# RFC: File Format Changes for Enabling Sparse Storage in HDF5

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

The document describes HDF5 file format extensions for storing sparse data. It is work in progress.

---

We introduce new storage paradigms called "Structured Chunk" and "Filtered Structured Chunk" for storing and performing efficient I/O on sparse data, and describe extensions to the existing File Format elements. While current proposal focuses on sparse data arrays, the file format extensions can be used to support other variable-size data, for example, HDF5 variable-length data and arrays of the elements that have different HDF5 datatypes (non-homogeneous arrays).

Introduction of structured chunk storage paradigm does not require major changes to the HDF5 programming model and public APIs. For example, as for the existing chunked storage data filtering is supported with the structured chunk storage. For more information on the programming model and API updates see "RFC: Programming Model to Support Sparse Data in HDF5" [4].

*The proposed extensions to the HDF5 File Format [2] and new APIs found in [4] 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 after discussions with The HDF Group. HTML version of the HDF5 File Format specification with the proposed changes (Version 3.1) is available for review from the following link https://gamma.hdfgroup.org/ftp/pub/outgoing/vchoi/SPARSE/H5.format.html.
Prototype implementation can be found in
https://github.com/HDFGroup/hdf5/tree/feature/sparse_data .

**Update for 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 3.1.2 that adds "sectioned" data layout since sectioned data 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

[1] Contact person

updated to use "sectioned storage" and "sectioned chunk" terminology along with other proposed and accepted changes, see [5].

The RFC is work in progress and will be finalized when we agree on the File Format changes proposed in this version change and implementation is updated accordingly.

# 1 Introduction

Support for efficient storage of sparse data in HDF5 is a long-standing request from the HDF5 users' community. For background information on sparse data in HDF5 and implementation ideas we refer the reader to the original RFC [1] published in 2018. In January 2023 Lifeboat, LLC was awarded DOE SBIR Phase I grant to design and prototype sparse data storage in HDF5. This document is part of the design effort and outlines the necessary extensions to the HDF5 File Format. The work on sparse data implementation is ongoing after Lifeboat was awarded DOE SBIR Phase II in April 2024.

While the immediate need is for file format changes to support sparse data, we have given significant thought to a number of other potential HDF5 enhancements and noticed the commonalities with the sparse data problem. In particular, the idea of extending the concept of the chunk to contain multiple sections that describe different facets of the values stored in the chunk seems to be applicable to a number of problems, for example, to HDF5 variable-length data and non-homogeneous arrays. This in turn raises the problem of compressing these different sections efficiently, as different compression algorithms may be optimal for different sections. In the remainder of this document, we describe the necessary new storage paradigms and extensions to existing concepts needed to support sparse storage in HDF5.

The document is organized as follows.

In Section 2 we introduce the notion of the *Structured Chunk* for storing chunks composed of arbitrary variable-size sections and the notion of the *Filtered Structured Chunk* for storing filtered versions of the sections composing a structured chunk. These new storage paradigms will be used initially for storing sparse data in HDF5. In the future, the new paradigms will allow us to store not only sparse data, but also HDF5 variable-length data in both dense and sparse data sets, and non-homogeneous arrays. Minimal further modifications will be required.

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

In Section 4 we outline the extensions to the current chunk indexing schemas to support non-filtered and filtered structured chunks.

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

We would like to emphasize two important features of our proposal:

- The proposed extensions do not require major changes to the HDF5 programming model and public APIs. We encourage the reader to check [4] for more information.
- The extensions to the File Format and new APIs to support sparse storage will be contributed to the open source HDF5 software maintained by The HDF Group.

# 2 New Storage Paradigms: Structured Chunk and Filtered Structured Chunk

In this section we introduce the concepts of Structured Chunk and Filtered Structured Chunk. The new storage paradigms will be used for storing non-filtered and filtered sparse data. The following sections describe layouts of non-filtered and filtered structured chunk and their corresponding metadata.

