User Guide for Developing a Virtual Object Layer Plugin

Mohamad Chaarawi

18th July 2014

Contents

1 Introduction

The Virtual Object Layer (VOL) is an abstraction layer in the HDF5 library that intercepts all API calls that could potentially access objects in an HDF5 container and forwards those calls to plugin "object drivers". The plugins could store the objects in variety of ways. A plugin could, for example, have objects be distributed remotely over different platforms, provide a raw mapping of the model to the file system, or even store the data in other file formats (like native netCDF or HDF4 format). The user still gets the same data model where access is done to a single HDF5 "container"; however the plugin object driver translates from what the user sees to how the data is actually stored. Having this abstraction layer maintains the object model of HDF5 and would allow HDF5 developers or users to write their own plugins for accessing HDF5 data.

This user guide is for developers interested in developing a VOL plugin for the HDF5 library. The document is meant to be used in conjunction with the HDF5 reference manual. It is assumed that the reader has good knowledge of the VOL architecture obtained by reading the VOL architectural design document **??** MSC-ref. The document will cover the steps needed to create external and internal VOL plugins. Both ways have a lot of common steps and rules that will be covered first.

2 Creating a VOL Plugin

Each VOL plugin should be of type `H5VL_class_t` that is defined as:

```
/* Class information for each VOL driver */
typedef struct H5VL_class_t {
    H5VL_class_value_t value;
    const char *name;
    herr_t (*initialize)(void);
    herr_t (*terminate)(void);
    size_t info_size;
    void * (*fapl_copy)(const void *info);
    herr_t (*fapl_free)(void *info);
    H5VL_attr_class_t       attr_cls;
    H5VL_datatype_class_t   datatype_cls;
    H5VL_dataset_class_t    dataset_cls;
    H5VL_file_class_t       file_cls;
    H5VL_group_class_t      group_cls;
    H5VL_link_class_t       link_cls;
    H5VL_object_class_t     object_cls;
    H5VL_async_class_t      async_cls;
} H5VL_class_t;
```

The `value` field is an integer enum identifier that should be greater than 128 for external plugins and smaller than 128 for internal plugins. This plugin identifier is used to select the VOL plugin to be used when creating/accessing the HDF5 container in the application. Setting it in the VOL structure is required.

The `name` field is a string that identifies the VOL plugin name. Setting it is not required.

The `initialize` field is a function pointer - MSC not used now!.

The `terminate` field is a function pointer - MSC not used now!.

The `info_size` field indicates the size required to store the info data that the plugin needs. That info data is passed when the plugin is selected for usage with the file access property list (fapl) function. It might be that the plugin defined does not require any information from the user, which means the size in this field will be zero. More information about the info data and the fapl selection routines follow later.

The `fapl_copy` field is a function pointer that is called when the plugin is selected with the fapl function. It allows the plugin to make a copy if the info data since the user might free it when closing the fapl. It is required if there is info data needed by the plugin.

The `fapl_free` field is a function pointer that is called to free the info data when the fapl close routine is called. It is required if there is info data needed by the plugin.

The rest of the fields in the `H5VL_class_t` struct are "subclasses" that define all the object VOL function callbacks that are mapped to from the HDF5 API layer and will be detailed in the following sub-sections.

2.1 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 plugin developers since that would mean reworking their VOL plugin 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 plugins 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 (beyond the 1.8 series). That pointer is set by the VOL plugin 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 plugin would not execute the operation asynchronously. If the plugin 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 to not have a one-to-one mapping between API operation and callbacks. Furthermore, to keep the callbacks themselves short

and not cluttered with a lot of parameters, some of the parameters are passed in as properties in property lists included with the callback. The value of those properties can be retrieved by calling the public routine (or its private version if this is an internal plugin):

```
herr_t H5Pget(hid_t plist_id, const char *property_name, void *value);
```

The property names and value 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 as follows:

```
/*
 * Structure to hold parameters for object locations.
 * either: BY_ID, BY_NAME, BY_IDX, BY_ADDR, BY_REF
 */

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 */
        struct H5VL_loc_by_name loc_by_name;
        struct H5VL_loc_by_idx loc_by_idx;
        struct H5VL_loc_by_addr loc_by_addr;
        struct H5VL_loc_by_ref loc_by_ref;
    }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 */
    H5VL_OBJECT_BY_NAME,

    /* location defined by object, path, and index in H5VL_loc_by_idx */
    H5VL_OBJECT_BY_IDX,
```

```
    /* location defined by physical address in H5VL_loc_by_addr */
    H5VL_OBJECT_BY_ADDR,

    /* NOT USED */
    H5VL_OBJECT_BY_REF
} H5VL_loc_type_t;

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

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 plist_id; /* The link access property list */
};

struct H5VL_loc_by_addr {
    haddr_t addr; /* physical address of location */
};

/* Not used for now */
struct H5VL_loc_by_ref {
    H5R_type_t ref_type;
    const void *_ref;
    hid_t plist_id;
};
```

Another large set of operations that would make a one-to-one mapping difficult are the `Get` operations that retrieve something from an object; for example a property list or a datatype of a dataset, etc... To handle that, each class of objects has a general get callback with a `get_type` and a `va_list` argument to handle the multiple get operations. More information about types and the arguments for each type will be detailed in the corresponding class arguments.

Finally there are a set of functions for the file and general object (H5O) classes that are not widely used or interesting enough for plugin developers to implement. Those routines are mapped to a `misc` callback in their respective class.

2.2 The File Function 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 plugin 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 (*flush)(void *obj, H5VL_loc_params_t loc_params,
        H5F_scope_t scope, hid_t dxpl_id, void **req);

    herr_t (*get)(void *obj, H5VL_file_get_t get_type, hid_t dxpl_id,
        void **req, va_list arguments);

    herr_t (*misc)(void *obj, H5VL_file_misc_t misc_type,
        hid_t dxpl_id, void **req, va_list arguments);

    herr_t (*optional)(void *obj, H5VL_file_optional_t op_type,
        hid_t dxpl_id, void **req, va_list arguments);

    herr_t (*close) (void *file, hid_t dxpl_id, void **req);
} H5VL_file_class_t;
```

The `create` callback in the file class should create a container and returns a pointer to the file structure 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 plugin.

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

Signature:

```
    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 plugin.

The `flush` callback in the file class should flush all buffers associated with the container to disk and returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*flush)(void *obj, H5VL_loc_params_t loc_params,
    H5F_scope_t scope, hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): Pointer to a file or object in the file to be flushed.
`loc_params`	(IN): The location parameters as explained in section 2.1. The type can be only `H5VL_OBJECT_BY_SELF` in this callback.
`scope`	(IN): The scope of the flushing action.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.

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:

```
herr_t (*get)(void *obj, H5VL_file_get_t get_type, hid_t dxpl_id,
    void **req, va_list arguments);
```

The `get_type` argument is an `enum`:

```
/* types for all file get API routines */
typedef enum H5VL_file_get_t {
    H5VL_FILE_GET_FAPL,    /* file access property list */
    H5VL_FILE_GET_FCPL,    /* file creation property list */
    H5VL_FILE_GET_INTENT,  /* file intent              */
    H5VL_FILE_GET_NAME,    /* file name                */
    H5VL_FILE_GET_OBJ_COUNT, /* object count in file   */
    H5VL_FILE_GET_OBJ_IDS, /* object ids in file       */
    H5VL_OBJECT_GET_FILE
} H5VL_file_get_t;
```

Arguments:

`obj`	(IN): The container or object where information needs to be retrieved from.
`get_type`	(IN): The type of the information to retrieve.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.
`arguments`	(IN/OUT): argument list containing parameters and output pointers for the get operation.

