1 Overview

This document is a work in progress (5-30-2024) census of property callbacks and property class callbacks in the HDF5 library, for the purpose of identifying potential issues with the implementation of threadsafety in the H5P module.

MAPL, LCPL, FMPL have no unique callbacks and only reuse encoding/decoding callbacks defined in other modules, so they are not given their own sections here.

1.1 Property Callback Overview

Each property (instance of `H5P_genprop_t`) has up to nine unique callbacks assigned to it when it is registered to a property list class via `H5P__register_real`. Each of these callbacks is optional, although properties which are objects with their own memory allocation typically require unique get/set/copy/create and del/close callbacks to properly implement the copy-by-value semantics expected of properties.

- Create (`H5P_prp_create_func_t`) - Invoked during the creation of a new property list which contains the given property. May change the initial value of the property from the default value given to `H5P__register_real()`.
- Set (`H5P_prp_set_func_t`) - Invoked before a new value is copied into the given property.
- Get (`H5P_prp_get_func_t`) - Invoked before a value is retrieved from the given property.
- Encode (`H5P_prp_encode_func_t`) - Invoked when a property list with the given property is encoded.
- Decode (`H5P_prp_decode_func_t`) - Invoked when a property list with the given property is decoded.
- Delete (`H5P_prp_delete_func_t`) - Invoked before the given property is deleted from a property list.
- Copy (`H5P_prp_copy_func_t`) - Invoked when a property list containing the given property is copied.
- Compare (`H5P_prp_compare_func_t`) - Invoked when a property list containing this property is compared to another property list.
- Close (`H5P_prp_close_func_t`) - Invoked before a property list that contains this property is destroyed.

There is substantial overlap in how these callbacks are implemented internally. Create, set, get, and copy frequently act as wrappers around a shared copy method for the property. This custom copy method is necessary for more complex objects that cannot be entirely copied with a simple `memcpy` from generic H5P routines. For example, consider the property for external file prefixes. The external file prefix's value is a pointer to a pointer to a string, `char**`. The H5P routine which invokes the `get` callback and copies the value only copies the intermediate pointer to the external file prefix (`char*`). The `get` callback is used to copy the underlying string using `strdup()`. The same principle applies to the `set`, `create`, and `copy` callbacks.

For the same reason, delete and close callbacks are often necessary for properties which are more complicated objects that cannot be entirely freed with a single top-level `free` call.

The library's default properties and their callbacks can be broadly separated into the following categories:

- Property with callbacks which use object message callbacks
- Property with callbacks which use module-specific routines
- Property with callbacks which have no significant external dependencies (besides e.g. H5MM, H5VM, and H5E for error reporting)

- Property with only encode/decode callbacks. These callbacks may be unique for this property, or generic type encode/decode functions defined in `H5Pencdec.c`.
- Properties with no defined callbacks. These are typically properties that cannot be encoded because they depend on local context (e.g. type conversion background buffer information).

This document is an attempt to compile every unique property callback, along with comments about its dependencies and implications for threadsafety, if any. Callbacks which are confirmed to be non-threadsafe are marked in red. Threadsafety issues due to H5E are ignored, since this module has a threadsafe implementation planned.

1.2 Property Class Callback Overview

Each property list class (instance of `H5P_genclass_t`) has up to three unique callbacks.

- Create (`H5P_cls_create_func_t`) - Invoked during creation of a property list of this class.
- Copy (`H5P_cls_copy_func_t`) - Invoked during copying of a property list of this class.
- Close (`H5P_cls_close_func_t` Invoked during closing of a property list of this class.

Each callback is paired with a data field in the property list class.

At the time of this document's creation (HDF5 1.14.4.3), the library does not define any property list callbacks on any of its predefined property list classes.

2 DAPL Callbacks

2.1 DAPL Property Callbacks

These callbacks are defined in `H5Pdapl.c`. The properties they belong to are chunk cache parameters, virtual dataset views, virtual dataset file prefixes, and external file prefixes. The only dependencies on other library modules are trivial invocations of the `H5VM` and `H5MM` API, all of which are threadsafe.

