RFC: Setting Bounds for Object Creation in HDF5 1.10.0

The HDF Group

The public routine *H5Pset_libver_bounds()* sets the low and high bounds on library release versions to be used when creating objects. The current implementation supports only two combinations of low/high bounds for this routine.

This RFC proposes additions to the enumerated define for low/high bounds so as to allow user flexibility in setting bounds for object creation.

Introduction

The public routine *H5Pset_libver_bounds(*hid_t *fapl_id*, H5F_libver_t *low*, H5F_libver_t *high)* sets the *low* and *high* bounds for the libver_bounds property in the file access property list *fapl_id*. The bounds are library release versions, which indirectly determines the format versions that the library will use when creating objects in the file.

Currently, the library defines two values for *H5F_libver_t* and supports two pairs of combinations via *H5Pset_libver_bounds()*.

This RFC proposes additions to the enumerated define for *H5F_libver_t*, which results in six possible pairs of *(low, high)* combinations. This document describes the changes in the library to provide such versioning support.

Terms used

The following terms are abbreviated as listed below for convenience and they are used interchangeably in the document:

a) H5F_LIBVER_EARLIEST *earliest*

b) H5F_LIBVER_V18 *v18*

c) H5F_LIBVER_V110 *v110*

d) H5F_LIBVER_LATEST *latest*

e) File access property list *fapl*

f) File creation property list *fcpl*

Current implementation

The public routine *H5Pset_libver_bounds* sets the bounds for the libver_bounds property in a file access property list, indicating the object creation behavior set for an existing file, and can vary according to application needs each time a file is opened. The setting to this routine will not affect the reading and writing of existing objects, but only the creation of new objects in the file.

The two parameters to the routine, *low* and *high*, control the range of library release versions that the library will use to create objects in the file. The parameter *low* sets the earliest format versions that the HDF5 library will use when creating objects in the file. Note that *earliest possible* is different from *earliest*, as some features introduced in library versions later than 1.0.0 resulted in updates to object formats. The parameter *high* sets the latest format versions that the library will be allowed to use when creating objects in the file.

The enumerated values for *H5F_libver_t* are:

```
typedef enum H5F_libver_t {
   H5F_LIBVER_EARLIEST,
   H5F_LIBVER_LATEST
} H5F_libver_t;
```

The value H5F_LIBVER_EARLIEST indicates the earliest possible format versions while H5F_LIBVER_LATEST indicates the latest format versions available.

The library supports only two pairs of (*low*, *high*) combinations as derived from the above values:

1. *low* = H5F_LIBVER_EARLIEST, *high* = H5F_LIBVER_LATEST
   - The library will create objects with the earliest possible format versions.
   - The library will allow objects to be created with the latest format versions available and there is no upper limit on the format versions to use. For example, if a newer format version is required to support a feature, this setting will allow the object to be created.
   - This is the library default setting and provides the greatest format compatibility.
2. *low* = H5F_LIBVER_LATEST, *high* = H5F_LIBVER_LATEST
   - The library will create objects using the latest format versions available.
   - This setting will allow users to take advantage of the latest features and performance enhancements in the library. However, earlier versions of the library may not be able to access objects created with this setting.

Use cases

As can be seen, the current implementation for *H5Pset_libver_bounds* is rather restricted. The following use cases will benefit from the proposed additions to *H5Pset_libver_bounds* as described in the next section:

- A user wants to use HDF5 1.10.x library to create a file that is compatible with 1.8.x library release.
- A user wants to use HDF5 1.10.x library to create a file that uses the latest features available to 1.8.x library release like compact-or-indexed link storage.
- A user wants to use HDF5 1.10.x library to create a file that uses at least the latest features available to 1.8.x library release. At the same time, the user also wants to use the latest features available to 1.10.x library release like dataset with the latest chunk indexing, no-filter-

partial-bound-chunks, or virtual dataset.

- A user wants to use the very latest features in HDF5 1.10.x library release.

Public Routines and Data Structures

0.1 *H5F_libver_t* enumerated values:

```
typedef enum H5F_libver_t {
        H5F_LIBVER_EARLIEST = 0,
        H5F_LIBVER_V18 = 1,
        H5F_LIBVER_V110 = 2,
        H5F_LIBVER_NBOUNDS
} H5F_libver_t;

#define H5F_LIBVER_LATEST  H5F_LIBVER_V110
```

The two new values added to *H5F_libver_t* are H5F_LIBVER_V18 and H5F_LIBVER_V110. They indicate that the library will create objects with the format versions available in library releases 1.8.x and 1.10.x respectively. We also define H5F_LIBVER_LATEST as a macro, which is aliased to the highest enumerated value in *H5F_libver_t*, indicating that this is currently the latest format available.

