FILE_GET_INFO               4
/* H5Fget_mdc_config */
#define H5VL_NATIVE_FILE_GET_MDC_CONF           5
/* H5Fget_mdc_hit_rate */
#define H5VL_NATIVE_FILE_GET_MDC_HR             6
/* H5Fget_mdc_size */
#define H5VL_NATIVE_FILE_GET_MDC_SIZE           7
/* H5Fget_filesize */
#define H5VL_NATIVE_FILE_GET_SIZE               8
/* H5Fget_vfd_handle */
#define H5VL_NATIVE_FILE_GET_VFD_HANDLE         9
/* H5Freset_mdc_hit_rate_stats */
#define H5VL_NATIVE_FILE_RESET_MDC_HIT_RATE     10
/* H5Fset_mdc_config */
#define H5VL_NATIVE_FILE_SET_MDC_CONFIG         11
/* H5Fget_metadata_read_retry_info */
#define H5VL_NATIVE_FILE_GET_METADATA_READ_RETRY_INFO 12
/* H5Fstart_swmr_write */
#define H5VL_NATIVE_FILE_START_SWMR_WRITE       13
/* H5Fstart_mdc_logging */
#define H5VL_NATIVE_FILE_START_MDC_LOGGING      14
/* H5Fstop_mdc_logging */
#define H5VL_NATIVE_FILE_STOP_MDC_LOGGING       15
/* H5Fget_mdc_logging_status */
#define H5VL_NATIVE_FILE_GET_MDC_LOGGING_STATUS 16
/* H5Fformat_convert */
```

```
#define H5VL_NATIVE_FILE_FORMAT_CONVERT          17
/* H5Freset_page_buffering_stats */
#define H5VL_NATIVE_FILE_RESET_PAGE_BUFFERING_STATS 18
/* H5Fget_page_buffering_stats */
#define H5VL_NATIVE_FILE_GET_PAGE_BUFFERING_STATS 19
/* H5Fget_mdc_image_info */
#define H5VL_NATIVE_FILE_GET_MDC_IMAGE_INFO      20
/* H5Fget_eoa */
#define H5VL_NATIVE_FILE_GET_EOA                 21
/* H5Fincrement_filesize */
#define H5VL_NATIVE_FILE_INCR_FILESIZE           22
/* H5Fset_latest_format/libver_bounds */
#define H5VL_NATIVE_FILE_SET_LIBVER_BOUNDS       23
/* H5Fget_dset_no_attrs_hint */
#define H5VL_NATIVE_FILE_GET_MIN_DSET_OHDR_FLAG  24
/* H5Fset_dset_no_attrs_hint */
#define H5VL_NATIVE_FILE_SET_MIN_DSET_OHDR_FLAG  25
/* H5Fget_mpi_atomicity */
#define H5VL_NATIVE_FILE_GET_MPI_ATOMICITY 26
/* H5Fset_mpi_atomicity */
#define H5VL_NATIVE_FILE_SET_MPI_ATOMICITY 27
/* Adjust file after open, with wrapping context */
#define H5VL_NATIVE_FILE_POST_OPEN 28

/* H5VL_SUBCLS_GROUP */
/* HG5Giterate (deprecated routine) */
#define H5VL_NATIVE_GROUP_ITERATE_OLD 0
/* HG5Gget_objinfo (deprecated routine) */
#define H5VL_NATIVE_GROUP_GET_OBJINFO 1

/* H5VL_SUBCLS_OBJECT */
/* H5G|H5Oget_comment, H5Oget_comment_by_name */
#define H5VL_NATIVE_OBJECT_GET_COMMENT         0
/* H5G|H5Oset_comment, H5Oset_comment_by_name */
#define H5VL_NATIVE_OBJECT_SET_COMMENT         1
/* H5Odisable_mdc_flushes */
#define H5VL_NATIVE_OBJECT_DISABLE_MDC_FLUSHES 2
/* H5Oenable_mdc_flushes */
#define H5VL_NATIVE_OBJECT_ENABLE_MDC_FLUSHES  3
/* H5Oare_mdc_flushes_disabled */
#define H5VL_NATIVE_OBJECT_ARE_MDC_FLUSHES_DISABLED 4
/* H5Oget_native_info(_by_idx, _by_name) */
#define H5VL_NATIVE_OBJECT_GET_NATIVE_INFO     5
```