RFC: Finalizing Sparse Data Programming Model and APIs

Elena Pourmal elena.pourmal@lifeboat.llc

This document provides recommendations regarding the Sparse Data Programming Model and APIs. Detailed discussions can be found in the "RFC: Programming Model to Support Sparse Data in HDF5" (RFC Lifeboat 2023-03-03.v1-21), available in the GitHub repo https://github.com/LifeboatLLC/SparseHDF5/tree/main/design_docs. This document does not replace the aforementioned RFC but is intended to support its finalization. It should be considered a supplementary resource that highlights issues raised during multiple discussions. Recommendations made following the review of this document will be reflected in version 22 of the RFC.

1 Introduction

Our discussions have revealed several issues in the proposed programming model and public APIs that need to be addressed before releasing version 0 of HDF5 Sparse Data. The following sections outline proposals for the programming data model and the public APIs.

2 Programming Model

There is general agreement that we should implement structured chunk storage for handling sparse and variable-length data in the native HDF5 format. The goal is to define a programming model that introduces a minimal set of new APIs while enabling users to configure storage for sparse fixed-size data, sparse variable-size data, and dense variable-size data—with support for compression in each section—without requiring knowledge of the underlying structure of the chunk.

The following sections revisit the initial proposal for the programming model, highlight its deficiencies, and provide recommendations for moving forward.

2.1 Two ways of setting structured chunk storage

We explored two approaches for configuring storage of sparse data. Initially, we considered using the existing APIs `H5Pset_layout` and `H5Pset_chunk`, and introduced a new storage layout type, `H5D_STRUCT_CHUNK`, to the `H5D_layout_t` enumeration (see `H5Dpublic.h`). In this approach, `H5Pset_layout` was used with the `H5D_STRUCT_CHUNK` parameter to specify structured chunk storage, while `H5Pset_chunk` defined logical chunk sizes, similar to the dense data case. However, this method required a new API to explicitly indicate that the data is sparse.
To address this limitation, two alternative proposals were considered. Both require new APIs and come with their own drawbacks, as outlined below:

1. Introduce the `H5Pset_density` function on dataset creation property list to indicate storage of sparse data. There are two advantages of using this function:
    a. It works with the existing `H5Pset_layout` and `H5Pset_chunk` functions.
    b. It works with a VOL connector that has a special mechanism for storing sparse data.

    This approach presents two major problems. First, setting up sparse storage for variable-length data requires yet another new function. Second, to use the new functions for applying a filter pipeline to a section, we also need a function to retrieve the number of sections and their corresponding indices within a structured chunk.
2. Introduce the `H5Pset_struct_chunk` function and use a parameter `flag` to specify sparse chunk storage type or variable-length storage. We rely on a datatype provided in the `H5Dcreate` call to set up storage for sparse variable-length data. To use the new functions for applying a filter pipeline to a section, we also need a function to retrieve the number of sections and their corresponding indices within a structured chunk.

Two major criticisms raised during discussions with the HDF5 developers were the cumbersome process of describing the type of data stored in a structured chunk, and the lack of a method to programmatically determine the composition of a structured chunk.

2.2 Proposed programming model

In this section, we propose modifications to the current programming model and APIs to address the issues discussed above. A summary of the proposed changes is provided below, with detailed function descriptions in the following subsections.

We propose modifying `H5Pset_struct_chunk` to accept a bitfield parameter (`flag`) that indicates the type of data to be stored. This approach is easily extensible, allowing support for new data types that may require additional sections—for example, sections for storing nonhomogeneous data.

A new API, `H5Pget_struct_chunk_sections`, is introduced to retrieve the number of sections and their associated "names" from the dataset creation property list modified by `H5Pset_struct_chunk`. Additionally, `H5Dcreate` should be updated to validate that the datatype matches the data type specified in the dataset creation property list, reporting an error if there is a mismatch.

The “names” can be used instead of the current sections numbers as inputs for APIs to set filter pipeline on a specific section of the structured chunk.

To mitigate the requirement of *a priori* sections usage, we can provide a simple function that returns corresponding “names” for `enum` values defined in this new data `H5_section_type_t` structure as shown in section 2.2.2.

2.2.1 H5Pset_struct_chunk

Sets structured chunked storage. The storage is used for sparse data of any datatype, and for dense data with variable-length datatype. Updates to the signature are shown in green.

2.2.1.1 Signature

```
herr_t H5Pset_struct_chunk (hid_t plist_id, int ndims, const hsize dim[], uint8_t flag, size_t data_size, void *data)
```

2.2.1.2 Parameters

`plist_id`	IN/OUT: Dataset creation property identifier
`ndims`	IN: The number of chunk dimensions
`dim`	IN: An array defining the size, in dataset elements, of each chunk
`flag`	IN: Data characteristics flag. Possible values are H5D_SPARSE_DATA, H5D_VL_DATA or their combination using the bit-wise OR operator.
`data_size`	IN: Size of user data buffer `data; reserved for future use; pass NULL`

data	IN: User data; reserved for future usage. Pass NULL for now

2.2.1.3 Description

H5Pset_struct_chunk() sets structured chunk storage layout, chunk sizes and a type of structured chunk storage for a dataset. This function is only valid for dataset creation property lists.

The ndims parameter must be the same size as the rank of the dataset.

The values of the dim array define the size of the chunks. The unit of measure for dim values is in dataset elements.

As a side-effect of this function, the creation property is modified to H5D_STRUCT_CHUNK storage layout, if it was previously set using H5Pset_layout function with any other storage layout type.