2.2 Chunk Cache Parameter Callbacks

- `H5P__encode_chunk_cache_nslots` - Encodes number of chunk slots in the raw data chunk cache into provided buffer.
- `H5P__decode_chunk_cache_nslots` - Decodes number of chunk slots in the raw data chunk cache from provided buffer.
- `H5P__encode_chunk_cache_nbytes` - Encodes size of the raw data chunk cache into provided buffer.
- `H5P__decode_chunk_cache_nbytes` - Decodes size of the raw data chunk cache from provided buffer.

2.3 Virtual dataset view callbacks

- `H5P__dacc_vds_view_enc` - Encodes a virtual dataset view into provided buffer.
- `H5P__dacc_vds_view_dec` - Decodes a virtual dataset view from a provided buffer.

2.4 Virtual dataset file prefix callbacks

- `H5P__dapl_vds_file_pref_set` - Copies a virtual dataset file prefix, replacing provided value.
- `H5P__dapl_vds_file_pref_get` - Copies a virtual dataset file prefix, replacing provided value.
- `H5P__dapl_vds_file_pref_enc` - Encodes virtual dataset file prefix into provided buffer.
- `H5P__dapl_vds_file_pref_dec` - Decodes virtual dataset file prefix from provided buffer.
- `H5P__dapl_vds_file_pref_del` - Frees memory used to store the virtual dataset file prefix.
- `H5P__dapl_vds_file_pref_copy` - Copies a virtual dataset file prefix, replacing provided value.
- `H5P__dapl_vds_file_pref_cmp` - Compares two virtual dataset file prefixes.
- `H5P__dapl_vds_file_pref_close` - Frees memory used to store the virtual dataset file prefix.

2.5 External file prefix callbacks

- `H5P__dapl_efile_pref_set` - Copies an external file prefix. Uses threadsafe H5MM function.
- `H5P__dapl_efile_pref_get` - Copies an external file prefix property. Uses threadsafe H5MM function.
- `H5P__dapl_efile_pref_enc` - Encodes the external file prefix.
- `H5P__dapl_efile_pref_dec` - Decodes the external file prefix.
- `H5P__dapl_efile_pref_del` - Frees memory used to store the external file prefix.
- `H5P__dapl_efile_pref_copy` - Creates a copy of the external file prefix.
- `H5P__dapl_efile_pref_cmp` - Compares two external file prefixes.
- `H5P__dapl_efile_pref_close` - Frees memory used to store an external file prefix.

3 DCPL Property Callbacks

These callbacks are found in H5Pdcpl.c. The properties they belong to are object storage layouts, fill values, external file lists, space allocation time, and object headers. Space allocation time and object header property callbacks use only generic encoding/decoding functions defined in H5Pencdec.c.

The property callbacks for object layouts, fill values, and external file lists invoke object message class callbacks from H5O_MSG_LAYOUT, H5O_MSG_FILL, and H5O_MSG_EFL. Each of these object message classes has several callbacks, but only 'copy' and 'reset' are ever used by these property callbacks.

Due to an indirect dependence on skip lists (and free lists, though those can be disabled) several layout property callbacks are not threadsafe.

3.1 Dataset layout callbacks

The object layout message copy callback H5O__layout_copy depends on H5D due to H5D_chunk_idx_reset and H5D__virtual_copy_layout.H5D__virtual_copy_layout depends on H5SL, H5FL, H5S, and H5I. Even if free lists are disabled at configure time, due to an indirect use of skip lists in H5D_close, this function is not threadsafe, and so H5O__layout_copy and all property callbacks that use it are not threadsafe. It also interacts with the metadata cache via H5AC_cork and H5AC_flush_tagged_metadata.

H5D_chunk_idx_reset uses the reset callback H5D_chunk_reset_func_t from H5D_chunk_ops_t, which has a distinct implementation for B-Trees, v2 B-Trees, extensible arrays, fixed arrays, non-indexed chunks, and single chunk operations. At the time of this, each of these reset callbacks is threadsafe and extremely simple.

The object layout reset callback H5O__layout_reset also depends on H5D_close via H5D__virtual_reset_source_dset, and is not threadsafe for the same reasons.

