# HDF5 Subfiling VFD User's Guide

**Jordan Henderson**

This document aims to be a helpful guide on how to use the HDF5 Subfiling Virtual File Driver within an HDF5 application.

# Revision History

| Version Number | Date | Comments |
|---|---|---|
| v1 | Jan. 25, 2023 | Version 1 drafted. |
| v2 | Jun. 19, 2023 | Added notes about Subfiling VFD's stripe count setting |

# Contents

# Glossary

**File Access Property List (FAPL)** An HDF5 Property List object that is used to control how a file is accessed.. 9, 11, 19–23

**Virtual File Driver (VFD)** An implementation of a Virtual File Driver ”class” – a structure containing function pointers and other data required for Virtual File operations.. 5, 19–22

# 1. Introduction and overview

The HDF5 Subfiling Virtual File Driver (VFD) is an MPI-based File Driver that was introduced in the HDF5 1.14.0 release. This File Driver allows an HDF5 application to create HDF5 files which are distributed across a collection of ”*subfiles*” in equal-sized data segment ”*stripes*.” I/O to the logical HDF5 file is directed to the appropriate ”*subfile*” according to the Subfiling VFD configuration and a system of ”*I/O Concentrators*.” For more information on the general design of this VFD, refer to the Subfiling RFC.

By allowing a user to configure its various parameters, such as the data stripe size, number of subfiles, etc., the Subfiling VFD aims to enable an HDF5 application to achieve a middle ground between the single shared file and file-per-process approaches to parallel file I/O for a particular machine that the application is running on. In general, the goal is to enable these applications to avoid some of the complexity of the file-per-process approach while also minimizing locking issues that can occur on a parallel file system with the single shared file approach at large scales.

The Subfiling feature consists of two major components: the Subfiling VFD and an underlying I/O Concentrator VFD. The Subfiling VFD is responsible for managing Subfiling metadata and configuration information, as well as breaking down I/O requests into the appropriate offset/length pairs based on the Subfiling configuration and forwarding those requests to an I/O Concentrator VFD that is stacked under the Subfiling VFD. The I/O Concentrator VFD is responsible for receiving I/O offset/length pairs from the Subfiling VFD and forwarding each of these I/O requests to the I/O Concentrator (typically, an MPI rank/dedicated thread/etc.) that is controlling the relevant portion of the logical HDF5 file.

The Subfiling feature was designed to allow different types of I/O Concentrator VFDs in the future, but currently includes a reference implementation of an I/O Concentrator VFD (called the ”IOC VFD”), which controls a dedicated set of MPI ranks acting as I/O Concentrators. Each of those MPI ranks hosts an I/O thread pool and uses POSIX I/O calls (e.g., pread/pwrite) within those I/O threads. Figure 1 graphically presents the current architecture of the Subfiling feature.

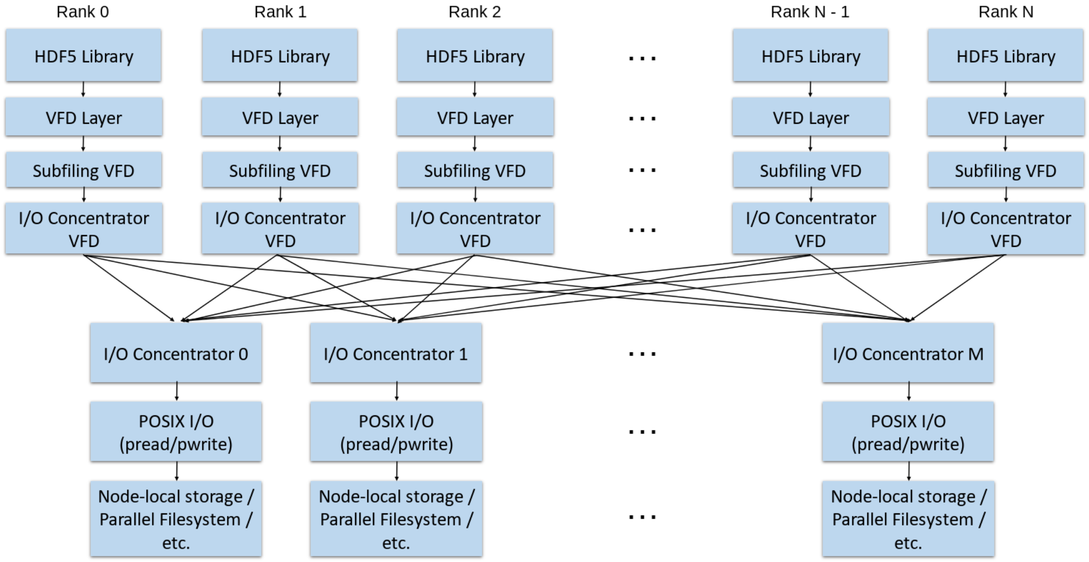
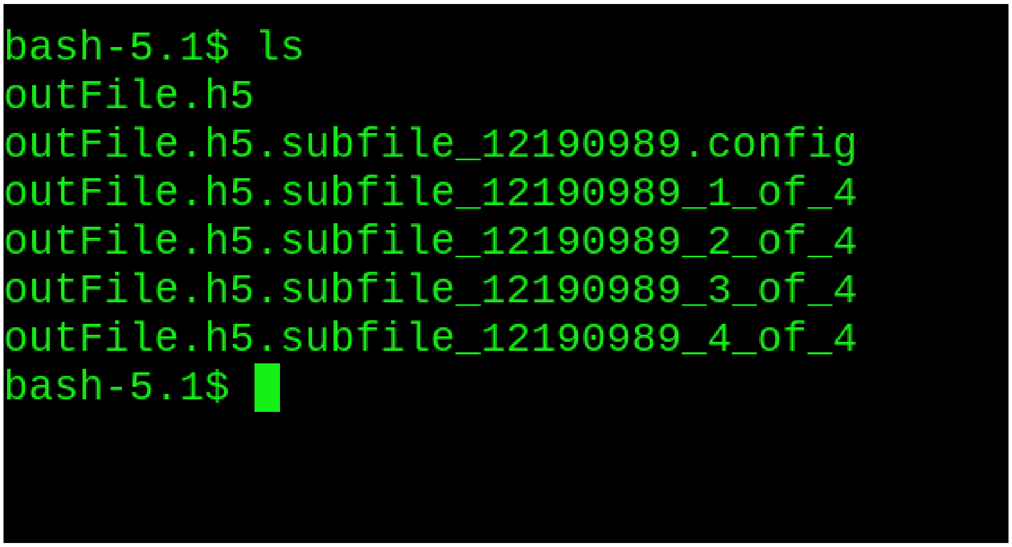


