

HDF5 Virtual Object Layer (VOL) User Guide

The HDF Group

5th December 2019



Contents

1	Introduction	1
1.1	The VOL Abstraction Layer	1
1.2	VOL Connectors	2
2	Quickstart	3
2.1	Read The Documentation For The New VOL Connector	3
2.2	Use A VOL-Enabled HDF5 Library	3
2.3	Determine How You Will Set The VOL Connector	3
2.4	If Needed: Update Your Code To Load And Use A VOL Connector	4
2.5	If Using A Plugin: Make Sure The VOL Connector Is In The Search Path	4
2.6	Optional: Set The VOL Connector Via The Environment Variable	4
3	Connector Use	4
3.1	Registration	5
3.2	Connector Versioning	6
3.3	Connector-Specific Registration Calls	6
3.4	H5Pset_vol()	6
3.5	VOL Connector Search Path	6
3.6	Parameter Strings	7
3.7	Environment Variable	7
4	Adapting HDF5 Software to Use the VOL	7
4.1	Specific API Call Substitutions	8
4.1.1	H5Fis_hdf5() → H5Fis_accessible()	8
4.1.2	H5Oget_info[1 2]() → H5Oget_info3()H5Oget_native_info()	8
4.1.3	Optional: H5Ovisit[1 2]() → H5Ovisit3()	8
4.2	Protect Native-Only API Calls	9
5	Using VOL Connectors With The HDF5 Command-Line Tools	9
6	Compatibility	9
6.1	Introspection / Feature Flags	9
6.2	List of HDF5 Native VOL API Calls	10
6.3	List of HDF5 VOL-Independent API Calls	11

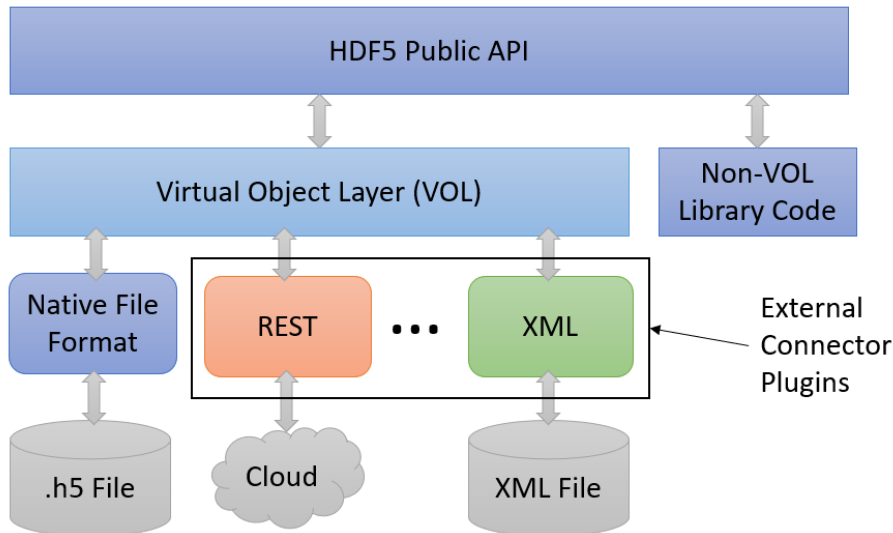
1 Introduction

The virtual object layer 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 a VOL connector, which implements the storage. The user or application gets the benefit of using the familiar and widely-used HDF5 data model and API, but can map the physical storage of the HDF5 file and objects to storage that better meets the application's data needs.

1.1 The VOL Abstraction Layer

The VOL lies just under the public API. When a storage-oriented public API call is made, the library performs a few sanity checks on the input parameters and then immediately invokes a VOL callback, which resolves to an implementation in the VOL connector that was selected when opening or creating the file. The VOL connector then performs whatever operations are needed before control returns to the library, where any final library operations such as assigning IDs for newly created/opened datasets are performed before returning. This means that, for calls that utilize the VOL, all of the functionality is deferred to the VOL connector and the HDF5 library does very little work. An important consideration of this is that most of the HDF5 caching layers (metadata and chunk caches, page buffering, etc.) will not be available as those are implemented in the HDF5 native VOL connector and cannot be easily reused by external connectors.