- H5P__dcrt_layout_set - Copies a layout property. Uses H5O_msg_copy() which depends on non-threadsafe H5SL.
- H5P__dcrt_layout_get - Copies a layout property. Uses H5O_msg_copy() which depends on non-threadsafe H5SL.
- H5P__dcrt_layout_enc - Encodes layout property. Threadsafe dependent on H5S.
- H5P__dcrt_layout_dec - Decodes layout property. Threadsafe dependent on H5S.
- H5P__dcrt_layout_del - Frees memory used to store layout. Uses H5O__layout_reset, which depends on non threadsafe H5SL.
- H5P__dcrt_layout_copy - Copy layout property. Uses H5O_msg_copy() which depends on non-threadsafe H5SL.
- H5P__dcrt_layout_cmp - Compare two layout properties. Threadsafe dependent on H5S.
- H5P__dcrt_layout_close - Frees memory used to store layout. Uses H5O__layout_reset, which depends on non threadsafe H5SL.

3.2 Dataset fill value callbacks

The fill value copy callback H5O__fill_copy uses H5T callbacks to deal with potential type conversion. Reading from and writing to the global type conversion path table H5T_g is threadsafe, since H5T_g is local to the H5T module, which exists under the global mutex. H5CX is used through H5T_convert.

The fill value reset callback H5O__fill_reset is similar to H5O_fill_copy, and is also threadsafe.

- H5P__dcrt_fill_value_set - Copies fill value property for a property list.
- H5P__dcrt_fill_value_get - Copies a fill value property from a property list.
- H5P__dcrt_fill_value_enc - Encodes the fill value.
- H5P__dcrt_fill_value_dec - Decodes the fill value.

3.3 External File List callbacks

The external file list copy and reset callbacks (`H5O__efl_copy`, `H5O__efl_reset`) only depend on H5MM and are both threadsafe.

- `H5P__dcrt_ext_file_list_set` - Copies external file list to a plist.
- `H5P__dcrt_ext_file_list_get` - Copies an external file list from a plist.
- `H5P__dcrt_ext_file_list_enc` - Encode the external file list.
- `H5P__dcrt_ext_file_list_dec` - Decodes the external file list.
- `H5P__dcrt_ext_file_list_del` - Frees memory used to store external file list.
- `H5P__dcrt_ext_file_list_copy` - Copies external file list.
- `H5P__dcrt_ext_file_list_cmp` - Compares two external file lists.
- `H5P__dcrt_ext_file_list_close` - Frees memory used to store the external file list.

4 DXPL Property Callbacks

These property callbacks are found in `H5Pdxpl.c`. The properties they belong to are data transformations and dataset I/O selections.

The data transformation property callbacks act as wrappers around `H5Z` functions. Because `H5Z` doesn't read or write any global structures, these callbacks are threadsafe.

The dataset I/O selection callbacks act as wrappers around `H5S` functions. Since `H5S` has a threadsafe implementation planned, these callbacks are considered threadsafe.

4.1 Data Transformation Property Callbacks

- `H5P__dxfr_xform_set` - Copies a data transform into a property list. Uses threadsafe H5Z call.
- `H5P__dxfr_xform_get` - Copies a data transform from a property list. Uses threadsafe H5Z call.
- `H5P__dxfr_xform_enc` - Encodes a data transform. Uses a threadsafe H5Z call.
- `H5P__dxfr_xform_dec` - Decodes a data transform. Uses a threadsafe H5Z call.
- `H5P__dxfr_xform_del` - Frees memory allocated for a data transform. Uses a threadsafe H5Z call.
- `H5P__dxfr_xform_copy` - Copies data transform string and associated parse tree. Uses a threadsafe H5Z call.
- `H5P__dxfr_xform_cmp` - Compares two data transforms. Uses a threadsafe H5Z call.
- `H5P__dxfr_xform_close` - Frees memory allocated for a data transform. Uses a threadsafe H5Z call.

4.2 Dataset I/O Selection Property Callbacks

- `H5P__dxfr_dset_io_hyp_sel_copy` - Copies dataset I/O selection.
- `H5P__dxfr_dset_io_hyp_sel_cmp` - Compares two dataset I/O selections.
- `H5P__dxfr_dset_io_hyp_sel_close` - Frees resources for a dataset I/O selection.