The `arguments` argument contains a variable list of arguments depending on the `get_type` parameter. The following list shows the argument list, in-order, for each type:

- `H5VL_FILE_GET_FCPL`, to retrieve the file creation property list:
  1. `hid_t *ret_id` (OUT): buffer for the identifier of the file creation property list.
- `H5VL_FILE_GET_FAPL`, to retrieve the file access property list:
  1. `hid_t *ret_id` (OUT): buffer for the identifier of the file access property list.
- `H5VL_FILE_GET_OBJ_COUNT:`, to retrieve the object count in the container:
  1. `unsigned types` (IN): type of objects to look for.
  2. `ssize_t *ret` (OUT): buffer for the object count.
- `H5VL_FILE_GET_OBJ_IDS:`, to retrieve object identifiers in the container:
  1. `unsigned types` (IN): type of objects to look for.
  2. `size_t max_objs` (IN): maximum number of objects to open.
  3. `hid_t *oid_list` (OUT): buffer for the object identifiers.
  4. `ssize_t *ret` (OUT): buffer for the object count.
- `H5VL_FILE_GET_INTENT`, get access intent of the container:
  1. `unsigned *ret` (OUT): buffer for the intent value.
- `H5VL_FILE_GET_NAME`, get container name an object belongs to:
  1. `H5I_type_t type` (IN): the object type in `obj`.
  2. `size_t size` (IN): size of the buffer for the file name.
  3. `char *name` (OUT): buffer for the file name.
  4. `ssize_t *ret` (OUT): buffer for the entire size of the file name.
- `H5VL_OBJECT_GET_FILE`, get the container that the object belongs to:
  1. `H5I_type_t type` (IN): the object type in `obj`.
  2. `void **ret` (OUT): pointer to the file structure returned by the plugin.

The misc callback in the file class should execute some not very common operations on the container as specified in the misc_type parameter. It returns an herr_t indicating success or failure.

Signature:

```
herr_t (*misc)(void *obj, H5VL_file_misc_t misc_type, hid_t dxpl_id,
    void **req, va_list arguments);
```

The misc_type argument is an enum:

```
/* types for all file misc operations */
typedef enum H5VL_file_misc_t {
    H5VL_FILE_MOUNT,       /* H5Fmount                 */
    H5VL_FILE_UNMOUNT,     /* H5Funmount               */
    H5VL_FILE_IS_ACCESSIBLE /* is the container accessible? */
} H5VL_file_misc_t;
```

Arguments:

obj	(IN): The container or object where the operation needs to happen.
misc_type	(IN): The type of the operation.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.
arguments	(IN/OUT): argument list containing parameters and output pointers for the get operation.

The arguments argument contains a variable list of arguments depending on the misc_type parameter. The following list shows the argument list, in-order, for each type:

- H5VL_FILE_MOUNT, Mounts a file on the location object:
  1. H5I_type_t type (IN): the object type in obj.
  2. char *name (IN): name of the group onto which the file specified by file is to be mounted.
  3. void *file (IN): child file to be mounted.
  4. hid_t *fmpl_id (IN): file mount property list.
- H5VL_FILE_UNMOUNT, un-mounts a file from the location object:
  1. H5I_type_t type (IN): the object type in obj.
  2. char *name (IN): name of the mount point.
- H5VL_FILE_IS_ACCESSIBLE, checks if a container is accessible using a specific file access property list:
  1. hid_t *fapl_id (IN): file access property list.
  2. char *name (IN): name of the container to check.
  3. htri_t *result (OUT): buffer for the result; 0 if no, 1 if yes.

The `optional` callback in the file class should execute some operations considered native HDF5 specific operations on the container as specified in the `optional_type` parameter. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*optional)(void *obj, H5VL_file_optional_t op_type,
    hid_t dxpl_id, void **req, va_list arguments);
```

The `optional_type` argument is an `enum`:

```
/* types for all file optional operations */
typedef enum H5VL_file_optional_t {
    H5VL_FILE_CLEAR_ELINK_CACHE, /* Clear external link cache    */
    H5VL_FILE_GET_FILE_IMAGE,  /* file image                     */
    H5VL_FILE_GET_FREE_SECTIONS, /* file free selections         */
    H5VL_FILE_GET_FREE_SPACE,  /* file freespace                 */
    H5VL_FILE_GET_INFO,        /* file info                      */
    H5VL_FILE_GET_MDC_CONF,    /* file metadata cache configuration */
    H5VL_FILE_GET_MDC_HR,      /* file metadata cache hit rate   */
    H5VL_FILE_GET_MDC_SIZE,    /* file metadata cache size       */
    H5VL_FILE_GET_SIZE,        /* file size                      */
    H5VL_FILE_GET_VFD_HANDLE,  /* file VFD handle                */
    H5VL_FILE_REOPEN,          /* reopen the file                */
    H5VL_FILE_RESET_MDC_HIT_RATE, /* get metadata cache hit rate */
    H5VL_FILE_SET_MDC_CONFIG   /* set metadata cache configuration */
} H5VL_file_optional_t;
```

Arguments:

`obj`	(IN): The container or object where the operation needs to happen.
`optional_type`	(IN): The type of the operation.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.
`arguments`	(IN/OUT): argument list containing parameters and output pointers for the get operation.

The `arguments` argument contains a variable list of arguments depending on the `optional_type` parameter. The following list shows the argument list, in-order, for each type:

- `H5VL_FILE_GET_SIZE`, retrieve the size of the container in `obj`:
  1. `hsize_t *ret` (OUT): file size.
- `H5VL_FILE_GET_FILE_IMAGE`, retrieve file image from the container in `obj`:
  1. `void *buf_ptr` (OUT): buffer to return the file image.
  2. `ssize_t *ret` (OUT): buffer for the total size needed for the file image.
  3. `size_t buf_len` (IN): size of the buffer passed in.

- `H5VL_FILE_GET_FREE_SPACE`, retrieve amount of free space in the container in `obj`:
  1. `hssize_t *ret` (OUT): buffer for the free space.
- `H5VL_FILE_GET_FREE_SECTIONS`, retrieve free sections from the container in `obj`:
  1. `H5F_sect_info_t *sinfo` (OUT): pointer to the section info structure to fill.
  2. `ssize_t *ret` (OUT): buffer for the total number of free sections.
  3. `H5F_mem_t type` (IN): type of the memory space to check for.
  4. `size_t nsects` (IN): number of section allocate in `sinfo`.
- `H5VL_FILE_GET_INFO`, retrieve file info from the object in `obj`:
  1. `H5I_type_t type` (IN): the object type in `obj`.
  2. `H5F_info2_t *finfo` (OUT): pointer to info structure to fill.
- `H5VL_FILE_GET_MDC_CONF`, retrieve the meta data cache configuration from the container in `obj`:
  1. `H5I_type_t type` (IN): the object type in `obj`.
  2. `H5AC_cache_config_t *conf` (OUT): pointer to configuration structure to fill.
- `H5VL_FILE_GET_MDC_HR`, retrieve the meta data cache hit rate from the container in `obj`:
  1. `double *ret` (OUT): buffer for the hit rate.
- `H5VL_FILE_GET_MDC_SIZE`, retrieve the meta data cache size information from the container in `obj`:
  1. `size_t max_size_ptr` (OUT): buffer for maximum size.
  2. `size_t min_clean_size_ptr` (OUT): buffer for minimum clean size.
  3. `size_t cur_size_ptr` (OUT): buffer for current size.
  4. `int cur_num_entries_ptr` (OUT): buffer for number of current cache entries.
- `H5VL_FILE_GET_VFD_HANDLE`, retrieve the virtual file driver handle from the container in `obj`:
  1. `void **handle` (OUT): pointer to a buffer the plugin sets to the VFD handle.
  2. `hid_t fapl` (IN): File access property list.
- `H5VL_FILE_CLEAR_ELINK_CACHE`, clears the external link file cache. Takes no extra arguments.
- `H5VL_FILE_REOPEN`, reopen the container in `obj`:
  1. `void **ret` (OUT): pointer to be set to the opened file structure.

- H5VL_FILE_RESET_MDC_HIT_RATE, resets the hit rate statistics for the metadata cache on the container in obj. Takes no extra arguments.
- H5VL_FILE_SET_MDC_CONFIG, sets the meta data cache configuration for the container in obj:
  1. H5AC_cache_config_t *conf (IN): pointer to configuration structure to use.

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 plugin.

2.3 The Group Function 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 plugin needs to implement:

```
typedef struct H5VL_group_class_t {
    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);

    void *(*open)(void *obj, 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_t get_type, hid_t dxpl_id,
        void **req, va_list arguments);

    herr_t (*close) (void *grp, hid_t dxpl_id, void **req);
} H5VL_group_class_t;
```

The create callback in the group class should create 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:

```
    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): The location parameters as explained in section 2.1. 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 plugin.

