RFC: File Format Changes for Enabling Sparse Storage in HDF5
Implemented Prototype

Vailin Choi (vchoi@hdfgroup.org)
John Mainzer (john.mainzer@lifeboat.llc)
Elena Pourmal (elena.pourmal@lifeboat.llc)[1]

The document introduces new storage paradigms—**Structured Chunk** and **Filtered Structured Chunk**—for storing and performing efficient I/O on sparse data, and describes implemented extensions to the HDF5 file format. While the current implementation focuses on sparse data arrays, these file format extensions can also support other types of data arrays—for example, arrays with HDF5 variable-length data elements and non-homogeneous arrays, i.e., arrays which data elements have different datatypes.

For details on sparse data, programming model and APIs, see "RFC: Programming Model to Support Sparse Data in HDF5" [1]. The proposed extensions to the HDF5 File Format [2] and new APIs to support structured chunk storage will be contributed to the open source HDF5 software maintained by The HDF Group.

This document went through multiple revisions that are available from https://github.com/LifeboatLLC/SparseHDF5. RFC Version 23 and later describe only implemented changes and are in sync with version 3.1 of the File Format HTML document at https://gamma.hdfgroup.org/ftp/pub/outgoing/vchoi/SPARSE/H5.format.html. The File Format document is also available from the GitHub repo [5].

Prototype implementation can be found in https://github.com/HDFGroup/hdf5/tree/feature/sparse_data.

[1] Contact person

1 Introduction

Support for efficient storage of sparse data in HDF5 has been a long-standing request from the HDF5 user community. For background information on sparse data in HDF5 and related implementation concepts, we refer the reader to the original RFC [3] published in 2018. In January 2023, Lifeboat, LLC was awarded a DOE SBIR Phase I grant to design and prototype sparse data storage in HDF5. This document is part of that design effort and outlines the necessary extensions to the HDF5 file format. The implementation work is ongoing, following the award of a DOE SBIR Phase II grant to Lifeboat in April 2024.

While the immediate focus is on file format changes to support sparse data, we have also considered several other potential HDF5 enhancements and identified commonalities with the sparse data challenge. In particular, extending the concept of a chunk to include multiple sections—each describing different facets of the values stored—appears broadly applicable. This approach could benefit other use cases, such as HDF5 variable-length data and non-homogeneous arrays. However, it also introduces the challenge of efficiently compressing these distinct sections, as different compression algorithms may be optimal for different data types. The remainder of this document outlines the new storage paradigms and extensions to existing concepts required to support sparse data storage in HDF5.

The document is organized as follows.

In Section 2, we introduce the concept of the **Structured Chunk** for storing chunks composed of arbitrary variable-size sections, and the **Filtered Structured Chunk** for storing filtered versions of those sections. These new storage paradigms are used to support sparse data in HDF5. In the future, they will also enable efficient storage of HDF5 variable-length data [4]—within both dense and sparse datasets—as well as non-homogeneous arrays. Only minimal additional modifications will be required to support these extended use cases.

In Section 3, we introduce an extension to the existing dataset header message—the **Data Layout Message**—along with a new **Structured Chunk Filter Pipeline Message** to enable the storage of both non-filtered and filtered structured chunks.

In Section 3.1.3, we outline the extensions to the existing chunk indexing schemes needed to support both non-filtered and filtered structured chunks.

Section 5 is currently reserved for documenting recommendations after proposal acceptance.

2 New Storage Paradigms: Structured Chunk and Filtered Structured Chunk

In this section, we introduce the concepts of the **Structured Chunk** and **Filtered Structured Chunk**, and describe the layouts of both non-filtered and filtered structured chunks, along with their corresponding metadata.

2.1 Structured Chunk

Like a regular chunk, a structured chunk stores the values contained within an n-dimensional rectangular volume of a dataset. However, unlike regular chunks, a structured chunk is composed of two or more sections[2] that together describe the values in that volume. The number of sections is defined by Structured Chunk Type and its properties (for more information see Section 3, Structured Chunk Composition field in Table 13 and Table 14.) All structured chunks in a dataset have the same number of sections, although some sections may be empty for a particular chunk. Because each section typically requires different management, we need to store the offset of each section within every structured chunk. Since the size of each section typically varies from chunk to chunk, this information—referred to as chunk metadata—must be maintained on a per-chunk basis. The details of the structured chunk metadata are discussed in the next section. The details of metadata's storage is covered in Section 3.1.3.

Deleted: Table 13

Deleted: Table 14

Formatted: Font: (Default) +Body (Calibri)

Formatted: Font: (Default) +Body (Calibri)

In the context of sparse data, structured chunks typically contain two or three sections: one for the encoded selection indicating which values are defined within the chunk, one for the actual values[3] themselves, and if needed, one for a heap containing the variable-length data[4]. Similarly, structured chunks can be used to store dense datasets with variable-length data. In this case, the first section would contain data similar to that of a current chunk in a dense dataset with variable-length data, while the second section would hold a heap for the variable-length data, which is currently stored in global heaps within the HDF5 file.