Figure 1: The VOL Architecture



Not all public HDF5 API calls pass through the VOL. Only calls which require manipulating storage go through the VOL and require a VOL connector author

to implement the appropriate callback. Dataspace, property list, error stack, etc. calls have nothing to do with storage manipulation or querying and do not use the VOL. This may be confusing when it comes to property list calls, since many of those calls set properties for storage. Property lists are just collections of key-value pairs, though, so a particular VOL connector is not required to set or get properties.

Another thing to keep in mind is that not every VOL connector will implement the full HDF5 public API. In some cases, a particular feature like variable-length types may not have been developed yet or may not have an equivalent in the target storage system. Also, many HDF5 public API calls are specific to the native HDF5 file format and are unlikely to have any use in other VOL connectors. A feature/capabilities flag scheme is being developed to help navigate this.

For more information about which calls go through the VOL and the mechanism by which this is implemented, see the connector author and library internals documentation.

1.2 VOL Connectors

A VOL connector can be implemented in several ways: as a shared or static library linked to an application; as a dynamically loaded plugin, implemented as a shared library; and even as an internal connector, built into the HDF5 library itself. This document mostly focuses on external connectors, both libraries and plugins, as those are expected to be much more common than internal implementations.

Many VOL connectors (at various stages of implementation) can be found here:

<https://bitbucket.hdfgroup.org/projects/HDF5VOL>

Not every connector in this collection is actively maintained by The HDF Group. It simply serves as a single location where important VOL connectors can be found. See the documentation in a connector's repository to determine its development status and the parties responsible for it. The collection also includes a VOL template that contains build scripts (Autotools and CMake) and an empty VOL connector "shell" which can be copied and used as a starting point for building new connectors. See the VOL Connector Author Guide for more information on this.

The only current (non-test) internal VOL connector distributed with the library is the native file format connector (the "native VOL connector") which contains the code that handles native HDF5 (*.h5/hdf5) files. In other words, even the canonical HDF5 file format is implemented via the VOL, making it a core part of the HDF5 library and not an optional component which could be disabled. It has not been completely abstracted from the HDF5 library, though, and is treated as a special case. For example, it cannot be unloaded and is always present.

2 Quickstart

The following steps summarize how one would go about using a VOL connector with an application. More information on particular steps can be found later on in this document.

2.1 Read The Documentation For The New VOL Connector

Many VOL connectors will require specific setup and configuration of both the application and the storage. Specific permissions may have to be set, configuration files constructed, and connector-specific setup calls may need to be invoked in the application. In many cases, converting software to use a new VOL connector will be more than just a straightforward drop-in replacement done by specifying a name in the VOL plugin environment variable.

2.2 Use A VOL-Enabled HDF5 Library

The virtual object layer was introduced in HDF5 1.12.0, so you will need that version or later to use it. The particular configuration of the library (serial vs parallel, thread-safe, debug vs production/release) does not matter. The VOL is a fundamental part of the library and cannot be disabled, so any build will do.

On Windows, it's probably best to use the same debug vs release configuration for the application and all libraries in order to avoid C runtime (CRT) issues. On pre-2015 versions of Visual Studio, you'll probably also want to stick to the same Visual Studio version (and thus same CRT version) as well.

When working with a debug HDF5 library, it's probably also wise to build with the "memory sanity checking" feature disabled to avoid accidentally clobbering our memory tracking infrastructure when dealing with buffers obtained from the HDF5 library.

2.3 Determine How You Will Set The VOL Connector

Fundamentally, setting a VOL connector involves modifying the file access property list (fapl) that will be used to open or create the file.

There are essentially three ways to do this:

- Direct use of `H5Pset_vol()`
- Library-specific API calls that call `H5Pset_vol()` for you
- Use the VOL environment variable, which will also call `H5Pset_vol()` for you

Exactly how you go about setting a VOL connector in a fapl, will depend on the complexity of the VOL connector and how much control you have over the application's source code.

2.4 If Needed: Update Your Code To Load And Use A VOL Connector

There are two concerns when modifying the application:

- It may be convenient to add connector-specific setup calls to the application.
- You will also need to protect any API calls which are only implemented in the native VOL connector as those calls will fail when using a non-native VOL connector. See the section entitled "Adapting HDF5 Software to Use the VOL", below. A list of native VOL API calls has been included in an appendix.

