RFC: H5Sencode/H5Sdecode Format Change

Vailin Choi

John Mainzer

Neil Fortner

Jira issue HDFFV-9947 reported an overflow problem when encoding a dataspace with selected elements exceeding 2^32 (32 bits integer limit).

This RFC describes the current format of the encoded buffer that causes the overflow problem and proposes a solution to address the issue.

Introduction

The public routine *H5Sencode()* encodes a dataspace description into a buffer. The description contains information about the dataspace message and the dataspace selection. There are four types of dataspace selections:

1) H5S_SEL_NONE: nothing selected
2) H5S_SEL_POINTS: sequence of points selected
3) H5S_SEL_HYPER: hyperslab selected
4) H5S_SEL_ALL: entire extent selected

The problem documented in jira issue HDFFV-9947 centers on the H5S_SEL_HYPER selection encoding, in which only 32 bits are used to encode counts and block offsets. Not only has this become insufficient, but the existing code fails to flag an error when the 32 bit limit is exceeded. On investigation, we discovered similar issues with H5S_SEL_POINTS encoding.

A further issue springs from the current lack of any mechanism for associating selection encodings with specific files. This presents a problem when introducing new selection encoding formats, as in the absence of such a mechanism, there is no way to control the encodings used to match the library version bounds for the target file. With extended, 64 bit versions of selection encodings, this has become necessary, and thus this RFC addresses this issue as well.

Except as noted, the changes discussed in this RFC are directed at HDF5 1.12.

Proposed Changes

0.1 New Public API Routines

As intimated in the introduction, both public API and selection encoding changes are needed. In this section, we address the API changes, leaving file format changes for Section 3 below.

2.1.1 H5Sencode

A new version of H5Sencode() is needed to address the following issues:

- The existing public routine *H5Sencode()* encodes dataspace selection to a buffer but does not tie the selection to a file or otherwise allow the user to control the encodings used.
- The file format changes needed to address the 32 bit limitation in turn require us to allow the user to select the encoding that corresponds to the target file (as specified by file format set in the file access property list). Failure to do so would prevent the maintenance of forward compatibility, allowing older versions of the library to decode the buffer.
- Again for forwards compatibility, we must allow internal library calls that encode dataspace selections to a buffer to do so according to the file format set in the file access property list.

With these points in mind, we propose the new API call:

```
herr_t H5Sencode2(hid_t space_id,  void *buf, size_t *nalloc,
hid_t fapl)
```

Where the first three parameters are the same as the existing H5Sencode() API call, and the fourth "`fapl`" parameter is used to control the encoding used via the `libver_bounds` property. If the `libver_bounds` property is missing, `H5Sencode2()` proceeds as if the `libver_bounds` property was set to *(H5F_LIBVER_EARLIEST, H5F_LIBVER_LATEST)*. See Section 2.2 & 2.3 for details on the encodings used for different `libver_bounds` settings.

As usual when adding a new version of an existing API, we propose renaming the existing API call `H5Sencode()` to `H5Sencode1()`, and adding the compatibility macro `H5Sencode()`.

Functionally, `H5Sencode1()` will be identical to `H5Sencode2()` with `libver_bounds` set to *(H5F_LIBVER_EARLIEST, H5F_LIBVER_LATEST)*.

There is no API change for `H5Sdecode()`, which will be updated to handle the new encoding formats.

To address the overflow problem, `H5Sencode()` in HDF5 1.8 must be modified to fail if the 32 bit limit is exceeded when encoding either offsets or counts in the selection.

The situation is a bit more complicated in HDF5 1.10, as the version 2 hyperslab encoding exists in this release, and there are three different cases that must be addressed – `H5Sencode()` proper, region references, and VDS.

For `H5Sencode()` proper, the issue is that without a new API, the function cannot receive direction on whether and how to employ the version 2 hyperslab encoding, and thus we must select some behavior that will cause minimal pain pending the API change in HDF5 1.12. My inclination is to duplicate the behavior of `H5Sencode()` in HDF5 1.8, specifically to use only the original 32 bit selection encodings, and fail if the 32 bit limit is exceeded. This has the twin advantages of being simple, and ensuring no forward compatibility issues from this quarter. See table 1 below.

