RFC: Finalizing Sparse Data Programming Model and APIs

Elena Pourmal [elena.pourmal@lifeboat.llc](mailto: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 2](#_Toc196986218)

[2 Programming Model 2](#_Toc196986219)

[2.1 Two ways of setting structured chunk storage 2](#_Toc196986220)

[2.2 Proposed programming model 3](#_Toc196986221)

[2.2.1 H5Pset\_struct\_chunk 3](#_Toc196986222)

[2.2.2 H5Pget\_struct\_chunk\_sections 4](#_Toc196986223)

[2.2.3 H5Pset\_filter2 5](#_Toc196986224)

[2.3 Changes to direct chunk functions 6](#_Toc196986225)

[2.4 Programming examples 7](#_Toc196986226)

# 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.

# 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.

## 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:
   1. It works with the existing H5Pset\_layout and H5Pset\_chunk functions.
   2. 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.

1. 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.

## 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.

### 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.

#### Signature

*herr\_t* [H5Pset\_struct\_chunk](https://gamma.hdfgroup.org/ftp/pub/outgoing/vchoi/SPARSE/hdf5lib_docs/html/group___d_c_p_l.html#title34) (*hid\_t* plist\_id, *int* ndims, *const hsize* dim[], *uint8\_t* flag, *size\_t* data\_size, *void* \*data)

#### 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 |

#### 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]](#footnote-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.

#### Returns

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

### H5Pget\_struct\_chunk\_sections

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

#### Signature

*herr\_t* H5Pget\_struct\_chunk\_sections (*hid\_t* plist\_id, *int* \*num, *H5\_section\_type\_t* buf[])

#### 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. |

#### 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;

#### Returns

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

### H5Pset\_filter2

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

#### 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)

#### 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 |

#### 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.

#### Returns

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

## 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)

#### 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 |

#### 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.

## 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. |

1. The names of the flags are subject to change. [↑](#footnote-ref-1)