In some cases, using the VOL environment variable will work well for setting the connector and any associated storage setup and the application will not use API calls that are not supported by the VOL connector. In this case, no application modification will be necessary.

2.5 If Using A Plugin: Make Sure The VOL Connector Is In The Search Path

The default location for all HDF5 plugins is set at configure time when building the HDF5 library. This is true for both CMake and the Autotools. The default locations for the plugins on both Windows and POSIX systems is listed further on in this document.

2.6 Optional: Set The VOL Connector Via The Environment Variable

In place of modifying the source code of your application, you may be able to simply set the `HDF5_VOL_CONNECTOR` environment variable (see below). This will automatically use the specified VOL in place of the native VOL connector.

3 Connector Use

Before a VOL connector can be set in a fapl, it must be registered with the library (`H5Pset_vol()` requires the connector's `hid_t` ID) and, if a plugin, it must be discoverable by the library at run time.

3.1 Registration

Before a connector can be used, it must be registered. This loads the connector into the library and give it an HDF5 `hid_t` ID. The `H5VLregister_connector` API calls are used for this.

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

hid_t H5VLregister_connector_by_name(const char *connector_name,
hid_t vipl_id)

hid_t H5VLregister_connector_by_value(H5VL_class_value_t connector_value,
hid_t vipl_id)
```

When used with a plugin, these functions will check to see if an appropriate plugin with a matching name, value, etc. is already loaded and check the plugin path (see above) for matching plugins if this is not true. The functions return `H5I_INVALID_HID` if they are unable to register the connector. Most people will use the `_by_name()` or `_by_value` versions of this API call. The call that takes an `H5VL_class_t` struct is considered a lower-level call this is more commonly used by connector authors in connector-specific setup/registration code. Many VOL connectors will provide a connector-specific init call that will load and register the connector for you.

Note the two ways that a VOL connector can be identified: by a name or by a connector-specific numerical value (`H5VL_class_value_t` is typedef'd to an integer). The name and value for a connector can be found in the connector's documentation or public header file.

Each call also takes a VOL initialization property list (`vipl`). The library adds no properties to this list, so it is entirely for use by connector authors. Set this to `H5P_DEFAULT` unless instructed differently by the documentation for the VOL connector.

As far as the library is concerned, connectors do not need to be explicitly unregistered as the library will unload the plugin and close the ID when the library is closed. If you want to close a VOL connector ID, either `H5VLunregister_connector()` or `H5VLclose()` can be used (they have the same internal code path). The library maintains a reference count on all open IDs and will not do the actual work of closing an ID until its reference count drops to zero, so it's safe to close IDs anytime after they are used, even while an HDF5 file that was opened with that connector is still open.

Note that it's considered an error to unload the native VOL connector. The library will prevent this. This means that, for the time being, the native VOL connector will always be available. This may change in the future so that the memory footprint of the native VOL connector goes away when not in use.

3.2 Connector Versioning

PRELIMINARY

This feature is under active development and is either incomplete or may change in behavior before the final release.

There is currently no recommended or library-inherent way to version VOL connectors. A `version` field is included in the `H5VL_class_t` struct which holds a connector's information and callbacks, but this is intended for use when versioning the class struct.

3.3 Connector-Specific Registration Calls

Most connectors will provide a special API call which will set the connector in the `fapl`. These will often be in the form of `H5Pset_fapl_<name>()`. For example, the DAOS VOL connector <https://bitbucket.hdfgroup.org/projects/HDF5VOL/repos/daos-vol> provides a `H5Pset_fapl_daos()` API call which will take MPI parameters and make this call. See the connector's documentation or public header file(s) for more information.

3.4 H5Pset_vol()

This is the main library API call for setting the VOL connector in a file access property list. Its signature is:

```
herr_t H5Pset_vol(hid_t plist_id, hid_t vol_id, vol_info)
```

It takes the ID of the file access property list, the ID of the registered VOL connector, and a pointer to whatever connector-specific data the connector is expecting. This will usually be a data struct specified in the connector's header.

As mentioned above, many connectors will provide their own replacement for this call. See the connector's documentation for more information.

3.5 VOL Connector Search Path

Dynamically loaded VOL connector plugins are discovered and loaded by the library using the same mechanism as dataset/group filter plugins. The default locations are:

Default locations:

POSIX systems: `/usr/local/hdf5/lib/plugin`

Windows: `%ALLUSERSPROFILE%/hdf5/lib/plugin`

These default locations can be overridden by setting the `HDF5_PLUGIN_PATH` environment variable. There are also public H5PL API calls which can be used

to add, modify, and remove search paths. The library will only look for plugins in the specified plugin paths. By default, it will NOT find plugins that are simply located in the same directory as the executable.

3.6 Parameter Strings

Each VOL connector is allowed to take in a parameter string which can be parsed via `H5VLconnector_str_to_info()` to get an info struct which can be passed to `H5Pset_vol()`.

```
herr_t H5VLconnector_str_to_info(const char *str, hid_t connector_id,
void **info)
```

And the obtained info can be freed via:

```
herr_t H5VLfree_connector_info(hid_t connector_id, void *vol_info)
```

Most users will not need this functionality as they will be using either connector-specific setup calls which will handle registering and configuring the connector for them or they will be using the environment variable (see below).

3.7 Environment Variable

The HDF5 library allows specifying a default VOL connector via an environment variable: `HDF5_VOL_CONNECTOR`. The value of this environment variable should be set to "`vol_connector_name <parameters>`".

This will perform the equivalent of:

1. `H5VLregister_connector_by_name()` using the specified connector name
2. `H5VLconnector_str_to_info()` using the specified parameters. This will go through the connector we got from the previous step and should return a VOL info struct from the parameter string in the environment variable.
3. `H5Pset_vol()` on the default `fppl` using the obtained ID and info.

The environment variable is parsed once, at library startup. Since the environment variable scheme just changes the default connector, it can be overridden by subsequent calls to `H5Pset_vol()`. The `<parameters>` is optional, so for connectors which do not require any special configuration parameters you can just set the environment variable to the name.

4 Adapting HDF5 Software to Use the VOL

The VOL was engineered to be as unobtrusive as possible and, when a connector which implements most/all of the data model functionality is in use, most applications should require little, if any, modification. As mentioned in the quick start section, most modifications will probably consist of connector setup code (which can usually be accomplished via the environment variable) and protecting native file format functions.

4.1 Specific API Call Substitutions

4.1.1 H5Fis_hdf5() → H5Fis_accessible()

H5Fis_hdf5() does not take a file access property list (fapl). As this is where the VOL connector is specified, this call cannot be used with arbitrary connectors. As a VOL-enabled replacement, H5Fis_accessible() has been added to the library. It has the same semantics as H5Fis_hdf5(), but takes a fapl so it can work with any VOL connector.

Note that, at this time, H5Fis_hdf5() *always* uses the native VOL connector, regardless of the settings of environment variables, etc.

4.1.2 H50get_info[1|2]() → H50get_info3()H50get_native_info()

PRELIMINARY

This feature is under active development and is either incomplete or may change in behavior before the final release.

The H50get_info1() and H50get_info2() family of HDF5 API calls are often used by user code to obtain information about an object in the file, however these calls return a struct which contains native information and are thus currently handled via the native connector's *object optional* callback and so are not handled by most external VOL connectors.

These calls will be separated into two functions: one for getting an object's data model information (H50get_info3(), which will go through the *object get* callback), and another for returning an object's native file format information (H50get_native_info(), which will go through the native connector's *object optional* callback). These API calls will use new H50_info2_t and H50_native_info_t structs, respectively, to hold the returned data.

This feature is under development and this section will be updated when the feature is complete.

4.1.3 **Optional:** H50visit[1|2]() → H50visit3()

PRELIMINARY

This feature is under active development and is either incomplete or may change in behavior before the final release.

The H50visit() family of API functions uses the H50_info_t struct in its callback function:

