RFC: File Format Changes in HDF5 1.10.0 and Recommendations for the File Format Changes in the Future Releases

The HDF Group

This document gives an overview of the proposed file format changes in HDF5 1.10.0 and gives recommendations on the file format changes that should be done to integrate HPC features in the 1.10.1 release. It also discusses future enhancements to the file format for the next major release of HDF5 (aka 1.12.0)

1 Introduction 4

2 HDF5 File Format Changes to the Superblock 5

2.1 File Locking 5

2.2 SWMR Backward Compatibility Issue 5

2.2.1 Proposed Change to Address SWMR-79 5

2.2.2 Alternative Change to Address SWMR-79 7

2.2.3 Final Recommendation 7

2.3 File Space Management 8

2.3.1 Proposed Format Change for File Space Management 8

2.3.2 Alternative Change for File Space Management 9

2.3.3 Final Recommendation 9

2.4 Avoid Truncate 10

2.4.1 Proposed Format Change for Avoid Truncate 10

2.4.2 Alternate Change for Avoid Truncate 10

2.4.3 Final Recommendation 11

2.5 Cache Image 12

2.5.1 Proposed Format Change for Cache Image 12

2.5.2 Alternative Change for Cache Image 12

2.5.3 Final Recommendation 12

3 HDF5 File Format Changes to Support New Chunk Indexing 13

3.1 Proposed Format Change for the Data Layout Message 13

3.2 Alternative Change for the Data Layout Message 14

3.3 Final Recommendation 15

4 HDF5 File Format Changes to Support VDS 16

4.1 Proposed Format Change for the Data Layout Message 16

4.2 Alternative Change for Data Layout Message 16

4.3 Final Recommendation 16

Revision History 17

References 18

# 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 1.8 version of the HDF5 Library[1].

Every new feature that required a file format change implemented the change independently of the changes done for other features. The intent of this document is to summarize all proposed changes and to finalize the HDF5 file format that will be implemented by the 1.10 version of the HDF5 Library.

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 the virtual dataset (VDS) feature.

Recommended final changes are documented in Section 5 and will be added to the *HDF5 File Format Specification* version 1.10.0 when approved. The *HDF5 File Format Specification* document is published on [The HDF Group](https://www.hdfgroup.org/) website at the time of each release.

# HDF5 File Format Changes to the Superblock

This section discusses proposed or currently implemented changes to the superblock for file locking with or without the single-writer multiple-reader (SWMR), file free space management, avoid truncate, and cache image features.

## File Locking

The 1.10 library implements file locking ensuring 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 and 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. This behavior of the library violates The HDF Group backward compatibility policy that requires any new version of the HDF5 library to read files created by previous versions of HDF5.

The investigation showed that the earlier versions of HDF5 1.8 may accidently write garbage to the status\_flags field in the superblock. The HDF5 1.10 Library uses values stored in status\_flags to verify that a file can be opened for a specified access. The issue was fixed in the later versions of HDF5 1.8[[1]](#footnote-1), and The HDF Group now provides the h5clear tool to fix the values stored in the garbled field. Since this solution requires action by the user that is not always possible, The HDF Group was asked to provide a solution that would be transparent to 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 the Superblock Version

The library determines the superblock version number to use based on whether the file is created with or without the latest format[[2]](#footnote-2).

1. When a file is created **without the latest format**, the library will determine the superblock version number based on the file access flags:
   1. Write access:

The library uses version 0 by default, but it will bump the version number based on the existence of the following file creation properties:

1. If a non-default v1 B-tree K value is set, the version is set to 1.
2. If the shared object header message index (SOHM) is enabled, the version is set to 2.
3. If a non-default file space info value is set, the version is set to 2.
   1. SWMR write access: file creation will fail

File without latest format

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Write Access* | | | | *SWMR Write Access* |
| *--* | *Non-default v1 B-tree K value* | *SOHM* | *Non-default file space info* |
| v. 0 | v. 1+ | v. 2\* | v. 2\* | fail |

1. When a file is created **with the latest format**, the library sets superblock version to 2 and activates the latest version support.

File with latest format

|  |  |  |  |
| --- | --- | --- | --- |
| *Write Access/SWMR Write Access* | | | |
| *--* | *Non-default v1 B-tree K value* | *SOHM* | *Non-default file space info* |
| v. 2 | v. 2+ | v. 2\* | v. 2\* |

\*The message will be stored in the superblock extension.

+The message will either be stored in the superblock or superblock extension.

#### The Proposed Format Change of the Superblock Version

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

1. When a file is created **without the latest format**, the implementation will be similar to case A described in the previous section. The difference is when the file is created with SWMR write access: file creation succeeds, and the library will set the superblock version to 3 and will activate the latest version support.

File without latest format

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Write Access* | | | | *SWMR Write Access* |
| *--* | *Non-default v1 B-tree K value* | *SOHM* | *Non-default file space info* |
| v. 0 | v. 1+ | v. 2\* | v. 2\* | v. 3\* |

1. When a file is created **with the latest format**, the library will set the superblock version to 3 and will activate the latest version support.

File with latest format

|  |  |  |  |
| --- | --- | --- | --- |
| *Write Access/SWMR Write Access* | | | |
| *--* | *Non-default v1 B-tree K value* | *SOHM* | *Non-default file space info* |
| v. 3 | v. 3+ | v. 3\* | v. 3\* |

The File Consistency Flags field in the superblock is:

* For superblock version 0 and 1: a 4-byte field starting at byte 20th
* For superblock version 2-3: a 1-byte field at byte 11th

\*The message will be stored in the superblock extension.

