User Guide for Developing a Virtual Object Layer Plugin

Mohamad Chaarawi

16th 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. 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 = 0,

    /* 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 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 *attr, 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.

2.3 The Named Datatype Function Callbacks

```
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;
```

2.4 The Dataset Function Callbacks

```
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 xfer_plist_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 xfer_plist_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;
```

2.5 The File Function Callbacks

```
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 *file, H5VL_file_get_t get_type, hid_t dxpl_id,
        void **req, va_list arguments);

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

    herr_t (*optional)(void *file, 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;
```

2.6 The Group Function Callbacks

```
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;
```

2.7 The Link Function Callbacks

```
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;
```

2.8 The Object Function Callbacks

```
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);

    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);

    herr_t (*close) (void *obj, H5VL_loc_params_t loc_params,
        hid_t dxpl_id, void **req);
} H5VL_object_class_t;
```

2.9 The Asynchronous Function Callbacks

```
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;
```

3 Creating 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.

4 Creating 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.