## 2.1 Structured Chunk

Like a regular chunk, a structured chunk will be used to store the values contained in some n-dimensional rectangular volume in a dataset. However, unlike regular chunks, it will be composed of two or more sections[2], which in combination, will describe the values contained in the volume. Note that all structured chunks in a dataset will have the same number of sections – although some of these sections may be empty for a particular chunk. As each of these sections will typically require different management, it follows that we must store the offset of each section within each structured chunk. Since the size of each section will typically vary from structured chunk to structured chunk, this structured chunk metadata must be stored on a per chunk basis. This structured chunk metadata is discussed in the next section 2.1.1. The details of how it will be stored is covered in Section 4.

In the context of sparse data, structured chunks will contain two or three sections – one for the encoded selection indicating the values defined within the chunk, one for the actual values[3] themselves, and if required one for a heap containing the variable-length data[4]. Similarly, structured chunks can be used to store dense datasets with variable-length data. The content of the first section would be the same (or similar to) as for the content of current chunk in a dense dataset with variable-length data, and the second section would contain a heap for the variable-length data that is currently stored in global heaps in HDF5 file.

### 2.1.1 Structured Chunk Metadata

The structured Chunk Metadata contains information on how to interpret the data in the Structured Chunk. Note that the offset of the first section of the structured chunk (the encoded selection in the initial application) is not specified in the Structured Chunk Metadata as it is presumed to be zero.

While we don't propose to implement variable-length data at this time, in the dense / variable-length case, the encoded selection would be absent, and the offset of the fixed length data would be zero.

---

[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 will not be supported in the initial prototype implementation.*

**Table 1: Structured Chunk Metadata**

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

**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).<br><br>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.

**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).<br><br>If the section is empty, the offset of section *(i+1)* will be the same as that of section *i*. |
| Section *i* | If section *i* contains metadata, the section *i* checksum must appear.<br><br>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**

<table>
<tr><th>byte</th><th>byte</th><th>byte</th><th>byte</th></tr>
<tr><td colspan="4" align="center">Encoded Selection of Defined Elements (Section 0)</td></tr>
<tr><td colspan="4" align="center">Section 0 Checksum (4 bytes)</td></tr>
<tr><td colspan="4" align="center">Fixed Length Data Section (Section 1)</td></tr>
</table>

Here, the structured chunk stores an encoded selection of the values defined in this chunk in Section 0. A checksum is required, as in this context, the encoded selection is effectively metadata. To see this, observe that each entry in the selection contains a reference into the fixed length data section (Section 1).

The Fixed Length Data Section (Section 1) contains the values associated with the encoded selection of defined values. Its name is derived from the fact that each datum in it must be of the same length. If the selection is empty, this section will be of zero length.

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

| byte | byte | byte | byte |
|---|---|---|---|
| Encoded Selection of Defined Elements (Section 0) | | | |
| Section 0 Checksum (*4 bytes*) | | | |
| Fixed Length 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 with its checksum. In essence, nothing changes here when we add support for variable size data.

Similarly, Section 1 still contains the fixed length data section. Entries in this section are still fixed length, and any variable-length data is represented with offset/length pairs referencing entries in the variable size data heap. However, since these offset / length pairs are metadata, Section 1 now requires a checksum.

Conceptually, the variable size data heap (Section 2) is just a buffer containing the variable-length data referenced in the fixed length data section. To allow tracking of the number of unused bytes in the heap, the first four bytes of the heap are reserved to store this value. Since variable-length data can contain references to other variable-length data, it is possible that the variable-length data heap will contain metadata – which makes a checksum necessary for Section 2 as well.

If the fixed length data section contains no references to variable-length data, the variable-size data heap will be empty. In this case, the variable size data heap doesn't exist, and Section 2 will be of zero length.

Similarly, if the selection is the empty selection, the fixed length data section is empty, and thus Section 1 will be of zero length.

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

| byte | byte | byte | byte |
|---|---|---|---|
| Fixed Length 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 data (see Table 7), the content of the fixed length data section (Section 0) is all but identical to the contents of an existing chunk in a dataset containing variable-length data. The difference is that instead of representing variable-length data with references into global heaps, variable size data is stored in the variable size data heap in Section 1, and referenced by offset / length pairs. Section 0 requires a checksum, as these offset / length pairs are metadata.

