RFC: File Format Changes for Enabling Sparse Storage in HDF5

John Mainzer (john.mainzer@lifeboat.llc) ,
Elena Pourmal (elena.pourmal@lifeboat.llc)
Lifeboat, LLC

The document describes HDF5 file format extensions for storing sparse data.

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.

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.

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[1], 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[2] themselves, and if required one for a heap containing the variable-length data[3]. 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.

[1] *In this document we use 0-based numbering of the sections.*
[2] *Which may be of any HDF5 data type.*
[3] *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			
…			
Offset of section N			

Table 2: Fields: Structured Chunk Metadata

Field Name	Description
Offset of section 1	Offset in the structured chunk of the beginning of section 1. Note that the offset of section 0 is presumed to be zero, and is therefore not recorded.
Offset of section N	Offset in the structured chunk of the beginning of the N-th 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 (may not exist)			
…			
Section N (variable size - may be empty)			
Section N Checksum (may not exist)			

Table 4: Fields: Structured Chunk Layout

Field Name	Description
Section I, where I = 0,..,N	The data contained in the I-th section of the structured chunk. If the section is empty, the offset of the (I+1)-th section will be the same as that of the I-th section.
Section I Checksum, where I = 0,..,N	If section I may contain metadata, the section I-th 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			
Fixed Length Data Section (Section 1)			

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			
Fixed Length Data Section (Section 1)			
Section 1 Checksum			
Variable-size data heap (Section 2)			
Checksum for Variable-size data heap (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			
Variable-size data heap (Section 1)			
Checksum for Variable-size data heap (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[4] (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
Filter Masks (one per section)[5]			
Section Offsets (one per section less 1)			
Section Unfiltered Sizes (one per section)			

[4] *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.*
[5] *Filter Mask is 0 for non-filtered section*

Table 9: Fields: Filtered Structured Chunk Metadata

Field Name	Description
Filter Masks	One filter mask per section of the filtered structured chunk. If no pipeline is defined for the section, the Filter Mask is 0.
Sections Offsets	Offset in the Filtered Structured Chunk of each section. Since section 0 always starts at offset zero, the offset of section 0 is omitted.
Section Unfiltered Sizes	Unfiltered size of each section, or zero if the section is empty.

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 with Checksum (Section 0)			
Filtered Fixed Length Data with Checksum (Section 1)			
Filtered Variable Size Data Heap with Checksum (Section 2)			

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

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) is 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.
<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</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>Layout Class</th><th>Description</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.

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 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">Properties (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>Version number of Structured Chunk Property Description</td></tr>
<tr><td>Structured Chunk Type</td><td>Type of the structured chunk[8]
<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>Properties</td><td>This variable-size field encodes information specific to a Structured Chunk Type as follows:
<table>
<tr><th>Layout Type</th><th>Description</th></tr>
<tr><td>Sparse Chunk</td><td>See Sparse Chunk Property Description</td></tr>
<tr><td>Future: Variable-Length Type Data</td><td>See Variable-Length Data Property Description</td></tr>
<tr><td>Future: …</td><td>See … Property Description</td></tr>
<tr><td></td><td></td></tr>
</table></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 Property Description

byte	byte	byte	byte
Offset Size	Number of Sections	Number of Sections Containing Metadata	This space inserted to align table nicely
Number of First Section with Metadata	…	Number of N-th Section with Metadata	This space inserted to align table nicely

Table 16: Fields: Structured Chunk Property Description

Field Name	Description
Offset size	Number of bytes used to store offsets in the structured chunk; currently it is 4 bytes due to API limitation; HDF5 File Format allows chunk size bigger than 4GB.
Number of Sections	Number of sections 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 which may contain metadata, and which therefore require 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.
Number of First Section with Metadata	Number of the first section that may contain metadata.
Number of N-th Section with Metadata	Number of the N-th section that may contain metadata, where N equals the "Number of Sections Containing Metadata" above.

3.1.2 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.

Table 17: Structured Chunk Filter Pipeline Message – Version 0

Version 0 of Structured Chunk 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).

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[9]	
Filter Description List (variable size)			
…			
Number of N-th Filtered Section	Number of filters for N-th Filtered Section	Size of the N-th Filter Description List	
Filter Description List (variable size)			

Table 18: Fields: Structured Chunk Filter Pipeline Message

Field Name	Description
Version	The version number of this message; currently 0.
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 filtered section. 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. A filter description as it appears in Filter Pipeline Message Version 2. See [2] for details.
Number of N-th Filtered Section	Number of the N-th filtered section.

[9] To make parsing easier

Field Name	Description
Number of filters for N-th Filtered Section	The total number of filters specified for the N-th filtered section. The maximum possible number of filters in a message is 32.
Size of the N-th Filter Description List	Size of the N-th Filter Description List in bytes.
Filter Description List	A description of each filter. A filter description as it appears in Filter Pipeline Message Version 2. See [2] for details.

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

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

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 version is increased to 1 and new clients IDs were added as shown in **Table 21**.

Table 21: Fields[10] : Fixed Array Header Version 1

<table>
<tr><td>Version</td><td>This document describes version 1.</td></tr>
<tr><td>Client ID</td><td>The ID for identifying the client of the Fixed Array:
<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>
</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.

[10] *We do not provide the whole table to save space in this document since the rest of the fields are the same as for Version 0.*

4.2.2 Changes to Fixed Array Data Block Fields

The version is increased to 1 and new clients IDs and Elements were added as shown in Table 22.

Table 22: Fields: Fixed Array Data Block Version 1

<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 1.</td></tr>
<tr><td>Client ID</td><td>The ID for identifying the client of the Fixed Array:
<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.

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:
<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

New elements were added as shown in Table 23.

Table 23: Fields: Fixed Array Data Block Page Version 1

<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 (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 (variable size, but fixed within any one index)			

4.3 Extensible Array Indexing Information

As with the Fixed Array Index, we will need to increase the versions of the Extensible Array Header, Index Block, Secondary Block, Data Block and add 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:

- Bump a version of Version 2 B-tree Header from 0 to 1 and 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 (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 just extended Type 11 Record Layout with the Filtered Structured Chunk metadata info.

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.

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/