Neil would prefer to have it use the 32 bit encodings if possible, but use the version 2 hyperslab encoding if possible and necessary rather than fail. (Neil: please present your arguments for this option.) See table 2 below.

For region references, if it is practical to access the libver_bounds setting for the file, we could use the version 2 hyperslab encoding or not based on this setting. See tables 2 and 3 below.

For VDS, we are already using the version 2 hyperslab encoding in the regular/unlimited case. If I recall my discussion with Neil correctly, the version 1 encoding must be used for irregular encodings (and must fail if count or offset exceeds 32 bits as the version 2 hyperslab encoding does not support irregular selections). For regular encodings, Neil would like the version 1 encoding to be used if there are less than 4 blocks, and the offsets fit in 32 bits. In all other cases, he would prefer that the version 2 hyperslab be used. Neil's argument here is that this ensures the most compact representation possible given the available hyperslab encodings in 1.10. See table 3 below.

Table 1: H5Sencode for hyperslab selection (proposed for 1.10 H5Sencode proper)

version	hyperslab type	H5S_UNLIMITED in selection specification	offset	# of blocks in the selection
1	irregular	N/A	up to (2^32 – 1)	up to (2^32 – 1)
FAIL	irregular	N/A	> (2^32 – 1)	> (2^32 – 1)
1	regular	no	up to (2^32 – 1)	up to (2^32 – 1)
FAIL	regular	no	> (2^32 – 1)	> (2^32 – 1)
2	regular	yes	up to (2^64 – 1)	N/A

Note: the proposed actions in table 1 above to handle compatibility issues can be mitigated by informing users with the following:

- Do not use H5S_UNLIMITED in the selection.
- Use the public routine `H5Sget_select_bounds(space, start, end)` to ensure that the upper bound for the selection does not exceed 32 bits.

Table 2: H5Sencode for hyperslab selection (proposed for 1.10 earliest format)

version	hyperslab type	H5S_UNLIMITED in selection specification	offset	# of blocks in the selection
1	irregular	N/A	up to (2^32 – 1)	up to (2^32 – 1)
FAIL	irregular	N/A	> (2^32 – 1)	> (2^32 – 1)
1	regular	no	up to (2^32 – 1)	up to (2^32 – 1)
2	regular	N/A (yes or no)	up to (2^64 – 1)	N/A

Table 3: H5Sencode for hyperslab selection (proposed for 1.10 latest format)

version	hyperslab type	H5S_UNLIMITED in selection specification	offset	# of blocks in the selection
1	irregular	N/A	up to (2^32 – 1)	up to (2^32 – 1)
FAIL	irregular	N/A	> (2^32 – 1)	> (2^32 – 1)
1	regular	no	up to (2^32 – 1)	< 4
2	regular	N/A (yes or no)	up to (2^64 – 1)	N/A

Table 4 is proposed for point selections in 1.10.

Table 4: H5Sencode for point selections (proposed for 1.10 earliest and latest format)

version	Coordinates in the selection	# of points in the selection
1	up to (2^32 – 1)	up to (2^32 – 1)
FAIL	> (2^32 – 1)	> (2^32 – 1)

2.1.2 H5Pencode

Unfortunately, the API changes cannot be restricted to H5Sencode(). H5Pencode() encodes properties on a property list into a buffer, and property lists can include selections via the virtual storage layout property. Thus similar API changes are required here as well – we propose the new API call:

```
herr_t H5Pencode2(hid_t plist_id,   void *buf, size_t *nalloc,
hid_t fapl)
```

As before, the first three parameters are the same as the existing `H5Pencode()` API call, and the `fapl` parameter allows the user to control the encoding used for selections via the `libver_bounds` property.

Similarly, `H5Pencode()` must be renamed `H5Pencode1()` and the `H5Pencode()` compatibility macro added.

As any selection encoding will be done via calls to the internal H5S_encode() function, all necessary functional changes will be implemented by that call.

No API changes are needed for H5Pdecode()

0.2 H5S_SEL_POINTS

Currently, the library uses the version 1 point selection encoding format to encode point selections (see section 3 for descriptions of the current and proposed encoding formats for selections).