4.3 Encode/Decode Callbacks

- `H5P__dxfr_bkgr_buf_type_enc` - Encodes the background buffer type.
- `H5P__dxfr_bkgr_buf_type_dec` - Decodes the background buffer type.
- `H5P__dxfr_btree_split_ratio_enc` - Encodes the B-tree split ratio.
- `H5P__dxfr_btree_split_ratio_dec` - Decodes the B-tree split ratio.
- `H5P__dxfr_io_xfer_mode_enc` - Encodes the I/O transfer mode.
- `H5P__dxfr_io_xfer_mode_dec` - Decodes the I/O transfer mode.
- `H5P__dxfr_mpio_collective_opt_enc` - Encodes the MPI-I/O collective optimization.
- `H5P__dxfr_mpio_collective_opt_dec` - Decodes the MPI-I/O collective optimization.
- `H5P__dxfr_mpio_chunk_opt_hard_enc` - Encodes the MPI-I/O chunk optimization.
- `H5P__dxfr_mpio_chunk_opt_hard_dec` - Decodes the MPI-I/O chunk optimization.
- `H5P__dxfr_edc_enc` - Encodes the error detect property.
- `H5P__dxfr_edc_dec` - Decodes the error detect property.
- `H5P__dxfr_selection_io_mode_enc` - Encodes selection I/O mode.

- `H5P__dxfr_selection_io_mode_dec` - Decodes selection I/O mode.
- `H5P__dxfr_modify_write_buf_enc` - Encodes the modify write buffer.
- `H5P__dxfr_modify_write_buf_dec` - Decodes the modify write buffer.

5 FAPL Property Callbacks

These property callbacks are found in `H5Pfapl.c`. The properties they belong to are file driver ID and information, file image info, cache configuration, metadata cache log location, metadata cache image property, VOL connector, MPI communicator, and MPI info.

5.1 File Driver ID and Information Callbacks

The create, set, get, and copy callbacks are all wrappers around the in-place copy operation `H5P__file_driver_copy`. Delete and close are wrappers around `H5P__file_driver_free`, which is a wrapper around `H5FD_free_driver_info`. The comparison callback uses `H5FD`, which in turn depends on `H5I` and `H5P`. Since all of these dependent modules are planned for threadsafe implementation, the comparison function is also threadsafe.

- `H5P__facc_file_driver_create` - Creates a file driver ID and info.
- `H5P__facc_file_driver_set` - Sets file driver ID and info in a plist.
- `H5P__facc_file_driver_get` - Gets file driver ID and info from a plist.
- `H5P__facc_file_driver_del` - Frees memory used to store file driver ID and info.
- `H5P__facc_file_driver_copy` - Copies a file driver ID and info. Dependent on H5FD.
- `H5P__facc_file_driver_cmp` - Compares two sets of file driver ID and info.
- `H5P__facc_file_driver_close` - Frees memory used to store file driver ID and info.

5.2 File Image Info Callbacks

The set, get, and copy operations are all wrappers around `H5P__file_image_info_copy`. This shared copy function uses callbacks defined on the file image info struct (`H5FD_file_image_info_t`): `image_malloc`, `image_memcpy`. The delete and close operations are wrappers around `H5P__file_image_info_free`. This shared free function uses the file image info callback `image_free`.

These file image memory callbacks default to being wrappers around the threadsafe `malloc` and `memcpy`. However, the file image API was designed to allow application programs to use their own file image callbacks which provide the illusion of allocating new memory while actually re-using buffers internally in order to improve performance [1]. If such a set of callbacks is used, then these callbacks deal with a resource shared between the application and the library, and are potentially non-threadsafe. However, managing this is the responsibility of the application providing the custom file image callbacks. Additionally, as long as the locking behavior that normally applies to library objects is upheld for these file image operations, no threadsafety issues should arise even if such optimized callbacks are used.

- `H5P__facc_file_image_info_set` - Copies file image info upon being set for a plist.
- `H5P__facc_file_image_info_get` - Copies file image info upon being retrieved from a plist.
- `H5P__facc_file_image_info_del` - Frees memory used to store file image info.
- `H5P__facc_file_image_info_copy` - Copies file image information.
- `H5P__facc_file_image_info_cmp` - Compares two sets of file image information.
- `H5P__facc_file_image_info_close` - Frees memory used to store file image information.

5.3 Cache Configuration Callbacks

These callbacks have only a trivial dependence on H5VM, and are otherwise entirely self-contained.

- `H5P__facc_cache_config_enc` - Encodes the cache configuration to a plist.
- `H5P__facc_cache_config_dec` - Decodes the cache configuration from a plist.
- `H5P__facc_cache_config_cmp` - Compares two cache configurations.