The Variable size data heap in this context is identical to that in the sparse data sets with variable-length data heap above.

## 2.2 Filtered Structured Chunk

A Filtered Structured Chunk is essentially a Structured Chunk with one or more of its sections passed through filter pipelines.

Managing filtering of the various sections requires extended metadata, which is discussed in the next section. As with Structured Chunks, the details of how this metadata will be stored is covered in Section 4.

### 2.2.1 Filtered Structured Chunk Metadata

The Filtered Structured Chunk metadata contains the information needed to interpret and find data in a Filtered Structured Chunk.

This includes a filter mask for each section, the offset of each section in the filtered structured chunk, and the un-filtered sized of each section[5] (see Table 8 for chunk layout and Table 9 for fields description). The (possibly filtered) size of each section is easily computable from the section offsets.

**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*) | | | |

[5] *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.*

**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).<br><br>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).<br><br>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).<br><br>If no pipeline is defined for the section, the filter mask is 0. |

### 2.2.2 Filtered Structured Chunk Layout

The layout of the Filtered Structured Chunk is the same as that of the Structured Chunk – the only difference being that at least one of the sections has a filter pipeline defined for it and may be filtered. Note that as with computing the size of a section, the checksum (if any) associated with a section is treated as part of the section for purposes of filtering.

The specifics of the layout of a filtered structured chunk depend on its application, and the filter pipelines defined (or not) for each section. The following example (see Table 10) shows the layout for a Filtered Structured Chunk used to store a sparse dataset with variable-length data. It presumes that filter pipelines are defined for all sections.

Note that as in the unfiltered example above, all sections have associated checksums, which are treated as part of the section for purposes of filtering. Further, it is possible that the latter two sections will be empty – in which case they will have length zero.

**Table 10: Filtered Structured Chunk – Sparse Dataset with Variable-Length Data Case[6]**

| byte | byte | byte | byte |
|---|---|---|---|
| Filtered Encoded Selection | | | |
| Section 0 Checksum | | | |

[6] *Note that at least one section is filtered.*

| |
|---|
| Filtered Fixed Length Data (Section 1) |
| Section 1 Checksum |
| Filtered Variable Size Data Heap (Section 2) |
| Section 2 Checksum |

# 3 HDF5 File Format Extensions

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

In the Data Layout Message, we introduce the new Structured Chunk Storage and Sparse Layout. Structured Chunk with Sparse Layout will be used for storing sparse data. In the Filter Pipeline Message, we extend the existing Flags field to support different filter pipelines for the different 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 Version 4. We will need to introduce a new layout class and its properties as described in the next section. One should note here that introduction of the new version for the message is not really required unless during our design work we discover that the current structure of the message has to be changed to accommodate new storage paradigm. The versions of the library 1.10.0 and later that are not aware of the new Layout Class for indicating Structured Chunk Storage should fail gracefully. For now, we propose to increase the version number to 5 in 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.

**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".

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

<table>
<tr><th>Field Name</th><th>Description</th></tr>
<tr><td>Version</td><td>The value for this field is 5</td></tr>
<tr><td>Layout Class</td><td>The layout class specifies the type of storage for the data and how the other fields of the layout message are to be interpreted.<br><table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>Compact Storage</td></tr><tr><td>1</td><td>Contiguous Storage</td></tr><tr><td>2</td><td>Chunked Storage</td></tr><tr><td>3</td><td>Virtual Storage</td></tr><tr><td>4</td><td>Structured Chunk Storage (see Note).</td></tr></table></td></tr>
<tr><td>Properties</td><td>This variable-size field encodes information specific to a layout class as follows:<br><table><tr><th><u>Layout Class</u></th><th><u>Description</u></th></tr><tr><td>Compact Storage</td><td>See Compact Storage Property Description for the version 3 Data Layout message.</td></tr><tr><td>Contiguous Storage</td><td>See Contiguous Storage Property Description for the version 3 Data Layout message.</td></tr><tr><td>Chunked Storage</td><td>See Chunked Storage Property Description below.</td></tr><tr><td>Virtual Storage</td><td>See Virtual Storage Property Description below.</td></tr><tr><td>Structured Chunk Storage</td><td>See Structured Chunk Storage Property Description below.</td></tr></table></td></tr>
</table>