2.1.1 Structured Chunk Metadata

The Structured Chunk Metadata contains information about the location of each section within a Structured Chunk. See Table 1. Rather than storing the sizes of each section, the metadata holds the offsets of each section. Note that the offset of the first section (e.g., the one containing the encoded selection for sparse data) is not included, as it is assumed to be zero.

Deleted: Table 1

Formatted: Font: (Default) +Body (Calibri)

Table 1: Structured Chunk Metadata

byte	byte	byte	byte
Offset of section 1 *(8 bytes)*			
…			
Offset of section N-1 *(8-bytes)*			

[2] In this document we use 0-based numbering of the sections. For example, Section 1 is the second section, and Section N-1 is the last section.
[3] Which may be of any HDF5 data type.
[4] Variable-length data is not supported in the initial prototype implementation.

Table 2: Fields: Structured Chunk Metadata

Field Name	Description
Offset of section *i*	Offset in the structured chunk of the beginning of section *i* where *i* is from 1 to N-1. Note that the numbering of sections is 0-based. As the offset of section 0 is presumed to be zero, it is therefore not recorded and starts with section 1 (the second section).

2.1.2 Structured Chunk Layout

Table 3 shows how the data is organized in a Structured Chunk and Table 4 contains descriptions of the fields. Sections that contain data necessary for interpreting the contents of other sections (i.e., metadata) include checksums. For example, in sparse data, the first section of the Structured Chunk holds the encoded locations of data and always includes a checksum—since corruption of this section would prevent proper interpretation of the data values stored in the second section for fixed-size datatype sparse datasets.

Deleted: Table 3
Deleted: Table 4
Formatted: Font: (Default) +Body (Calibri)
Formatted: Font: (Default) +Body (Calibri)

Table 3: Structured Chunk Layout

byte	byte	byte	byte
Section 0 (variable size – may be empty)			
Section 0 Checksum (*4 bytes*, may not exist)			
…			
Section N-1 (variable size - may be empty)			
Section N-1 Checksum (*4 bytes*, may not exist)			

Table 4: Fields: Structured Chunk Layout

Field Name	Description
Section *i*	The data contained in section *i* of the structured chunk where *i* is from 0 to N-1. If the section is empty, the offset of section *(i+1)* will be the same as that of section *i*.
Section *i* Checksum	If section *i* contains metadata for other section(s), the section *i* checksum must appear. Note that for purposes of computing section offsets, the section *i* checksum is part of section *i*.

Having presented the structured chunk in general terms, we now present the structured chunk as we propose to use it for the storage of sparse datasets – first without variable-length data, and then with.

Finally, we show the possible use of the structured chunk to represent dense datasets with variable-length data.

Table 5: Structured Chunk to store sparse dataset of fixed-size datatype

byte	byte	byte	byte
Encoded Selection of Defined Elements (Section 0)			
Section 0 Checksum (*4 bytes*)			
Data Section (Section 1)			

A structured chunk that stores **sparse data of a fixed-size datatype** consists of two sections. Section 0 contains an encoded selection that specifies the locations of the defined values within the chunk. A checksum is required for this section, as the encoded selection effectively serves as metadata. Corruption of this section would make it impossible to interpret the data stored in Section 1.

Section 1, the "Data Section," contains the defined values. For a fixed-size datatype, each datum in this section has the same length[5]. If the selection is empty, this section will have zero length.

Table 6: Structured Chunk to store sparse dataset of variable-length datatype

byte	byte	byte	byte
Encoded Selection of Defined Elements (Section 0)			
Section 0 Checksum (*4 bytes*)			
Data Section (Section 1)			
Section 1 Checksum (*4 bytes*)			
Variable-size Data Heap (Section 2)			
Checksum for Variable-size Data Heap (*4 bytes*; Section 2 Checksum)			

As before, Section 0 contains an encoded selection of values defined in the structured chunk, along with its checksum. In essence, nothing changes here when adding support for variable-size data.

Similarly, Section 1 contains references (format TBD—for example, offset/length pairs) to data stored in Section 2, the Variable-Size Data Heap. Each data element in this section has a uniform length and, in the programming interface, is referenced in the same manner as data in Section 1 for the sparse fixed-size datatype case described above. Since these references are metadata, Section 1 also requires a checksum.

[5] In the programming interface this section is referred to as `H5_SECTION_FIXED`

Conceptually, the Variable-size Data Heap (Section 2)[6] is a buffer that holds the variable-length data referenced by Section 1. To enable tracking of unused space within the heap, the first four bytes are reserved to store the number of unused bytes. Because variable-length data may contain references to other variable-length data, the heap itself may include metadata—therefore, a checksum is also required for Section 2.

Table 7: Structured Chunk to store dense dataset of variable-length datatype

byte	byte	byte	byte
Data Section (Section 0)			
Section 0 Checksum (*4 bytes*)			
Variable-size Data Heap (Section 1)			
Checksum for Variable-size Data Heap (*4 bytes*; Section 1 Checksum)			

When used for dense datasets with variable-length data (see Table 7), the content of the "Data Section" (Section 0) is similar to that of an existing chunk in a dataset containing variable-length data. The key difference is that, instead of referencing variable-length data stored in global heaps as in the current HDF5 implementation, the data is stored directly in Section 1, "Variable-size Data Heap**"**, i.e., within the chunk itself. Section 0 requires a checksum, as it is considered as metadata.