The open callback in the group class should open 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:

```
    void *(*open)(void *obj, H5VL_loc_params_t loc_params,
        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): The location parameters as explained in section 2.1. 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 plugin.

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

Signature:

```
    herr_t (*get)(void *obj, H5VL_group_get_t get_type, hid_t dxpl_id,
        void **req, va_list arguments);
```

The get_type argument is an enum:

```
/* types for all group get API routines */
typedef enum H5VL_group_get_t {
    H5VL_GROUP_GET_GCPL,    /*group creation property list */
```

```
    H5VL_GROUP_GET_INFO     /*group info                */
} H5VL_group_get_t;
```

Arguments:

`obj`	(IN): The group object where information needs to be retrieved from.
`get_type`	(IN): The type of the information to retrieve.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.
`arguments`	(IN/OUT): argument list containing parameters and output pointers for the get operation.

The `arguments` argument contains a variable list of arguments depending on the `get_type` parameter. The following list shows the argument list, in-order, for each type:

- `H5VL_GROUP_GET_GCPL`, to retrieve the group creation property list of the group specified in `obj`:
  1. `hid_t *ret_id` (OUT): buffer for the identifier of the group creation property list.
- `H5VL_GROUP_GET_INFO`, to retrieve the attribute info:
  1. `H5VL_loc_params_t loc_params` (IN): The location parameters explained in section 2.1.
  2. `H5G_info_t *ginfo` (OUT): info structure to fill the group info in.

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

Signature:

```
    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 plugin.

2.4 The Dataset Function 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 plugin needs to implement:

```
typedef struct H5VL_dataset_class_t {
    void *(*create)(void *obj, H5VL_loc_params_t loc_params,
```

```
        const char *name, hid_t dcpl_id, hid_t dapl_id,
        hid_t dxpl_id, void **req);

    void *(*open)(void *obj, 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 (*set_extent)(void *dset, const hsize_t size[],
        hid_t dxpl_id, void **req);

    herr_t (*get)(void *dset, H5VL_dataset_get_t get_type,
        hid_t dxpl_id, void **req, va_list arguments);

    herr_t (*close) (void *dset, hid_t dxpl_id, void **req);
} H5VL_dataset_class_t;
```

The `create` callback in the dataset class should create 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 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): The location parameters as explained in section 2.1. The type can be only `H5VL_OBJECT_BY_SELF` in this callback.
`name`	(IN): The name of the dataset to be created.
`dcpl_id`	(IN): The dataset creation property list. It contains all the dataset creation properties in addition to the dataset datatype (an `hid_t`), dataspace (an `hid_t`), and the link creation property list of the create operation (an `hid_t`) that can be retrieved with the properties, `H5VL_DSET_TYPE_ID`, `H5VL_DSET_SPACE_ID`, and `H5VL_DSET_LCPL_ID` respectively.
`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 plugin.

The `open` callback in the dataset class should open 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): The location parameters as explained in section 2.1. 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 plugin.

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

Signature:

```
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);
```

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 plugin.

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

Signature:

```
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);
```

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 plugin.

The `set_extent` callback in the dataset class should extend the dataset dimensions and returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*set_extent)(void *dset, const hsize_t size[],
    hid_t dxpl_id, void **req);
```

Arguments:

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

The `get` callback in the dataset class should retrieve 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_t get_type,
    hid_t dxpl_id, void **req, va_list arguments);
```

The `get_type` argument is an `enum`:

```
/* types for all dataset get API routines */
typedef enum H5VL_dataset_get_t {
    H5VL_DATASET_GET_SPACE,      /* dataspace              */
    H5VL_DATASET_GET_SPACE_STATUS, /* space status         */
    H5VL_DATASET_GET_TYPE,       /* datatype               */
    H5VL_DATASET_GET_DCPL,       /* creation property list */
    H5VL_DATASET_GET_DAPL,       /* access property list   */
    H5VL_DATASET_GET_STORAGE_SIZE, /* storage size         */
    H5VL_DATASET_GET_OFFSET      /* offset                 */
} H5VL_dataset_get_t;
```

Arguments:

`dset`	(IN): The dataset object where information needs to be retrieved from.
`get_type`	(IN): The type of the information to retrieve.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.
`arguments`	(IN/OUT): argument list containing parameters and output pointers for the get operation.

The `arguments` argument contains a variable list of arguments depending on the `get_type` parameter. The following list shows the argument list, in-order, for each type:

- `H5VL_DATASET_GET_SPACE`, to retrieve the dataspace of the dataset specified in `obj`:
    1. `hid_t *ret_id` (OUT): buffer for the identifier of the dataset dataspace.
- `H5VL_DATASET_GET_SPACE_STATUS`, to retrieve the information whether space has been allocated for the dataset:
    1. `H5D_space_status_t *allocation` (OUT): buffer for the space status.
- `H5VL_DATASET_GET_TYPE`, to retrieve the datatype of the dataset specified in `obj`:
    1. `hid_t *ret_id` (OUT): buffer for the identifier of the dataset datatype.
- `H5VL_DATASET_GET_DCPL`, to retrieve the dataset creation property list of the dataset specified in `obj`:
    1. `hid_t *ret_id` (OUT): buffer for the identifier of the dataset creation property list.
- `H5VL_DATASET_GET_DAPL`, to retrieve the dataset access property list of the dataset specified in `obj`:
    1. `hid_t *ret_id` (OUT): buffer for the identifier of the dataset access property list.
- `H5VL_DATASET_GET_STORAGE_SIZE`, to retrieve the storage size of the dataset specified in `obj`:
    1. `hsize_t *ret` (OUT): buffer for the storage size of the dataset in the container.
- `H5VL_DATASET_GET_OFFSET`, to retrieve the offset of the dataset specified in `obj` in the container:
    1. `haddr_t *ret` (OUT): buffer for the offset of the dataset in the container.

The `close` callback in the dataset class should terminate 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 plugin.

2.5 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 plugin needs to implement:

```
typedef struct H5VL_attr_class_t {
    void *(*create)(void *obj, H5VL_loc_params_t loc_params,
        const char *attr_name, hid_t acpl_id, hid_t aapl_id,
        hid_t dxpl_id, void **req);

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

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

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

    herr_t (*iterate)(void *obj, H5VL_loc_params_t loc_params,
        H5_index_t idx_type, H5_iter_order_t order, hsize_t *n,
        H5A_operator2_t op, void *op_data, hid_t dxpl_id, void **req);

    herr_t (*get)(void *obj, H5VL_attr_get_t get_type, hid_t dxpl_id,
        void **req, va_list arguments);

    herr_t (*remove)(void *obj, H5VL_loc_params_t loc_params,
        const char *attr_name, hid_t dxpl_id, void **req);

    herr_t (*close)(void *attr, hid_t dxpl_id, void **req);
} H5VL_attr_class_t;
```

The `create` callback in the attribute class should create 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:

```
    void *(*create)(void *obj, H5VL_loc_params_t loc_params,
        const char *attr_name, hid_t acpl_id, hid_t aapl_id,
        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): The location parameters as explained in section 2.1.
attr_name	(IN): The name of the attribute to be created.
acpl_id	(IN): The attribute creation property list. It contains all the attribute creation properties in addition to the attribute datatype (an hid_t) and dataspace (an hid_t) that can be retrieved with the properties, H5VL_ATTR_TYPE_ID and H5VL_ATTR_SPACE_ID.
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 plugin.

The open callback in the attribute class should open 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:

```
void *(*open)(void *obj, H5VL_loc_params_t loc_params,
    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): The location parameters as explained in section 2.1.
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 plugin.

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

Signature:

```
herr_t (*read)(void *attr, hid_t mem_type_id, void *buf,
    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 plugin.

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

Signature:

```
herr_t (*write)(void *attr, hid_t mem_type_id, const void *buf,
    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 plugin.

The `iterate` callback in the attribute class should iterate over the attributes in the container of the location object and call the user defined function on each one. It returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*iterate)(void *obj, H5VL_loc_params_t loc_params,
    H5_index_t idx_type, H5_iter_order_t order, hsize_t *n,
    H5A_operator2_t op, void *op_data, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): Pointer to an object where the iteration needs to happen or where the look-up of the target object needs to start.
loc_params	(IN): The location parameters as explained in section 2.1.
idx_type	(IN): Type of index.
order	(IN): Order in which to iterate over index.
n	(IN/OUT): Initial and return offset withing index.
op	(IN): User-defined function to pass each attribute to.
op_data	(IN/OUT): User data to pass through to and to be returned by iterator operator function.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.