**Figure 1** – Subfiling feature architecture overview

When an HDF5 file is written with the Subfiling VFD, several different files are created: The HDF5 stub file, a Subfiling configuration file and the various subfiles. For example, an HDF5 file called ”outFile.h5” is created with the Subfiling VFD, along with four subfiles, Figure 2.

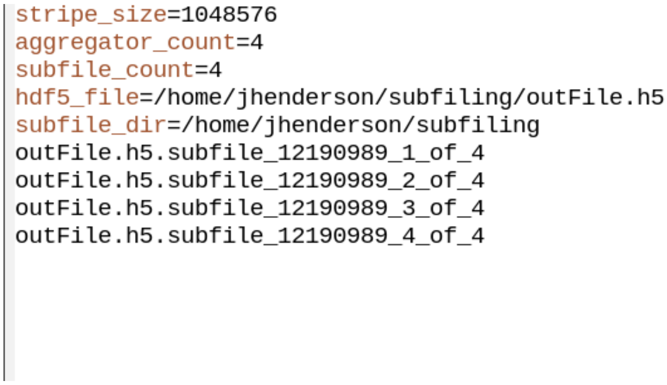


```
bash-5.1$ ls
outFile.h5
outFile.h5.subfile_12190989.config
outFile.h5.subfile_12190989_1_of_4
outFile.h5.subfile_12190989_2_of_4
outFile.h5.subfile_12190989_3_of_4
outFile.h5.subfile_12190989_4_of_4
bash-5.1$ █
```

**Figure 2** – Subfiling feature output files

The first of these files, the HDF5 stub file, looks like a standard HDF5 file but only contains a small portion of the file's metadata and is primarily used for HDF5 to determine when a file was written with the Subfiling VFD.

The next file created is a Subfiling configuration file. The Subfiling VFD keeps track of its configuration by writing it into the previously mentioned HDF5 stub file as metadata. However, this configuration information also gets written out to a Subfiling configuration file so external tools can obtain that information and access Subfiling VFD-based HDF5 files without needing to involve HDF5. An example of the Subfiling configuration file is presented in Figure 3.



```
stripe_size=1048576
aggregator_count=4
subfile_count=4
hdf5_file=/home/jhenderson/subfiling/outFile.h5
subfile_dir=/home/jhenderson/subfiling
outFile.h5.subfile_12190989_1_of_4
outFile.h5.subfile_12190989_2_of_4
outFile.h5.subfile_12190989_3_of_4
outFile.h5.subfile_12190989_4_of_4
```

**Figure 3** – Subfiling configuration file

Finally, the created subfiles (`outFile.h5.subfile_12190989_X_of_4`) simply contain the raw file data, distributed in a round-robin fashion across the subfiles according to the Subfiling stripe size.

# 2. Usage

## 2.1. Building

Before enabling the Subfiling VFD, it has a few requirements that have to be satisfied:

- The VFD requires that HDF5 is built with parallel support using an MPI implementation that supports at least the MPI-3 standard,
- The VFD requires a C11-compliant compiler since it currently makes use of the C11 atomic type primitives from `stdatomic.h`,
- The VFD is officially only supported on Linux systems due to the direct use of pthreads for threading. While the VFD may work under MinGW or similar, this is not tested.

If these requirements are satisfied, the Subfiling VFD can be enabled by building using either CMake, Section 2.1.1, or Autotools, Section 2.1.2. HDF5 will check for these requirements and then build the VFD into the library.

