Virtual File Drivers

What is a File Driver ?

In HDF5, a file driver is a mapping between the HDF5 format address space and storage. By default, HDF5 simply maps the format address space directly onto a single file.

However, users may want the ability to map the format address space onto different types of storage with various types of maps. With HDF5 we provide a small set of pre-defined file drivers, and users can implement their own mappings using the Virtual File Layer APIs. With HDF5-1.12 users can create Virtual Object Layer plugins to HDF5 to store objects with file drivers that they have created.

Detailed information on file drivers can be found under the Technical Notes documentation on the Virtual File Layer.

File Drivers Defined in HDF5

Following are the file drivers that HDF5 provides.

- **H5FD_CORE:** This driver performs I/O directly to memory and can be used to create small temporary files that never exist on permanent storage.
- **H5FD_FAMILY:** This driver partitions a large format address space into smaller chunks (separate storage of a user's choice).
- **HDF5_HDFS**: This driver enables access to an HDF5 file with the Hadoop Distributed File System (HDFS). ***New***
- **H5FD_MPIIO:** This driver is used with Parallel HDF5, and is only pre-defined if the library is compiled with parallel I/O support.
- **H5FD_MULTI:** This driver enables different types of HDF5 data and metadata to be written to separate files. The H5FD_SPLIT driver is an example of what the H5FD_MULTI driver can do.
- **H5FD_ROS3**: This driver enables access to an HDF5 file via the Amazon Simple Storage Service (Amazon S3). ***New***
- **H5FD_SEC2:** This is the default driver which uses Posix file-system functions like `read` and `write` to perform I/O to a single file.
- **H5FD_SPLIT:** This driver splits the meta data and raw data into separate storage of a user's choice.
- **H5FD_STDIO:** This driver uses functions from 'stdio.h' to perform buffered I/O to a single file.

Programming Model for Using a Pre-Defined File Driver

- Create a copy or instance of the File Access property list:
  ```
  fapl = H5Pcreate (H5P_FILE_ACCESS);
  ```

- Initialize the file driver. Each pre-defined file driver has it's own initialization function, whose name is `H5Pset_fapl_` followed by the driver name and which takes a file access property list as the first argument, followed by additional driver-dependent arguments. For example:

  ```
  size_t member_size = 100*1024*1024;  /* 100 MB */
    status = H5Pset_fapl_family (fapl, member_size, H5P_DEFAULT);
  ```

  An alternative to using the driver initialization function is to set the driver directly using `H5Pset_driver`, which is not covered here.

- Call `H5Fcreate`, passing in the identifier of the property list just modified.

  ```
  file_id = H5Fcreate (HDF5FILE, H5F_ACC_TRUNC, H5P_DEFAULT, fapl);
  ```

- Close the File Access Property List:
  ```
  status = H5Pclose (fapl);
  ```

- Perform I/O on the file, if need be. To do so, a Data Access/Transfer property must be copied, modified, and passed in to `H5Dread` or `H5Dwrite`.

  For example, the following sets the MPI-IO driver to use independent access for I/O operations:

  ```
  dxpl = H5Pcreate (H5P_DATA_XFER);
    status = H5Pset_dxpl_mpio (dxpl, H5FD_MPIO_INDEPENDENT);
    status = H5Dread (dataset_id, type, mspace, fspace, buffer, dxpl);
  ```

User Designed File Drivers

These are out of the scope of this tutorial. Refer to the Technical Notes documentation on the Virtual File Layer.

How Does a General Application Open an HDF5 File ?

A general application does not know what drivers were used to create a file. It would have to try different file drivers until it succeeds. An example of a general application is the h5dump tool that we provide with HDF5.