The version 1 point selection encoding format uses:

- 32 bits to encode the number of points in the selection
- 32 bits to encode the coordinates of the points in the selection

Table A below shows the current version used to encode point selections. At present when version 1 is used to encode the coordinates or # of points that exceed 32 bits, the encoding is incorrect and the library fails to flag an error

Table A: H5Sencode for point selections (current)

version	Coordinates in the selection	# of points in the selection
1	up to (2^32 – 1)	up to (2^32 – 1)

We propose adding version 2 as follows:

- Use a *size of point info* field to indicate the size that will be used to encode point info, i.e. the number of points and the coordinates of points in the selection. The size can be 2, 4 or 8 bytes.

- See section 3 on the detailed format description for version 2 points selection info.

Tables B and C below list the versions we will use to encode point selections for the earliest and latest file format respectively.

Table B: H5Sencode for point selections (proposed for earliest format)

version	Coordinates in the selection	# of points in the selection
1	up to (2^32 – 1)	up to (2^32 – 1)
2	up to (2^64 – 1)	up to (2^64 – 1)

Table C: H5Sencode for point selections (proposed for latest format)

version	Coordinates in the selection	# of points in the selection
2	up to (2^64 – 1)	up to (2^64 – 1)

0.3 H5S_SEL_HYPER

Currently, the library uses two encoding formats to encode hyperslab selections:

- Version 1
  - Uses 32 bits to encode the number of blocks in the selection
  - Uses 32 bits to encode the starting and ending offsets of each block in the selection
- Version 2
  - Uses 64 bits to encode the *start/stride/count/block* arrays of the selection specification

See table D below for the encoding versions used to encode hyperslab selections. At present when version 1 is used to encode offset or # of blocks that exceed 32 bits, the encoding is incorrect and the code fails to flag an error.

Table D: H5Sencode for hyperslab selections (current)

version	hyperslab type	H5S_UNLIMITED in selection specification	offset	# of blocks in the selection
1	irregular	N/A	up to (2^32 – 1)	up to (2^32 – 1)
1	regular	no	up to (2^32 – 1)	up to (2^32 – 1)
2	regular	yes	up to (2^64 – 1)	N/A

We propose to adding the version 3 hyperslap encoding as follows:

- Use a *flag* field to indicate regular or irregular hyperslab type:

- For regular hyperslab, we will use *start/stride/count/block* arrays to specify selection info.
- For irregular hyperslab, we will encode the starting and ending offsets of each block in the selection.

- Use a *size of offset info* field to indicate the size that will be used to encode offset info, i.e. offset and/or # of blocks in the selection. The size can be 2, 4 or 8 bytes.
- See section 3 on the detailed format description for version 3 hyperslab selection info.

Tables E and F below show the hyperslab encoding versions we will use to encode hyperslab selections for the earliest and latest file format respectively.

Table E: H5Sencode for hyperslab selection (proposed for earliest format)

version	hyperslab type	H5S_UNLIMITED in selection specification	offset	# of blocks in the selection
1	irregular	N/A	up to (2^32 – 1)	up to (2^32 – 1)
1	regular	no	up to (2^32 – 1)	up to (2^32 – 1)
2	regular	yes	up to (2^64 – 1)	N/A
2	regular	no	up to (2^64 – 1)	N/A
3	irregular	N/A	up to (2^64 – 1)	up to (2^64 – 1)

Table F: H5Sencode for hyperslab selection (proposed for latest format)

version	hyperslab type	H5S_UNLIMITED in selection specification	offset	# of blocks in the selection
3	irregular	N/A	up to (2^64 – 1)	up to (2^64 – 1)
3	regular	N/A	up to (2^64 – 1)	N/A

0.4 Internal library usage of selection encoding

2.4.1 Selection class callback

`H5S_select_class_t` (defined in H5Spkg.h) requires two functions, *`serial_size`* and *`serialize`*, that are used to encode dataspace selections. At present, these callbacks are used by virtual datasets, and by the region reference code. As both of these applications result in selection encodings that are incorporated into HDF5 files, the parameter lists of the function types of serial_size and serialize:

```
typedef herr_t (*H5S_sel_serial_size_func_t)(const H5S_t *space);
typedef herr_t (*H5S_sel_serialize_func_t)(const H5S_t *space,
uint8_t *pp);
```

must be augmented to allow access to the libver_bounds associated with the target file.