### 2.1.1. CMake

```
cmake -DHDF5_ENABLE_PARALLEL=ON -DHDF5_ENABLE_SUBFILING_VFD=ON ..
```

### 2.1.2. Autotools

```
./configure --enable-parallel --enable-subfiling-vfd=yes
```

## 2.2. Loading

To load and use the Subfiling VFD within an HDF5 application, a few steps must first be taken to ensure proper operation.

### 2.2.1. Initialize MPI with MPI_THREAD_MULTIPLE threading support

The Subfiling VFD uses a combination of MPI and threading internally to achieve its purpose. This means that a parallel HDF5 application wishing to use the Subfiling VFD should call `MPI_Init_thread` (rather than `MPI_Init`) at the beginning of the application. Further, any particular thread created by the VFD may make MPI calls at any time, so the VFD requires that MPI is initialized with support for this behavior by passing the `MPI_THREAD_MULTIPLE` flag to `MPI_Init_thread`. See 2.4 for a minimal example of using the Subfiling VFD which does this. After calling `MPI_Init_thread`, the application should always check that the level of threading support provided by the MPI implementation is at least `MPI_THREAD_MULTIPLE` before attempting to use the VFD. For example:

```
int
main(int argc, char **argv)
{
    int mpi_thread_required = MPI_THREAD_MULTIPLE;
    int mpi_thread_provided = 0;

    if (MPI_SUCCESS != MPI_Init_thread(&argc, &argv,
                                       mpi_thread_required,
                                       &mpi_thread_provided))
        return -1;

    if (mpi_thread_provided < mpi_thread_required)
        return -1;

    ...

    MPI_Finalize();

    return 0;
}
```

Failing to perform this check may result in odd and difficult to diagnose errors when the MPI implementation doesn't provide the MPI_THREAD_MULTIPLE level of threading support.

**NOTE:** If using MPICH for an MPI implementation, it may be necessary on some systems to set the MPICH_MAX_THREAD_SAFETY environment variable to "multiple", as in:

```
export MPICH_MAX_THREAD_SAFETY=multiple
```

in order to have access to the required MPI_THREAD_MULTIPLE level of threading support when calling MPI_Init_thread.

### 2.2.2. Set any MPI parameters

The Subfiling VFD checks for any specially set MPI parameters during initialization, so if the HDF5 application needs to use a special MPI Communicator and/or MPI Info object, it should make use of the H5Pset_mpi_params API routine to set those MPI parameters on a File Access Property List (FAPL) to be used for file access before making calls to any Subfiling VFD-related API routines. In the absence of any specially set MPI parameters, the Subfiling VFD will default to using MPI_COMM_WORLD for the MPI Communicator and MPI_INFO_NULL for the MPI Info object.

```
hid_t fapl_id = H5Pcreate(H5P_FILE_ACCESS);
/* Set any special MPI parameters */
H5Pset_mpi_params(fapl_id, MPI_COMM_WORLD, MPI_INFO_NULL);
```

### 2.2.3. Set Subfiling VFD on a FAPL

Once these initial steps have been performed within an HDF5 application, using the Subfiling VFD simply entails setting up a File Access Property List (FAPL) with Subfiling VFD access, as in:

```
hid_t fapl_id = H5Pcreate(H5P_FILE_ACCESS);
/* Set any special MPI parameters */
H5Pset_mpi_params(fapl_id, MPI_COMM_WORLD, MPI_INFO_NULL);
/* Optional Subfiling configuration can be provided rather than NULL */
H5Pset_fapl_subfiling(fapl_id, NULL);
hid_t file_id = H5Fcreate("file.h5", H5F_ACC_TRUNC, H5P_DEFAULT, fapl_id);
```

See sections 2.4 and 2.5 for minimal examples of using the Subfiling VFD with default and custom configurations.

### 2.2.4. Load by environment variable

Though mostly for testing purposes, the Subfiling VFD can also be loaded by setting the `HDF5_DRIVER` environment variable, as in:

```
export HDF5_DRIVER=subfiling
```

When the Subfiling VFD is loaded this way, the only way to configure the VFD in a non-default manner is through the use of environment variables, as described in section 2.3.2.

**NOTE:** The HDF5 application must still fulfill the requirements in the previous sections (2.2.1 and 2.2.2) to load the Subfiling VFD in this manner.

## 2.3. Configuring

There are two main ways to configure the Subfiling VFD: providing a Subfiling configuration structure to the `H5Pset_fapl_subfiling` API routine or setting environment variables.

### 2.3.1. By File Access Property List

The `H5Pset_fapl_subfiling` API routine accepts a pointer to a `H5FD_subfiling_config_t` structure for configuration (defined in A.5). If this parameter is `NULL`, the Subfiling VFD will use the default settings of:

- Default ”IOC” VFD used for the I/O Concentrator VFD
- 32MiB data stripe size
- 1 I/O Concentrator per machine node
- 4 worker threads per I/O Concentrator

### 2.3.2. By environment variables

The Subfiling VFD can also be configured with a few different environment variables that generally take precedence over any Subfiling configuration parameters set on a FAPL:

- `H5FD_SUBFILING_STRIPE_SIZE` - This environment variable controls the size of the data stripes written to subfiles. It is interpreted as a `long long` value and must be $> 0$.