There will be six pairs of (*low*, *high*) combinations as derived from the above values in *H5F_libver_t* but the library will only support #1 to #5:

1. *low* = H5F_LIBVER_EARLIEST, *high* = H5F_LIBVER_V18
   - The library will create objects with the earliest possible format versions.
   - The library will allow objects to be created with the latest format versions available to library release 1.8.x.
   - API calls that create objects or features that are available to versions of the library greater than 1.8.x release will fail.
2. *low* = H5F_LIBVER_EARLIEST, *high* = H5F_LIBVER_V110
   - The library will create objects with the earliest possible format versions.
   - The library will allow objects to be created with the latest format versions available to library release 1.10.x. Since 1.10.x is also H5F_LIBVER_LATEST, there is no upper limit on the format versions to use. For example, if a newer format version is required to support a feature e.g. virtual dataset, this setting will allow the object to be created.
   - This is the library default setting and provides the greatest format compatibility.
3. *low* = H5F_LIBVER_V18, *high* = H5F_LIBVER_V18
   - The library will create objects with the latest format versions available to library release 1.8.x.
   - API calls that create objects or features that are available to versions of the library greater than 1.8.x release will fail.

- Earlier versions of the library may not be able to access objects created with this setting.

4. *low* = H5F_LIBVER_V18, *high* = H5F_LIBVER_V110
   - The library will create objects with the latest format versions available to library release 1.8.x.
   - The library will allow objects to be created with the latest format versions available to library release 1.10.x. Since 1.10.x is also H5F_LIBVER_LATEST, there is no upper limit on the format versions to use. For example, if a newer format version is required to support a feature e.g. virtual dataset, this setting will allow the object to be created.
   - Earlier versions of the library may not be able to access objects created with this setting.
5. *low* = H5F_LIBVER_V110, *high* = H5F_LIBVER_V110
   - The library will create objects with the latest format versions available to library release 1.10.x.
   - The library will allow objects to be created with the latest format versions available to library release 1.10.x. Since 1.10.x is also H5F_LIBVER_LATEST, there is no upper limit on the format versions to use. For example, if a newer format version is required to support a feature e.g. virtual dataset, this setting will allow the object to be created.
   - This setting allows users to take advantage of the latest features and performance enhancements in the library. However, objects written with this setting may be accessible to a smaller range of library versions than would be the case if *low* is set to H5F_LIBVER_EARLIEST.
   - Earlier versions of the library may not be able to access objects created with this setting.
6. *low* = H5F_LIBVER_EARLIEST, *high* = H5F_LIBVER_EARLIEST
   - The library does not support this setting, as it requires the library to use the earliest format versions, some of which may be buggy.

0.2 H5Pset/get_libver_bounds

We will modify the public routines *H5Pset_libver_bounds/H5Pget_libver_bounds* to accept the new values for *H5F_libver_t*. The routine will also reject invalid combinations as below:

- *low* = H5F_LIBVER_EARLIEST, *high* = H5F_LIBVER_EARLIEST
- *low* = H5F_LIBVER_V110, *high* = H5F_LIBVER_EARLIEST
- *low* = H5F_LIBVER_V110, *high* = H5F_LIBVER_V18
- *low* = H5F_LIBVER_V18, *high* = H5F_LIBVER_EARLIEST

0.3 H5Fset_libver_bounds

There is an existing public routine called *H5Fset_latest_format(*hid_t *file_id*, hbool_t *latest)*, which enables switching between latest or non-latest format while a file is opened. Currently, this routine is used in *test/genall.c*.

As new enumerated values are added to *H5F_libver_t*, *H5Fset_latest_format()* is no longer sufficient. Therefore, we will deprecate this routine and introduce the following routine to set the low and high bounds while a file is opened:

H5Fset_libver_bounds(hid_t *file_id*, H5F_libver_t *low*, H5F_libver_t *high*)

The parameters are:

- *file_id*: the file identifier
- *low*: the low bound as in *H5Pset_libver_bound()*
- *high*: the high bound as in *H5Pset_libver_bound()*

0.4 File struct

We will add two new fields to *H5F_file_t*:

```
struct H5F_file_t {
        :
        :
        H5F_libver_t low_bound;
        H5F_libver_t high_bound;
        :
        :
};
```

These two fields record the *low* and *high* bound values for the libver_bounds property set in the *fapl*, which is used to create or open a file. They replace the existing field *latest_flags*, which will be removed.