Deleted: Table 7

Formatted: Font: (Default) +Body (Calibri)

The Variable-size Data Heap in this context is identical to the one described above for sparse datasets with variable-length data.

2.2 Filtered Structured Chunk

A Filtered Structured Chunk is essentially a Structured Chunk in which one or more of its sections have been processed through filter pipelines.

Managing the filtering of individual sections requires additional metadata, which is discussed in the following section. As with Structured Chunks, the details of how this metadata is stored are covered in Section 4.

2.2.1 Filtered Structured Chunk Metadata

The **Filtered Structured Chunk metadata** contains the information necessary to locate and interpret data within a Filtered Structured Chunk.

[6] In the programming interface this section is referred to as `H5_SECTION_VL`

This includes a filter mask for each section, the offset of each section within the chunk, and the unfiltered size of each section[7] (see Table 8 for chunk layout and Table 9 for fields description). The size of each section is easily computable from the section offsets.

Deleted: Table 8
Deleted: Table 9
Formatted: Font: (Default) +Body (Calibri)
Formatted: Font: (Default) +Body (Calibri)

Table 8: Filtered Structured Chunk Metadata

byte	byte	byte	byte
Offset of Section 1 (*8 bytes*)			
…			
Offset of Section N-1 (*8 bytes*)			
Unfiltered size of Section 0 (*8 bytes*)			
…			
Unfiltered size of Section N-1 (*8 bytes*)			
Filter Mask of Section 0 (*4 bytes*)			
…			
Filter Mask of Section N-1 (*4 bytes*)			

Table 9: Fields: Filtered Structured Chunk Metadata

Field Name	Description
Offset of Section *i*	Offset in the filtered structured chunk of each section where *i* is from 1 to N-1. Note that the numbering of sections is 0-based. Since section 0 always starts at offset zero, the offset of section 0 is omitted and starts with section 1 (the second section).
Unfiltered Size of Section *i*	Unfiltered size of each section where *i* is from 0 to N-1. It will be zero if the section is empty.
Filter Mask of Section *i*	One filter mask per section of the filtered structured chunk where *i* is from 0 to N-1. If no pipeline is defined for the section, the filter mask is 0.

2.2.2 Filtered Structured Chunk Layout

The layout of a Filtered Structured Chunk is identical to that of a Structured Chunk, with the only difference being that one or more sections have an associated filter pipeline and may be filtered. Note

[7] The un-filtered section sizes are included to allow immediate allocation of correctly sized buffers when un-filtering. There are differing opinions as to the utility of this – we propose to try it and see.

that, as with section size computation, any checksum associated with a section is considered part of the section for the purposes of filtering.

The specific layout of a Filtered Structured Chunk depends on its application and the filter pipelines defined (or not) for each section. The following example (see Table 10) illustrates the layout for a Filtered Structured Chunk used to store a sparse dataset with variable-length data, assuming that all sections have filter pipelines defined.

Deleted: Table 10

Formatted: Font: (Default) +Body (Calibri)

As in the unfiltered example above, each section includes an associated checksum, which is treated as part of the section for filtering purposes. Additionally, the latter two sections may be empty—in which case, their lengths will be zero.

Table 10: Filtered Structured Chunk – Sparse Dataset with Variable-Length Data Case[8]

byte	byte	byte	byte
Filtered Encoded Selection			
Section 0 Checksum			
Filtered Data (Section 1)			
Section 1 Checksum			
Filtered Variable-size data heap (Section 2)			
Section 2 Checksum			

[8] Note that at least one section is filtered.

3 HDF5 File Format Extensions

This section describes extensions to the HDF5 File Format to support **structured chunk storage**. Extensions are required for only two dataset header messages: the **Data Layout Message** and the **Filter Pipeline Message**.

In the Data Layout Message, we introduce support for Structured Chunk Storage and a new Sparse Layout, which will be used to store sparse data. In the Filter Pipeline Message, we extend the existing *Flags* field to allow different filter pipelines to be applied to the individual sections of a Filtered Structured Chunk. All changes are highlighted in **bold**.

3.1 Structured Chunk Storage for Encoding Sparse Data Arrays

Sparse data storage will require modifications to the Data Layout Message, potentially resulting in a new **Version 5**. This will involve introducing a new layout class and its associated properties, as described in the next section. It's important to note that introducing a new message version is not strictly necessary—unless, during the design process, we determine that changes to the existing message structure are required to support the new storage paradigm.

Versions of the HDF5 library from 1.10.0 onward that are unaware of the new layout class for indicating Structured Chunk Storage should fail gracefully when encountering it. *For now, we propose incrementing the message version to 5 out of an abundance of caution.*

3.1.1 Data Layout Message (Version 5) Extension

Data Layout Message (Version 5) added a new Layout Class "Structured Chunk Storage" as shown for the reference and reader's convenience in Table 11.