- `H5FD_SUBFILING_IOC_PER_NODE` - This environment variable controls the number of I/O Concentrators that will be assigned to each machine node. It is interpreted as a `long` value and must be $> 0$.
- `H5FD_SUBFILING_IOC_SELECTION_CRITERIA` - This environment variable is intended to be used in conjunction with the I/O Concentrator selection method specified in a Subfiling configuration structure set on a File Access Property List (FAPL). It enables the user to specify extra criteria for how the Subfiling VFD should operate when selection I/O Concentrators. Refer to the Subfiling VFD header for more information on allowed values for this environment variable.
- `H5FD_SUBFILING_SUBFILE_PREFIX` - This environment variable allows setting of a prefix that is prepended to the filenames generated for subfiles. It is interpreted as a string value.
- `H5FD_IOC_THREAD_POOL_SIZE` - This environment variable controls the number of worker threads per I/O Concentrator. It is interpreted as an `int` value and must be $> 0$.

Note that modifying the stripe count from its default setting is expected to be a less common operation with the Subfiling VFD. Therefore, an environment variable for changing the value has not currently been provided at this time. If an HDF5 application needs to modify the stripe count value (refer to section 3 for information on when modifying the stripe count value is necessary), the Subfiling VFD will need to be configured by a File Access Property List. However, note that any settings in environment variables will still take precedence, even when the VFD is configured by a File Access Property List.

## 2.4. Minimal example

The following is a minimal example of creating a Subfiling-based HDF5 file with the default Subfiling configuration of a 32MiB stripe size, 1 I/O Concentrator per machine node and 4 worker threads per I/O Concentrator.

```
#include "hdf5.h"

int
main(int argc, char **argv)
{
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Info info = MPI_INFO_NULL;
    hid_t    file_id = H5I_INVALID_HID;
    hid_t    fapl_id = H5I_INVALID_HID;
    int      mpi_thread_required = MPI_THREAD_MULTIPLE;
    int      mpi_thread_provided = 0;

    /* Initialize MPI with threading support */
    MPI_Init_thread(&argc, &argv, mpi_thread_required, &mpi_thread_provided);
    if (mpi_thread_provided != mpi_thread_required)
        MPI_Abort(comm, -1);

    /*
     * Set up File Access Property List with MPI
     * parameters for the Subfiling VFD to use
     */
    fapl_id = H5Pcreate(H5P_FILE_ACCESS);
    H5Pset_mpi_params(fapl_id, comm, info);

    /*
     * Set Subfiling VFD on FAPL using default settings
     * (use IOC VFD, 1 IOC per node, 32MiB stripe size)
     */
    H5Pset_fapl_subfiling(fapl_id, NULL);

    /*
     * Create a new Subfiling-based HDF5 file
     */
    file_id = H5Fcreate("my_h5_file.h5", H5F_ACC_TRUNC, H5P_DEFAULT, fapl_id);

    /* Close/free resources */
    H5Pclose(fapl_id);
    H5Fclose(file_id);

    MPI_Finalize();

    return 0;
}
```

## 2.5. Minimal example with custom Subfiling configuration

The following is a minimal example of creating a Subfiling-based HDF5 file with a custom Subfiling configuration of a 1MiB stripe size and 2 worker threads per I/O Concentrator.

```
#include "hdf5.h"

int
main(int argc, char **argv)
{
    H5FD_subfiling_config_t subf_config;
    H5FD_ioc_config_t ioc_config;
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Info info = MPI_INFO_NULL;
    hid_t    file_id = H5I_INVALID_HID;
    hid_t    fapl_id = H5I_INVALID_HID;
    int      mpi_thread_required = MPI_THREAD_MULTIPLE;
    int      mpi_thread_provided = 0;

    /* Initialize MPI with threading support */
    MPI_Init_thread(&argc, &argv, mpi_thread_required, &mpi_thread_provided);
```

