# H5Z_REGISTER

Expand all Collapse all

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

Summary
Description
Example
JAVA
FORTRAN
C++
C

# H5Z_REGISTER

Registers a new filter with the HDF5 library

**Procedure:**

H5Z_REGISTER (filter_class)

**Signature:**

```
herr_t H5Zregister(const H5Z_class_t *filter_class) )
```

**Parameters:**

| | |
|---|---|
| *const H5Z_class_t* `*filter_class` | IN: A pointer to a buffer for the struct containing filter-definition information |

**Description:**

H5Z_REGISTER registers a new filter with the HDF5 library.

Making a new filter available to an application is a two-step process. The first step is to write the three filter callback functions described below: `can_apply`, `set_local`, and `filter`. This call to H5Z_REGISTER, registering the filter with the library, is the second step. The `can_apply` and `set_local` fields can be set to `NULL` if they are not required for the filter being registered.

H5Z_REGISTER accepts a single parameter, a pointer to a buffer for the `filter_class` data structure. That data structure must conform to one of the following definitions:

```
typedef struct H5Z_class1_t {
            H5Z_filter_t id;
            const char  *name;
            H5Z_can_apply_func_t can_apply;
            H5Z_set_local_func_t set_local;
            H5Z_func_t filter;
        } H5Z_class1_t;

        typedef struct H5Z_class2_t {
            int version;
            H5Z_filter_t id;
            unsigned encoder_present;
            unsigned decoder_present;
            const char  *name;
            H5Z_can_apply_func_t can_apply;
            H5Z_set_local_func_t set_local;
            H5Z_func_t filter;
        } H5Z_class2_t;
```

`version` is a library-defined value reporting the version number of the *H5Z_class_t* struct. This currently must be set to `H5Z_CLASS_T_VERS`.

`id` is the identifier for the new filter. This is a user-defined value between `H5Z_FILTER_RESERVED` and `H5Z_FILTER_MAX`. These values are defined in the HDF5 source file `H5Zpublic.h`, but the symbols `H5Z_FILTER_RESERVED` and `H5Z_FILTER_MAX` should always be used instead of the literal values.

`encoder_present` is a library-defined value indicating whether the filter's encoding capability is available to the application.

`decoder_present` is a library-defined value indicating whether the filter's encoding capability is available to the application.

`name` is a descriptive comment used for debugging, may contain a descriptive name for the filter, and may be the null pointer.

`can_apply`, described in detail below, is a user-defined callback function which determines whether the combination of the dataset creation property list values, the datatype, and the dataspace represent a valid combination to apply this filter to.

`set_local`, described in detail below, is a user-defined callback function which sets any parameters that are specific to this dataset, based on the combination of the dataset creation property list values, the datatype, and the dataspace.

`filter`, described in detail below, is a user-defined callback function which performs the action of the filter.

The statistics associated with a filter are not reset by this function; they accumulate over the life of the library.

`H5Z_class_t` is a macro which maps to either `H5Z_class1_t` or `H5Z_class2_t`, depending on the needs of the application. To affect only this macro, `H5Z_class_t_vers` may be defined to either `1` or `2`. Otherwise, it will behave in the same manner as other API compatibility macros. See API Compatibility Macros in HDF5 for more information. `H5Z_class1_t` matches the `H5Z_class_t` structure that is used in the 1.6.x versions of the HDF5 library.

H5Z_REGISTER will automatically detect which structure type has been passed in, regardless of the mapping of the `H5Z_class_t` macro. However, the application must make sure that the fields are filled in according to the correct structure definition if the macro is used to declare the structure.

**The callback functions**
Before H5Z_REGISTER can link a filter into an application, three callback functions must be defined as described in the HDF5 library header file `H5Zpublic.h`.

When a filter is applied to the fractal heap for a group (e.g., when compressing group metadata) and if the *can apply* and *set local* callback functions have been defined for that filter, HDF5 passes the value `-1` for all parameters for those callback functions. This is done to ensure that the filter will not be applied to groups if it relies on these parameters, as they are not applicable to group fractal heaps; to operate on group fractal heaps, a filter must be capable of operating on an opaque block of binary data.

The *can apply* callback function is defined as follows:

*typedef htri_t* (*`H5Z_can_apply_func_t`) (*hid_t* `dcpl_id`, *hid_t* `type_id`, *hid_t* `space_id`)

Before a dataset is created, the *can apply* callbacks for any filters used in the dataset creation property list are called with the dataset's dataset creation property list, `dcpl_id`, the dataset's datatype, `type_id`, and a dataspace describing a chunk, `space_id`, (for chunked dataset storage).