5.4 Metadata Cache Log Location Callbacks

The metadata cache log location is a string, and these callbacks are mostly wrappers around system string and memory operations. The only dependencies are trivial ones to H5VM and H5MM.

- `H5P__facc_mdc_log_location_enc` - Encodes the metadata cache log location to a plist.
- `H5P__facc_mdc_log_location_dec` - Decodes the metadata cache log location from a plist.
- `H5P__facc_mdc_log_location_del` - Frees memory used to store a metadata cache log location.
- `H5P__facc_mdc_log_location_copy` - Copies the metadata cache log location.
- `H5P__facc_mdc_log_location_cmp` - Compares two metadata cache log locations.
- `H5P__facc_mdc_log_location_close` - Frees memory used to store a metadata cache log location.

5.5 Cache Image Configuration Callbacks

These callbacks use no functions from other modules.

- `H5P__facc_cache_image_config_cmp` - Compares two cache image configurations.
- `H5P__facc_cache_image_config_enc` - Encodes a cache image configration to a plist.
- `H5P__facc_cache_image_config_dec` - Decodes a cache image configuration.

5.6 VOL Connector Callbacks

The create, set, get, and copy callbacks are wrappers around `H5VL_conn_copy`. The delete and close callbacks are wrappers around `H5VL_conn_free`. The compare callback uses H5I and H5VL routines. Because these modules are planned for threadsafe implementations, these callbacks are considered threadsafe.

- `H5P__facc_vol_create` - Creates a VOL connector ID and information property in a plist.
- `H5P__facc_vol_set` - Sets VOL connector ID and info in a plist.
- `H5P__facc_vol_get` - Gets VOL connector ID and info from a plist.
- `H5P__facc_vol_del` - Frees memory used to store VOL connector ID and info from a plist.
- `H5P__facc_vol_copy` - Copies VOL connector ID and info.
- `H5P__facc_vol_cmp` - Compares two sets of VOL connector ID and info.
- `H5P__facc_vol_close` - Frees memory used to store VOL connector ID and info.

5.7 MPI Communicator Callbacks

These callbacks act as wrappers around `H5mpi.c` functions, which in turn make use of MPI routines. Get, set, and copy callbacks all use `H5_mpi_comm_dup`, delete and close callbacks both use `H5_mpi_comm_free`.

All MPI routines used are either guaranteed threadsafe, or threadsafe as long as the MPI object they modify is not being operated on by another thread - a fact which should be guaranteed true by the global lock and/or the user application.

- `H5P__facc_mpi_comm_set` - Copies an MPI communicator for a plist
- `H5P__facc_mpi_comm_get` Copies an MPI communicator from a plist
- `H5P__facc_mpi_comm_del` - Frees memory used to store an MPI communicator
- `H5P__facc_mpi_comm_copy` - Copies an MPI communicator.
- `H5P__facc_mpi_comm_cmp`- Compares two MPI communicators.
- `H5P__facc_mpi_comm_close` - Frees memory used to store an MPI communicator.

5.8 MPI Info Callbacks

Like the MPI Communicator callbacks, these callbacks are wrappers around `H5mpi.c` functions, which are in turn wrappers around MPI routines. Just as for those callbacks, all MPI routines used are threadsafe or threadsafe as long as the target MPI object is not externally modified during operation.

The set, get, and copy callbacks are wrappers around `H5_mpi_info_dup`. The delete and close callbacks are wrappers around `H5_mpi_info_free`.

- `H5P__facc_mpi_info_set` - Sets MPI info object in a plist
- `H5P__facc_mpi_info_get` - Gets MPI info object from a plist
- `H5P__facc_mpi_info_del` - Frees memory used to store an MPI info object
- `H5P__facc_mpi_info_copy` - Copies an MPI info object
- `H5P__facc_mpi_info_cmp` - Compares two MPI info objects
- `H5P__facc_mpi_info_close` - Frees memory used to store an MPI info object

5.9 Encode/Decode Callbacks

None of these callbacks use routines from any other module.