Deleted: Table 11
Formatted: Font: (Default) +Body (Calibri)

Table 11: Data Layout Message (Version 5)

byte	byte	byte	byte
Version	Layout Class	*This space inserted only to align table nicely*	
Properties (*variable size*)			

Table 12 shows new Layout Class "Structured Chunk Storage".

Deleted: Table 12
Formatted: Font: (Default) +Body (Calibri)

Table 12: Fields: Data Layout Message (Version 5)

Field Name	**Description**
Version	The value for this field is 5
Layout Class	The layout class specifies the type of storage for the data and how the other fields of the layout message are to be interpreted.
Properties	This variable-size field encodes information specific to a layout class as follows:

Value	**Description**
0	Compact Storage
1	Contiguous Storage
2	Chunked Storage
3	Virtual Storage
4	**Structured Chunk Storage (see Note).**

Layout Class	**Description**
Compact Storage	See Compact Storage Property Description for the version 3 Data Layout message.
Contiguous Storage	See Contiguous Storage Property Description for the version 3 Data Layout message.
Chunked Storage	See Chunked Storage Property Description below.
Virtual Storage	See Virtual Storage Property Description below.
Structured Chunk Storage	**See Structured Chunk Storage Property Description below.**

Table 13 describes the layout of the **Structured Chunk Storage Property**[9]. This property is similar to the standard chunked storage property but includes two additional fields: a **"Structured Chunk Type"** field, which specifies the type of structured chunk, and a **"Structured Chunk Composition"** field, which defines type-specific properties.

Deleted: Table 13

Formatted: Font: (Default) +Body (Calibri)

At present, this section outlines only the properties relevant to structured chunks used for sparse data. Note that dimension numbering in the table is 0-based**.**

[9] It was suggested to call the new layout class "sectioned storage" to distinguish from chunk storage. For the initial implementation of new storage for sparse and variable-length data, we will be using chunk storage to store all sections that describe the data together. If desired, sections can be stored separately and can be used for other types of data storage access, e.g., column storage of 2-dim arrays and tables. Therefore, we reserve "sectioned storage" for the future.

Table 13: Structured Chunk Storage Property Description

byte	byte	byte	byte
Version	**Structured Chunk Type**		This space inserted to align table nicely
Flags	Dimensionality	Dimension Size Encoding Length	This space inserted to align table nicely
Dimension 0 Size (*variable size*)			
Dimension 1 Size (*variable size*)			
…			
Dimension #N-1 Size (*variable size*)			
Structured Chunk Composition (*variable size*)			
Chunk Indexing Type[10]	This space inserted to align table nicely		
Indexing Type Information			
Address ^{O}			

Fields of Structured Chunk Property are described in Table 14.

Deleted: Table 14

Formatted: Font: (Default) +Body (Calibri)

[10] We decided to use current indexing schemas for the Structured Chunk to accommodate different types of the Structured Chunks in the future instead of extending schemas when we add support for new type of the Structured Chunk.

Table 14: Fields: Structured Chunk Storage Property Description

Field Name	Description
Version	This is the version number for the Structured Chunk Storage property. The value for this field is 0 and is introduced to support structured chunk.
Structured Chunk Type	Type of the structured chunk[11]

Bit	Description
0	Sparse Chunk Storage Property Description
1	HDF5 Variable-Length Chunk Storage Property Description
2-15	Reserved

Field Name	Description
Flags	The same as in Chunked Storage Property Description
…	The same as in Chunked Storage Property Description
Address	The same as in Chunked Storage Property Description
Structured Chunk Composition	See Structured Chunk Composition Description

Table 15 specifies properties for Structured Chunk that is applicable to the types listed in Table 14.

Deleted: Table 15

Deleted: Table 14

[11] Currently we plan to have only one for sparse storage but can add more in the future (for example, HDF5 variable-length data dense chunks, non-homogeneous arrays, dense or sparse chunks with missing values). Since structured chunk may have more than one type (e.g., sparse variable-length data), we use a bit-field.

Table 15: Structured Chunk Composition Description

byte	byte	byte	byte
Offset Size (8 bytes)			
Number of Sections	Number of Sections Containing Metadata	This space inserted to align table nicely	
Sequence number of First Section with Metadata	Sequence number of Second Section with Metadata	…	Sequence number of Last Section with Metadata

Table 16: Fields: Structured Chunk Composition Description

Field Name	Description
Offset size	Number of bytes used to store offsets in the structured chunk; currently it is 8 bytes. Please notice that structured chunk does not have 4GB limit on the chunk size as the “dense” chunk has due to API limitations.
Number of sections	Number of sections N in each structured chunk in the dataset. At present, this number is implied by the structured chunk type above. However, this need not always be the case.
Number of sections containing metadata	Number of sections K which may contain metadata, and which therefore requires a checksum if non-empty. At present, this number and the list of sections with metadata below is implied by the *Structured Chunk Type* above. However, this need not always be the case.
Sequence number of first section with metadata	Sequence number M_1, of the first section that may contain metadata, $N-1 >= M_1 >= 0$.
Sequence number of second section with metadata	Sequence number $M_2 > M_1$ of the second section that may contain metadata.
Sequence number of last section with metadata	Sequence number M_K of the last section that may contain metadata, where $M_K > \ldots > M_2 > M_1$. The total number of sections from *first* through *last* section will be equal to the *"Number of Sections containing metadata"* K above.