0.5 Array of versions

Arrays of versions will be set up for the format versions listed in table I (see next section). Each array will be indexed by the values defined for *H5F_libver_t*: H5F_LIBVER_EARLIEST, H5F_LIBVER_V18, H5F_LIBVER_V110.

```
/* Superblock */
#define HDF5_SUPERBLOCK_VERSION_DEF             0
#define HDF5_SUPERBLOCK_VERSION_LATEST          HDF5_SUPERBLOCK_VERSION_3
unsigned HDF5_superblock_ver_bounds[H5F_LIBVER_NBOUNDS] =
  {     HDF5_SUPERBLOCK_VERSION_DEF,
        HDF5_SUPERBLOCK_VERSION_2,
        HDF5_SUPERBLOCK_VERSION_LATEST };

/* Object header */
#define H5O_VERSION_LATEST          H5O_VERSION_2
unsigned H5O_obj_ver_bounds[H5F_LIBVER_NBOUNDS] =
  {     H5O_VERSION_1,
```

```
        H5O_VERSION_2,
        H5O_VERSION_LATEST };

/* Attribute message */
#define H5O_ATTR_VERSION_LATEST H5O_ATTR_VERSION_3
unsigned H5O_attr_ver_bounds[H5F_LIBVER_NBOUNDS] =
  {     H5O_ATTR_VERSION_1,
        H5O_ATTR_VERSION_3,
        H5O_ATTR_VERSION_LATEST };

/* Dataspace message */
#define H5O_SDSPACE_VERSION_LATEST H5O_SDSPACE_VERSION_2
unsigned H5O_sdspace_ver_bounds[H5F_LIBVER_NBOUNDS] =
  {     H5O_SDSPACE_VERSION_1,
        H5O_SDSPACE_VERSION_2,
        H5O_SDSPACE_VERSION_LATEST };

/* Datatype message */
#define H5O_DTYPE_VERSION_LATEST H5O_DTYPE_VERSION_3
unsigned H5O_dtype_ver_bounds[H5F_LIBVER_NBOUNDS] =
  {     H5O_DTYPE_VERSION_1,
        H5O_DTYPE_VERSION_3,
        H5O_DTYPE_VERSION_LATEST  };

/* Layout message */
#define H5O_LAYOUT_VERSION_LATEST H5O_LAYOUT_VERSION_4
unsigned H5O_layout_ver_bounds[H5F_LIBVER_NBOUNDS] =
  {     H5O_LAYOUT_VERSION_1,
        H5O_LAYOUT_VERSION_3,
        H5O_LAYOUT_VERSION_LATEST };

/* Fill value message */
#define H5O_FILL_VERSION_LATEST H5O_FILL_VERSION_3
unsigned H5O_fill_ver_bounds[H5F_LIBVER_NBOUNDS] =
  {     H5O_FILL_VERSION_1,
        H5O_FILL_VERSION_3,
        H5O_FILL_VERSION_LATEST };

/* Filter pipeline message */
#define H5O_PLINE_VERSION_LATEST H5O_PLINE_VERSION_2
unsigned H5O_pline_ver_bounds[H5F_LIBVER_NBOUNDS] =
  {     H5O_PLINE_VERSION_1,
        H5O_PLINE_VERSION_2,
        H5O_PLINE_VERSION_LATEST };
```

The library will use the *low_bound* field in *H5F_file_t* to index into the corresponding array to determine the earliest format version that can be used. It will also perform bounds check by indexing the *high_bound* field into the corresponding array to make sure the version set does not exceed the version allowed. For example, when creating a dataset, the bounds check for *layout* version will be:

If(layout version > *H5O_layout_ver_bounds[high_bound])*
error

If the user creates a virtual dataset, the library will generate a version 4 *layout* message. If *high_bound* is *v1*8, the above check will trigger failure for the dataset creation.

Format versions

The following two tables list the format versions that the library will use when creating objects in the file. The first table lists those formats that have multiple versions for supporting certain features and the public routines that will trigger such features. The second table lists those formats that have just the basic versions.

The two tables also include the library release that a feature is introduced. This information will help in setting the *low* and *high* parameters for *H5Pset_libver_bounds()* when an application uses a feature.

Table I

Format version		Feature	Introduced in library release	Trigger
HDF5_SUPERBLOCK_VERSION (Superblock)	0	Basic	1_6	
	1	Non-default v1-btree K value	1_6	*H5Pset_istore_k*
	2	a) SOHM b) Non-default free-space info	1_8 1_10	*H5Pset_shared_mesg_index* *H5Pset_file_space* *H5Pset_libver_bounds:* *(v18, v18), (v18, v10)*
	3	Enable SWMR write	1_10	*H5Fcreate: H5F_ACC_SWMR_WRITE* *H5Pset_libver_bounds: (v110, v110)*
H5O_VERSION (Object header)	1	Basic	1_0	
	2	a) Creation order for attributes b) Shared attributes	1_8 1_8	*H5Pset_attr_creation_order* *H5Pset_shared_mesg_index:* *H5O_SHMESG_ATTR_FLAG* *H5Pset_libver_bounds:* *(v18, v18), (v18, v110) (v110, v110)*
H5O_ATTR_VERSION (Attribute message)	1	Basic	1_0	
	2	a) Shared datatype b) Shared dataspace	1_6 1_8	*H5Tcommit2* *H5Pset_shared_mesg_index:*