The `get` callback in the attribute class should retrieve 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_t get_type, hid_t dxpl_id,
    void **req, va_list arguments);
```

The `get_type` argument is an `enum`:

```
/* types for all attr get API routines */
typedef enum H5VL_attr_get_t {
    H5VL_ATTR_EXISTS,        /* attribute exists?    */
    H5VL_ATTR_GET_SPACE,     /* dataspace            */
    H5VL_ATTR_GET_TYPE,      /* datatype             */
```

```
    H5VL_ATTR_GET_ACPL,       /* creation property list */
    H5VL_ATTR_GET_NAME,       /* access property list */
    H5VL_ATTR_GET_STORAGE_SIZE, /* storage size       */
    H5VL_ATTR_GET_INFO        /* offset              */
} H5VL_attr_get_t;
```

Arguments:

`attr`	(IN): An attribute or location object where information needs to be retrieved from.
`get_type`	(IN): The type of the information to retrieve.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.
`arguments`	(IN/OUT): argument list containing parameters and output pointers for the get operation.

The `arguments` argument contains a variable list of arguments depending on the `get_type` parameter. The following list shows the argument list, in-order, for each type:

- `H5VL_ATTR_EXISTS`, to check if an attribute exists on a particular object specified in `obj`:
  1. `H5VL_loc_params_t loc_params` (IN): The location parameters explained in section 2.1.
  2. `char *attr_name` (IN): the attribute name to check.
  3. `htri_t *ret` (OUT): existence result, 0 if false, 1 if true.
- `H5VL_ATTR_GET_SPACE`, to retrieve the dataspace of the attribute specified in `obj`:
  1. `hid_t *ret_id` (OUT): buffer for the identifier of the attribute dataspace.
- `H5VL_ATTR_GET_TYPE`, to retrieve the datatype of the attribute specified in `obj`:
  1. `hid_t *ret_id` (OUT): buffer for the identifier of the attribute datatype.
- `H5VL_ATTR_GET_ACPL`, to retrieve the attribute creation property list of the attribute specified in `obj`:
  1. `hid_t *ret_id` (OUT): buffer for the identifier of the attribute creation property list.
- `H5VL_ATTR_GET_NAME`, to retrieve an attribute name on a particular object specified in `obj`:
  1. `H5VL_loc_params_t loc_params` (IN): The location parameters explained in section 2.1. The type could be either `H5VL_OBJECT_BY_SELF` meaning `obj` is the attribute, or `H5VL_OBJECT_BY_IDX` meaning the attribute to retrieve the name for should be looked up using the index information on the object in `obj` and the index information in `loc_params`.

2. `size_t buf_size` (IN): the size of the buffer to store the name in.
3. `void *buf` (OUT): Buffer to store the name in.
4. `ssize_t *ret_val` (OUT): return the actual size needed to store the fill attribute name.

- `H5VL_ATTR_GET_INFO`, to retrieve the attribute info:
  1. `H5VL_loc_params_t loc_params` (IN): The location parameters explained in section 2.1.
  2. `H5A_info_t *ainfo` (OUT): info structure to fill the attribute info in.
- `H5VL_ATTR_GET_STORAGE_SIZE`, to retrieve the storage size of the attribute specified in `obj`:
  1. `hsize_t *ret` (OUT): buffer for the storage size of the attribute in the container.

The `remove` callback in the attribute class should remove an attribute object in the container of the location object and returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*remove)(void *obj, H5VL_loc_params_t loc_params,
    const char *attr_name, hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): Pointer to an object where the attribute needs to be removed or where the look-up of the target object needs to start.
`loc_params`	(IN): The location parameters as explained in section 2.1.
`attr_name`	(IN): The name of the attribute to be removed.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.

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

Signature:

```
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 plugin.

2.6 The Named Datatype Function 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 plugin needs to implement:

```
typedef struct H5VL_datatype_class_t {
    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);

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

    ssize_t (*get_binary)(void *obj, unsigned char *buf, size_t size,
        hid_t dxpl_id, void **req);

    herr_t (*get) (void *obj, H5VL_datatype_get_t get_type,
        hid_t dxpl_id, void **req, va_list arguments);

    herr_t (*close) (void *dt, hid_t dxpl_id, void **req);
} H5VL_datatype_class_t;
```

The `commit` callback in the named datatype class should create 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): The location parameters as explained in section 2.1. 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 plugin.

The open callback in the named datatype class should open 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:

```
void *(*open) (void *obj, H5VL_loc_params_t loc_params,
    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): The location parameters as explained in section 2.1. 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 plugin.

The get_binary callback in the named datatype class should serialize the original transient HDF5 datatype that was committed, or return the size that is required for it be serialized if the passed in buffer is NULL. The HDF5 library provides two functions to encode and decode datatypes in their transient form, H5Tencode() and H5Tdecode(). When a datatype is committed, the plugin is required to keep the serialized form of the transient datatype stored somewhere in the container (which is usually the case anyway when committing a named datatype), so it can be retrieved with this call. This is needed to generate the higher level HDF5 datatype identifier that allows all the H5T "transient" routines to work properly on the named datatype.

Signature:

```
ssize_t (*get_binary)(void *obj, unsigned char *buf, size_t size,
    hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): Pointer to the named datatype object.
buf	(OUT): Buffer to out the binary form of the datatype in.
size	(IN): The size of the buffer passed in (0 if NULL).
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.

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

Signature:

```
herr_t (*get) (void *obj, H5VL_datatype_get_t get_type,
    hid_t dxpl_id, void **req, va_list arguments);
```

The get_type argument is an enum:

```
/* types for all datatype get API routines */
typedef enum H5VL_datatype_get_t {
    H5VL_DATATYPE_GET_TCPL /*datatype creation property list */
} H5VL_datatype_get_t;
```

Arguments:

obj	(IN): The named datatype to retrieve information from.
get_type	(IN): The type of the information to retrieve.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.
arguments	(IN/OUT): argument list containing parameters and output pointers for the get operation.

The arguments argument contains a variable list of arguments depending on the get_type parameter. The following list shows the argument list, in-order, for each type:

- H5VL_DATATYPE_GET_TCPL, to retrieve the datatype creation property list:
    1. hid_t *ret_id (OUT): buffer for the identifier of the type creation property list.

The close callback in the named datatype class should terminate 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 plugin.

2.7 The Object Function 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 plugin needs to implement:

```
typedef struct H5VL_object_class_t {
    void *(*open)(void *obj, H5VL_loc_params_t loc_params,
        H5I_type_t *opened_type, hid_t dxpl_id, void **req);

    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);

    herr_t (*visit)(void *obj, H5VL_loc_params_t loc_params,
        H5_index_t idx_type, H5_iter_order_t order,
        H5O_iterate_t op, void *op_data, hid_t dxpl_id, void **req);

    herr_t (*get)(void *obj, H5VL_loc_params_t loc_params,
        H5VL_object_get_t get_type, hid_t dxpl_id,
        void **req, va_list arguments);

    herr_t (*misc)(void *obj, H5VL_loc_params_t loc_params,
        H5VL_object_misc_t misc_type, hid_t dxpl_id,
        void **req, va_list arguments);

    MSC - NOT USED
    herr_t (*optional)(void *obj, H5VL_loc_params_t loc_params,
        H5VL_object_optional_t op_type, hid_t dxpl_id,
        void **req, va_list arguments);

    MSC - NOT USED
    herr_t (*close) (void *obj, H5VL_loc_params_t loc_params,
        hid_t dxpl_id, void **req);
} H5VL_object_class_t;
```

The `open` callback in the object class should open 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): The location parameters as explained in section 2.1.
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 plugin.

The copy callback in the object class should copy 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): The location parameters as explained in section 2.1. 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): The location parameters as explained in section 2.1. 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 plugin.

The visit callback in the object class should recursively visit all objects accessible from a specified object and call the user defined function on each one. It returns an herr_t indicating success or failure.

Signature:

```
herr_t (*visit)(void *obj, H5VL_loc_params_t loc_params,
    H5_index_t idx_type, H5_iter_order_t order,
    H5O_iterate_t op, void *op_data, hid_t dxpl_id, void **req);
```

Arguments:

obj	(IN): Pointer to an object where the iteration needs to happen or where the look-up of the target object needs to start.
loc_params	(IN): The location parameters as explained in section 2.1. The type could be H5VL_OBJECT_BY_SELF or H5VL_OBJECT_BY_NAME for this callback.
idx_type	(IN): Type of index.
order	(IN): Order in which to iterate over index.
op	(IN): User-defined function to pass each object to.
op_data	(IN/OUT): User data to pass through to and to be returned by iterator operator function.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.