We propose that this be done by adding "`H5F_t *f`" to the parameter lists of both of these function types as follows:

```
typedef herr_t (*H5S_sel_serial_size_func_t)(const H5S_t *space,
H5F_t *f);
typedef herr_t (*H5S_sel_serialize_func_t)(const H5S_t *space,
uint8_t *pp, H5F_t *f);
```

and propagating the change through the code. This change gives these functions access to the libver_bounds information in the shared instance `H5F_file_t`, which is required to select the appropriate encoding as indicated by the tables referenced below:

- For point selection, see tables B and C in section 2.2.
- For hyperslab selection, see tables E and F in section 2.3

2.4.2 Property class callback

Instances of the structure type H5P_genprop_t (defined in H5Ppkg.h) are used to store a variety of information concerning properties. Of particular interest in this context is the encode field, which must be set to point to a function of type H5P_prp_encode_func_t (currently defined in H5Ppublic.h). This callback (along with the prp_decode callback, of type H5P_prp_decode_func_t) is used to encode and decode properties of the indicated type on the property list.

In the context of virtual storage layout property, this is a problem as H5P__dcrt_layout_enc() calls H5S_encode() to encode the selections associated with each constituent of the virtual dataset. If H5S_encode() is to respect the libver_bounds associated with the target file, the FAPL must be passed to it. However, at present, H5P_prp_encode_func_t does not support any way of passing the FAPL through. Further, as it is a public type, modifying it would seem to be awkward.

Fortunately, while H5P_prp_encode_func_t and H5P_prp_decode_func_t are currently defined in H5Ppublic.h, they are not in fact used to register new properties in either H5Pregister1() or H5Pregister2(). Instead, these callbacks are set to NULL for user defined property list entries.

Thus, we can square this circle by moving the declarations of H5P_prp_encode_func_t and H5P_prp_decode_func_t to H5Pprivate.h, and adding a udata field to H5P_prp_encode_func_t. The udata field will be NULL in most cases, but will point to an instance of H5P_enc_cb_ud_t whose sole field at present is an hid_t which will be set to the fapl_id to be passed through to H5S_encode().

The proposed revised definition of H5P_prp_encode_func_t and H5P_enc_cb_ud_t are shown below.

```
typedef herr_t (*H5P_prp_encode_func_t)(const void *value, void
**buf, size_t *size, void *udata);


typedef struct H5P_enc_cb_ud_t {
    hid_t fapl_id;
} H5P_enc_cb_ud_t;
```

As the actual selection encoding is done by `H5S_encode()`, the behavior is as described above for `H5Sencode()`.

Also, since the `H5S_encode()` calls in H5P__dcrt_layout_enc() are exclusively used by VDS (Virtual Data Sets) in 1.10, the `fapl_id` via `udata` from the encode callback is set to latest format so that the encoding is done as described in table 3 and table 4 for hyperslab and point selections.

Format of Dataspace Description

The following tables illustrate the format of the dataspace description including the new versions introduced in the previous section.

Layout: Dataspace Description for H5Sencode/H5Sdecode

Byte	**Byte**	**Byte**	**Byte**
Dataspace ID	Encode version	SizeSizeof_size	*This exists to align table nicely.*
Size of extent			
Dataspace message (variable size)			
Dataspace selection (variable size)			

Fields: Dataspace Description for H5Sencode/H5Sdecode

Field Name	Description
Dataspace ID	The dataspace message ID which is 1.
Encode version	H5S_ENCODE_VERSION which is 0.
Sizeof_size	The number of bytes used to store the size of an object
Size of extent	Size of the dataspace message
Dataspace message	The dataspace message information. See *The Format Specification* for the format of the dataspace message.
Dataspace selection	See *Dataspace Selection* below.