This callback must determine whether the combination of the dataset creation property list settings, the datatype, and the dataspace represent a valid combination to which to apply this filter. For example, an invalid combination may involve the filter not operating correctly on certain datatypes, on certain datatype sizes, or on certain sizes of the chunk dataspace. If this filter is enabled through H5P_SET_FILTER as optional and the *can apply* function returns FALSE, the library will skip the filter in the filter pipeline.

This callback can be the `NULL` pointer, in which case the library will assume that the filter can be applied to a dataset with any combination of

dataset creation property list values, datatypes, and dataspaces.

The *can apply* callback function must return a positive value for a valid combination, zero for an invalid combination, and a negative value for an error.

The *set local* callback function is defined as follows:

*typedef herr_t* (*`H5Z_set_local_func_t`) (*hid_t* `dcpl_id`, *hid_t* `type_id`, *hid_t* `space_id`)

After the *can apply* callbacks are checked for a new dataset, the *set local* callback functions for any filters used in the dataset creation property list are called. These callbacks receive `dcpl_id`, the dataset's private copy of the dataset creation property list passed in to `H5Dcreate` (i.e. not the actual property list passed in to `H5Dcreate`); `type_id`, the datatype identifier passed in to `H5Dcreate`, which is not copied and should not be modified; and `space_id`, a dataspace describing the chunk (for chunked dataset storage), which should also not be modified.

The *set local* callback must set any filter parameters that are specific to this dataset, based on the combination of the dataset creation property list values, the datatype, and the dataspace. For example, some filters perform different actions based on different datatypes, datatype sizes, numbers of dimensions, or dataspace sizes.

The *set local* callback may be the `NULL` pointer, in which case, the library will assume that there are no dataset-specific settings for this filter.

The *set local* callback function must return a non-negative value on success and a negative value for an error.

The *filter operation* callback function, defining the filter's operation on the data, is defined as follows:

*typedef size_t* (*`H5Z_func_t`) (*unsigned int* `flags`, *size_t* `cd_nelmts`, *const unsigned int* `cd_values[]`, *size_t* `nbytes`, *size_t* *`buf_size`, *vo id* **`buf`)

The parameters `flags`, `cd_nelmts`, and `cd_values` are the same as for the function `H5Pset_filter`. The one exception is that an additional flag, `H5Z_FLAG_REVERSE`, is set when the filter is called as part of the input pipeline.

The parameter `*buf` points to the input buffer which has a size of `*buf_size` bytes, `nbytes` of which are valid data.

The filter should perform the transformation in place if possible. If the transformation cannot be done in place, then the filter should allocate a new buffer with `malloc()` and assign it to `*buf`, assigning the allocated size of that buffer to `*buf_size`. The old buffer should be freed by calling `free()`.

If successful, the *filter operation* callback function returns the number of valid bytes of data contained in `*buf`. In the case of failure, the return value is `0` (zero) and all pointer arguments are left unchanged.

> **Programming Note for C++ Developers Using C Functions:**
>
> If a C routine that takes a function pointer as an argument is called from within C++ code, the C routine should be returned from normally.
>
> Examples of this kind of routine include callbacks such as `H5Pset_elink_cb` and `H5Pset_type_conv_cb` and functions such as `H5Tconvert` and `H5Ewalk2`.
>
> Exiting the routine in its normal fashion allows the HDF5 C library to clean up its work properly. In other words, if the C++ application jumps out of the routine back to the C++ “catch” statement, the library is not given the opportunity to close any temporary data structures that were set up when the routine was called. The C++ application should save some state as the routine is started so that any problem that occurs might be diagnosed.

### Returns:

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

### Example:

Coming Soon!

### History:

| Release | Change |
|---|---|
| 1.6.0 | This function was substantially revised in Release 1.6.0 with a new `H5Z_class_t` struct and new *set local* and *can apply* callback functions. |

| | |
|---|---|
| 1.8.0 | The fields `version`, `encoder_present`, and `decoder_present` were added to the `H5Z_class_t` struct in this release. |
| 1.8.3 | `H5Z_class_t` renamed to `H5Z_class2_t`, `H5Z_class1_t` structure introduced for backwards compatibility with release 1.6.x, and `H5Z_class_t` macro introduced in this release. Function modified to accept either structure type. |
| 1.8.5 | Semantics of the *can apply* and *set local* callback functions changed to accommodate the use of filters with group fractal heaps. |
| 1.8.6 | Return type for the *can apply* callback function, `H5Z_can_apply_func_t`, changed to *`htri_t`*. |

--- Last Modified: May 28, 2019 | 12:25 PM