The get callback in the object class should retrieve 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_t get_type, hid_t dxpl_id,
    void **req, va_list arguments);
```

The get_type argument is an enum:

```
/* types for all object get API routines */
typedef enum H5VL_object_get_t {
    H5VL_OBJECT_EXISTS,    /* Object exists?     */
    H5VL_OBJECT_GET_INFO,  /* object info        */
    H5VL_OBJECT_GET_COMMENT, /* object comment   */
    H5VL_REF_GET_REGION,   /* dataspace of region */
    H5VL_REF_GET_TYPE,     /* type of object     */
    H5VL_REF_GET_NAME      /* object name        */
} H5VL_object_get_t;
```

Arguments:

obj	(IN): A location object where information needs to be retrieved from.
loc_params	(IN): The location parameters as explained in section 2.1.
get_type	(IN): The type of the information to retrieve.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.
arguments	(IN/OUT): argument list containing parameters and output pointers for the get operation.

The arguments argument contains a variable list of arguments depending on the get_type parameter. The following list shows the argument list, in-order, for each type:

- H5VL_OBJECT_EXISTS, to check if an object with name specified in loc_params (type H5VL_OBJECT_BY_NAME) exists:
  1. htri_t *ret (OUT): existence result, 0 if false, 1 if true.
- H5VL_OBJECT_GET_INFO, to retrieve object info:
  1. H5O_info_t *oinfo (OUT): info structure to fill the object info in.
- H5VL_OBJECT_GET_COMMENT, to retrieve the comment on the object specified with obj and loc_params which could have types H5VL_OBJECT_BY_SELF or H5VL_OBJECT_BY_NAME here:
  1. void * (OUT): Buffer to store the comment in.
  2. size_t buf_size (IN): the size of the buffer passed in.
  3. ssize_t *ret (OUT): return the actual size needed to store the comment.
- H5VL_REF_GET_REGION, to retrieve a region reference contained in obj:
  1. hid_t *ret_id (OUT): buffer for the dataspace created from the region reference.
  2. H5R_type_t ret_type (IN): type of region reference (should be H5R_DATASET_REGION.
  3. void *ref (IN): the region reference to open.
- H5VL_REF_GET_TYPE, to retrieve object type a reference points to:
  1. H5O_type_t *type (OUT): buffer to return the object type.
  2. H5R_type_t ret_type (IN): type of region reference to query.
  3. void *ref (IN): the region reference to query.
- H5VL_REF_GET_NAME, to retrieve a name for a referenced object:
  1. ssize_t *ret (OUT): buffer to return the length of the name.
  2. char* name (OUT): buffer to copy the name into.
  3. size_t size (IN): size of the buffer name, if 0, return only the buffer size needed.
  4. H5R_type_t ret_type (IN): type of region reference to query.
  5. void *ref (IN): the region reference to query.

The misc callback in the object class should execute operations in the container on objects as specified in the misc_type parameter.It returns an herr_t indicating success or failure.

Signature:

```
herr_t (*misc)(void *obj, H5VL_loc_params_t loc_params,
    H5VL_object_misc_t misc_type, hid_t dxpl_id,
    void **req, va_list arguments);
```

The misc_type argument is an enum:

```
/* types for all object general operations */
typedef enum H5VL_object_misc_t {
    H5VL_ATTR_RENAME,            /* H5Arename              */
    H5VL_OBJECT_CHANGE_REF_COUNT, /* H5Oincr/decr_refcount */
    H5VL_OBJECT_SET_COMMENT,     /* H5Oset_comment(_by_name) */
    H5VL_REF_CREATE              /* H5Rcreate              */
} H5VL_object_misc_t;
```

Arguments:

`obj`	(IN): A location object for the operation.
`loc_params`	(IN): The location parameters as explained in section 2.1.
`misc_type`	(IN): The type of the operation.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.
`arguments`	(IN/OUT): argument list containing parameters and output pointers for the misc operation.

The `arguments` argument contains a variable list of arguments depending on the `misc_type` parameter. The following list shows the argument list, in-order, for each type:

- `H5VL_ATTR_RENAME`, to rename an attribute under the location object where the `loc_params` could have types `H5VL_OBJECT_BY_SELF` or `H5VL_OBJECT_BY_NAME`:
  1. `char *old_name` (IN): old name of the attribute to rename.
  2. `char *new_name` (IN): the new attribute name to set for the attribute.
- `H5VL_OBJECT_CHANGE_REF_COUNT`, to update the reference count for the object in `obj`:
  1. `int ref_count` (IN): reference count to set on the object.
- `H5VL_OBJECT_SET_COMMENT`, to set a comment on the object where the `loc_params` could have types `H5VL_OBJECT_BY_SELF` or `H5VL_OBJECT_BY_NAME`:
  1. `char *comment` (IN): comment to set on the object.
- `H5VL_REF_CREATE`, to create a reference of an object under the location object `obj`:
  1. `void *ref` (OUT): the region reference created.
  2. `char* name` (IN): Name of the object at the location `obj`.
  3. `H5R_type_t ret_type` (IN): type of region reference to create.
  4. `hid_t* space_id` (IN): Dataspace identifier with selection. Used only for dataset region references; passed as -1 if reference is an object reference, i.e., of type `H5R_OBJECT`.

2.8 The Link Function 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 plugin needs to implement:

```
typedef struct H5VL_link_class_t {
    herr_t (*create)(H5VL_link_create_type_t create_type, void *obj,
        H5VL_loc_params_t loc_params, hid_t lcpl_id,
        hid_t lapl_id, hid_t dxpl_id, void **req);

    herr_t (*move)(void *src_obj, H5VL_loc_params_t loc_params1,
        void *dst_obj, H5VL_loc_params_t loc_params2,
        hbool_t copy_flag, hid_t lcpl, hid_t lapl,
        hid_t dxpl_id, void **req);

    herr_t (*iterate)(void *obj, H5VL_loc_params_t loc_params,
        hbool_t recursive, H5_index_t idx_type, H5_iter_order_t order,
        hsize_t *idx, H5L_iterate_t op, void *op_data, hid_t dxpl_id,
        void **req);

    herr_t (*get)(void *obj, H5VL_loc_params_t loc_params,
        H5VL_link_get_t get_type, hid_t dxpl_id, void **req,
        va_list arguments);

    herr_t (*remove)(void *obj, H5VL_loc_params_t loc_params,
        hid_t dxpl_id, void **req);
} H5VL_link_class_t;
```

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

Signature:

```
    herr_t (*create)(H5VL_link_create_type_t create_type, void *obj,
        H5VL_loc_params_t loc_params, hid_t lcpl_id,
        hid_t lapl_id, hid_t dxpl_id, void **req);
```

The `create_type` argument is an `enum`:

```
/* link create types for VOL */
typedef enum H5VL_link_create_type_t {
    H5VL_LINK_CREATE_HARD, /* Hard Link       */
    H5VL_LINK_CREATE_SOFT, /* Soft Link       */
    H5VL_LINK_CREATE_UD   /* External / UD Link */
} H5VL_link_create_type_t;
```

Arguments:

`create_type`	(IN): type of the link to be created.
`obj`	(IN): Pointer to an object where the link needs to be created from.
`loc_params`	(IN): The location parameters as explained in section 2.1 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 plugin.

As mentioned in the argument list, the `lcpl_id` contains the parameters for the link creation operation depending on the creation type:

- `H5VL_LINK_CREATE_HARD` contains two properties:
  1. `H5VL_LINK_TARGET` (with type `void*`): The target object where the hard link needs to be created to.
  2. `H5VL_LINK_TARGET_LOC_PARAMS` (with type `H5VL_loc_params_t`): The location parameters as explained in section 2.1 for the target object.
- `H5VL_LINK_CREATE_SOFT` contains one property:
  1. `H5VL_LINK_TARGET_NAME` (with type `char*`): The target link where the soft link should point to.
- `H5VL_LINK_CREATE_UD` contains two properties:
  1. `H5VL_LINK_TYPE` (with type `H5L_type_t`): The user defined link class. `H5L_TYPE_EXTERNAL` suggests an external link is to be created.
  2. `H5VL_LINK_UDATA` (with type `void*`): User supplied link information (contains the external link buffer for external links).
  3. `H5VL_LINK_UDATA_SIZE` (with type `size_t`): size of the `udata` buffer.

The `move` callback in the link class should copy or move 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,
    hbool_t copy_flag, hid_t lcpl, hid_t lapl,
    hid_t dxpl_id, void **req);
```

Arguments:

src_obj	(IN): original/source object or file.
loc_params1	(IN): The location parameters for the source object as explained in section 2.1. The type can be only H5VL_OBJECT_BY_NAME in this callback.
dst_obj	(IN): destination object or file.
loc_params1	(IN): The location parameters for the destination object as explained in section 2.1. The type can be only H5VL_OBJECT_BY_NAME in this callback.
copy_flag	(IN): flag to indicate whether link is to be copied (value 1) or moved (value 0).
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 plugin.