Table 13 describes layout of the Structured Chunk Storage Property. The Property is similar to chunked storage property except it also has two new fields: “Structured Chunk Type” field to specify a type of structured chunk and “Properties” field to specify specific properties for the type. Currently, the section outlines only the properties for the structured chunks to support sparse data. Dimension numbering in the table is 0-based.

Note: It was suggested to call the new layout class “sectioned storage” to distinguished from chunk storage. Currently, we will be using chunk storage to store all sections that describe the data together, but If desired, sections can be stored separately. See section 3.1.2 for update on the Data Layout Message.

### Table 13: Structured Chunk Storage Property Description

<table>
<tr><th>byte</th><th>byte</th><th>byte</th><th>byte</th></tr>
<tr><td>Version</td><td colspan="2">Structured Chunk Type</td><td>This space inserted to align table nicely</td></tr>
<tr><td>Flags</td><td>Dimensionality</td><td>Dimension Size Encoding Length</td><td>This space inserted to align table nicely</td></tr>
<tr><td colspan="4">Dimension 0 Size (variable size)</td></tr>
<tr><td colspan="4">Dimension 1 Size (variable size)</td></tr>
<tr><td colspan="4">…</td></tr>
<tr><td colspan="4">Dimension #N-1 Size (variable size)</td></tr>
<tr><td>Chunk Indexing Type[7]</td><td colspan="3">This space inserted to align table nicely</td></tr>
<tr><td colspan="4">Indexing Type Information</td></tr>
<tr><td colspan="4">Address [O]</td></tr>
<tr><td colspan="4">Structured Chunk Composition (variable size)</td></tr>
</table>

Fields of Structured Chunk Property are described in Table 14.

[7] 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**

<table>
<tr><th>Field Name</th><th>Description</th></tr>
<tr><td>Version</td><td>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.</td></tr>
<tr><td>Structured Chunk Type</td><td>Type of the structured chunk[8]<br><table><tr><th><u>Bit</u></th><th><u>Description</u></th></tr><tr><td>0</td><td>Sparse Chunk Storage Property Description</td></tr><tr><td>1</td><td>HDF5 Variable-Length Chunk Storage Property Description</td></tr><tr><td>2-15</td><td>Reserved</td></tr></table></td></tr>
<tr><td>Flags</td><td>The same as in Chunked Storage Property Description</td></tr>
<tr><td>…</td><td>The same as in Chunked Storage Property Description</td></tr>
<tr><td>Address</td><td>The same as in Chunked Storage Property Description</td></tr>
<tr><td>Structured Chunk Composition</td><td>See Structured Chunk Composition Description</td></tr>
</table>

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

[8] *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 | |
| Number of First Section with Metadata | Number of Second Section with Metadata | … | 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.<br><br>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 in each structured chunk in the dataset.<br><br>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 which may contain metadata, and which therefore requires a checksum if non-empty.<br><br>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. |
| Number of first section with metadata | Number of the first section that may contain metadata. |
| Number of second section with metadata | Number of the second section that may contain metadata. |
| Number of last section with metadata | Number of the last section that may contain metadata. The total number of sections from *first* through *last* section will be equal to the *"Number of Sections containing metadata"* above. |

### 3.1.2 Updated Data Layout Message

This section is an addition to version 22 of this RFC to reflect our discussions on the storage of data that consists of several sections. Sparse and variable-length data and their corresponding metadata fall in this category. In the future we may consider sectioned storage for implementing columnar storage or extensible compound datatype datasets (i.e., when compound datatype can be updated to add or remove fields). Currently, we only support structured chunk storage. We will be using "sectioned" instead of "structured" as in the HTML File format document for this version [5].