3.1.2 Filtering of Sparse Data Arrays

The HDF5 library allows up to 32 filters to be applied to a dense dataset's data during I/O operations. We propose extending this capability to datasets that use Structured Chunk Storage by allowing a separate filter pipeline to be applied to each section of a structured chunk.

For example, in the case of sparse datasets, we should anticipate scenarios where users may wish to apply one set of filters (if any) to the HDF5 dataspace selection, and a different set of filters to the actual data elements. Similarly, for variable-length data, there may be compression advantages to applying one filter pipeline to the section containing references to data elements, and a different one to the section containing the variable-length values themselves.

APIs for managing filtering of datasets with structured chunk storage are discussed in [**Error! Reference source not found.**].

Table 17 below shows version 3 of Structured Chunk Filter Pipeline Message.

Deleted: Table 17

Formatted: Font: (Default) +Body (Calibri), English (US)

Version 3 of Filter Pipeline Message may be present in the object headers of a dataset that uses structured chunk storage. It specifies the filters to apply to each section of the structured chunk by providing a filter description. A filter description is the same as in Filter Pipeline Message Version 2 (i.e., filter identification numbers, flags, a name, and client data).

Table 17: Filter Pipeline Message – Version 3[12]

byte	byte	byte	byte
Version	Number of Filtered Sections	This space inserted to align table nicely	
Sequence number of First Filtered Section	Number of filters for First Filtered Section	Size of the first Filter Description List[13]	
Filter Description List (variable size)			
…			
Sequence number of Last Filtered Section	Number of filters for Last Filtered Section	Size of the Last Filter Description List	
Filter Description List (variable size)			

[12] Filter Pipeline Message Version 3 is used only for structured chunk storage and is not used for dense storage even when H5F_LIBVER_LATEST is specified by "low" parameter in the H5Pset_libver_bounds. This limitation will be removed when dense chunks will use Shared Chunk Cache in the future.

[13] To make parsing easier.

Table 18: Fields: Filter Pipeline Message

Field Name	Description
Version	The version number of this message. The value for this field is 3 and is introduced to support structured chunk.
Number of Filtered Sections	Total number L of the filtered sections in structured chunk.
Sequence number of First Filtered Section	Sequence number M_1 of the first filtered section N-1 >= M_1 >= 0, where N is number of sections in structured chunk.
Number of filters for first Filtered Section	The total number of filters specified for the first section that is filtered. The maximum possible number of filters in a message is 32.
Size of the first Filter Description List	Size of the first Filter Description List in bytes.
Filter Description List	A description of each filter as it appears in the filter description list of the version 2 Filter Pipeline message. See [5] or details.
Sequence number of Last Filtered Section	Sequence number M_L of the last filtered section, where N-1 >= M_L >…>M_1>= 0.
Number of filters for Last Filtered Section	The total number of filters specified for the last section that is filtered. The maximum possible number of filters in a message is 32.
Size of the Last Filter Description List	Size of the Filter Description List in bytes for the last section that is filtered.
Filter Description List	A description of each filter as it appears in the filter description list of the version 2 Filter Pipeline. See [5] or details.

3.1.3 Discussion of Filter Description List Versions and H5Pset_filter2

The reader can skip this section unless is interested in the rationale for using Version 2 of the Filter Description List and the current implementation of the `H5Pset_filter2` function.

Initially, we proposed new H5Pset_filter2 function with the following signature to address some known issues in passing data to the filter functions:

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

It was anticipated that the new function would offer greater flexibility when used with VOL connectors and complex filter data provided by an application. Its implementation led to the proposal of the Filter Description List Version 3 message:

byte	**byte**	**byte**	**byte**
Filter Identification Value		Name Length *(optional)*	
Flags (8 bytes)			
Client Data Buffer Length (8 bytes)			
Name *(variable size, optional)*			
Client Data *(variable size, optional)*			

Fields: Filter Description – Version 3

Field Name	**Description**
Filter Identification Value	As in Filter Description – Version 2
Name Length	Each filter has an optional null-terminated **UTF-8** name and this field holds the length of the name including the null termination padded with nulls to be a multiple of eight. If the filter has no name, then a value of zero is stored in this field. Filters with IDs less than 256 (in other words, filters that are defined in this format documentation) do not store the *Name Length* or *Name* fields.
Flags	The flags indicate certain properties for a filter. The bit values defined so far are: Bit — Description 0 — If set then the filter is an optional filter. During output, if an optional filter fails it will be silently skipped in the pipeline. 1-63 — Reserved (zero)
Client Data Buffer Length	Each filter can store values to control how the filter operates. The number of bytes in the *Client Data* buffer is stored in this field.
Name	If the *Name Length* field is non-zero, then it will contain the size of this field, *not* padded to a multiple of eight. This field contains a *non*-null-terminated, **UTF-8** string to serve as a comment/name for the filter. Filters that are defined in this format documentation such as deflate and shuffle do not store the *Name Length* or *Name* fields.
Client Data	This is a buffer which will be passed to the filter function. The Client Data is a binary buffer of length stored in Data Buffer Length field.

