# RFC: File Format Changes in HDF5 1.10.0

**The HDF Group**

This document gives an overview of the file format changes in HDF5 1.10.0.

The first version of this document contains the proposed changes to the superblock.

When finished, it should contain the overview of all recommended changes to the HDF5 file format that would be implemented by the HDF5 libraries version 1.10. The final document can be used to inform the HDF5 users about the file format changes in 1.10 and serve as a high-level checklist for the changes to the HDF5 File Format Specification document 1.10.0.

## Introduction

The HDF5 version 1.10.0 will introduce several new features and bug fixes that require extensions or modifications to the HDF5 File Format implemented by the HDF5 libraries version 1.8 [1].

Every new feature that required a file format change, implemented the change independently of the changes done for other features. We need to take a look at all proposed file format changes and finalize them. The intent of this document is to summarize all proposed changes and to finalize the HDF5 file format that will be implemented by the HDF5 library versions 1.10.

The document is organized as follows. Section 2 of the document gives an overview of the changes proposed to the superblock and its extensions, File Space Management, Avoid Truncate, and Cache Image features. Section 3 documents the changes needed to introduce new chunk indexing structures. Section 4 describes the file format changes required for VDS.

Recommended final changes are documented in Section 5 and will be documented in the HDF5 File Format Specification version 1.10.0 (<<<location TBD>>>) when approved.

## HDF5 File Format changes to Superblock

This section discusses proposed or currently implemented changes to superblock for File Locking with/without SWMR, File Free Space Management, Avoid Truncate and Cache Image features.

### File Locking

The SWMR enabled library (version 1.10) implements file locking to ensure file consistency. It uses the *file consistency flags* field in the superblock (*status_flags* in *H5F_super_t* structure) as part of the mechanism to lock the file, which can be open with or without SWMR access. Please see *RFC: File Locking Under SWMR – Semantics*, *Programming Model*, *and Implementation* for details.

The *file consistency flags* field in the superblock is:

- For superblock version 0 & 1: a 4-byte field starting at byte 20th
- For superblock version 2: a 1-byte field at byte 11th

### SWMR backward compatibility issue

It was brought to The HDF Group developers' attention that the SWMR enabled HDF5 library cannot open some HDF5 files created by HDF5 1.8. The issue was reported and documented in JIRA SWMR-79. Behavior of the library violates The HDF Group backward compatibility policy that requires any new version of the HDF5 library to read the files created by the previous versions of HDF5.

Performed investigation showed that the earlier versions of HDF5 1.8 may accidently write garbage to the *status_flags* field in the superblock. The values stored in *status_flags* are used by the HDF5 1.10 library to verify that a file can be opened for a specified access. The issue was fixed in the later versions of HDF5 1.8 and The HDF Group provides the *h5clear* tool to fix the values stored in the garbled field, but this solution requires user's interference that is not always possible. The HDF Group was asked to provide a solution that would be transparent to the applications.

#### Proposed change to address SWMR-79

We propose to bump the version of the superblock to 3 when the HDF5 library version 1.10 creates a file with the latest format. Any other HDF5-based process that opens the file will behave according to semantics described in the *RFC: File Locking Under SWMR – Semantics*, *Programming Model*, *and Implementation*. The library will ignore values in *status_flags* when the version of the superblock is less than 3.

#### *The current implementation of superblock version*

The library determines the superblock version # to use based on whether the file is created with or without the latest format.

A. When a file is created without the latest format, the library will determine the superblock version # based on the file access flags:

a) non-SWMR file access:

The library uses version 0 by default. But it will bump the version # based on the existence of the following file creation properties:

1) If non-default v1 B-tree K value is set, the version is set to 1.
2) If shared object header message index (SOHM) is enabled, the version is set to 2.
3) If non-default file space info is set, the version is set to 2.