The iterate callback in the link class should iterate over links in a group and apply a user defined routine. It returns an herr_t indicating success or failure.

Signature:

```
herr_t (*iterate)(void *obj, H5VL_loc_params_t loc_params,
    hbool_t recursive, H5_index_t idx_type, H5_iter_order_t order,
    hsize_t *idx, H5L_iterate_t op, void *op_data, hid_t dxpl_id,
    void **req);
```

Arguments:

obj	(IN): object where to start iteration or where the lookup for the target object needs to start.
loc_params	(IN): The location parameters for the source object as explained in section 2.1. The type can be only H5VL_OBJECT_BY_NAME or H5VL_OBJECT_BY_SELF in this callback.
recursive	(IN): whether to recursively follow links into subgroups of the specified group.
idx_type	(IN): Type of index which determines the order.
idx	(IN/OUT): iteration position where to start and return position where an interrupted iteration may restart.
op	(IN): User-defined function for the iterator.
op_data	(IN/OUT): User data to pass through to and to be returned by iterator operator function.
dxpl_id	(IN): The data transfer property list.
req	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.

The get callback in the link class should retrieve 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_t get_type, hid_t dxpl_id, void **req,
    va_list arguments);
```

The `get_type` argument is an `enum`:

```
/* types for all link get API routines */
typedef enum H5VL_link_get_t {
    H5VL_LINK_EXISTS,       /* link existence */
    H5VL_LINK_GET_INFO,     /* link info      */
    H5VL_LINK_GET_NAME,     /* link name      */
    H5VL_LINK_GET_VAL       /* link value     */
} H5VL_link_get_t;
```

Arguments:

`obj`	(IN): The file or group object where information needs to be retrieved from.
`loc_params`	(IN): The location parameters for the source object as explained in section 2.1. The type can be only `H5VL_OBJECT_BY_NAME` or `H5VL_OBJECT_BY_IDX` in this callback.
`get_type`	(IN): The type of the information to retrieve.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.
`arguments`	(IN/OUT): argument list containing parameters and output pointers for the get operation.

The `arguments` argument contains a variable list of arguments depending on the `get_type` parameter. The following list shows the argument list, in-order, for each type:

- `H5VL_LINK_EXITS`, to determine whether the link specified in the `loc_params` exits (`loc_params` is of type `H5VL_OBJECT_BY_NAME` only with this type):
  1. `htri_t *ret` (OUT): buffer for the existence of the link (0 for no, 1 for yes).
- `H5VL_LINK_GET_INFO`, to retrieve the link info from the link specified in the `loc_params`:
  1. `H5L_info_t linfo` (OUT): pointer to info structure to fill.
- `H5VL_LINK_GET_NAME`, to retrieve the name of the link specified by the index information in `loc_params` (`loc_params` is of type `H5VL_OBJECT_BY_IDX` only with this type):
  1. `char* name` (OUT): buffer to copy the name into.
  2. `size_t size` (IN): size of the buffer name, if 0, return only the buffer size needed.
  3. `ssize_t *ret` (OUT): buffer to return the length of the link name.
- `H5VL_LINK_GET_VAL`, to retrieve the link value from the link specified in the `loc_params`:

1. `void *buf` (OUT): buffer to put the value into.
2. `size_t size` (IN): size of the passed in buffer.

The `remove` callback in the link class should remove a link from an HDF5 container, and returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*remove)(void *obj, H5VL_loc_params_t loc_params,
    hid_t dxpl_id, void **req);
```

Arguments:

`obj`	(IN): group object or file containing the link.
`loc_params`	(IN): The location parameters for the link to be deleted.The type can be only `H5VL_OBJECT_BY_NAME` or `H5VL_OBJECT_BY_IDX` in this callback.
`dxpl_id`	(IN): The data transfer property list.
`req`	(IN/OUT): A pointer to the asynchronous request of the operation created by the plugin.

2.9 The Asynchronous Function Callbacks

As of now, the HDF5 library does not provide asynchronous API operations. An asynchronous class to manage asynchronous operations was added nevertheless to handle an asynchronous API that might be added in the future:

```
typedef struct H5VL_async_class_t {
    herr_t (*cancel)(void **, H5ES_status_t *);

    herr_t (*test) (void **, H5ES_status_t *);

    herr_t (*wait) (void **, H5ES_status_t *);
} H5VL_async_class_t;
```

The `H5ES_status_t` argument is an `enum`:

```
/* 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_CANCEL     /* Operation has not completed and has
                              been cancelled */
} H5ES_status_t;
```

The `cancel` callback attempts to cancel an asynchronous operation and returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*cancel)(void **req, H5ES_status_t *status);
```

Arguments:
`req` (IN): A pointer to the asynchronous request.
`status` (OUT): result of the cancel operation.

The `test` callback tests an asynchronous operation completion and returns an `herr_t` indicating success or failure.

Signature:

```
herr_t (*test)(void **req, H5ES_status_t *status);
```

Arguments:
`req` (IN): A pointer to the asynchronous request.
`status` (OUT): result of the test operation.

The `wait` callback waits for an asynchronous operation completion and returns an `herr_t` indicating success or failure.

Signature:

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

Arguments:
`req` (IN): A pointer to the asynchronous request.
`status` (OUT): result of the wait operation.

3 Creating and Using an Internal Plugin

Internal plugins are developed internally with the HDF5 library and are required to ship with the entire library to be used. Typically those plugins need to use internal features and functions of the HDF5 library that are not available publicly from the user application.

After implementing the VOL class as described in section 2, the next step would be to allow users to select this plugin to be used. This is done by creating a new API routine to set the plugin on the file access property list. For example, if we create an internal plugin called "dummy" that needs an MPI communicator and info object as information from the user, that routine signature should look like:

```
herr_t H5Pset_fapl_dummy(hid_t fapl_id, MPI_Comm comm, MPI_Info info);
```

The implementation for the above routine should use the internal function:

```
herr_t H5P_set_vol(H5P_genplist_t *plist, H5VL_class_t *vol_cls, const
void *vol_info);
```

that will set the file access using that `fapl_id` to go through the "dummy" plugin. It will also call the copy callback of the "dummy" plugin on the info object (`comm and info`).

A sample implementation for the `H5Pset_fapl_dummy()` could like this:

```
/* DUMMY-specific file access properties */
typedef struct H5VL_dummy_fapl_t {
    MPI_Comm          comm;    /* communicator     */
    MPI_Info          info;    /* MPI information */
} H5VL_dummy_fapl_t;

herr_t
H5Pset_fapl_dummy(hid_t fapl_id, MPI_Comm comm, MPI_Info info)
{
    H5VL_dummy_fapl_t fa;
    H5P_genplist_t *plist;    /* Property list pointer */
    herr_t         ret_value;

    FUNC_ENTER_API(FAIL)

    if(fapl_id == H5P_DEFAULT)
        HGOTO_ERROR(H5E_PLIST, H5E_BADVALUE, FAIL, "can't set values in
            default property list")

    if(NULL == (plist = H5P_object_verify(fapl_id, H5P_FILE_ACCESS)))
        HGOTO_ERROR(H5E_ARGS, H5E_BADTYPE, FAIL, "not a file access
            property list")

    if(MPI_COMM_NULL == comm)
        HGOTO_ERROR(H5E_PLIST, H5E_BADTYPE, FAIL, "not a valid
            communicator")

    /* Initialize driver specific properties */
    fa.comm = comm;
    fa.info = info;

    ret_value = H5P_set_vol(plist, &H5VL_dummy_g, &fa);

done:
    FUNC_LEAVE_API(ret_value)
} /* end H5Pset_fapl_dummy() */
```

At this point, the internal plugin is ready to be used. For more information on how to implement an internal plugin, the native plugin for the HDF5 library is a comprehensive plugin that implements all features of the library and can be used as guide.

4 Creating and Using an External Plugin

External plugins are developed outside of the HDF5 library and do not use any internal HDF5 private functions. They do not require to be shipped with the HDF5 library, but can just link to it from userspace just like an HDF5 application.

4.1 New API Routines for External Plugins

Some callbacks in the VOL class require new API routines for the implementation to be possible. Two new API routines have been added for that matter:

```
hid_t H5VLobject_register(void *obj, H5I_type_t obj_type, const
    H5VL_class_t *cls);
```

to register an `hid_t` with an object `obj` associated with the VOL plugin of class `cls`. This is needed in all iterate and visit callbacks where the plugins internally need to wrap an `hid_t` around an object to call the user defined operation `op` on.