#### Data Layout Message (Version 5)

<table>
<tr><th>byte</th><th>byte</th><th>byte</th><th>byte</th></tr>
<tr><td>Version</td><td>Layout Class</td><td colspan="2">This space inserted only to align table nicely</td></tr>
<tr><td colspan="4">Properties (variable size)</td></tr>
</table>

#### Fields: Data Layout Message (Version 5)

<table>
<tr><th>Field Name</th><th>Description</th></tr>
<tr><td>Version</td><td>The value for this field is 5. It is introduced for sectioned storage</td></tr>
<tr><td>Layout Class</td><td>The layout class specifies the type of storage for the data and how the other fields of the layout message are to be interpreted.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>Compact Storage</td></tr>
<tr><td>1</td><td>Contiguous Storage</td></tr>
<tr><td>2</td><td>Chunked Storage</td></tr>
<tr><td>3</td><td>Virtual Storage</td></tr>
<tr><td>4</td><td>Sectioned Storage</td></tr>
</table>
</td></tr>
<tr><td>Properties</td><td>This variable-size field encodes information specific to a layout class as follows:
<table>
<tr><th><u>Layout Class</u></th><th><u>Description</u></th></tr>
<tr><td>Compact Storage</td><td>See Compact Storage Property Description for the version 3 Data Layout message.</td></tr>
<tr><td>Contiguous Storage</td><td>See Contiguous Storage Property Description for the version 3 Data Layout message.</td></tr>
<tr><td>Chunked Storage</td><td>See Chunked Storage Property Description below.</td></tr>
<tr><td>Virtual Storage</td><td>See Virtual Storage Property Description below.</td></tr>
<tr><td>Sectioned Storage</td><td>See Sectioned Storage Property Description below.</td></tr>
</table>
</td></tr>
</table>

### Sectioned Storage Property Description

| **byte** | **byte** | **byte** | **byte** |
|---|---|---|---|
| Version | **Sectioned Data Type** | | **Sectioned Storage Type** |
| Number of Sections | Number of Sections Containing Metadata | This space inserted to align table nicely | |
| Number of First Section with Metadata | Number of Second Section with Metadata | ... | Number of Last Section with Metadata |
| Sectioned Storage Type Info (*variable size; depends on "Sectioned Storage Type"*) | | | |

### Fields: Sectioned Storage Property Description

<table>
<tr><th>Field Name</th><th>Description</th></tr>
<tr><td>Version</td><td>This is the version number for the Sectioned Storage property. The value for this field is 0.</td></tr>
<tr><td>Sectioned Data Type</td><td>Type of data that uses sectioned storage<br>
<table>
<tr><th><u>Bit</u></th><th><u>Description</u></th></tr>
<tr><td>0</td><td>Sparse</td></tr>
<tr><td>1</td><td>Variable-Length</td></tr>
<tr><td>2-15</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>Sectioned Storage Type</td><td>Integer from 0 to 255<br>
<table>
<tr><th><u>Value</u></th><th><u>Description</u></th></tr>
<tr><td>0</td><td>Chunked Sectioned Storage[9]</td></tr>
<tr><td>1-255</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>Number of sections</td><td>Number of sections in sectioned storage.</td></tr>
<tr><td>Number of sections containing metadata</td><td>Number of sections 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 Sectioned Data Type above. However, this need not always be the case.</td></tr>
<tr><td>Number of first section with metadata</td><td>Number of the first section that may contain metadata.</td></tr>
<tr><td>Number of second section with metadata</td><td>Number of the second section that may contain metadata.</td></tr>
<tr><td>Number of last section with metadata</td><td>Number of the last section that may contain metadata. The total number of sections from first through last section will be equal to the "Number of Sections containing metadata" above.</td></tr>
</table>

[9] Please notice that at this point we know that is "sectioned chunked data" and we will be using updated chunk indexing schemas we already discussed for "structured chunk".