+The message will either be stored in the superblock or superblock extension.

### Alternative Change to Address SWMR-79

No alternative solutions were proposed.

### Final Recommendation

The solution proposed above was implemented in the *revise\_chunks* branch and released in HDF5 1.10.0-alpha0 on December 30, 2015.

The HDF5 format specification will be updated for the proposed format change.

## 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 proposes three file space management strategies based on four mechanisms used to allocate space. See the not yet completed *File Space Management User Guide* at https://svn.hdfgroup.org/hdf5doc/trunk/projects/1\_10\_alpha/FileSpaceManagement. > for more information.

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 used are free-space managers and virtual file driver

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

### Proposed Format 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

### Alternative Change for File Space Management

The following will be re-evaluated later for release 1.12:

***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 the superblock to store the needed information for paged aggregation.

### Final Recommendation

This will be implemented for library release 1.10.1.

Library release 1.10.0 has already implemented a *File Space Info* message but with different file space information. The following needs to be done:

* Make the necessary modifications in library release 1.10.0 to ignore unknown messages.
* Do not publish documentation for file space management.

## Avoid Truncate

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 the 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 will store the file’s EOA information in an *EOA* message, which is located in the superblock extension.

See the as yet unwritten <<*Avoid Truncate documentation*>> for more information.

### Proposed Format 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)

### Alternate Change for Avoid Truncate

The following will be re-evaluated later for release 1.12:

***PENDING***: We might consider storing the EOA value (sec2) in the superblock. For file drivers with multiple file backend, we might put the EOA values in the *Driver Info* message in the superblock extension.

### Final Recommendation

This will be implemented for library release 1.10.1.

## Cache Image

The Cache Image feature allows the library to save the image of the metadata cache at file closing. When the file is re-opened, the library reads the saved cache image, decodes the image, and then loads the contents into the metadata cache. See the as yet unwritten <*Cache Image documentation*> for more information.

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

### Proposed Format 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*)

### Alternative Change for Cache Image

The following will be evaluated for release 1.12:

***PENDING***: We will consider including this feature or not. If so, there might be changes to the superblock on EOA/EOF.

### Final Recommendation

This will be evaluated for library release 1.12.

# HDF5 File Format Changes to Support New Chunk Indexing

Currently, the library uses version 1 B-trees 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
* *Single Chunk* indexing for datasets with only 1 chunk (in other words, the current, maximum, and chunk dimension sizes are the same)

To support these new chunk indexing types, the library will use the version 4 *Data Layout* message to describe the dataset layout information in the object header.

## Proposed Format Change for the 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 where the raw data is located (*size of offsets*)
* Size of the raw data (*size of lengths*)
* *Chunked*:
  + Chunked layout feature flag (1 byte)
    - H5O\_LAYOUT\_CHUNK\_DONT\_FILTER\_PARTIAL\_BOUND\_CHUNKS
      * Do not apply filter to a partial edge chunk
    - H5O\_LAYOUT\_CHUNK\_SINGLE\_INDEX\_WITH\_FILTER
      * A filtered chunk for Single Chunk indexing
  + Dimensionality (1 byte)
  + Encoded number of bytes for chunk dimensions (1 byte)
  + N dimension sizes (variable size)
  + Chunk indexing type (1 byte):
    - 0—*Version 1 B-tree indexing: should not be in version 4 layout*
    - 1— *Single Chunk indexing*
    - 2*—Implicit indexing*
    - 3—*Fixed Array indexing*
    - 4—*Extensible Array indexing*
    - 5—*Version 2 B-tree indexing*
  + Creation parameters information specific to an indexing type (variable size):
    - *Single Chunk indexing:*
      * Filtered info if it is a filtered chunk*: size of lengths* + 4 bytes
    - *Implicit indexing*: none
    - *Fixed Array indexing*: 1 byte
    - *Extensible Array indexing*: 5 bytes
    - *Version 2 B-tree indexing*: 6 bytes
  + Address (*size of offsets*) specific to an indexing type:
    - *Single Chunk* indexing: address of the single chunk; address may be undefined if the chunk is not allocated yet
    - *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
* *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)

## Alternative Change for the Data Layout Message

No alternative solutions were proposed.

## Final Recommendation

The solution proposed above was implemented in the *revise\_chunks* branch and released in HDF5 1.10.0-alpha0 on December 30, 2015.

The HDF5 format specification will be updated for the proposed format change.

# HDF5 File Format Changes to Support VDS

The VDS feature (Virtual Dataset) allows users to manage data stored across a collection of HDF5 files in a similar way as if the 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. See the as yet unwritten <*VDS documentation*> for for information.

To support this feature, the library will modify 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.

## Proposed Format Change for the Data Layout Message

See the description of the version 4 *Data Layout* message in the “Proposed Format Change for the Data Layout Message” section on page 12.

## Alternative Change for Data Layout Message

No alternative solutions were proposed.

## Final Recommendation

The solution proposed above was implemented in the *revise\_chunks* branch and released in HDF5 1.10.0-alpha0 on December 30, 2015.

The HDF5 format specification will be updated for the proposed format change.

# 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. |
| February 12, 2016: | Version 3 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*>

1. The exact version that fixed garbled superblock fields has to be found yet. [↑](#footnote-ref-1)
2. *Under “latest format” we mean 1.10.0 format.* We need to implement library versioning to distinguish “1.8 latest” format from “1.10 latest” format. This task was pushed to the 1.10.1 release. [↑](#footnote-ref-2)