			H5O_SHMESG_SDSPACE_FLAG *H5O_SHMESG_DTYPE_FLAG*	
3	Character encoding	1_8	*H5Pset_char_encoding* *H5Pset_libver_bounds:* *(v18, v18), (v18, v110) (v110, v110)*	
H5O_SDSPACE_VERSION (Dataspace message)	1	Basic	1_0	
	2	Support for NULL dataspace & more efficient packing in message	1_8	*H5Screate: H5S_NULL* *H5Pset_libver_bounds:* *(v18, v18), (v18, v110) (v110, v110)*
H5O_DTYPE_VERSION (Datatype message)	1	Basic: (all except array types)	1_0	
	2	Array datatypes	1_4	*H5Tarray_create2*
	3	More efficient encoding and packing in message for *compound*, *enum*, *array* types; also *vlen* type with above types as base type	1_8	*H5Pset_libver_bounds:* *(v18, v18), (v18, v110) (v110, v110)*
H5O_LAYOUT_VERSION (Layout message)	1	Basic	1_0	
	2	Delayed space allocation	1_4	
	3	Default (more efficient encoding and fix bug in previous versions)	1_8	
	4[#]	a) Virtual Dataset b) Do not filter partial bound chunks	1_10	*H5Pset_virtual* *H5Pset_chunk_opts* *H5Pset_libver_bounds: (v110, v110)*
H5O_FILL_VERSION (Fill value message)	1	Basic	1_6	
	2	Default (more condensed packing in message)	1_8	
	3	More efficient packing in message	1_8	*H5Pset_libver_bounds:* *(v18, v18), (v18, v110) (v110, v110)*
H5O_PLINE_VERSION (Filter pipeline message)	1	Basic	1_0	
	2	More efficient packing in message	1_8	*H5Pset_libver_bounds:* *(v18, v18), (v18, v110) (v110, v110)*

* This will enable all latest version support as *H5Pset_libver_bounds: (v10, v10)*

This will enable latest chunk indexing

Table II

Format version		Feature	Introduced in library release
HDF_FREESPACE_VERSION (File's free-space information)	0	Basic	1_0
HDF5_OBJECTDIR_VERSION (Root group symbol table entry)	0	Basic	1_0
HDF5_DRIVERINFO_VERSION (Driver information block)	0	Basic	1_4
HDF5_SHAREDHEADER_VERSION (Shared header message format)	0	Basic	1_0
H5HG_VERSION (Global heap)	1	Basic	1_0
H5HL_VERSION (Local heap)	0	Basic	1_6
H5B2_HDR_VERSION (Version 2 B-tree header)	0	Basic	1_8
H5B2_INT_VERSION (Version 2 B-tree internal node)	0	Basic	1_8
H5B2_LEAF_VERSION (Version 2 B-tree leaf node)	0	Basic	1_8
H5FS_HDR_VERSION (Free-space manager header)	0	Basic	1_8
H5FS_SINFO_VERSION (Free-space manager section info)	0	Basic	1_8
H5HF_HDR_VERSION (Fractal heap header)	0	Basic	1_8
H5HF_DBLOCK_VERSION (Fractal heap direct block)	0	Basic	1_8
H5HF_IBLOCK_VERSION (Fractal heap indirect block)	0	Basic	1_8
H5SM_LIST_VERSION (Shared object header message list)	0	Basic	1_8
H5EA_HDR_VERSION (Extensible array header)	0	Basic	1_10
H5EA_IBLOCK_VERSION (Extensible array index block)	0	Basic	1_10
H5EA_SBLOCK_VERSION (Extensible array secondary block)	0	Basic	1_10
H5EA_DBLOCK_VERSION (Extensible array data block)	0	Basic	1_10
H5FA_HDR_VERSION (Fixed array header)	0	Basic	1_10
H5FA_DBLOCK_VERSION (Fixed array data block)	0	Basic	1_10
H5O_FSINFO_VERSION (File space info message)	0	Basic	1_10
H5O_MDCI_VERSION (Metadata cache image message)	0	Basic	1_10
H5O_EFL_VERSION (External file list message)	1	Basic	1_0
H5O_MTIME_VERSION (Object modification time message)	1	Basic	1_6
H5O_LINK_VERSION (Link message)	1	Basic	1_8
H5O_AINFO_VERSION (Attribute info message)	0	Basic	1_8
H5O_BTREEK_VERSION (B-tree 'K' bbvalues message)	0	Basic	1_8
H5O_DRVINFO_VERSION (Driver Info message)	0	Basic	1_8
H5O_GINFO_VERSION (Group info message)	0	Basic	1_8
H5O_LINFO_VERSION (Link info message)	0	Basic	1_8

H5O_REFCOUNT_VERSION (Object reference count message)	0	Basic	1_8
H5L_EXT_VERSION (External link)	0	Basic	1_8

Cycle of Operation

0.6 Setting file access property list

A user can call the public routine *H5Pset_libver_bounds*() to specify the bounds for library releases when creating objects in a file. When not specified, the default setting for low/high bounds is (*earliest*, *latest*).

When a file is created or opened, the library will set the *low_bound* and *high_bound* fields in *H5F_file_t* to the *low* and *high* parameters set via *H5Pset_libver_bounds*() in the *fapl*.

0.7 Creating a file

For file creation, the features enabled in *fcpl*, the bounds setting in *fapl*, and the file access permission will determine the following:

- The superblock version #
- Fail or succeed in creating the file

Depending on the file's access permission, the library will initially set the superblock version as follows:

- When creating a file with write access, the superblock version is set according to the features enabled in *fcpl*:
    - Version 0: default
    - Version 1: non-default v1-btree K value
    - Version 2: shared message index, non-default file space info
- When creating a file with SWMR-write access, the superblock version is set to 3 and the *low_bound* field in *H5F_file_t* is set to at least v110.

  Upgrading the low_bound will give the best format versions available in library release 1.10.x for SWMR. Version 3 superblock is introduced in 1.10 for SWMR due to the problem of the *status_flags* field in the superblock. See jira issue SWMR-79 and also the *RFC: File Format Changes in HDF5 1.10.0*.