### Chunked Sectioned Storage Type Info

| byte | byte | byte | byte |
|---|---|---|---|
| Offset Size (8 bytes) | | | |
| 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*) | | | |
| Chunk Indexing Type | This space inserted to align table nicely | | |
| Indexing Type Information | | | |
| Address [O] | | | |

### Fields: Chunked Sectioned Storage Type Data

| Field Name | Description |
|---|---|
| Offset size | Number of bytes used to store offsets in the structured chunk; currently it is 8 bytes. Please notice that sectioned chunk does not have 4GB limit on the chunk size as the “dense” chunk has due to API limitations. Present only for sectioned chunk storage. |
| *…..The rest of the fields are the same as for chunked storage* | |

### 3.1.3 Filtering of Sparse Data Arrays

HDF5 library allows to defined up to 32 filters to be applied to dense dataset’s data during I/O operations. We would like to extend this capability for datasets that use Structured Chunk Storage by applying a filter pipeline to each *section* of the Structured Chunk. For example, for sparse datasets we should anticipate scenarios where users may want to apply one set of filters (if any) to HDF5 dataspace selection and another set of filters to data elements. For variable-length data one would anticipate compression benefits of applying one compression method to the section that contains elements’ offset / length pairs, and another one to the section that contains variable-length values themselves. We discuss APIs to manage filtering of the datasets with structured chunk storage in [4].

Table 17 below shows version 0 of Structured Chunk Filter Pipeline Message.

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

| byte | byte | byte | byte |
|---|---|---|---|

| Version | Number of Filtered Sections | This space inserted to align table nicely |
|---|---|---|
| Number of First Filtered Section | Number of filters for First Filtered Section | Size of the first Filter Description List[10] |
| Filter Description List (variable size) | | |
| … | | |
| Number of Last Filtered Section | Number of filters for Last Filtered Section | Size of the Last Filter Description List |
| Filter Description List (variable size) | | |

**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 N of the filtered sections in structured chunk. |
| Number of First Filtered Section | Number of the first filtered section. |
| 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 [2] for details. |
| Number of Last Filtered Section | Number of the last filtered section. |
| 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 message. See [2] for details. |

[10] To make parsing easier

# 4 Structured Chunk Indexing

We will need to modify existing chunk indexing to accommodate non-filtered and filtered structured chunks metadata. Next sections discuss changes to for indexing schemas: Single Chunk Indexing, Fixed Array Indexing, Extensible Array Indexing and Version 2 B-tree Chunk Indexing. Implicit indexing cannot be used for structured chunk due to its variable-size nature. Version 1 B-tree Chunk Indexing is obsolete and is not considered for new features.

Throughout this section, we refer to both Structured Chunk Metadata and Filtered Structured Chunk Metadata as described in sections 2.1.1 and 2.2.1 correspondingly. While the size of this metadata is variable, it is a function of the number of sections – which is specified in the Data Layout message, and is constant within any dataset. All extensions are shown in **bold**.

## 4.1 Single Chunk Indexing[11]

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

**Table 19: Layout: Structured Chunk Indexing Information**

<table>
<tr><th>byte</th><th>byte</th><th>byte</th><th>byte</th></tr>
<tr><td colspan="4" align="center">Chunk Size (variable size; at most 8 bytes)</td></tr>
<tr><td colspan="4" align="center">Structured Chunk Metadata (variable)</td></tr>
</table>

**Table 20: Layout: Filtered Structured Chunk Indexing Information**

<table>
<tr><th>byte</th><th>byte</th><th>byte</th><th>byte</th></tr>
<tr><td colspan="4" align="center">Chunk Size (variable size; at most 8 bytes)</td></tr>
<tr><td colspan="4" align="center">Filtered Structured Chunk Metadata (variable)</td></tr>
</table>

## 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.

[11] 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[12] : Fixed Array Header Version 0**