Layout: Dataspace Selection

Byte	Byte	Byte	Byte
Selection Type			
Selection Info (variable size)			

Fields: Dataspace Selection

Field Name	Description
Selection Type	There are 4 types of selection: Value — Description 0 — H5S_SEL_NONE: Nothing Selected 1 — H5S_SEL_POINTS: Sequence of points selected 2 — H5S_SEL_HYPER: Hyperslab selected 3 — H5S_SEL_ALL: Entire extent selected
Selection Info	Value — Description 0 — See *Selection Info for H5S_SEL_NONE* 1 — See *Selection Info for H5S_SEL_POINTS* 2 — See *Selection Info for H5S_SEL_HYPER* 3 — See *Selection Info for H5S_SEL_ALL*

Layout: Selection Info for H5S_SEL_NONE

Byte	**Byte**	**Byte**	**Byte**
Version			
Reserved (zero, 8 bytes)			

Fields: Selection Info for H5S_SEL_NONE

Field Name	**Description**
Version	The version number for the H5S_SEL_NONE Selection info. The value is 1.

Layout: Selection Info for H5S_SEL_POINTS

Byte	**Byte**	**Byte**	**Byte**
Version			
Points Selection Info (variable size)			

Fields: Selection Info for H5S_SEL_POINTS

Field Name	**Description**
Version	The version number for the H5S_SEL_POINTS Selection Info. The value is either 1 or 2.
Points Selection Info	Depending on *version*: Version — Description 1 — See *Version 1 Points Selection Info*. 2 — See *Version 2 Points Selection Info*.

Layout: Version 1 Points Selection Info

Byte	**Byte**	**Byte**	**Byte**
Reserved (zero)			
Length			
Rank			
Num Points (4 bytes)			
Point #1: coordinate #1 (4 bytes)			
:			
:			
Point #1: coordinate #u (4 bytes)			
:			
:			
:			
Point #n: coordinate #1 (4 bytes)			
:			
:			
Point #n: coordinate #u (4 bytes)			

Fields: Version 1 Points Selection Info

Field Name	**Description**
Length	The size in bytes from *Rank* to the end of the Selection Info.
Rank	The number of dimensions.
Num Points	The number of points in the selection
Point #n: coordinate #u	The array of points in the selection. The points selected are #1 to #n where n is *Num Points*. The list of coordinates for each point are #1 to #u where u is *Rank*.

Layout: Version 2 Points Selection Info

Byte	**Byte**	**Byte**	**Byte**
Size of Point Info	*This space inserted only to align table nicely*		
Rank			
Num Points (2, 4 or 8 bytes)			
Point #1: coordinate #1 (2, 4 or 8 bytes)			
: :			
Point #1: coordinate #u (2, 4 or 8 bytes)			
: : :			
Point #n: coordinate #1 (2, 4 or 8 bytes)			
: :			
Point #n: coordinate #u (2, 4 or 8 bytes)			

Fields: Version 2 Points Selection Info

Field Name	Description
Size of Point Info	The size for point info, which can be 2, 4 or 8 bytes.
Rank	The number of dimensions.
Num Points	The number of points in the selection. *size of point info* indicates the size of this field.
Point #n: coordinate #u	The array of points in the selection. The points selected are #1 to #n where n is *Num Points*. The list of coordinates for each point are #1 to #u where u is *Rank*. *size of point info* indicates the size of this field.

Layout: Selection Info for H5S_SEL_HYPER

Byte	Byte	Byte	Byte
Version			
Hyperslab Selection Info (variable size)			

Fields: Selection Info for H5S_SEL_HYPER

Field Name	Description
Version	The version number for the H5S_SEL_HYPER Selection Info. The value is 1, 2 or 3.
Hyperslab Selection Info	Depending on *version*: Version — Description 1 — See *Version 1 Hyperslab Selection Info* 2 — See *Version 2 Hyperslab Selection Info* 3 — See *Version 3 Hyperslab Selection Info*