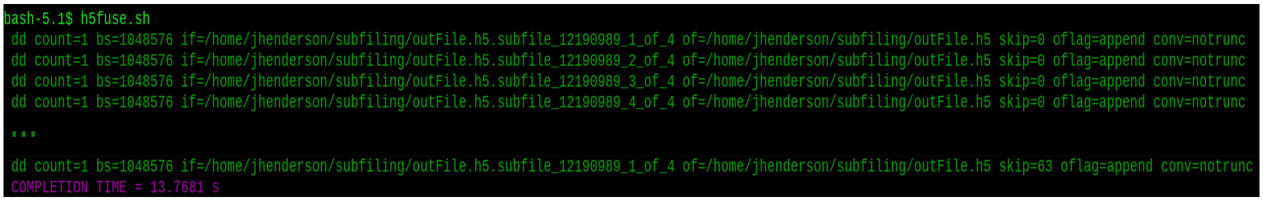
```
if (mpi_thread_provided != mpi_thread_required)
    MPI_Abort(comm, -1);

/*
 * Set up File Access Property List with MPI
 * parameters for the Subfiling VFD to use
 */
fapl_id = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_mpi_params(fapl_id, comm, info);

/*
 * Get a default Subfiling and IOC configuration
 */
H5Pget_fapl_subfiling(fapl_id, &subf_config);
H5Pget_fapl_ioc(fapl_id, &ioc_config);

/*
 * Set Subfiling configuration to use a 1MiB
 * stripe size.
 */
subf_config.shared_cfg.stripe_size = 1048576;

/*
 * Set IOC configuration to use 2 worker threads
 * per IOC instead of the default setting.
 */
ioc_config.thread_pool_size = 2;

/*
 * Set our new configuration on the IOC
 * FAPL used for Subfiling
 */
H5Pset_fapl_ioc(subf_config.ioc_fapl_id, &ioc_config);

/*
 * Finally, set our new Subfiling configuration
 * on the original FAPL
 */
H5Pset_fapl_subfiling(fapl_id, &subf_config);

/*
 * Create a new Subfiling-based HDF5 file
 */
file_id = H5Fcreate("my_h5_file.h5", H5F_ACC_TRUNC, H5P_DEFAULT, fapl_id);

/* Close/free resources */
```

```
    H5Pclose(fapl_id);
    H5Fclose(file_id);

    MPI_Finalize();

    return 0;
}
```

## 2.6. H5fuse script

Included with the Subfiling feature is a new HDF5 tool, called *H5fuse*, which comes in the form of a Bash script ("h5fuse.sh"). This script reads a Subfiling configuration file and then uses dd(1) to combine the various subfiles back into a single, regular HDF5 file. By default, the H5fuse tool looks for a Subfiling configuration file in the directory that it is run from. Still, it can also be pointed to a particular configuration file with a command-line option.

During operation, the H5fuse tool will print out the `dd` operations that it is performing to stitch each of the subfiles together and will print out the total time elapsed once it has finished, Figure 4. The newly-created file can then be accessed by HDF5 or HDF5-related tools as though it had been written without the Subfiling VFD.



```
bash-5.1$ h5fuse.sh
 dd count=1 bs=1048576 if=/home/jhenderson/subfiling/outFile.h5.subfile_12190989_1_of_4 of=/home/jhenderson/subfiling/outFile.h5 skip=0 oflag=append conv=notrunc
 dd count=1 bs=1048576 if=/home/jhenderson/subfiling/outFile.h5.subfile_12190989_2_of_4 of=/home/jhenderson/subfiling/outFile.h5 skip=0 oflag=append conv=notrunc
 dd count=1 bs=1048576 if=/home/jhenderson/subfiling/outFile.h5.subfile_12190989_3_of_4 of=/home/jhenderson/subfiling/outFile.h5 skip=0 oflag=append conv=notrunc
 dd count=1 bs=1048576 if=/home/jhenderson/subfiling/outFile.h5.subfile_12190989_4_of_4 of=/home/jhenderson/subfiling/outFile.h5 skip=0 oflag=append conv=notrunc

 ...

 dd count=1 bs=1048576 if=/home/jhenderson/subfiling/outFile.h5.subfile_12190989_1_of_4 of=/home/jhenderson/subfiling/outFile.h5 skip=63 oflag=append conv=notrunc
 COMPLETION TIME = 13.7681 s
```

**Figure 4** – Output from H5fuse script

## 2.7. HDF5 tools support

There are a few different ways that HDF5's tools can be used with a Subfiling VFD-based HDF5 file. The most obvious way would be to use the H5fuse tool to convert the Subfiling VFD-based file into a regular HDF5 file and then utilize HDF5's tools normally.

Alternatively, HDF5's tools can also load and use the Subfiling VFD directly to access one of these files. For example,

```
# Load the Subfiling VFD by environment variable and
# dump "my_subfiling_file.h5"
HDF5_DRIVER=subfiling h5dump my_subfiling_file.h5
```

OR

```
# Load the Subfiling VFD by FAPL and dump
# "my_subfiling_file.h5"
h5dump --filedriver=subfiling my_subfiling_file.h5
```

The former method should work for most use cases, but the latter approach may be needed when an HDF5 tool works with more than one file simultaneously and only one of the files is a Subfiling VFD-based HDF5 file. For example, this method would be needed if using `h5repack` to convert a Subfiling file into a different format through the use of another VFD.