```
typedef herr_t (*H50_iterate_t)(hid_t obj, const char *name, const
H50_info_t *info, void *op_data)
```

A new callback that takes the updated, data-model-only struct will replace the old callback in new `H5Ovisit3` API calls. Note that replacing this function is optional as it goes through the VOL's *object specific* callback.

As with the `H5Oget_info()` work, this feature is under development and this section will be updated when the feature is complete.

4.2 Protect Native-Only API Calls

There is currently no simple way to determine if the VOL connector used for a file or object is the native VOL connector. You will have to track this in your application and not make native-only VOL calls with non-native VOL connectors.

A list of native-only connector calls is given below.

We are working on a mechanism that will allow detecting when the VOL connector is the native VOL connector, however a good mechanism for this is complicated as some VOL connectors will be passthrough VOL connectors (or even splitters) which ultimately resolve down to the native VOL connectors.

5 Using VOL Connectors With The HDF5 Command-Line Tools

The best way to use a particular VOL connector with the HDF5 tools, both command-line and GUI tools, is to specify the VOL connector using the VOL connector environment variable (`HDF5_VOL_CONNECTOR` - see above).

The HDF5 tools currently do not have an option to specify a VOL connector on the command line, though this option will be added soon.

6 Compatibility

6.1 Introspection / Feature Flags

PRELIMINARY

This feature is under active development and is either incomplete or may change in behavior before the final release.

Some VOL connectors will not implement all the functionality available in the HDF5 library, especially in the early stages of development. Unfortunately, there is currently no easy way to determine if a VOL connector implements particular API calls or groups of calls.

We are working on an introspection mechanism, possibly involving a set of feature or capabilities flags and will have that available in a future release.

6.2 List of HDF5 Native VOL API Calls

These API calls will probably fail when used with terminal VOL connectors other than the native HDF5 file format connector. Their use should be protected in code that uses arbitrary VOL connectors. Note that some connectors may, in fact, implement some of this functionality as it is possible to mimic the native HDF5 connector, however this will probably not be true for most non-native VOL connectors.

H5Aget_num_attrs (deprecated) H5Aiterate1 (deprecated)
H5Ddebug H5Dformat_convert H5Dget_chunk_index_type H5Dget_chunk_info H5Dget_chunk_info_by_coord H5Dget_chunk_storage_size H5Dget_num_chunks H5Dread_chunk H5Dwrite_chunk
H5FD*
H5Fclear_elink_file_cache H5Fformat_convert H5Fget_dset_no_attrs_hint H5Fget_eoa H5Fget_file_image H5Fget_filesize H5Fget_free_sections H5Fget_freespace H5Fget_info1 (deprecated) H5Fget_info2 H5Fget_mdc_config H5Fget_mdc_hit_rate H5Fget_mdc_image_info H5Fget_mdc_logging_status H5Fget_mdc_size H5Fget_metadata_read_retry_info H5Fget_mpi_atomicity H5Fget_page_buffering_stats H5Fget_vfd_handle H5Fincrement_filesize H5Fis_hdf5 (deprecated) H5Freset_mdc_hit_rate_stats H5Freset_page_buffering_stats H5Fset_dset_no_attrs_hint H5Fset_latest_format (deprecated) H5Fset_libver_bounds H5Fset_mdc_config H5Fset_mpi_atomicity H5Fstart_mdc_logging H5Fstart_swmr_write H5Fstop_mdc_logging

H5Gget_comment (deprecated) H5Giterate (deprecated) H5Gget_info H5Gget_info_by_name H5Gget_info_by_idx H5Gget_objinfo (deprecated) H5Gget_objname_by_idx (deprecated) H5Gget_objtype_by_idx (deprecated) H5Gset_comment (deprecated)
H5Oare_mdc_flushes_disabled H5Odisable_mdc_flushes H5Oenable_mdc_flushes H5Oget_comment H5Oget_comment_by_name H5Oget_info_by_idx1 (deprecated) H5Oget_info_by_idx2 H5Oget_info_by_name1 (deprecated) H5Oget_info_by_name2 H5Oget_info1 (deprecated) H5Oget_info2 H5Oset_comment H5Oset_comment_by_name

Table 1: Alphabetical list of HDF5 API calls specific to the native VOL connector

6.3 List of HDF5 VOL-Independent API Calls

These HDF5 API calls do not depend on a particular VOL connector being loaded.

H5*
H5Dfill H5Dgather H5Diterate H5Dscatter H5Dvlen_reclaim (deprecated) H5Dvlen_get_buf_size
H5E*
H5I*
H5Lis_registered H5Lregister H5Lunpack_elink_val H5Lunregister
H5PL*
H5P*
H5S*
H5T* (non-committed)

H5VL*
H5Z*

Table 2: Alphabetical list of VOL-independent HDF5 API calls