- `H5P__facc_fclose_degree_enc` - Encodes file close degree
- `H5P__facc_fclose_degree_dec` - Decodes file close degree
- `H5P__facc_multi_type_enc` - Encodes multi VFD memory type
- `H5P__facc_multi_type_dec` - Decodes multi VFD memory type
- `H5P__facc_libver_type_enc` - Encodes a library version bound
- `H5P__facc_libver_type_dec` - Decodes a library version bound
- `H5P__encode_coll_md_read_flag_t` - Encodes the collective metadata read flag
- `H5P__decode_coll_md_read_flag_t` - Decodes the collective metadata read flag

6 FCPL Property Callbacks

These property callbacks are found in `H5Pfcpl.c`. This module contains only custom encode/decode callbacks. None of these callbacks use any external routines.

6.1 Encode/Decode Callbacks

- `H5P__fcrt_btree_rank_enc` - Encodes the minimum rank of a btree internal node
- `H5P__fcrt_btree_rank_dec` - Decodes the minimum rank of a btree internal node
- `H5P__fcrt_shmsg_index_types_enc` - Encodes the shared message index types
- `H5P__fcrt_shmsg_index_types_dec` - Decodes the shared message index types
- `H5P__fcrt_shmsg_index_minsize_enc` - Encodes the shared message index minimum size
- `H5P__fcrt_shmsg_index_minsize_dec` - Decodes the shared message index minimum size
- `H5P__fcrt_fspace_strategy_enc` - Encodes the free-space strategy
- `H5P__fcrt_fspace_strategy_dec` - Decodes the free-space strategy

7 GCPL Property Callbacks

These property callbacks are found in `H5Pgcpl.c`. This module contains only custom encode/decode callbacks. None of these callbacks use any external routines.

- `H5P__gcrt_group_info_enc` - Encodes group info
- `H5P__gcrt_group_info_dec` - Decodes group info
- `H5P__gcrt_link_info_enc` - Encodes link info
- `H5P__gcrt_link_info_dec` - Decodes link info

8 LAPL Property Callbacks

These property callbacks are found in `H5Plapl.c`. The properties they belong to are external link prefixes, and external link FAPLs.

8.1 External Link Prefix Callbacks

These callbacks have only trivial dependencies on H5VM and H5MM routines.

- `H5P__lacc_elink_pref_set` - Sets an external link prefix in a plist
- `H5P__lacc_elink_pref_get` - Gets an external link prefix from a plist
- `H5P__lacc_elink_pref_enc` - Encodes the external link prefix
- `H5P__lacc_elink_pref_dec` - Decodes the external link prefix
- `H5P__lacc_elink_pref_del` - Frees memory used to store the external link prefix
- `H5P__lacc_elink_pref_copy` - Creates a copy of the external link prefix
- `H5P__lacc_elink_pref_cmp` - Compares two external link prefixes
- `H5P__lacc_elink_pref_close` - Frees memory used to store the external link prefix

8.2 External Link FAPL Callbacks

These callbacks depend on routines from H5P, H5I, and trivial functions from H5VM. Because H5P and H5I have threadsafe implementations planned, these callbacks are considered threadsafe.

An entire FAPL is stored as a single property for external links. Callbacks which usually copy their property internally (get, set, copy) only do so if the FAPL is non-default, otherwise the callback is a noop. The encode/decode callbacks serialize the FAPL to/from a single indicator byte if it is default, or a single indicator byte following by the FAPL itself if it is non-default.

- `H5P__lacc_elink_fapl_set` - Sets an external link FAPL
- `H5P__lacc_elink_fapl_get` - Gets an external link FAPL
- `H5P__lacc_elink_fapl_enc` - Encodes an external link FAPL to a plist.
- `H5P__lacc_elink_fapl_dec` - Decodes an external link FAPL from a plist.
- `H5P__lacc_elink_fapl_del` - Frees memory used to store an external link FAPL. Uses reference counting managed through H5I.
- `H5P__lacc_elink_fapl_copy` - Copies an external link FAPL.
- `H5P__lacc_elink_fapl_cmp` - Compares two external link FAPLs.
- `H5P__lacc_elink_fapl_close` - Frees memory used to store an external link FAPL. Uses reference counting managed through H5I.

9 OCPL Property Callbacks

These callbacks are defined in `H5Pocpl.c`. The only property with callbacks defined in this module is the filter pipeline for object creation.

9.1 Filter Pipeline Property Callbacks

Trivial dependence on H5VM, H5MM. Decode uses a threadsafe H5Z callback.

The set, get, and copy callbacks invoke the object message copy callback for filter pipelines, which is `H5O__pline_copy`. Besides a dependence on H5FL, this callback is threadsafe, and so the callbacks which use it are threadsafe.

