# Usage of the Chunk Layout Outside of the Layout Interface

Neil Fortner

7/2/24



## Introduction

The purpose of this document is to investigate the level of entanglement between the chunk layout and the higher levels of the dataset code in HDF5, to help assess the difficulty of implementing a new layout class with a new and (initially) separate chunk cache. Thus we are not concerned with interactions with the dataset code that flow through the existing layout interface, H5D_layout_ops_t, only with interactions that bypass that interface.

In many cases in this document, we give suggestions for how to improve the interface to make the upper levels of the dataset code more agnostic to the specific type of layout being used. While this would improve the maintainability of the HDF5 library, it is not necessary to do so in order to implement a shared chunk cache or sparse storage.

## Inclusion in H5D_shared_t

The main data struct for the chunk cache, H5D_rdcc_t, is embedded within the main dataset object struct H5D_shared_t. This gives all code direct access to the root of the chunk cache information, though the struct describing each entry (chunk) in the chunk cache, H5D_rdcc_ent_t, is declared in H5Dchunk.c and therefore individual entries cannot be inspected or manipulated outside of the chunk code. This is used in several places:

- Test code in H5Dtest.c
- Dataset close code in H5D_close() and H5D_mult_refresh_close(), where it is used to close the skip lists and single chunk dataspace. I am not sure why this code isn't in H5D__chunk_dest().
- In H5D__set_extent(), where it is used to adjust the number of bits used to encode the chunk scaled coordinates. We may want to consider adding a set_extent() layout callback to mitigate this entanglement.
- In H5D_get_access_plist(), where it is used to retrieve chunk cache properties from the dataset in order to place them in a property list.

## Inclusion in H5O_layout_t

In addition to the embed of H5D_rdcc_t in H5D_shared_t, chunk specific information is also embedded in the layout message struct, H5O_layout_t, which is also embedded in H5D_shared_t. The

chunk specific information is present in 2 unions in H5O_layout_t, u.chunk and storage.u.chunk. These are used in many places outside of the chunk code:

# Functions Declared in H5Dpkg.h

## H5D__chunk_cacheable

This function is currently only used in H5Dchunk.c and its declaration could be moved there.

## H5D__chunk_create

This function creates an (empty) chunk index for a dataset. It is called by H5D__alloc_storage() when creating a new dataset and allocation time is early, when writing to one whose chunk index hasn't been created, or when extending a dataset with early allocation. To clean up the interface we could add a alloc_storage layout callback and move most or all of the logic in H5D__alloc_storage() to layout specific callbacks.

## H5D__chunk_set_info

This function is called by H5D__set_extent() after updating the dataset's extent, and is used to recalculate cache indices and rebuild the hash table. This function, and others called by H5D__set_extent(), could probably be replaced by a set_extent_notify layout callback.

## H5D__chunk_is_space_alloc

This function is called by H5O__dset_bh_info() to decide whether it is necessary to call H5D__chunk_bh_info(), and by H5O__layout_copy_file() to decide whether to call H5D__chunk_copy(). In the first case, we could create a "bh_info" layout callback and leave it to that callback to decide how to handle it regardless of the allocated state. In the second case, we could add a "copy" layout callback and similarly leave it up to that callback to determine the exact behaviour. It is also worth noting that H5D__chunk_is_space_alloc() is part of the layout interface and could be called through the layout callback, though in these cases the code is written specifically for chunked datasets (and there are other branches for other layouts), and we want to eliminate these entire blocks if possible.

## H5D__chunk_is_data_cached

This function is used by H5O__layout_copy_file() to determine if the data chunks should be copied using H5D__chunk_copy().

## H5D__chunk_lookup

This is used by the parallel compression code to look up chunk addresses.

## H5D__chunk_allocated

This is used by H5D__get_storage_size() to determine the allocated storage size of the raw chunk data. This could be replaced with a get_storage_size layout callback.

## H5D__chunk_allocate

This function is called by H5D__init_storage() to allocate space for all raw data in the dataset, and potentially write the fill value. H5D__init_storage() is called by H5D__alloc_storage() for any chunked dataset with early allocation. This could probably be replaced by an init_storage layout callback. The logic in H5D__alloc_storage() could likely be greatly simplified by moving the logic handling decisions on allocation time to the layout callbacks.