## 3. Tips and best practices

- While the default configuration for the Subfiling VFD has been found to be generally performant across different machines, it is still a good idea to initially test a range of Subfiling configurations to see if there is a particular one that generally performs best on the machine of interest.
- When using a parallel file system that stripes data across different storage targets (e.g., Lustre, GPFS, etc.), it is generally a good idea to choose a Subfiling stripe size that is a multiple of the size of the data stripes. For the best performance, it is also generally recommended that the HDF5 application use the `H5Pset_alignment` API routine to set the alignment of objects in the file to match the Subfiling stripe size or to at least be a multiple of the size of the file system's data stripes. However, setting the alignment of objects in the file can potentially waste space and vastly increase the size of the resulting file, so caution may be needed when trying to find a good balance with this suggestion.
- If using the Subfiling VFD with a parallel HDF5 application that pre-creates an HDF5 file on a single MPI rank or some other subset of `MPI_COMM_WORLD`, it is necessary to adjust the ”stripe count” (or number of subfiles) value that the Subfiling VFD uses internally in order to ensure that the proper number of subfiles gets generated when the logical HDF5 file is created. Without setting this value, the VFD will default to creating one subfile per machine node, which will cause it to create only one subfile in the case of pre-creating an HDF5 file on a single MPI rank. Note that as pre-creating an HDF5 file on a subset of `MPI_COMM_WORLD` is expected to be a less common operation when using the Subfiling VFD, an environment variable for modifying this value has not been provided at this time and the VFD will need to be configured on a File Access Property List for this use case. Using the example in section 2.5 as a guide, one can configure the stripe count value by setting the `stripe_count` field in the `H5FD_subfiling_params_t` structure, which is part of the `H5FD_subfiling_config_t` structure that is passed to `H5Pset_fapl_subfiling`. For that particular example, one would set the `subf_config.shared_cfg.stripe_count` field to the desired total number of subfiles.
- While not always leading to an improvement in performance, it can sometimes be useful to enable collective metadata writes and, if the application's access pattern allows for it, collective metadata reads in a parallel HDF5 application through use of the `H5Pset_coll_metadata_write` and `H5Pset_all_coll_metadata_ops` API routines.
- When measuring the performance of the Subfiling feature, be aware that results may sometimes be artificially high due to file system caching. Since all I/O to a particular subfile is directed to a single I/O Concentrator on a machine node, there are many opportunities for data from subfiles to be served from or written to file system caching layers.

# 4. Known problems and limitations

- Collective I/O is currently not supported, except for collective metadata reads/writes, as enabled by `H5Pset_all_coll_metadata_ops` and `H5Pset_coll_metadata_write`
- The VFD should currently not be used with an HDF5 library that has been built with thread-safety enabled. This can cause deadlocks when failures occur due to interactions between the VFD's internal threads and HDF5's global lock.
- The internal I/O task queue doesn't currently have any sophisticated error handling method, so errors are only reported at a course-grained level if an I/O request fails at the lowest level (pread/pwrite/etc.)

## Acknowledgments

This work was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

# A. Reference Manual

The section contains a list of the C APIs related to the Subfiling feature. The Fortran APIs have similar signatures and can be found online in the HDF5 Reference Manual.

## A.1. H5Pset_fapl_subfiling

### Synopsis:

```
herr_t H5Pset_fapl_subfiling(hid_t fapl_id,
                             const H5FD_subfiling_config_t * vfd_config);
```

### Purpose:

Sets the Subfiling Virtual File Driver (VFD) on a File Access Property List (FAPL).

### Description:

`H5Pset_fapl_subfiling` modifies the given File Access Property List (FAPL) to use the Subfiling Virtual File Driver (VFD). Configuration for the File Driver is provided by a pointer to a `H5FD_subfiling_config_t` structure (A.5). MPI must have been initialized within the HDF5 application using `MPI_Init_thread` with the `MPI_THREAD_MULTIPLE` flag prior to calling this routine. Any special MPI Communicator and MPI Info parameters must also have been set on the given FAPL using the `H5P_set_mpi_params` routine prior to calling this routine.

### Parameters:

| | |
|---|---|
| `hid_t fapl_id` | IN: File Access Property List (FAPL) ID |
| `const H5FD_subfiling_config_t * vfd_config` | IN: Pointer to Subfiling VFD configuration structure. May be NULL. |

### Returns:

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

## A.2. H5Pget_fapl_subfiling

### Synopsis:

```
herr_t H5Pget_fapl_subfiling(hid_t fapl_id,
                             H5FD_subfiling_config_t * config_out);
```

### Purpose:

Retrieves the Subfiling Virtual File Driver (VFD) configuration set on the given File Access Property List (FAPL).

### Description:

`H5Pget_fapl_subfiling` returns the Subfiling configuration set on the given File Access Property List (FAPL) through the provided `H5FD_subfiling_config_t` structure pointer. If no such configuration has been set on the given FAPL, a default Subfiling configuration is returned. An HDF5 application may use this functionality to configure the Subfiling VFD by obtaining a default configuration through calling `H5Pget_fapl_subfiling` on a newly-created File Access Property List (FAPL), adjusting the default values and then calling `H5Pset_fapl_subfiling` with the configured `H5FD_subfiling_config_t` structure.

**NOTE:** `H5Pget_fapl_subfiling` only returns the Subfiling configuration parameters as set by calling `H5Pset_fapl_subfiling`. Any configuration parameters set through the use of environment variables will not be reflected in the configuration returned by this routine, so an application may need to check those environment variables to get accurate values for the Subfiling VFD configuration.