Layout: Version 1 Hyperslab Selection Info

Byte	**Byte**	**Byte**	**Byte**
Reserved			
Length			
Rank			
Num Blocks (4 bytes)			
Start offset #1 for block #1 (4bytes)			
: :			
Start offset #n for block #1 (4 bytes)			
End offset #1 for block #1 (4 bytes)			
: :			
End offset #n for block #1 (4 bytes)			
: : :			
Start offset #1 for block #u (4 bytes)			
: :			
Start offset #n for block #u (4 bytes)			
End offset #1 for block #u (4 bytes)			
: :			
End offset #n for block #u (4 bytes)			

Fields: Version 1 Hyperslab Selection Info

Field Name	**Description**
Length	The size in bytes from the field *Rank* to the end of the Selection Info.
Rank	The number of dimensions in the dataspace.
Num Blocks	The number of blocks in the selection.
Start offset #n for block #u	The offset #n of the starting element in block #u. #n is from 1 to *Rank*. #u is from 1 to *Num Blocks* moving from the fastest changing dimension to the slowest changing dimension.
End offset #n for block #u	The offset #n of the ending element in block #u. #n is from 1 to *Rank*. #u is from 1 to *Num Blocks* moving from the fastest changing dimension to the slowest changing dimension.

Layout: Version 2 Hyperslab Selection Info

Byte	**Byte**	**Byte**	**Byte**
Flags	*This space inserted only to align table nicely*		
Length			
Rank			
Start #1 (8 bytes)			
Stride #1 (8 bytes)			
Count #1 (8 bytes)			
Block #1 (8 bytes)			
:			
:			
:			
Start #n (8 bytes)			
Stride #n (8 bytes)			
Count #n (8 bytes)			
Block #n (8 bytes)			

Fields: Version 2 Hyperslab Selection Info

Field Name	**Description**
Flags	This is a bit field with the following definition. Currently, this is always set to 0x1. Bit Description 0 If set, it is a regular hyperslab, otherwise, irregular.
Length	The size in bytes from the field *Rank* to the end of the Selection Info.
Rank	The number of dimensions in the dataspace.
Start #n	The offset of the starting element in the block. #n is from 1 to *Rank*.
Stride #n	The number of elements to move in each dimension. #n is from 1 to *Rank*.
Count #n	The number of blocks to select in each dimension. #n is from 1 to *Rank*.
Block #n	The size (in elements) of each block in each dimension. #n is from 1 to *Rank*.

Layout: Version 3 Hyperslab Selection Info

Byte	**Byte**	**Byte**	**Byte**
Flags	Size of Offset Info	*This space inserted only to align table nicely*	
Rank			
Offset Info (*variable size*)			

Fields: Version 3 Hyperslab Selection Info

Field Name	**Description**
Flags	This is a bit field with the following definition. Bit Description 0 If set, it is a regular hyperslab, otherwise irregular.
Size of Offset Info	The size for offset info, which can be 2, 4 or 8 bytes.
Rank	The number of dimensions in the dataspace.
Offset Info	Depending on *flag* value, see *offset info for version 3 regular* or *irregular hyperslab*.

Layout: Offset Info for Version 3 Regular Hyperslab

Byte	**Byte**	**Byte**	**Byte**
Start #1 (2, 4 or 8 bytes)			
Stride #1 (2, 4 or 8 bytes)			
Count #1 (2, 4 or 8 bytes)			
Block #1 (2, 4 or 8 bytes)			
: : :			
Start #n (2, 4 or 8 bytes)			
Stride #n (2, 4 or 8 bytes)			
Count #n (2, 4 or 8 bytes)			
Block #n (2, 4 or 8 bytes)			

Fields: Offset Info for Version 3 Regular Hyperslab

Field Name	Description
Start #n	The offset of the starting element in the block. #n is from 1 to *Rank*. *size of offset info* indicates the size of this field.
Stride #n	The number of elements to move in each dimension. #n is from 1 to *Rank*. *size of offset info* indicates the size of this field.
Count #n	The number of blocks to select in each dimension. #n is from 1 to *Rank*. *size of offset* info indicates the size of this field.
Block #n	The size (in elements) of each block in each dimension. #n is from 1 to *Rank*. *size of offset info* indicates the size of this field.