The library will create the superblock extension to store the messages for items (1) to (3) above.

b) SWMR file access: file creation will fail

***File without latest format***

| *non-SWMR file access* | | | | *SWMR file access* |
|---|---|---|---|---|
| -- | *non-default v1 B-tree K value* | *SOHM* | *non-default file space info* | |
| v. 0 | v. 1* | v. 2* | v. 2* | fail |

B. When a file is created with the latest format, the library sets superblock version to 2.

***File with latest format***

| *non-SWMR/SWMR file access* | | | |
|---|---|---|---|
| -- | *non-default v1 B-tree K value* | *SOHM* | *non-default file space info* |
| v. 2 | v. 2* | v. 2* | v. 2* |

*indicates superblock extension is used to store the message information

***The proposed implementation of superblock version***

The library determines the superblock version # to use based on whether the file is created with or without the latest format.

A. When a file is created without the latest format, the implementation will be the same as case #A described in the previous section.
B. When a file is created with the latest format, the library will set superblock version to 3.

***File with latest format***

| *non-SWMR/SWMR file access* | | | |
|---|---|---|---|
| -- | *non-default v1 B-tree K value* | *SOHM* | *non-default file space info* |

| | | | |
|---|---|---|---|
| v. 3 | v. 3* | v. 3* | v. 3* |

**Alternative change to address SWMR-79**

No alternative solutions were proposed.

### File Space Management

The HDF5 library performs file space management activities such as tracking free space and allocating space to store file metadata and raw data. The library provides 3 file space management strategies based on 4 mechanisms used to allocate space. Please refer to the <*File Space Management User Guide*—TO BE UPDATED> for detailed description.

The mechanisms are:

- Free-space managers
- Free-space managers with embedded paged aggregation
- Aggregators
- Virtual file driver

The strategies are:

- H5F_FSPACE_STRATEGY_AGGR
    - The mechanisms used are free-space managers, aggregators and virtual file driver
    - This is the library default
- H5F_FSPACE_STRATEGY_PAGE
    - The mechanisms used are free-space managers with embedded paged aggregation and virtual file driver
- H5F_FSPACE_STRATEGY_NONE
    - The mechanisms are free-space managers and virtual file driver

To support this feature, the library stores the file space handling information in the *File Space Info* message, which is located in the superblock extension.

#### Implemented change for File Space Management

The information stored in the *File Space Info* message is listed below:

- Message version (1 byte)
    - Version is 0
- File space strategy (1 byte)
    - The strategies are:
        - H5F_FSPACE_STRATEGY_AGGR
        - H5F_FSPACE_STRATEGY_PAGE
        - H5F_FSPACE_STRATEGY_NONE
- Persisting free-space (1 byte)
- Free-space section threshold (*size of lengths*)

- For paged aggregation: file space page size (*size of lengths*)
- For paged aggregation: page end metadata threshold (2 bytes)
- For paged aggregation: EOF file space section type (1 byte)
- Addresses of 6 free-space managers (*size of offsets*)
    - Exist only when persisting free-space
    - For paged aggregation: only 3 managers will be defined

Please refer to the *HDF5 format specification* (TO BE UPDATED) for detailed description of the message.

**Alternative change for File Space Management**

***PENDING***: The library's default file space strategy might be changed to H5F_FSPACE_STRATEGY_PAGE depending on the performance result for paged aggregation/page buffering. If that is the case, there might be changes to superblock version 3 to store the needed information for paged aggregation.

## Avoid Truncate Feature

The HDF5 library tracks two pieces of information about the size of an HDF5 file in memory. The "end of allocation" (EOA) value indicates how much of the file has been allocated for use by some piece of the HDF5 file format. The "end of file" (EOF) value indicates the location of the highest byte actually written in the file by the HDF5 library. These two values are frequently not the same during normal operation of the library. Currently, the library changes the file's size from its current size (EOF) to the EOA value before setting the EOF value to the EOA value and stores the [modified] EOF value in the file's superblock.

As setting the file's size is fairly expensive, this feature allows the library to not change the file's size by storing the EOA value along with the unmodified EOF value in the superblock. To support this feature, the library stores the file's EOA information in an *EOA* message, which is located in the superblock extension.