### Parameters:

| | |
|---|---|
| `hid_t fapl_id` | IN: File Access Property List (FAPL) ID |
| `H5FD_subfiling_config_t * config_out` | OUT: Pointer to Subfiling VFD configuration structure through which the Subfiling VFD configuration information will be returned. |

### Returns:

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

## A.3. H5Pset_fapl_ioc

### Synopsis:

```
herr_t H5Pset_fapl_ioc(hid_t fapl_id,
                       H5FD_ioc_config_t * vfd_config);
```

### Purpose:

Sets the ”IOC” I/O Concentrator Virtual File Driver (VFD) on a File Access Property List (FAPL).

### Description:

`H5Pset_fapl_ioc` modifies the given File Access Property List (FAPL) to use the ”IOC” I/O Concentrator Virtual File Driver (VFD). Configuration for the File Driver is provided by a pointer to a `H5FD_ioc_config_t` structure (A.8).

### Parameters:

| | |
|---|---|
| `hid_t fapl_id` | IN: File Access Property List (FAPL) ID |
| `H5FD_ioc_config_t * vfd_config` | IN: Pointer to ”IOC” I/O Concentrator VFD configuration structure. May be NULL. |

### Returns:

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

## A.4. H5Pget_fapl_ioc

### Synopsis:

```
herr_t H5Pget_fapl_ioc(hid_t fapl_id,
                       H5FD_ioc_config_t * config_out);
```

### Purpose:

Retrieves the "IOC" I/O Concentrator Virtual File Driver (VFD) configuration set on the given File Access Property List (FAPL).

### Description:

`H5Pget_fapl_ioc` returns the "IOC" I/O Concentrator configuration set on the given File Access Property List (FAPL) through the provided `H5FD_ioc_config_t` structure pointer. If no such configuration has been set on the given FAPL, a default "IOC" I/O Concentrator configuration is returned. An HDF5 application may use this functionality to configure the "IOC" I/O Concentrator VFD by obtaining a default configuration through calling `H5Pget_fapl_ioc` on a newly-created File Access Property List (FAPL), adjusting the default values and then calling `H5Pset_fapl_ioc` with the configured `H5FD_ioc_config_t` structure.

**NOTE:** `H5Pget_fapl_ioc` only returns the "IOC" I/O Concentrator configuration parameters as set by calling `H5Pset_fapl_ioc`. Any configuration parameters set through the use of environment variables will not be reflected in the configuration returned by this routine, so an application may need to check those environment variables to get accurate values for the "IOC" I/O Concentrator VFD configuration.

### Parameters:

| | |
|---|---|
| `hid_t fapl_id` | IN: File Access Property List (FAPL) ID |
| `H5FD_ioc_config_t * config_out` | OUT: Pointer to "IOC" I/O Concentrator VFD configuration structure through which the "IOC" I/O Concentrator VFD configuration information will be returned. |

### Returns:

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

## A.5. H5FD_subfiling_config_t

### Type declaration:

```
typedef struct H5FD_subfiling_config_t {
    uint32_t magic;
    uint32_t version;
    hid_t    ioc_fapl_id;
    hbool_t  require_ioc;
    H5FD_subfiling_params_t shared_cfg;
} H5FD_subfiling_config_t;
```

### Purpose:

Subfiling configuration structure provided to the `H5Pset_fapl_subfiling` API routine

### Description of fields:

`uint32_t magic` - This field is a ”magic” number used for Subfiling configuration verification and must currently always be set to the value represented by the `H5FD_SUBFILING_FAPL_MAGIC` macro.

`uint32_t version` - This field is the version number of the Subfiling configuration structure and should currently always be set to the value represented by the `H5FD_SUBFILING_CURR_FAPL_VERSION` macro.

`hid_t ioc_fapl_id` - This field is the ID of the File Access Property List (FAPL) that the Subfiling VFD will use for accessing subfiles and must be setup with an I/O Concentrator VFD in order for the Subfiling VFD to operate properly. Currently, this FAPL will be setup with the ”IOC” VFD by default and shouldn’t be modified, but future work may allow other I/O Concentrator VFDs to be set on this FAPL instead of the default reference implementation VFD.

`hbool_t require_ioc` - This field determines whether the Subfiling VFD should require an underlying I/O Concentrator VFD and should currently always be set to `true`.

`H5FD_subfiling_params_t shared_cfg` - This field is a structure that contains the various parameters for the Subfiling VFD, such as the data stripe size, number of subfiles to be used, etc. Refer to A.6 for information about its fields.

### A.6. H5FD_subfiling_params_t

#### Type declaration:

```
typedef struct H5FD_subfiling_params_t {
    H5FD_subfiling_ioc_select_t ioc_selection;
    int64_t                     stripe_size;
    int32_t                     stripe_count;
} H5FD_subfiling_params_t;
```

#### Purpose:

Structure containing various parameters for the Subfiling VFD

#### Description of fields:

`H5FD_subfiling_ioc_select_t ioc_selection` - This field is an enumeration that determines how the Subfiling VFD chooses MPI ranks, threads, etc. as I/O Concentrators. Refer to A.7 for information about its defined values.

