H5P_SET_FILTER

Expand all Collapse all

- Jump to ...
- Summary
- Description
- Example
- Switch language ...
- C
- C++
- FORTRAN
- JAVA

H5P_SET_FILTER

Adds a filter to the filter pipeline

Procedure:

H5P_SET_FILTER (plist_id, filter_id, flags, cd_nelmts, cd_values)

Signature:

```
herr_t H5Pset_filter(hid_t plist_id,
        H5Z_filter_t filter_id,
        unsigned int flags,
        size_t cd_nelmts,
        const unsigned int cd_values[]
    )
```

```
Fortran90 Interface: h5pset_filter_f

SUBROUTINE h5pset_filter_f(prp_id, filter, flags, cd_nelmts, cd_values,  hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Gropu or dataset creation property
                                        ! list identifier
  INTEGER, INTENT(IN) :: filter         ! Filter to be added to the pipeline
  INTEGER, INTENT(IN) :: flags          ! Bit vector specifying certain
                                        ! general properties of the filter
  INTEGER(SIZE_T), INTENT(IN) :: cd_nelmts
                                        ! Number of elements in cd_values
  INTEGER, DIMENSION(*), INTENT(IN) :: cd_values
                                        ! Auxiliary data for the filter
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pset_filter_f
```

Parameters:

hid_t `plist_id`	IN: Dataset or group creation property list identifier
H5Z_filter_t `filter_id`	IN: Filter identifier for the filter to be added to the pipeline
unsigned int `flags`	IN: Bit vector specifying certain general properties of the filter
size_t `cd_nelmts`	IN: Number of elements in `cd_values`
const unsigned int `cd_values[]`	IN: Auxiliary data for the filter

Description:

H5P_SET_FILTER adds the specified `filter_id` and corresponding properties to the end of an output filter pipeline.

plist_id must be either a dataset creation property list or group creation property list identifier. If `plist_id` is a dataset creation property list identifier, the filter is added to the raw data filter pipeline.

If `plist_id` is a group creation property list identifier, the filter is added to the link filter pipeline, which filters the fractal heap used to store most of the link metadata in certain types of groups. The only predefined filters that can be set in a group creation property list are the gzip filter (H5Z_FILTER_DEFLATE) and the Fletcher32 error detection filter (H5Z_FILTER_FLETCHER32).

The array `cd_values` contains `cd_nelmts` integers which are auxiliary data for the filter. The integer values will be stored in the dataset object header as part of the filter information.

The `flags` argument is a bit vector with the following fields specifying certain general properties of the filter:

`H5Z_FLAG_OPTIONAL`	If this bit is set then the filter is optional. If the filter fails (see below) during an H5D_WRITE operation then the filter is just excluded from the pipeline for the chunk for which it failed; the filter will not participate in the pipeline during an H5D_READ of the chunk. This is commonly used for compression filters: if the filter result would be larger than the input, then the compression filter returns failure and the uncompressed data is stored in the file. This flag should not be set for the Fletcher32 checksum filter as it will bypass the checksum filter without reporting checksum errors to an application.

H5Z_FLAG_MANDATORY	If the filter is required, that is, set to mandatory, and the filter fails, the library's behavior depends on whether the chunk cache is in use: • If the chunk cache is enabled, data chunks will be flushed to the file during H5D_CLOSE and the library will return the failure in H5D_CLOSE. When the chunk cache is disabled or not big enough, or the chunk is being evicted from the cache, the failure will happen during H5D_WRITE. In each case, the library will still write to the file all data chunks that were processed by the filter before the failure occurred. For example, assume that an application creates a dataset of four chunks, the chunk cache is enabled and is big enough to hold all four chunks, and the filter fails when it tries to write the fourth chunk. The actual flush of the chunks will happen during H5D_CLOSE, not H5D_WRITE. By the time H5D_CLOSE fails, the first three chunks will have been written to the file. Even though H5D_CLOSE fails, all the resources will be released and the file can be closed properly. If, however, the filter fails on the second chunk, only the first chunk will be written to the file as nothing further can be written once the filter fails.