Several issues were identified with the proposed `H5Pset_filter2` function and Filter Description List Version 3 during the prototype implementation.

1. It would be necessary to update internal H5Z APIs to support handling of `void` buffers.
2. The HDF5 library currently lacks a mechanism for encoding void buffers. Users typically work around this by packing their data into an integer array, leaving it to the filter function to interpret the data. One possible solution is to define a grammar and parser, allowing data to be passed as a string formatted according to the grammar. This would enable portable data encoding.
3. If encoding the void buffer continues to rely on integer arrays, introducing Version 3 of the Filter Description List becomes less meaningful, as the remaining changes are relatively minor and not compelling.
    a. The rationale for using a 64-bit number for flags is unclear.
    b. ASCII strings are currently sufficient; switching to UTF-8 encoding is not critical.

Therefore, in the current implementation we use the `H5Pset_filter2` function with the following signature:

```
herr_t H5Pset_filter2 (hid_t plist_id, H5_section_type_t section_type,
                       H5Z_filter_t filter,
                       unsigned int flags,
                       size_t cd_nelmts,
                       const unsigned int cd_values[])
```

and Filter Description List 2 provided below for readers convenience.

Layout: Filter Description – Version 2

byte	**byte**	**byte**	**byte**
Filter Identification Value		Name Length *(optional)*	
Flags		Number Client Data Values	
Name *(variable size, optional)*			
Client Data *(variable size, optional)*			

Please notice that the function `H5Pset_filter2` can be used with dense chunk storage when NULL (or TBD) value is passed[14] in the `section_type` parameter.

4 Structured Chunk Indexing

We need to modify the existing chunk indexing mechanisms to support both non-filtered and filtered Structured Chunk Metadata**.** The following sections describe the required changes to the indexing schemes: Single Chunk Indexing**,** Fixed Array Indexing**,** Extensible Array Indexing**,** and Version 2 B-tree

[14] In the first implementation this function is not implemented to handle dense chunks.

Chunk Indexing**.** Implicit Indexing cannot be used for structured chunks due to their variable-size metadata. Version 1 B-tree Chunk Indexing is considered obsolete and is not being extended to support new features.

Throughout this section, we refer to both Structured Chunk Metadata and Filtered Structured Chunk Metadata, as defined in Sections 2.1.1 and 2.2.1, respectively. While the size of this metadata is variable, it depends on the number of sections, which is specified in the Data Layout message and remains constant within any given dataset. All proposed extensions are highlighted in **bold**.

4.1 Single Chunk Indexing[15]

For Single Chunk Index we will need change "Single Chunk Indexing Information" in Layout message as shown in Table 19 and Table 20.

Deleted: Table 19
Deleted: Table 20
Formatted: Font: (Default) +Body (Calibri)
Formatted: Font: (Default) +Body (Calibri)

Table 19: Layout: Structured Chunk Indexing Information

byte	byte	byte	byte
Chunk Size (variable size; at most 8 bytes)			
Structured Chunk Metadata (variable)			

Table 20: Layout: Filtered Structured Chunk Indexing Information

byte	byte	byte	byte
Chunk Size (variable size; at most 8 bytes)			
Filtered Structured Chunk Metadata (variable)			

4.2 Fixed Array Indexing Information

This section outlines the changes to The Fixed Array Index described in the section VII.C of the File Format Document. We can use current layouts of Fixed Array Header, Fixed Array Data Block and Fixed Array Data Page, but we need to change the versions and introduce new element types as shown in the tables below.

4.2.1 Changes to Fixed Array Header Fields

The additions of new clients' IDs are shown in **Table 21**.

Deleted: Table 21

[15] See "Structured Chunk Type Indexing for Single Chunk" in sub-section of "Version 5 Data Layout Message for the structured chunk layout class".

Table 21: Fields[16] : Fixed Array Header Version 1[17]

Signature	No changes
Version	This document describes version **1**.
Client ID	The ID for identifying the client of the Fixed Array:
Other fields	No changes

ID	Description
0	Non-filtered dataset chunks
1	Filtered dataset chunks
2	**Structured dataset chunks**
3	**Filtered structured dataset chunks**
4+	Reserved

Similar changes are necessary for the fields in Fixed Array Data Block and Fixed Array Data Block Page as described in the next sections.

4.2.2 Changes to Fixed Array Data Block Fields

The additions of new client IDs and Elements are shown in Table 22.

Deleted: Table 22

Formatted: Font: (Default) +Body (Calibri)

[16] We do not provide the whole table to save space in this document since the rest of the fields are the same.

[17] Current implementation uses new version; we agreed that it is not a necessary change, this will be fixed before the official prototype release.