It then increments the superblock version as required by the low_bound.

Lastly, the library will verify that the superblock version does not exceed the version allowed by the high_bound. File creation will fail if this condition does not hold.

The following two tables depict the actions described above when creating a file for the five pairs of low/high bounds setting in the *fapl*.

Creating a file with write access

Bounds setting	*(earliest, v18)*	*(earliest, v110)*	*(v18, v18)*	*(v18, v110)*	*(v110, v110)*
Superblock version	0, 1 or 2	0, 1, or 2	2	2	3
low_bound	No change				
File creation	Succeed				

Creating a file with SWMR-write access

Bounds setting	*(earliest, v18)*	*(earliest, v110)*	*(v18, v18)*	*(v18, v110)*	*(v110, v110)*
Superblock version	--	3	--	3	3
low_ bound	--	v110	--	v110	No change
File creation	Fail	Succeed	Fail	Succeed	Succeed

“” indicates “upgrade to”

0.8 Opening a file

On file open, the file’s superblock version, the bounds setting in *fapl*, and the file access permission determine:

- Whether the *low_bound* field in *H5F_file_t* is upgraded,
- Whether the file open succeeds

Depending on the file’s access permission, the library will upgrade *low_bound* based on the file’s superblock version:

- When opening a file with write access:
    - Version 0 or 1 superblock: no change to *low_bound*
    - Version 2 superblock: upgrade *low_bound* to at least v18
    - Version 3 superblock: upgrade *low_bound* to at least v110
- When opening a file with SWMR-write access:
    - Version 0 or 1 superblock: fail the file open
    - Version 3 superblock: upgrade *low_bound* to at least v110

The library will then verify that the superblock version does not exceed the version allowed by the high_bound. File open will fail if this condition does not hold

The following five tables depict the actions described above when opening a file for the five pairs of low/high bounds setting in the *fapl*.

Opening a file with write access

Superblock version 0, 1

Bounds setting	*(earliest, v18)*	*(earliest, v110)*	*(v18, v18)*	*(v18, v110)*	*(v110, v110)*
low_bound	No change				
File open	Succeed				

Superblock version 2

Bounds setting	*(earliest, v18)*	*(earliest, v110)*	*(v18, v18)*	*(v18, v110)*	*(v110, v110)*
low_ bound	v18	v18	No change	No change	No change
File open	Succeed				

“” indicates “upgrade to”

Superblock version 3

Bounds setting	*(earliest, v18)*	*(earliest, v110)*	*(v18, v18)*	*(v18, v110)*	*(v110, v110)*
low_ bound	--	v110	--	v110	No change
File open	Fail	Succeed	Fail	Succeed	Succeed

Opening a file with SWMR-WRITE access:

Superblock version 0, 1, 2

Bounds setting	*(earliest, v18)*	*(earliest, v110)*	*(v18, v18)*	*(v18, v110)*	*(v110, v110)*
low_bound	--				
File open	Fail				

Superblock version 3

Bounds setting	*(earliest, v18)*	*(earliest, v110)*	*(v18, v18)*	*(v18, v110)*	*(v110, v110)*
low_ bound	--	v110	--	v110	No change
File open	Fail	Succeed	Fail	Succeed	Succeed

“” indicates “upgrade to”

0.9 Setting message versions

Object header

When creating an object header for an object, the library will first select amongst the available versions:

- Version 1: default
- Version: 2: attribute creation order is tracked, shared attribute in file

via a call to *H5O_set_version()*. This is done without reference to the version bounds setting.

It then increments the selected object header version as required by the low_bound.

Finally, it tests to see if the selected object header version is permitted by the high_bound, and forces the object header creation to fail if it not.

Attribute

When creating an attribute attached to an object, the library will first select amongst the available attribute versions:

- Version 1: default
- Version 2: shared datatype or dataspace
- Version 3: character encoding

via a call to *H5A_set_version()*. This is done without reference to the version bounds setting.

It then increments the selected attribute version as required by the low_bound.

Finally, it tests to see if the selected attribute version is permitted by the high_bound, and forces the attribute creation to fail if it not.

Dataspace

When creating a dataspace, the library will initially set the dataspace message version as shown below without regard to the version bounds setting:

- Version 1: default
- Version 2: a null dataspace

When the dataspace is subsequently associated with an attribute or dataset, the library will call *H5S_set_version()* to:

- Increment the selected dataspace version as required by the low_bound
- Verify that the selected dataspace version is permitted by the high_bound

If the latter check fails, the attribute or dataset creation will fail.

Datatype

When creating a datatype, the library will initially set the datatype message version as shown below without regard to the version bounds setting:

- Version 1: default (version 3 is selected only as a result of the low_bound)
- Version 2: array type

When the datatype is subsequently associated with an attribute, a dataset, or a committed datatype , the library will call *H5T_set_version()* to:

- Increment the selected datatype version as required by the low_bound

- Verify that the selected datatype version is permitted by the high_bound

If the latter check fails, the attribute/dataset/committed datatype creation will fail.