The `filter_id` parameter specifies the filter to be set. Valid pre-defined filter identifiers are as follows:

H5Z_FILTER_DEFLATE	Data compression filter, employing the gzip algorithm
H5Z_FILTER_SHUFFLE	Data shuffling filter
H5Z_FILTER_FLETCHER32	Error detection filter, employing the Fletcher32 checksum algorithm
H5Z_FILTER_SZIP	Data compression filter, employing the SZIP algorithm
H5Z_FILTER_NBIT	Data compression filter, employing the N-Bit algorithm
H5Z_FILTER_SCALEOFFSET	Data compression filter, employing the scale-offset algorithm

Also see H5P_SET_EDC_CHECK and H5P_SET_FILTER_CALLBACK.

When a non-empty filter pipeline is used with a group creation property list, the group will be created with the new group file format (see Group Implementations in HDF5). The filters will come into play only when dense storage is used (see H5P_SET_LINK_PHASE_CHANGE) and will be applied to the group's fractal heap. The fractal heap will contain most of the the group's link metadata, including link names.

When working with group creation property lists, if you are adding a filter that is not in HDF5's set of predefined filters, i.e., a user-defined or third-party filter, you must first determine that the filter will work for a group. See the discussion of the *set local* and *can apply* callback functions in H5Z_REGISTER.

If multiple filters are set for a property list, they will be applied to each chunk of raw data for datasets or each block of the fractal heap for groups *in the order in which they were set*.

Filters can be applied only to chunked datasets; they cannot be used with other dataset storage methods, such as contiguous, compact, or external datasets.

Dataset elements of variable-length and dataset region reference datatypes are stored in separate structures in the file called heaps. Filters cannot currently be applied to these heaps.

Filter Behavior in HDF5:
Filters can be inserted into the HDF5 pipeline to perform functions such as compression and conversion. As such, they are a very flexible aspect of HDF5; for example, a user-defined filter could provide encryption for an HDF5 dataset.

A filter can be declared as either *required* or *optional*. Required is the default status; optional status must be explicitly declared.

A required filter that fails or is not defined causes an entire output operation to fail; if it was applied when the data was written, such a filter will cause an input operation to fail.

The following table summarizes required filter behavior.

	Required FILTER_X not available	**FILTER_X available**

H5Pset_<*FILTER_X*>	Will fail.	Will succeed.
H5Dwrite with FILTER_X set	Will fail.	Will succeed; FILTER_X will be applied to the data.
H5Dread with FILTER_X set	Will fail.	Will succeed.

An optional filter can be set for an HDF5 dataset even when the filter is not available. Such a filter can then be applied to the dataset when it becomes available on the original system or when the file containing the dataset is processed on a system on which it is available.

A filter can be declared as optional through the use of the H5Z_FLAG_OPTIONAL flag with H5P_SET_FILTER.

Consider a situation where one is creating files that will normally be used only on systems where the optional (and fictional) filter FILTER_Z is routinely available. One can create those files on system A, which lacks FILTER_Z, create chunked datasets in the files with FILTER_Z defined in the dataset creation property list, and even write data to those datasets. The dataset object header will indicate that FILTER_Z has been associated with this dataset. But since system A does not have FILTER_Z, dataset chunks will be written without it being applied.

HDF5 has a mechanism for determining whether chunks are actually written with the filters specified in the object header, so while the filter remains unavailable, system A will be able to read the data. Once the file is moved to system B, where FILTER_Z *is* available, HDF5 will apply FILTER_Z to any data rewritten or new data written in these datasets. Dataset chunks that have been written on system B will then be unreadable on system A; chunks that have not been re-written since being written on system A will remain readable on system A. All chunks will be readable on system B.

The following table summarizes optional filter behavior.

	FILTER_Z not available	**FILTER_Z available with encode and decode**	**FILTER_Z available decode only**
H5Pset_<*FILTER_Z*>	Will succeed.	Will succeed.	Will succeed.
H5Dwrite with FILTER_Z set	Will succeed; FILTER_Z will *not* be applied to the data.	Will succeed; FILTER_Z will be applied to the data.	Will succeed; FILTER_Z will *not* be applied to the data.
H5Dread with FILTER_Z set	Will succeed if FILTER_Z has not actually been applied to data.	Will succeed.	Will succeed.

The above principles apply generally in the use of HDF5 optional filters insofar as HDF5 does as much as possible to complete an operation when an optional filter is unavailable. (The SZIP filter is an exception to this rule; see H5P_SET_SZIP for details.)

Also see Data Flow Pipeline for H5Dread.

Returns:

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

Example:

Coming Soon!

History:

Release	Change
1.6.0	Function introduced in this release.
1.8.5	Function applied to group creation property lists.

--- Last Modified: July 14, 2020 | 04:01 PM