<table>
<tr><td>Signature</td><td>No changes</td></tr>
<tr><td>Version</td><td>No changes</td></tr>
<tr><td>Client ID</td><td>The ID for identifying the client of the Fixed Array:<br>
<table>
<tr><th>ID</th><th>Description</th></tr>
<tr><td>0</td><td>Non-filtered dataset chunks</td></tr>
<tr><td>1</td><td>Filtered dataset chunks</td></tr>
<tr><td>2</td><td>Structured dataset chunks</td></tr>
<tr><td>3</td><td>Filtered structured dataset chunks</td></tr>
<tr><td>4+</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>Other fields</td><td>No changes</td></tr>
</table>

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

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

### 4.2.2 Changes to Fixed Array Data Block Fields

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

**Table 22: Fields: Fixed Array Data Block Version 0**

<table>
<tr><th>Field Name</th><th>Description</th></tr>
<tr><td>Signature</td><td>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.</td></tr>
<tr><td>Version</td><td>This document describes version 0.</td></tr>
<tr><td>Client ID</td><td>The ID for identifying the client of the Fixed Array:<br><table>
<tr><th>ID</th><th>Description</th></tr>
<tr><td>0</td><td>Non-filtered dataset chunks</td></tr>
<tr><td>1</td><td>Filtered dataset chunks</td></tr>
<tr><td>2</td><td>Structured dataset chunks</td></tr>
<tr><td>3</td><td>Filtered structured dataset chunks</td></tr>
<tr><td>4+</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>Header Address</td><td>The address of the Fixed Array header. Principally used for file integrity checking.</td></tr>
<tr><td>Page Bitmap</td><td>A bitmap indicating which data block pages are initialized.<br><br>Exists only if the data block is paged.</td></tr>
<tr><td>Elements</td><td>Contains the elements stored in the data block and exists only if the data block is not paged. There are four element types:<br><table>
<tr><th>ID</th><th>Description</th></tr>
<tr><td>0</td><td>Non-filtered dataset chunks</td></tr>
<tr><td>1</td><td>Filtered dataset chunks</td></tr>
<tr><td>2</td><td>Structured dataset chunks</td></tr>
<tr><td>3</td><td>Filtered structured dataset chunks</td></tr>
<tr><td>4+</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>Checksum</td><td>The checksum for the Fixed Array data block.</td></tr>
</table>

### 4.2.3 Changes to the layout of Fixed Array Data Block Page

Additions of new elements are shown in Table 23.

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

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

### 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.

**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**<br>(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.

**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**<br>(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**.

**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**<br>(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**<br>(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<br>1. Used Version 3 for Filter Pipeline Message instead of the new type of message that was only applicable to structured chunk storage.<br>2. Changed the order of the fields in Table 8 to follow FF practices<br>3. Clarified structured chunk section numbering and fixed it all over the document (section N-1 is the last N$^{th}$ 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.<br>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:<br>1. Changed the order in the fields of Table 9 to reflect the change in Table 8. |

| | |
|---|---|
| | 2. Reworked # 4 from June 5.<br>a. We decided to use “Structured Chunk Composition” instead of “Structured Chunk Type Properties”<br>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 3.1.2 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]. |

## References


1. The HDF Group, Draft RFC: Sparse Chunks, https://docs.hdfgroup.org/hdf5/rfc/RFC_Sparse_Chunks180830.pdf
2. The HDF Group, “HDF5 File Format Specification” https://www.hdfgroup.org/HDF5/doc/H5.format.html
3. The HDF Group, Variable-Length Data in HDF5 Sketch Design, https://docs.hdfgroup.org/hdf5/rfc/var_len_data_sketch_design_190715.pdf
4. John Mainzer, Elena Pourmal, “RFC: Programming Model to Support Sparse Data in HDF5”. Available from https://github.com/LifeboatLLC/SparseHDF5/
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

## Appendix

~~III.A.2. Disk Format: Level 1A2 - Version 2 B-trees:~~ (done, see changes to 4.4 in the above RFC)

The version # for the index data structures should not be incremented for this change. It's not necessary for adding a new record type. The version # for the data structure should only be changed for actual changes to the index data structure itself, not the records it contains, which are totally outside of its scope.

Bumping the version # for new records will cause unnecessary churn for third-party parsers and library code.