Please refer to <<*Avoid Truncate documentation*>> for detailed description.

### Implemented change for Avoid Truncate

The information stored in the *EOA* message is listed below:

- Message version (1 byte)
  - Version is 0
- Avoid truncate setting (1 byte)
  - The settings are:
    - H5F_AVOID_TRUNCATE_OFF
    - H5F_AVOID_TRUNCATE_EXTEND
    - H5F_AVOID_TRUNCATE_ALL
- EOA value (*size of offsets*)
  - End of file addresses for up to 6 basic allocation types:

1. H5FD_MEM_SUPER (superblock data)
2. H5FD_MEM_BTREE (B-tree data)
3. H5FD_MEM_DRAW (raw data)
4. H5FD_MEM_GHEAP (global heap data)
5. H5FD_MEM_LHEAP (local heap data)
6. H5FD_MEM_OHDR (object header data)

Please refer to the *HDF5 format specification* (TO BE UPDATED) for detailed description of the message.

### Alternative change for Avoid Truncate

***PENDING***: As we are adding superblock version 3, we might revisit the implementation of this feature to store the EOA value (sec2) in version 3 superblock. For file drivers with multiple file backend, we might consider putting the EOA values in the *Driver Info* message in the superblock extension.

## Cache Image Feature

When this feature is enabled, the library saves the image of the metadata cache at file closing. When the file is re-opened, the library reads the saved cache image, decodes and loads its contents into the metadata cache. Please refer to the <*Cache Image documentation*> for detailed description.

To support this feature, the library stores the information about the saved image in a *Metadata Cache Image* message, which is located in the superblock extension.

### Implemented change for Cache Image

The information stored in the *Metadata Cache Image* message is listed below:

- Message version (1 byte)
  - Version is 0
- Address of the cache image block (*size of offsets*)
- Length of the cache image block (*size of lengths*)

Please refer to the *HDF5 format specification* (TO BE UPDATED) for detailed description of the message.

### Alternative change for Cache Image

***PENDING***: When the implementation of the cache image feature is finalized, there might be changes to the superblock version 3 on EOA/EOF??.

## HDF5 File Format changes to support new chunk indexing

Currently, the library uses version 1 B-tree to index chunked datasets in an HDF5 file with or without the latest format.

For chunked datasets in an HDF5 file with the latest format, the 1.10 library will use one of the following indexing types depending on a chunked dataset's dimension specification and the way it is extended:

- *Extensible Array* indexing for appending along a specified dimension
- *Version 2 B-tree* indexing for appending along multiple dimensions
- *Fixed Array* indexing for fixed-size datasets
- *Implicit* indexing for fixed-size datasets with early space allocation and without filters

Please refer to the *HDF5 format specification* (TO BE UPDATED) for detailed description of the indexing types.

To support these chunk indexing types, the library uses a pair of messages to describe the dataset layout information in the object header:

- Version 4 *Data Layout* message
- Version 0 *Data Storage* message

### Implemented Change for Data Layout Message

The information stored in the version 4 *Data Layout* message is listed below:

- Message version (1 byte)
    - Version is 4
- Layout class (1 byte)
    - The classes are: *compact*, *contiguous*, *chunked*
- Properties specific to each layout class (variable size)
    - Contains the following fields:
        - *Compact*: none
        - *Contiguous:* none
        - *Chunked*:
            - Chunked storage enabled flag (1 byte)
            - Dimensionality (1 byte)
            - Encoded # of bytes for chunk dimensions (1 byte)
            - N dimension sizes (variable size)

        - Chunk indexing type (1 byte):
            - 0—*Version 1 B-tree indexing*
            - 1—*Implicit indexing*
            - 2—*Fixed Array indexing*
            - 3—*Extensible Array indexing*
            - 4—*Version 2 B-tree indexing*
        - Creation parameters information specific to an indexing type (variable size):
            - *Implicit indexing*: none
            - *Fixed Array indexing*: 1 byte
            - *Extensible Array indexing*: 5 bytes
            - *Version 2 B-tree indexing*: 6 bytes

Please refer to the *HDF5 format specification* (TO BE UPDATED) for detailed description of the message.