The value of the flag parameter can be H5D_SPARSE_DATA to store sparse data of any datatype, H5D_VL_DATA[1] to store dense data of variable-length datatype, and H5D_SPARSE_DATA I H5D_VL_DATA to store sparse data of variable-length datatype.

2.2.1.4 Returns

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

2.2.2 H5Pget_struct_chunk_sections

Gets information about sections of the structured chunked that will be used to store specified type of data.

2.2.2.1 Signature

herr_t H5Pget_struct_chunk_sections (*hid_t* plist_id, *int* *num, *H5_section_type_t* buf[])

2.2.2.2 Parameters

plist_id	IN/OUT: Dataset creation property identifier
num	OUT: The number of the structured chunk sections
buf	OUT: And array of size num to hold sections identifiers; can be NULL to get the number of sections first to allocate the buffer of the appropriate size.

[1] The names of the flags are subject to change.

2.2.2.3 Description

H5Pget_struct_chunk_sections gets the number of sections and their identifiers. This function is only valid for dataset creation property lists. The buff array has type *H5_section_t that is* enum integers as show below.

```
typedef enum H5_section_type_t {
     H5_SECTION_UNKNOWN = -1,
     H5_SECTION_SELECTION,
     H5_SECTION_FIXED,
     H5_SECTION_VL,
     H5_SECTION_NUM /* Should be the last item */
} H5_section_type_t;
```

2.2.2.4 Returns

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

2.2.3 H5Pset_filter2

The function adds a filter to the filter pipeline for a specified section or to all sections of a structured chunk.

2.2.3.1 Signature

```
herr_t H5Pset_filter2 (hid_t plist_id, H5_section_type_t sec_type,
                       H5Z_filter_t filter,
                       uint64_t flags,
                       size_t buf_size,
                       const void *buf)
```

2.2.3.2 Parameters

plist_id	IN: dataset creation property list identifier
sec_type	IN: One of the array values returned by the H5Pget_struct_chunk_sections function. When applied to dense datasets, the flag is ignored.
filter	IN: Filter identifier for the filter to be added to the pipeline
flags	IN: Bit vector specifying certain general properties of the filter
buf_size	IN: Size in bytes of buf buffer
buf	IN: Buffer with an auxiliary data for the filter

2.2.3.3 Description

`H5Pset_filter2` adds the specified `filter` identifier and corresponding properties to the end of an output filter pipeline for the section of the structured chunk specified by the `sec_type` parameter. The parameter is one of the values returned by the `H5Pget_struct_chunk_sections` function and corresponds to a section that holds specific data in the structured chunk, e.g., encoded selection for sparse data, or fixed-length data that describes offsets into the section of the chunk that holds variable-length data.

`plist_id` is a dataset creation property identifier. The buffer `buf` of size `buf_size` contains auxiliary data for the filter. The values will be stored in the Structured Chunk Filter Pipeline message in the dataset object header as part of the filter information.

The `flags` argument is a bit vector with the fields specifying certain general properties of the filter as documented in the description of the current `H5Pset_filter` function.

Please note that `H5Pset_edc_check` function will be applicable to the structured chunk storage with enabled filtering but may not be available with the first release of the sparse feature.

2.2.3.4 Returns

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

2.3 Changes to direct chunk functions

The first change will be to the `H5D_struct_chunk_info_t structure` as indicated below.

```
typedef struct H5D_struct_chunk_info_t {
    H5_section_type_t sections[]  ;  /* Array of section types obtained by
                                        H5Pget_struct_chunk_sections          */
    uint8_t   num_sections;          /* Number of sections                    */
    uint16_t  filter_mask[];         /* Array of num_sections size.           */
                                     /* Contains filter mask for each section. */
                                     /* It is 0 when no filters are applied.  */
    size_t    section_size[];        /* Array of num_sections size            */
                                     /* Contains the size of each section.    */
    size_t    section_orig_size[];   /* Array of num_sections size            */
                                     /* Contains original size of each section */
} H5D_struct_chunk_info_t
```

We replace type of the structured chunk with information about all sections.

The second change is to the parameters list in the callback function for iteration routine: we add the size of the user data buffer as shown below to follow security recommendations from The HDF Grou developers. Please notice that sizes of the buffers in all other direct chunk APIs can be calculated from the provided sizes of each section.

```
typedef int(*H5D_struct_chunk_iter_op_t)(const hsize_t *offset,
                                         H5D_struct_chunk_info_t *chunk_info,
```

```
                                        haddr_t *addr,
                                        hsize_t *chunk_size, size_t op_data_size,
                                        void *op_data)
```

2.3.1.1 Parameters

offset	IN: Logical position of the chunk's first element in the array
chunk_info	IN: Information about the structured chunk
addr	IN: Chunk address in the file
chunk_size	IN: Chunk size in bytes; 0 if the chunk does not exist
op_data_size	IN: Size of the op_data buffer.
op_data	IN: User-defined pointer to data required by the callback function

2.3.1.2 Returns

- Zero (H5_ITER_CONT) causes the iterator to continue, returning zero when all elements have been processed.
- A positive value (H5_ITER_STOP) causes the iterator to immediately return that value, indicating short-circuit success.
- A negative (H5_ITER_ERROR) causes the iterator to immediately return that value, indicating failure.

Signatures of the proposed H5D*_struct_chunk_* direct chunk functions are not changed.

2.4 Programming examples

The example demonstrates the programming model and usage of APIs to write and read sparse data can be found in the GitHub repo
https://github.com/LifeboatLLC/SparseHDF5/tree/main/design_docs/examples

Revision History

April 28, 2025	Version 1 circulated for comment within Lifeboat, LLC.
April 27, 2025	Version 2 added link to examples.