Layout: Offset Info for Version 3 Irregular Hyperslab

Byte	**Byte**	**Byte**	**Byte**
Num Blocks (2, 4 or 8 bytes)			
Start offset #1 for block #1 (2, 4 or 8 bytes)			
: :			
Start offset #n for block #1 (2, 4 or 8 bytes)			
End offset #1 for block #1 (2, 4 or 8 bytes)			
: :			
End offset #n for block #1 (2, 4 or 8 bytes)			
: : :			
Start offset #1 for block #u			
: :			
Start offset #n for block #u (2, 4 or 4 bytes)			
End offset #1 for block #u (2, 4 or 8bytes)			
: :			
End offset #n for block #u (2, 4 or 8 bytes)			

Fields: Offset Info for Version 3 Irregular Hyperslab

Field Name	Description
Num Blocks	The number of blocks in the selection. *size of offset info* indicates the size of this field.
Start offset #n for block #u	The offset #n of the starting element in block #u. #n is from 1 to *Rank*. #u is from 1 to *Num Blocks* moving from the fastest changing dimension to the slowest changing dimension. *size of offset info* indicates the size of this field.
End offset #n for block #u	The offset #n of the ending element in block #u. #n is from 1 to *Rank*. #u is from 1 to *Num Blocks* moving from the fastest changing dimension to the slowest changing dimension. *size of offset info* indicates the size of this field.

Layout: Selection Info for H5S_SEL_ALL

Byte	Byte	Byte	Byte
Version			
Reserved (zero, 8 bytes)			

Fields: Selection Info for H5S_SEL_ALL

Field Name	Description
Version	The version number for the H5S_SEL_ALL Selection info; the value is 1.

1)

Testing

New Tests

Add the following tests to *test/th5s.c*:

- *test_h5s_encode_exceed32()*
    - Verifies selection encoding that exceeds (2^32 – 1) bits is correctly encoded (HDFFV-9947) with old or new file format setting
- *test_h5s_encode_length*
    - Verifies that version 2 hyperslab selection info has encoded the "*length*" field correctly (HDFFV-10271)
- *test_h5s_encode_regular_hyper()*
    - Verifies that the encoding of regular hyperslabs work as specified in this RFC:
        - Set up regular hyperslabs so that version 1, 2 or 3 will be used for encoding
        - Set up regular hyperslabs so that 2, 4, or 8 bytes offset size will be used to encode the selection
- *test_h5s_encode_irregular_hyper*
    - Verifies that the encoding of irregular hyperslabs work as specified in this RFC:
        - Set up irregular hyperslabs so that version 1 or 3 will be used for encoding
- *test_h5s_encode_points*
    - Verifies that the encoding of point selection work as specified in this RFC:
        - Set up point selection so that version 1 or 2 will be used for encoding
        - Set up point selection so that 2, 4, or 8 bytes offset size will be used to encode the selection

Existing Tests

Modify the following tests:

- test/th5s.c
    - Modify *test_h5s_encode()* in the file to verify that *H5Sencode2()* encodes correctly with old and new file format setting.
- test/vds.c
    - Modify *test_api_get_ex_dcpl ()* in the file to encode property list with *H5Pencode2()*.
    - Modify tests in the file to run with old/new file format setting.

- test/trefer.c
    - Modify *test_reference_region()* and *test_reference_region_1D()* to run with old/new file format setting.
- enc_dec_plist.c
    - Modify tests in the file to run with old/new file format setting.

Documentation

Add reference manual entries for the new public routines *H5Sencode2 and H5Pencode2*.

Update reference manual entries for existing public routines *H5Sencode and H5Pencode*.

Update HDF5 Format Specification for *H5Sencode*.

Acknowledgements

Revision History

July 07, 2017	Version 0 – initial draft for discussion
July 17, 2017	Version 1—modifications after code review meeting
July 26, 2017	Version 2—modifications after discussion with John Mainzer
August 2, 2017	Version 3—modifications after review by John Mainzer and Neil Fortner
August 16, 2017	Version 4—add *H5Pencode2()*
September 18, 2017	Version 5—add testing information
October 27, 2017	Version 6—modifications by John: discussions for 1.10 and edits for clarity and completeness. Modifications made after review by Vailin.