`int64_t stripe_size` - This field contains the size (in bytes) of data stripes written to subfiles. The default setting for this parameter is a 32MiB stripe size. The value set must either be the value defined by the `H5FD_SUBFILING_DEFAULT_STRIPE_SIZE` macro, or an integer value $> 0$.

`int32_t stripe_count` - This field contains the target number of subfiles to use. It is mostly useful when pre-creating an HDF5 file in a parallel application within some subset of `MPI_COMM_WORLD` (see section 3). The default setting for this parameter is set such that one subfile is used per machine node. The value set must either be the value defined by the `H5FD_SUBFILING_DEFAULT_STRIPE_COUNT` macro, or an integer value $> 0$.

## A.7. H5FD_subfiling_ioc_select_t

### Type declaration:

```
typedef enum {
    SELECT_IOC_ONE_PER_NODE = 0,
    SELECT_IOC_EVERY_NTH_RANK,
    SELECT_IOC_WITH_CONFIG,
    SELECT_IOC_TOTAL,
    ioc_selection_options
} H5FD_subfiling_ioc_select_t;
```

### Purpose:

Enumeration containing the values for different methods of selection I/O Concentrators

### Description of values:

SELECT_IOC_ONE_PER_NODE - The default I/O Concentrator selection method. I/O Concentrators will be selected such that only one is assigned per machine node. If this selection method is specified, the H5FD_SUBFILING_IOC_PER_NODE environment variable can be used to adjust the number of I/O Concentrators assigned per machine node.

SELECT_IOC_EVERY_NTH_RANK - I/O Concentrators will be selected such that an initial I/O Concentrator is selected from the available candidates and then each following I/O Concentrator is selected by applying a stride value N. For example, if I/O Concentrators are MPI ranks, MPI rank 0 will be the first I/O Concentrator and then the next I/O Concentrator will be MPI rank N and so on. The default value for N is 1 and can be adjusted with the H5FD_SUBFILING_IOC_SELECTION_CRITERIA environment variable.

SELECT_IOC_WITH_CONFIG - This method of I/O Concentrator selection is currently unsupported.

SELECT_IOC_TOTAL - I/O Concentrators will be selected such that a total number N of I/O Concentrators are assigned. A fixed stride value is applied when selecting I/O Concentrators so that they are evenly spaced across the set of available candidates. The default value for N is one and can be adjusted with the H5FD_SUBFILING_IOC_SELECTION_CRITERIA environment variable.

ioc_selection_options - This is a sentinel value and should not be used.

## A.8. H5FD_ioc_config_t

### Type declaration:

```
typedef struct H5FD_ioc_config_t {
    uint32_t magic;
    uint32_t version;
    int32_t  thread_pool_size;
} H5FD_ioc_config_t;
```

### Purpose:

”IOC” I/O Concentrator VFD configuration structure provided to the `H5Pset_fapl_ioc` API routine

### Description of fields:

`uint32_t magic` - This field is a ”magic” number used for ”IOC” I/O Concentrator VFD configuration verification and must currently always be set to the value represented by the `H5FD_IOC_FAPL_MAGIC` macro.

`uint32_t version` - This field is the version number of the ”IOC” I/O Concentrator VFD configuration structure and should currently always be set to the value represented by the `H5FD_IOC_CURR_FAPL_VERSION` macro.

`int32_t thread_pool_size` - This field contains the number of worker threads to use per I/O Concentrator. The default setting for this parameter is 4 worker threads per I/O Concentrator. The value set must either be the value defined by the `H5FD_IOC_DEFAULT_THREAD_POOL_SIZE` macro, or an integer value $> 0$.

## A.9. H5FD_SUBFILING_FILENAME_TEMPLATE

### Macro declaration:

```
#define H5FD_SUBFILING_FILENAME_TEMPLATE "%s.subfile_%" PRIu64 "_%0*d_of_%d"
```

### Description:

This is a convenience macro for external tooling that defines the printf-style format template used to generate the filenames for subfiles. Its format specifiers are as follows:

```
%s       -> base filename, e.g. "file.h5"
%PRIu64 -> file inode, e.g. 11273556
%0*d    -> number (starting at 1) signifying the Nth (out of total
           number of subfiles) subfile. Zero-padded according
           to the number of digits in the number of subfiles
           (calculated by log10(num_subfiles) + 1)
%d       -> number of subfiles
```

yielding filenames such as:

```
file.h5.subfile_11273556_01_of_10
file.h5.subfile_11273556_02_of_10
file.h5.subfile_11273556_10_of_10
```

## A.10. H5FD_SUBFILING_CONFIG_FILENAME_TEMPLATE

### Macro declaration:

```
#define H5FD_SUBFILING_CONFIG_FILENAME_TEMPLATE "%s.subfile_%" PRIu64
↪  ".config"
```

### Description:

This is a convenience macro for external tooling that defines the printf-style format template used to generate a Subfiling configuration file. Its format specifiers are as follows:

```
%s       -> base filename, e.g. "file.h5"
%PRIu64 -> file inode, e.g. 11273556
```

yielding a filename such as:

```
file.h5.subfile_11273556.config
```