Layout

When setting the layout property in a dataset creation property list, the library will initially set the layout message version as shown below without regard to the version bounds setting:

- Version 3: default (versions 1 and 2 appear to have been deprecated since HDF5 1.6)
- Version 4: virtual layout, do not filter partial bound chunks

When the layout message is subsequently associated with a dataset, the library will call *H5D__layout_set_version()* to:

- Increment the selected layout property version as required by the low_bound
- Verify that the selected layout property version is permitted by the high_bound

If the latter check fails, the dataset creation will fail.

Fill value

When setting the fill value property in a dataset creation property list, the library will initially set the fill value message version as shown below without regard to the version bounds setting:

- Version 2: default (version 1 appears to have been deprecated since HDF5 1.6, version 3 is selected only as a result of the low_bound)

When the fill value message is subsequently associated with a dataset, the library will call *H5D__fill_set_version()* to:

- Increment the selected fill value message version as required by the low_bound
- Verify that the selected fill value message version is permitted by the high_bound

If the latter check fails, the dataset creation will fail.

Filter pipeline

When setting the filter property in a dataset creation property list, the library will initially set the filter pipeline message version as shown below without regard to the version bounds setting:

- Version 1: default (version 2 is selected only as a result of the low_bound)

When the filter pipeline message is subsequently associated with a dataset, the library will call *H5O__pline_set_version()* to:

- Increment the selected filter pipeline message version as required by the low_bound
- Verify that the selected filter pipeline message version is permitted by the high_bound

If the latter check fails, the dataset creation will fail.

The filter property can also be set in a group creation property list. The library will perform similar

actions as described above except that the filter pipeline message will be associated with a fractal heap for dense storage.

0.10 Bounds check

Cache image

As this feature is introduced in library release 1.10.x and the creation of a cache image is done only on file close, we will perform the bounds check in the internal library routine *H5C__prep_image_for_file_close()*. This routine is called by the metadata cache on file close to do preparatory work prior to the generation of a cache image. We will verify the following:

- If the high_bound is less than v110, silently cancel the request for a cache image.

The silent failure follows the same rationale as the check for superblock version in this routine before the generation of a cache image – specifically that the cache image is an optimization, and thus it is not appropriate to trigger an error when this optimization cannot be applied. Please see the *RFC: Metadata Cache Image* for details.

H5Ocopy

H5Ocopy is a public routine that copies an object in an HDF5 file to a destination location in the same file or a different file. The library does the following bounds check when copying an object:

- The source object header to be copied does not exceed the version allowed by the destination file's high_bound. Otherwise *H5Ocopy* will fail.
- For each message in the source object header to be copied, check to ensure that the version of the source message to be copied does not exceed the version allowed by the destination file's high_bound. Otherwise *H5Ocopy* will fail. The bounds check is done in each message's pre_copy callback.

H5Sencode and H5Pencode

The new public routine *H5Sencode2()* will be introduced in library release 1.12.x. The new parameter *fapl* to this routine controls the encoding used via the bounds setting in *H5Pset_libver_bounds()*. In the *RFC: H5Sencode/H5Sdecode Format Change*, the encodings are described only for the *(earliest, latest)* and *(latest, latest)* settings. As new combinations of low/high bounds are added in this RFC, the RFC for *H5Sencode* and the internal library coding need to be modified accordingly. Also, tests need to be added.

Tools

The tool *h5repack* is modified to add two new parameters, *-j* and *-k*, so users can specify the *low* and *high* bounds when repacking a file.

Testing

NOTE: Binh-Minh, please modify this section accordingly for tests that you add/modify.

0.11 Regression tests

There are tests in *test/tfile.c* to verify the format versions listed in table I are as specified. Each test is tested with the five combinations of low/high bounds via *H5Pset_libver_bounds()*.

- *test_libver_bounds_super()*
  - Validate superblock versions
- *test_libver_bounds_obj()*
  - Validate object header versions
- *test_libver_bounds_dataset()*
  - Validate format versions for layout, filter pipeline, and fill value messages
- *test_libver_bounds_dataspace()*
  - Validate dataspace message versions
- *test_libver_bounds_datatype()*
  - Validate datatype message versions
- *test_libver_bounds_attributes()*
  - Validate attribute message versions

0.12 Compatibility tests

The program *gen_bounds.c* will generate the following HDF5 files:

1) *bounds_earliest_latest.h5*
   a. Version 0 superblock
   b. Contains a chunked dataset with layout version 3
   c. Contains a chunked dataset with layout version 4
2) *bounds_earliest_v18.h5*
   a. Version 0 superblock
   b. Contains a chunked dataset with layout version 3
3) *bounds_latest_latest.h5*
   a. Version 3 superblock
   b. Contains a chunked dataset with layout version 4
4) *bounds_v18_latest.h5*
   a. Version 2 superblock
   b. Contains a chunked dataset with layout version 3
5) *bounds_v18_v18.h5*
   a. Version 2 superblock
   b. Contains a chunked dataset with layout version 3
   c. Contains a chunked dataset with layout version 4

Testing with 1.8 library:

1) *bounds_earliest_latest.h5*
   a. Is able to open the chunked dataset with layout version 3
   b. Is unable to open the chunked dataset with layout version 4—see error #a below
2) *bounds_earliest_v18.h5*

a. Is able to open the chunked dataset with layout version 3

3) *bounds_latest_latest.h5*
   a. Is unable to open the file—see error #b below.
4) *bounds_v18_latest.h5*
   a. Is able to open the chunked dataset with layout version 3
5) *bounds_v18_v18.h5*
   a. Is able to open the chunked dataset with layout version 3
   b. Is unable to open the chunked dataset with layout version 4—see error #a below

- Error #a:

```
HDF5-DIAG: Error detected in HDF5 (1.8.18-snap1) thread 0:
  #000: ../../hdf5/src/H5D.c line 369 in H5Dopen2(): can't open dataset
    major: Dataset
    minor: Unable to initialize object
  #001: ../../hdf5/src/H5Dint.c line 1248 in H5D_open(): not found
    major: Dataset
    minor: Object not found
  #002: ../../hdf5/src/H5Dint.c line 1374 in H5D__open_oid(): can't get layout/pline/efl info
    major: Dataset
    minor: Can't get value
  #003: ../../hdf5/src/H5Dlayout.c line 354 in H5D__layout_oh_read(): unable to read data layout message
    major: Dataset
    minor: Unable to initialize object
  #004: ../../hdf5/src/H5Omessage.c line 484 in H5O_msg_read(): unable to read object header message
    major: Object header
    minor: Read failed
  #005: ../../hdf5/src/H5Omessage.c line 545 in H5O_msg_read_oh(): unable to decode message
    major: Object header
    minor: Unable to decode value
  #006: ../../hdf5/src/H5Olayout.c line 118 in H5O_layout_decode(): bad version number for layout message
    major: Object header
    minor: Unable to load metadata into cache
```

- Error #b:

```
HDF5-DIAG: Error detected in HDF5 (1.8.18-snap1) thread 0:
  #000: ../../hdf5/src/H5F.c line 604 in H5Fopen(): unable to open file
    major: File accessibilty
    minor: Unable to open file
  #001: ../../hdf5/src/H5Fint.c line 1087 in H5F_open(): unable to read superblock
    major: File accessibilty
    minor: Read failed
  #002: ../../hdf5/src/H5Fsuper.c line 294 in H5F_super_read(): unable to load superblock
    major: Object cache
    minor: Unable to protect metadata
  #003: ../../hdf5/src/H5AC.c line 1262 in H5AC_protect(): H5C_protect() failed.
    major: Object cache
    minor: Unable to protect metadata
  #004: ../../hdf5/src/H5C.c line 3574 in H5C_protect(): can't load entry
    major: Object cache
    minor: Unable to load metadata into cache
```

```
  #005: ../../hdf5/src/H5C.c line 7954 in H5C_load_entry(): unable to load entry
    major: Object cache
    minor: Unable to load metadata into cache
  #006: ../../hdf5/src/H5Fsuper_cache.c line 174 in H5F_sblock_load(): bad superblock version number
    major: File accessibilty
    minor: Bad value
```

Testing with 1.6 library:

1) *bounds_earliest_latest.h5*
   a. Is able to open the chunked dataset with layout version 3
   b. Is unable to open the chunked dataset with layout version 4—see error #c below
2) *bounds_earliest_v18.h5*
   a. Is able to open the chunked dataset with layout version 3
3) *bounds_latest_latest.h5*
   a. Is unable to open the file—see error #d below
4) *bounds_v18_latest.h5*
   a. Is unable to open the file—see error #d below
5) *bounds_v18_v18.h5*
   a. Is unable to open the file—see error #d below

- Error #c:

```
HDF5-DIAG: Error detected in HDF5 library version: 1.6.10-snap17 thread 0.  Back trace follows.
  #000: ../../hdf5/src/H5D.c line 1174 in H5Dopen(): not a dataset
    major(15): Dataset interface
    minor(03): Inappropriate type
  #001: ../../hdf5/src/H5G.c line 2088 in H5G_get_type(): unable to determine object type
    major(10): Symbol table layer
    minor(29): Unable to initialize object
  #002: ../../hdf5/src/H5D.c line 2343 in H5D_isa(): unable to read object header
    major(15): Dataset interface
    minor(29): Unable to initialize object
  #003: ../../hdf5/src/H5O.c line 953 in H5O_exists(): unable to verify object header message
    major(12): Object header layer
    minor(24): Read failed
  #004: ../../hdf5/src/H5O.c line 996 in H5O_exists_real(): unable to load object header
    major(12): Object header layer
    minor(40): Unable to load metadata into cache
  #005: ../../hdf5/src/H5AC.c line 764 in H5AC_protect(): H5C_protect() failed.
    major(08): Meta data cache layer
    minor(46): Unable to protect metadata
  #006: ../../hdf5/src/H5C.c line 2974 in H5C_protect(): can't load entry
    major(08): Meta data cache layer
    minor(40): Unable to load metadata into cache
  #007: ../../hdf5/src/H5C.c line 3914 in H5C_load_entry(): unable to load entry
    major(08): Meta data cache layer
    minor(40): Unable to load metadata into cache
  #008: ../../hdf5/src/H5Ocache.c line 243 in H5O_load(): bad object header version number
    major(12): Object header layer
    minor(56): Wrong version number
```