```
herr_t H5VLget_object(hid_t obj_id, void **obj, H5VL_t **vol_plugin);
```

to retrieve the VOL object and plugin structure from an HDF5 identifier (`hid_t`). The plugin structure is defined as:

```
struct H5VL_t {
    const H5VL_class_t *cls;  /* constant class info */
    const char *container_name; /* name of the underlying storage
                                   container */
    unsigned long feature_flags; /* VOL Driver feature Flags */
    int nrefs;                /* number of references by objects
                                   using this struct */
};
```

4.2 Using an External Plugin

Unlike internal plugins, the external plugins cannot create an API routine for applications to use to set the VOL plugin in the file access property list. After implementing the VOL class as described in section 2, the application has to register the plugin with HDF5 library. The function to do that is `H5VLregister()`:

```
hid_t H5VLregister(const H5VL_class_t *cls);
```

where `cls` is a pointer to the external plugin to be used. The identifier returned can be used to set this plugin to be used in the file access property list with this

API routine:

```
herr_t H5Pset_vol(hid_t fapl_id, hid_t plugin_id, const void
    *new_vol_info);
```

where `plugin_id` is the identifier returned from the `H5VLregister()` and `new_vol_info` is the plugin information needed from the application.

The user is required to un-register the plugin from the library when access to the container(s) is terminated using:

```
herr_t H5VLunregister(hid_t plugin_id);
```

5 Interchanging and Stacking VOL Plugins

Accessing an HDF5 container with a VOL plugin different than the one it was created with would be a valid approach as long as the underlying file format is the same. This would be the user's responsibility to ensure that the different plugins are interchangeable.

5.1 Stacking Plugins on Top of Each Other

It would be also possible to stack VOL plugins on top of each other. This notion is similar to the idea of the split VFD, where underneath the split VFD itself, two file drivers would be used, one for the file storing the metadata and another for raw data. Some stackings make sense and others would be erroneous. For example, stacking the native HDF5 plugin on top of a non-HDF5 backend plugin does not make sense and is erroneous. Figure 1 shows a stacking of a remote plugin, where data is distributed remotely, on top of the native h5 plugin, where servers that store the data at remote locations use the h5 file format.

5.2 Mirroring Plugins

Another useful design option is to allow a mirroring plugin, where the HDF5 API calls are forwarded through a mirror plugin to two or more VOL plugins. This is an extention to the stacking feature. Figure 2 shows an example of a VOL mirror that maps HDF5 API calls to an h5 backend plugin and an XML backend plugin.

Another possible VOL plugin could be a statistics plugin that just gathers information on HDF5 API calls and records statistics associated with the number of calls to a specific API functions and corresponding parameters. This plugin would be very useful for profiling purposes. The statistics plugin would be stacked on top of another VOL plugin that actually performs the required access to the file.

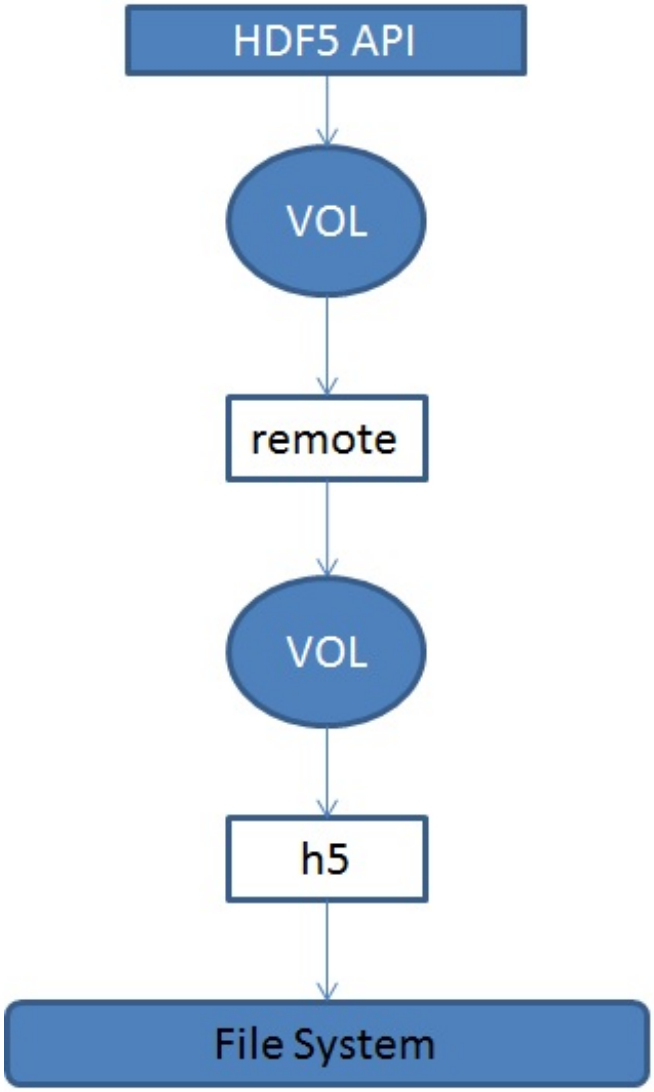

Figure 1: Stacked VOL plugins.

5.3 Implementing Stacked and Mirrored Plugins

A new set of API calls that map directly to the VOL callbacks have been added to the HDF5 library to make stacking and mirroring easy for plugin developers. Similarly to the public VFD (H5FD) routines that call the VFD callbacks directly, we added the following H5VL APIs to the library:

```
/* ATTRIBUTE OBJECT ROUTINES */
void *H5VLattr_create(void *obj, H5VL_loc_params_t loc_params, H5VL_t
    *vol_plugin, const char *attr_name, hid_t acpl_id, hid_t aapl_id,
    hid_t dxpl_id, void **req);
void *H5VLattr_open(void *obj, H5VL_loc_params_t loc_params, H5VL_t
```

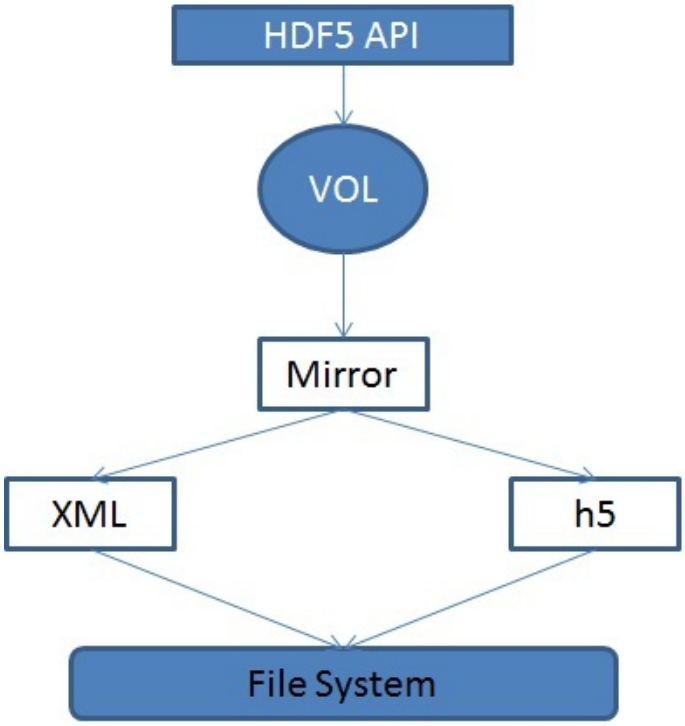

Figure 2: Mirrored VOL plugins.