The new record types should just be added to the list in the existing header, and the new record types listed in the list of record types there. Older versions of the library will throw an error if they encounter a record type they don't understand. And, they should never try to interpret these records anyway, since they won't understand the new layout message version that points to indices with these kinds of records.

====================

~~IV.A.2.i. The Data Layout Message~~ (updated, see changes to section 3.1.1)

This text should be removed:

"Structured Chunk: This is only supported for
<a href="#DataLayoutMessageV5">version 5 of the Data Layout message.</i></a>
See <a href="#AppendixE">“Appendix E: Layout of Structured
Chunk”</a></mark>
for the layout of structured chunk.</li>"

Structured chunks are not a new kind of layout, they are a particular way of storing chunks.

====================

IV.A.2.i. The Data Layout Message (rejected)

Bumping the version # (to v5) is appropriate, but all the information about structured chunks should be moved into a new (versioned) set of information for the Chunked Storage Property Description. Having a "chunk type" in there would allow for encoding the (existing) dense chunk information as well as the new structured chunk type of information.

Basically, you are creating two sub-types of chunked storage: dense (unfiltered and filtered) and structured.

Comment: See new section 3.1.2.

====================

"Fields: Structured Chunk Storage Property Description" table

The "Structured Chunk Type" field appears to be a type, not a bitfield. And, each type needs an explanation of its assumptions. i.e. "Sparse structured chunks contain two sections. They are …" This must be explicit, so that third-party parsers can know what to expect in each structured chunk and in the chunk index records.

Comment: No, it is bit-field. 0 and 1 bits are set to indicate *sparse VL* storage.

====================

"Fields: Structured Chunk Composition Description" table

- Why is there a "Number of sections" field? Each type of structured *must* be described (earlier) and therefore the number of sections is known.

- Why are the sections with metadata listed out? If it's only because they "need a checksum", these fields can be eliminated, all the sections need checksums.

Comment: Redundance will not hurt. Format should be self-describing. This information allows to double check what is stored. We do document sparse and VL layout (it is done in IX, Appendix E.) We will clarify description of Fixed-Length section for VL structured chunks.

====================

v3 of the Filter Pipeline Message

This needs an explanation of how it will apply to existing (i.e. "dense" chunks). Each new version must be a superset of the previous version. (This version of the message could be created if an application chooses the "latest format" property for the file)

Comment: Not exactly. This version is intended to be used when structured chunk storage is used. If dense chunk is stored as a structured chunk, then it is clear how it is used, since there is only one section.

====================

VII.C. ~~The Fixed Array Index:~~ (done, see changes in section 4.2.1)

The version # for the index data structures should not be incremented for this change. (Same reason as B-trees)

The new record types should be added to the list in the existing header, and the new record types listed in the list of record types there.

====================

VII.D. ~~The Extensible Array Index: (~~done, see changes to 4.3)

The version # for the index data structures should not be incremented for this change.  (Same reason as B-trees)

The new record types should be added to the list in the existing header, and the new record types listed in the list of record types there.

====================

IX. Appendix E: Layout of Structured Chunk

- "Metadata used to interpret and find the data in a structured chunk."   Should add an explanation that this metadata is located in the index records, not the chunk itself.

- All the data sections need a checksum.   This is table-stakes for today's environment.
Comment: Disagree. User can add checksum to data, all metadata has checksum.

- Please mention (somewhere) that there is only one selection in the selection section for sparse storage.

- The format of encoded selections needs a description.  (There might be one you can refer to in the region reference datatype info, in this doc)
Comment: Agree, it is mentioned in another RFC. Let's provide a reference to VIII.A and version???? (it is a buffer that H5Sencode2 currently returns).

- The examples that describe chunks with VL-info need a description of the offset+length info for heap pointers.   And they need a description of the format of the heap section's information.
Comment: This spec is deferred since we are not implementing VL storage yet. We will do it after initial implementation is completed (resource problem).

====================

Please rebase your changes on top of the most recent version of the file format doc in the git repo.  It's difficult to diff your changes against the current document, since there have been many changes to the current specification doc since the version after you forked a copy.

====================