- Error #d:

```
HDF5-DIAG: Error detected in HDF5 library version: 1.6.10-snap17 thread 0.  Back trace follows.
  #000: ../../hdf5/src/H5F.c line 2072 in H5Fopen(): unable to open file
    major(04): File interface
    minor(17): Unable to open file
  #001: ../../hdf5/src/H5F.c line 1852 in H5F_open(): unable to read superblock
    major(04): File interface
    minor(24): Read failed
  #002: ../../hdf5/src/H5Fsuper.c line 125 in H5F_read_superblock(): bad superblock version number
    major(04): File interface
    minor(05): Bad value
```

Future releases/maintenance

To handle future releases, for example, HDF5 release 1.12:

- Add H5F_LIBVER_V112 to *H5F_libver_t* struct defined in *H5Fpublic.h*.
- Change H5F_LIBVER_LATEST macro to H5F_LIBVER_V112 defined in *H5Fpublic.h*
- Add the version for *v112* to the arrays of format versions:
    - HDF5_superblock_ver_bounds[] defined in *H5Fsuper.c*
    - H5O_obj_ver_bounds[] defined in H5Oint.c
    - H5O_ layout_ver_bounds defined in *H5Dlayout.c*
    - H5O_attr_ver_bounds[] defined in *H5Aint.c*
    - H5O_dtype_ver_bounds[] defined in *H5T.c*
    - H5O_fill_ver_bounds[] defined in *H5Ofill.c*
    - H5O_pline_ver_bounds[] defined in *H5Opline.c*
    - H5O_sdspace_ver_bounds[] defined in *H5S.c*
- Modify the tests to accommodate H5F_LIBVER_V112

Appendix: Table of format versions for library releases

	1_0	*1_2*	*1_4*	*1_6*	*1_8*	*1_10*
HDF5_BOOTBLOCK_VERSION	0	0	0	-	-	-
HDF5_SUPERBLOCK_VERSION				0, 1	0, 1, 2	0, 1, 2, 3
HDF5_FREESPACE_VERSION	0	0	0	0	0	0
HDF5_OBJECTDIR_VERSION	0	0	0	0	0	0
HDF5_DRIVERINFO_VERSION	-	-	0	0	0	0
HDF5_SHAREDHEADER_VERSION	0	0	0	0	0	0
H5HL_VERSION	no vers	no vers	no vers	0	0	0

H5HG_VERSION	1	1	1	1	1	1
H5B2_HDR_VERSION					0	0
H5B2_INT_VERSION					0	0
H5B2_LEAF_VERSION					0	0
H5FS_HDR_VERSION					0	0
H5FS_SINFO_VERSION					0	0
H5HF_HDR_VERSION					0	0
H5HF_DBLOCK_VERSION					0	0
H5HF_IBLOCK_VERSION					0	0
H5SM_LIST_VERSION					0	0
H5EA_HDR_VERSION						0
H5EA_IBLOCK_VERSION						0
H5EA_SBLOCK_VERSION						0
H5EA_DBLOCK_VERSION						0
H5FA_HDR_VERSION						0
H5FA_DBLOCK_VERSION						0
H5O_VERSION	1	1	1	1	1, 2	1, 2
H5O_ATTR_VERSION	1	1	1	1, 2	1, 2, 3	1, 2, 3
H5O_DTYPE_VERSION	1	1	1, 2	1, 2	1, 2, 3	1, 2, 3
H5O_EFL_VERSION	1	1	1	1	1	1
H5O_FILL_VERSION	no vers	no vers	no vers	no vers, 1, 2	1, 2, 3	1, 2, 3
H5O_LAYOUT_VERSION	1	1	1, 2	1, 2, 3	1, 2, 3	1, 2, 3, 4

H5O_MTIME_VERSION	no vers	no vers	no vers	no vers, 1	no vers, 1	no vers, 1
H5O_PLINE_VERSION	1	1	1	1	1, 2	1, 2
H5O_SDSPACE_VERSION	1	1	1	1	1, 2	1, 2
H5O_SHARED_VERSION	1	1	1	1, 2	1, 2, 3	1, 2, 3
H5O_AINFO_VERSION					0	0
H5O_BTREEK_VERSION					0	0
H5O_DRVINFO_VERSION					0	0
H5O_FSINFO_VERSION						0
H5O_MDCI_VERSION						0
H5O_GINFO_VERSION					0	0
H5O_LINK_VERSION					1	1
H5L_EXT_VERSION					0	0
H5O_LINFO_VERSION					0	0
H5O_REFCOUNT_VERSION					0	0

Revision History

January 5, 2016: Version 1 sent for internal review.

September 15, 2017 Version 2: revised to reflect current implementation.

January 11, 2018 Version 3: another revision to reflect implementation.

Januray 19, 2018 Version 4: revised after review by John Mainzer

References

1) The HDF Group, “HDF5 File Format Specification”
https://www.hdfgroup.org/HDF5/doc/H5.format.html

2)