### Implemented change for Data Storage message

The information stored in the *Data Storage* message is listed below:

- Message version (1 byte)
    - Version is 0
- Layout class (1 byte)
    - The classes are: *compact*, *contiguous*, *chunked*
- Properties specific to each layout class (variable size)
    - Contains the following fields:
        - *Compact*:
            - Size of the raw data (2 bytes)
            - The raw data (variable size)
        - *Contiguous:*
            - Address where the raw data is located (*size of offsets*)
            - Size of the raw data (*size of lengths*)
        - *Chunked*:
            - Chunked storage enabled flag (1 byte)
            - Chunk indexing type (1 byte)
                - 0—*Version 1 B-tree indexing*

    - 1—*Implicit indexing*
    - 2—*Fixed Array indexing*
    - 3—*Extensible Array indexing*
    - 4—*Version 2 B-tree indexing*
- Address (*size of offsets*) specific to an indexing type:
    - *Implicit* indexing: address of the dataset chunks
    - *Fixed Array/Extensible Array/Version 2 B-tree* indexing: address where the indexing information is located; address may be undefined if storage information for the indexing type is not allocated yet

Please refer to the *HDF5 format specification* (TO BE UPDATED) for detailed description of the message.

### Alternative change for Data Layout/Data Storage messages

***PENDING***: We need to evaluate and decide whether to just use the *data layout* message or *data layout/storage* message pair. For either case, layout/storage information to support new chunk indexing and VDS will need to be merged.

## HDF5 File Format changes to support VDS

The VDS feature (Virtual Dataset) allows users to manage data stored across a collection of the HDF5 files in a similar way as if data was stored in a dataset in an HDF5 file. It provides a mapping from source dataset elements in some source HDF5 files to a set of elements in the VDS. The library stores the mapping information in the file's global heap. The Please refer to the <*VDS documentation*> for detailed description.

To support this feature, the library modifies the version 4 *Data Layout* message to store the global heap ID, which is used to locate the global heap collection containing the VDS mapping information.

### Implemented Change for Data Layout Message

The information stored in the version 4 *Data Layout* message is listed below:

- Message version (1 byte)
    - Version is 4
- Layout class (1 byte)
    - The classes are: *compact*, *contiguous*, *chunked*, *virtual*
- Properties specific to each layout class (variable size)
    - Contains the following fields:
        - *Compact*:
            - Size of the raw data (2 bytes)
            - The raw data (variable size)
        - *Contiguous:*
            - Address of the raw data (*size of offsets*)
            - Size of the raw data (*size of lengths*)
        - *Chunked*:
            - Dimensionality (1 byte)
            - Address where the *Version 1 B-tree* indexing information is located (*size of offsets*)
            - N dimension sizes (4 bytes for each dimension)
        - *Virtual*
            - Address of the global heap collection where the VDS mapping entries are stored (*size of offsets*)
            - Index of the data object within the global heap collection (4 bytes)

Please refer to the *HDF5 format specification* (TO BE UPDATED) for detailed description of the

message.

### Alternative Change for Data Layout Message

***PENDING***: We need to evaluate and decide whether to just use the *data layout* message or *data layout/storage* message pair. For either case, layout/storage information to support new chunk indexing and VDS will need to be merged.

## Final recommendation: HDF5 File Format changes to be implemented by HDF5 1.10

Summary of the recommended changes will be here

## Revision History

*September 15, 2015:* Version 1 sent to authors to fill their sections for file format changes and extensions.

*September 28, 2015:* Version 2 sent for internal review.

## References


1. The HDF Group, “HDF5 File Format Specification” https://www.hdfgroup.org/HDF5/doc/H5.format.html
2. *RFC: File Locking Under SWMR—Semantics, Programming Model, and Implementation*
3. *<File Space Management User Guide>*
4. *<Avoid Truncate documentation>*
5. *<Cache Image documentation>*
6. *<VDS documentation>*