Table 22: Fields: Fixed Array Data Block Version 1[18]

Field Name	Description
Signature	The ASCII character string "FADB" is used to indicate the beginning of a Fixed Array data block. This gives file consistency checking utilities a better chance of reconstructing a damaged file.
Version	This document describes version **1**.
Client ID	The ID for identifying the client of the Fixed Array: ID | Description 0 | Non-filtered dataset chunks 1 | Filtered dataset chunks **2** | **Structured dataset chunks** **3** | **Filtered structured dataset chunks** **4+** | **Reserved**
Header Address	The address of the Fixed Array header. Principally used for file integrity checking.
Page Bitmap	A bitmap indicating which data block pages are initialized. Exists only if the data block is paged.
Elements	Contains the elements stored in the data block and exists only if the data block is not paged. There are **four** element types: ID | Description 0 | Non-filtered dataset chunks 1 | Filtered dataset chunks **2** | **Structured dataset chunks** **3** | **Filtered structured dataset chunks** **4+** | **Reserved**
Checksum	The checksum for the Fixed Array data block.

[18] Current implementation uses new version; we agreed that it is not a necessary change, this will be fixed before the official prototype release.

4.2.3 Changes to the layout of Fixed Array Data Block Page

Additions of new elements are shown in Table 23.

Deleted: Table 23

Formatted: Font: (Default) +Body (Calibri)

Table 23: Fields: Fixed Array Data Block Page Version 0

Field Name	Description
Elements	Contains the elements stored in the data block page. There are four element types: ID | Description 0 | Non-filtered dataset chunks 1 | Filtered dataset chunks **2** | **Structured dataset chunks** **3** | **Filtered structured dataset chunks** **4+** | **Reserved**
Checksum	The checksum for a Fixed Array data block page.

4.2.4 Layout of Data Block Element for Structured Dataset Chunk

The layout of Data Block Element for Structured Dataset Chunk is shown in Table 24.

Deleted: Table 24

Formatted: Font: (Default) +Body (Calibri)

Table 24: Layout of Data Block Element for Structured Dataset Chunk

byte	byte	byte	byte
Address ^{O}			
Chunk Size (variable size; at most 8 bytes)			
Structured Chunk Metadata (variable size, but fixed within any one index)			

4.2.5 Layout of Data Block Element for Filtered Structured Dataset Chunk

The layout of Data Block Element for Filtered Structured Dataset Block is shown in Table 25.

Deleted: Table 25

Formatted: Font: (Default) +Body (Calibri)

Table 25: Layout of Data Block Element for Filtered Structured Dataset Chunk

byte	byte	byte	byte
Address ^{O}			
Chunk Size (variable size; at most 8 bytes)			
Filtered Structured Chunk Metadata (variable size, but fixed within any one index)			

4.3 Extensible Array Indexing Information

As with the Fixed Array Index, we will need to modify the Extensible Array Header, Index Block, Secondary Block, Data Block by adding new element types as for the Fixed Array Index. Layout of the new data block elements as in the Fixed Array Index.

4.4 Version 2 B-tree chunk indexing

Version 2 B-tree we will need to do the following updates to the File Format:

- Introduce new type for B-tree:
  - 12 – This B-tree is used for indexing structured chunks of datasets with no filters and with more than on dimension of unlimited extent.
  - 13 – This B-tree is used for indexing filtered structured chunks of datasets with more than on dimension of unlimited extent.

Record Layout for the new types are presented in Table 26 and Table 27.

Deleted: Table 26

Deleted: Table 27

Formatted: Font: Not Bold, Font color: Text 1

Formatted: Font: Not Bold, Font color: Text 1

Table 26: Version 2 B-tree, Type 12 Record Layout – Unfiltered Structured Dataset Chunk

byte	byte	byte	byte
Address ^{O}			
Chunk Size (variable size; at most 8 bytes)			
Dimension 0 Scaled Offset (8 bytes)			
Dimension 1 Scaled Offset (8 bytes)			
…			
Dimension N Scaled Offset (8 bytes)			
Structured Chunk Metadata (variable size, but fixed within any one B-Tree)			

As one can see, we just extended Type 10 Record Layout with the structured chunk metadata info.

Table 27: Version 2 B-tree, Type 13 Record Layout – Filtered Structured Dataset Chunk

byte	byte	byte	byte
Address ^{O}			
Chunk Size (variable size; at most 8 bytes)			
Dimension 0 Scaled Offset (8 bytes)			
Dimension 1 Scaled Offset (8 bytes)			
…			
Dimension N Scaled Offset (8 bytes)			
Filtered Structured Chunk Metadata (variable size, but fixed within any one B-Tree)			

As one can see, we modified Type 11 Record Layout for Filtered Dataset Chunks by removing “Filter Mask” field and adding “Filtered Structured Chunk Metadata” field instead.

5 Final recommendation: HDF5 File Format changes for Sparse Data

To be added after community discussion.

Acknowledgment

This work is supported by the U.S. Department of Energy, Office of Science under Award number DE-SC0023583 for SBIR project "Supporting Sparse Data in HDF5".