The delete and close callbacks invoke the object message reset callback for filter pipelines - `H5O__pline_reset`. This callback is threadsafe, so the property callbacks which use it are threadsafe.

- `H5P__ocrt_pipeline_set` - Sets a filter pipeline in a plist
- `H5P__ocrt_pipeline_get` - Retrieves a filter pipeline from a plist
- `H5P__ocrt_pipeline_enc` - Encodes the filter pipeline
- `H5P__ocrt_pipeline_dec` - Decodes the filter pipeline
- `H5P__ocrt_pipeline_del` - Frees memory used to store a filter pipeline.
- `H5P__ocrt_pipeline_copy` - Copies a filter pipeline.
- `H5P__ocrt_pipeline_cmp` - Compares two filter pipelines.
- `H5P__ocrt_pipeline_close` - Frees memory used to store a filter pipeline.

10 OCPYPL Property Callbacks

These callbacks are found in `H5Pocpypl.c`. The only property these callbacks belong to is the merge committed datatype list.

10.1 Merge Committed Datatype List Callbacks

The get, set, and copy callbacks are wrappers around `H5P__copy_merge_comm_dt_list()`. The delete and close callbacks are wrappers around `H5P__free_merge_comm_dtype_list()`. Besides a dependence on H5FL, these callbacks are threadsafe.

- `H5P__ocpy_merge_comm_dt_list_set` - Sets a merge committed datatype list in a plist. This callback is a wrapper around `H5P__copy_merge_comm_dt_list()`.
- `H5P__ocpy_merge_comm_dt_list_get` - Gets a merge committed datatype list from a plist. This callback is a wrapper around `H5P__copy_merge_comm_dt_list()`.
- `H5P__ocpy_merge_comm_dt_list_enc` - Encodes a merge committed datatype list.
- `H5P__ocpy_merge_comm_dt_list_dec` - Decodes a merge committed datatype list.
- `H5P__ocpy_merge_comm_dt_list_del` - Frees memory used to store a merge committed datatype list. This callback is a wrapper around `H5P__free_merge_comm_dtype_list()`.
- `H5P__ocpy_merge_comm_dt_list_copy` - Copies a merge committed datatype list. This callback is a wrapper around `H5P__copy_merge_comm_dt_list()`.
- `H5P__ocpy_merge_comm_dt_list_cmp` - Compares two merge committed datatype lists.
- `H5P__ocpy_merge_comm_dt_list_close` - Frees memory used to store a merge committed datatype list. This callback is a wrapper around `H5P__free_merge_comm_dtype_list()`.

11 H5Pstrcpl Property Callbacks

These callbacks are found in `H5strcpl.c`. This module contains only encode and decode callbacks for character set encodings, which depend on no routines from other modules.

11.1 Character Set Encoding Callbacks

- `H5P__strcrt_char_encoding_enc` - Encodes a character set.
- `H5P__strcrt_char_encoding_dec` - Decodes a character set.

12 Encoding/Decoding Callbacks

These callbacks are defined in `H5Pencdec.c`. They contain only trivial dependencies on H5VM.

- `H5P__encode_size_t` - Encodes a size_t value into a provided buffer.
- `H5P__decode_size_t` - Decodes a size_t value from a provided buffer.
- `H5P__encode_hsize_t` - Encodes an hsize_t value into a provided buffer.
- `H5P__decode_hsize_t` - Decodes an hsize_t value from a provided buffer.
- `H5P__encode_unsigned` - Encodes an unsigned value into a provided buffer.
- `H5P__decode_unsigned` - Decodes an unsigned value from a provided buffer.
- `H5P__encode_uint8_t` - Encodes a uint8_t value into a provided buffer.
- `H5P__decode_uint8_t` - Decodes a uint8_t value from a provided buffer.
- `H5P__encode_bool` - Encodes a boolean value into a provided buffer.
- `H5P__decode_bool` - Decodes a boolean value from a provided buffer.
- `H5P__encode_double` - Encodes a double value to provided buffer.
- `H5P__decode_double` - Decodes a double value from a provided buffer.
- `H5P__encode_uint64_t` - Encode a uint64_t value into a provided buffer.
- `H5P__decode_uint64_t` - Decodes a uint64_t value from a provided buffer.