## H5D__chunk_file_alloc

This function is called by the parallel compression code to allocate or reallocate filtered chunks after they have been written to.

## H5D__chunk_mem_alloc/free

These functions are simple custom memory allocation/free functions passed to H5T__fill_init() by the parallel compression code.

## H5D__chunk_mem_xfree/realloc

These functions are currently only used in H5Dchunk.c and their declarations could be moved there.

## H5D__chunk_update_old_edge_chunks

This function is called by H5D__set_extent() to update the status of chunks that no longer extend beyond the extent of the dataset. This function could likely be moved out of the generic dataset code and into the chunk specific code by adding a set_extent_notify callback, as suggested previously.

## H5D__chunk_is_partial_edge_chunk

This function is used by the parallel compression code to decide if it should skip filters due to a chunk being a partial bound edge chunk (partially outside of the extent).

## H5D__chunk_prune_by_extent

This function is called by H5D__set_extent() to remove chunks that are no longer in the dataset's extent. This function could likely be moved out of the generic dataset code and into the chunk specific code by adding a set_extent_notify callback, as suggested previously.

## H5D__chunk_set_sizes

This function is called by H5D__layout_oh_read() to finish setting up the chunk layout data (embedded in the dataset struct) after copying the layout to the internal DCPL. The only reason this is not simply included in H5D__chunk_init() appears to be the need to have a different number of dimensions for the chunks in the dataset struct (and file) vs the property list. The former has an extra dimension for the datatype size. In my opinion, this distinction should be eliminated, though the code must still be able to account for the older file format. We could add a set_plist layout callback to allow the chunk code to perform this twiddling while setting the property list, and other layout types would simply set this to NULL and H5D__layout_oh_read() would call H5P_set() as normal.

## H5D__chunk_addrmap

This function is called by H5D__obtain_mpio_mode(), part of the multi chunk algorithm for parallel compression, to obtain a map (implemented as a flat array) from chunk indices to addresses.

## H5D__chunk_update_cache

This function is called by H5D__set_extent() to update chunk indices after changing the dataset extent. See the earlier note about adding a set_extent() layout callback.

## H5D__chunk_copy

This function is called by H5O__layout_copy_file(), which is called by H5Ocopy(), to copy the dataset's chunk index, raw data, and cached data to the new dataset. This could probably be turned into a copy or copy_file layout callback. The call to H5D__chunk_is_data_cached() (mentioned earlier) could likely be integrated into this function as well, instead of being called in the upper layers of the dataset code.

## H5D__chunk_bh_info

This function is called by H5O__dset_bh_info() to calculate the amount of space used in the file for the chunk index. This could be replaced by a bh_info or bh_size layout callback.

## H5D__chunk_dump_index

This function is called by H5Ddebug(), used by h5debug, to print information about the chunk index and the chunks themselves as they exist on disk. This could be replaced by a layout callback, but this is probably not necessary as this functionality is secondary to the main library.

## H5D__chunk_delete

This function is called by H5O__layout_delete() when a dataset is deleted. H5O__layout_delete() contains only a simple switch statement that only calls delete functions for different layouts, so it would be trivial to convert it to a delete layout callback.

## H5D__chunk_get_offset_copy

This function is called by H5VL__native_dataset_optional() in the direct chunk read and write blocks, and it serves to create a local copy of the chunk offset buffer (so it can be modified), and perform bounds checking. The direct chunk read and write functionality is currently very specific to the existing chunk layout so this can probably stay as is for now.

## H5D__chunk_direct_write/read

These functions are also called by H5VL__native_dataset_optional() in the direct chunk read and write blocks, and they perform the actual direct chunk I/O. The direct chunk read and write functionality is currently very specific to the existing chunk layout so this can probably stay as is for now.

## H5D__chunk_stats

Called by H5D_close() when H5D_CHUNK_DEBUG is defined, in order to print chunk cache statistics.

## H5D__chunk_format_convert

This function is called by H5D__format_convert() to downgrade the file format version of a chunked dataset. This could possibly be replaced by a layout callback, though there is no need to implement it for a new layout type since these is initially only one file format for that layout. This can be left alone for now.

# Other Layout-Type Specific Code