The authors would like to thank The HDF Group developers and Quincey Koziol, Principal Engineer, AWS HPC for reviewing the numerous versions of the document and for fruitful discussions.

Revision History

February 28, 2023:	Version 1 is created for internal review
March 14, 2023	Version 2 is created for internal review; Version 1 was modified to introduce "structured chunk" and extended filtering.
April 13, 2023	Version 3 is created for internal review
April 18, 2023	Version 4 is created for internal review
April 20, 2023	Version 5 is created for internal review
April 29, 2023	Version 6 is created for internal review
April 30, 2023	Version 7 is created for THG review
May 1-31, 2023	Version 8 -10 is created after THG review. Structured Chunk and Filtered Structured Chunk headers described in the previous versions were updated and are now stored with the chunk index.
June 1-13, 2023	Versions 11-12 have simplified structured chunk metadata. V*ersion 12 sent to THG.*
June 21, 2023	Version 13 created with the new Structured Chunk Filter Pipeline Message for internal review
June 23, 2023	Version 14 created; Filter with the new Structured Chunk Filter Pipeline Message for internal review
June 26, 2023	Version 15 created for THG review
July 10, 2023	Version 16 created for public review. Checksum filed as removed from Chunk Record Layout as discussed with THG.
July 17, 2023	Version 17 created for public review after copy editing.
June 5, 2024	V*ersion 18:* Addressed THG reviewer comments 1. Used Version 3 for Filter Pipeline Message instead of the new type of message that was only applicable to structured chunk storage. 2. Changed the order of the fields in Table 8 to follow FF practices 3. Clarified structured chunk section numbering and fixed it all over the document (section N-1 is the last Nth section in the structured chunk); used 'First" and "Last" when we list of entities instead of index-type of reference that can be confused with section numberings. 4. Renamed "Properties" in Table 14 to "Structured Chunk Type Properties" to distinguish for Structured Chunk Properties
June 11, 2024	Version 19: Addressed THG reviewer fundings: 1. Changed the order in the fields of Table 9 to reflect the change in Table 8.

	2. Reworked # 4 from June 5. a. We decided to use "Structured Chunk Composition" instead of "Structured Chunk Type Properties" b. Removed sub-table in the last field of Table 14
June 13, 2024	Version 20: Explained the changes to the fields in Type 13 Record Layout (Table 27)
August 14, 2024	Version 21: Synced with HTML version https://gamma.hdfgroup.org/ftp/pub/outgoing/vchoi/SPARSE/H5.format.html#ChunkedStorage; added links to the HTML document
December 10, 2024	Version 22: This version of the document addressed comments provided in **Appendix.** Specifically, there is a new proposed version of "The Data Layout Message" in section **Error! Reference source not found.** that adds "sectioned" data storage that may not require chunked storage. Currently, only sectioned (former structured) chunk storage is supported. Versions numbers of chunk index were fixed (they are not incremented now). Minor copyedits were done. Comments in Appendix explain if the proposed change was done or if it was rejected and why. The Fille Format HTML document was updated to use "sectioned storage" and "sectioned chunk" terminology, see [5].
June 19, 2025	Version 23: Removed description of "The Data Layout Message" introduced in version 22, section 3.1.2 since we don't implement it in the current prototype. Added section 3.1.3 that discuss our decision to stay with Version 2 of Filter Description List. Fixed Array Header Version and Fixed Array Data Block Version was left as 1 (section 4.2.1 and 4.2.2) to reflect current implementation; it will be reverted to 0 before the release. Versions for the extensible array and Version 2 B-tree header are left 0 since no implementation was done yet.
September 3, 2025	Version 24: Removed Appendix and misc. copyediting.
September 8, 2025	Version 25: Moved "Structured Chunk Composition (*variable size*)" row in Table 13 right after dimension fields to enable decoding for "Indexing Type Information" for single chunk storage; misc. copyediting and clarifications for the current implementation.
September 19-October 9, 2025	Version 26-28: Copyedits and reworked section 3.1.3. Fixed a link to Draft RFC: Sparse Chunks; fixed several typos and accepted all changes.

References

1. John Mainzer, Elena Pourmal, "RFC: Programming Model to Support Sparse Data in HDF5". Available from https://github.com/LifeboatLLC/SparseHDF5/
2. The HDF Group, "HDF5 File Format Specification" https://www.hdfgroup.org/HDF5/doc/H5.format.html
3. The HDF Group, Draft RFC: Sparse Chunks, https://support.hdfgroup.org/releases/hdf5/documentation/rfc/RFC_Sparse_Chunks180830.pdf
4. The HDF Group, Variable-Length Data in HDF5 Sketch Design, https://docs.hdfgroup.org/hdf5/rfc/var_len_data_sketch_design_190715.pdf
5. File Format Specification (HTML) with the changes as described in RFC: File Format Changes for Enabling Sparse Storage in HDF5 , v. 22, https://github.com/LifeboatLLC/SparseHDF5/blob/main/design_docs/HDF5%20File%20Format%20Specification%20Version%203.1%202025-01-13-RFCv22.html