```
    *vol_plugin, const char *name, hid_t aapl_id, hid_t dxpl_id, void
    **req);
herr_t H5VLattr_read(void *attr, H5VL_t *vol_plugin, hid_t dtype_id, void
     *buf, hid_t dxpl_id, void **req);
herr_t H5VLattr_write(void *attr, H5VL_t *vol_plugin, hid_t dtype_id,
    const void *buf, hid_t dxpl_id, void **req);
herr_t H5VLattr_iterate(void *obj, H5VL_loc_params_t loc_params,
H5VL_t *vol_plugin, H5_index_t idx_type, H5_iter_order_t order,
hsize_t *n, H5A_operator2_t op, void *op_data, hid_t dxpl_id, void
    **req);
herr_t H5VLattr_get(void *attr, H5VL_t *vol_plugin, H5VL_attr_get_t
    get_type, hid_t dxpl_id, void **req, va_list arguments);
herr_t H5VLattr_remove(void *obj, H5VL_loc_params_t loc_params, H5VL_t
*vol_plugin, const char *attr_name, hid_t dxpl_id, void **req);
herr_t H5VLattr_close(void *attr, H5VL_t *vol_plugin, hid_t dxpl_id, void
     **req);

/* DATASE OBJECT ROUTINES */
void *H5VLdataset_create(void *obj, H5VL_loc_params_t loc_params,
H5VL_t *vol_plugin, const char *name, hid_t dcpl_id, hid_t dapl_id, hid_t
     dxpl_id, void **req);
void *H5VLdataset_open(void *obj, H5VL_loc_params_t loc_params, H5VL_t
*vol_plugin, const char *name, hid_t dapl_id, hid_t dxpl_id, void **req);
herr_t H5VLdataset_read(void *dset, H5VL_t *vol_plugin, hid_t
```

```
mem_type_id, hid_t mem_space_id, hid_t file_space_id, hid_t plist_id,
    void *buf, void **req);
herr_t H5VLdataset_write(void *dset, H5VL_t *vol_plugin, hid_t
mem_type_id, hid_t mem_space_id, hid_t file_space_id, hid_t plist_id,
    const void *buf, void **req);
herr_t H5VLdataset_set_extent(void *dset, H5VL_t *vol_plugin, const
    hsize_t size[], hid_t dxpl_id, void **req);
herr_t H5VLdataset_get(void *dset, H5VL_t *vol_plugin,
H5VL_dataset_get_t get_type, hid_t dxpl_id, void **req, va_list
    arguments);
herr_t H5VLdataset_close(void *dset, H5VL_t *vol_plugin, hid_t dxpl_id,
    void **req);

/* DATATYPE OBJECT ROUTINES */
void *H5VLdatatype_commit(void *obj, H5VL_loc_params_t loc_params, H5VL_t
     *vol_plugin, 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 *H5VLdatatype_open(void *obj, H5VL_loc_params_t loc_params, H5VL_t
    *vol_plugin, const char *name, hid_t tapl_id, hid_t dxpl_id, void
    **req);
ssize_t H5VLdatatype_get_binary(void *obj, H5VL_t *vol_plugin, unsigned
    char *buf, size_t size, hid_t dxpl_id, void **req);
herr_t H5VLdatatype_get(void *obj, H5VL_t *vol_plugin,
    H5VL_datatype_get_t get_type, hid_t dxpl_id, void **req, va_list
    arguments);
herr_t H5VLdatatype_close(void *dt, H5VL_t *vol_plugin, hid_t dxpl_id,
void **req);

/* FILE OBJECT ROUTINES */
void *H5VLfile_create(H5VL_t **vol_plugin, const char *name, unsigned
    flags, hid_t fcpl_id, hid_t fapl_id, hid_t dxpl_id, void **req);
void *H5VLfile_open(H5VL_t **vol_plugin, const char *name, unsigned
    flags, hid_t fapl_id, hid_t dxpl_id, void **req);
herr_t H5VLfile_flush(void *obj, H5VL_loc_params_t loc_params, H5VL_t
    *vol_plugin, H5F_scope_t scope, hid_t dxpl_id, void **req);
herr_t H5VLfile_misc(void *file, H5VL_t *vol_plugin, H5VL_file_misc_t
    misc_type, hid_t dxpl_id, void **req, va_list arguments);
herr_t H5VLfile_optional(void *file, H5VL_t *vol_plugin,
H5VL_file_optional_t optional_type, hid_t dxpl_id, void **req, va_list
    arguments);
herr_t H5VLfile_get(void *file, H5VL_t *vol_plugin, H5VL_file_get_t
    get_type, hid_t dxpl_id, void **req, va_list arguments);
herr_t H5VLfile_close(void *file, H5VL_t *vol_plugin, hid_t dxpl_id, void
     **req);

/* GROUP OBJECT ROUTINES */
void *H5VLgroup_create(void *obj, H5VL_loc_params_t loc_params, H5VL_t
*vol_plugin, const char *name, hid_t gcpl_id, hid_t gapl_id, hid_t
    dxpl_id, void **req);
void *H5VLgroup_open(void *obj, H5VL_loc_params_t loc_params, H5VL_t
*vol_plugin, const char *name, hid_t gapl_id, hid_t dxpl_id, void
**req);
herr_t H5VLgroup_get(void *obj, H5VL_t *vol_plugin, H5VL_group_get_t
    get_type, hid_t dxpl_id, void **req, va_list arguments);
```

```
herr_t H5VLgroup_close(void *grp, H5VL_t *vol_plugin, hid_t dxpl_id, void
     **req);

/* LINK OBJECT ROUTINES */
herr_t H5VLlink_create(H5VL_link_create_type_t create_type, void *obj,
H5VL_loc_params_t loc_params, H5VL_t *vol_plugin, hid_t lcpl_id, hid_t
lapl_id, hid_t dxpl_id, void **req);
herr_t H5VLlink_move(void *src_obj, H5VL_loc_params_t loc_params1,
void *dst_obj, H5VL_loc_params_t loc_params2, H5VL_t *vol_plugin,
hbool_t copy_flag, hid_t lcpl_id, hid_t lapl_id, hid_t dxpl_id, void
    **req);
herr_t H5VLlink_iterate(void *obj, H5VL_loc_params_t loc_params,
H5VL_t *vol_plugin, hbool_t recursive, H5_index_t idx_type,
H5_iter_order_t order, hsize_t *idx, H5L_iterate_t op, void *op_data,
    hid_t dxpl_id, void **req);
herr_t H5VLlink_get(void *obj, H5VL_loc_params_t loc_params, H5VL_t
*vol_plugin, H5VL_link_get_t get_type, hid_t dxpl_id, void **req, va_list
     arguments);
herr_t H5VLlink_remove(void *obj, H5VL_loc_params_t loc_params, H5VL_t
*vol_plugin, hid_t dxpl_id, void **req);

/* OBJECT ROUTINES */
void *H5VLobject_open(void *obj, H5VL_loc_params_t loc_params, H5VL_t
    *vol_plugin, H5I_type_t *opened_type, hid_t dxpl_id, void **req);
herr_t H5VLobject_copy(void *src_obj, H5VL_loc_params_t loc_params1,
H5VL_t *vol_plugin1, const char *src_name, void *dst_obj,
H5VL_loc_params_t loc_params2, H5VL_t *vol_plugin2, const char *dst_name,
     hid_t ocpypl_id, hid_t lcpl_id, hid_t dxpl_id, void **req);
herr_t H5VLobject_visit(void *obj, H5VL_loc_params_t loc_params,
H5VL_t *vol_plugin, H5_index_t idx_type, H5_iter_order_t order,
    H5O_iterate_t op, void *op_data, hid_t dxpl_id, void **req);
herr_t H5VLobject_get(void *obj, H5VL_loc_params_t loc_params, H5VL_t
*vol_plugin, H5VL_object_get_t get_type, hid_t dxpl_id, void **req,
    va_list arguments);
herr_t H5VLobject_misc(void *obj, H5VL_loc_params_t loc_params, H5VL_t
*vol_plugin, H5VL_object_misc_t misc_type, hid_t dxpl_id, void **req,
    va_list arguments);
herr_t H5VLobject_optional(void *obj, H5VL_loc_params_t loc_params,
H5VL_t *vol_plugin, H5VL_object_misc_t optional_type, hid_t dxpl_id, void
     **req, va_list arguments);
herr_t H5VLobject_close(void *obj, H5VL_loc_params_t loc_params, H5VL_t
    *vol_plugin, hid_t dxpl_id, void **req);

/* ASYNCHRONOUS ROUTINES */
herr_t H5VLrequest_cancel(void **req, H5VL_t *vol_plugin, H5ES_status_t
    *status);
herr_t H5VLrequest_test(void **req, H5VL_t *vol_plugin, H5ES_status_t
    *status);
herr_t H5VLrequest_wait(void **req, H5VL_t *vol_plugin, H5ES_status_t
    *status);
```

The above API calls should be used in the stacked or mirror plugin to call into the lower plugins indicated by the `vol_plugin` parameter that is added to all

the routines.