RFC: Improvements for SWMR File Access and Dataset Append

Vailin Choi

This RFC describes the changes to the HDF5 library that improve the SWMR (Single-writer/multiple-read) file access model and provide better support for dataset append operation.

1 Introduction

The modifications described in this RFC cover improvements to two areas in the HDF5 library:

- The process in enabling SWMR writing for an open HDF5 file
  - A new public routine *H5Fstart_swrm_write()* to simplify the steps in setting up and enabling a file for SWMR writing
- The flush behavior when appending to a dataset and when flushing an HDF5 object
  - Two new public routines *H5Pget/set_append_flush()* to control when a dataset flush will occur during an append operation and to invoke an application callback function
  - Two new public routines *H5Pget/set_object_flush_cb()* to invoke an application callback function when an object flush occurs

2 Enhancement to SWMR file access

The SWMR file access model follows the standard HDF5 model: the writer and readers will need to indicate SWMR access using file access flags with the *H5Fcreate* and *H5Fopen* calls. To switch to SWMR-safe operations after creating/opening a file, a writer application has to close and reopen the file with SWMR access flags. To improve usability for the writer applications, the library will provide a new public routine, `H5Fstart_swmr_write`, to activate SWMR writing mode for an opened file.

The HDF5 library will use the file consistency flags in the file's superblock data structure (*status_flags* field in *struct H5F_super_t*) to mark a file as safe for SWMR writing. The marking will be removed upon file closing. Once the file is marked as SWMR-safe, a user cannot switch back to the previous mode. Also, a user cannot activate SWMR writing mode more than once for an opened file.

2.1 H5Fstart_swmr_write

Name:

`H5Fstart_swmr_write`

Signature:

herr_t `H5Fstart_swmr_write`*(hid_t* `file_id`*)*

Purpose:

Enables SWMR writing mode for a file.

Description:

`H5Fstart_swmr_write` will activate SWMR writing mode for a file associated with `file_id`. This routine will prepare and ensure the file is safe for SWMR writing as follows:

- Check that the file is open with write access (H5F_ACC_RDWR).
- Check that the file is open with the latest library format to ensure data structures with check-summed metadata are used.
- Check that the file is not already marked in SWMR writing mode.
- Enable reading retries for check-summed metadata to remedy possible checksum failures from reading inconsistent metadata on a system that is not atomic.
- Turn off usage of the library's accumulator to avoid possible ordering problem on a system that is not atomic.
- Perform a flush of the file's data buffers and metadata to set a consistent state for starting SWMR write operations.

Library objects are groups, datasets, and committed datatypes. For the current implementation, groups and datasets can remain open when activating SWMR writing mode, but not committed datatypes. Attributes attached to objects cannot remain open either.

Parameters:

hid_t `file_id`	IN: A file identifier.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example Usage:

The example below illustrates the usage of this routine to activate SWMR writing mode for an opened file.

```
/*
 *   The writer process
 */
/* Create a copy of the file access property list */
fapl_id = H5Pcreate(H5P_FILE_ACCESS);

/* Set to use the latest library format */
H5Pset_libver_bounds(fapl_id, H5F_LIBVER_LATEST, H5F_LIBVER_LATEST);

/* Create a file with the latest library format */
file_id = H5Fcreate(filename, H5F_ACC_TRUNC, H5P_DEFAULT, fapl_id);
:
:
:
/* Perform operations that are not SWMR-safe. */
:
:
```

```
:
/* Start a concurrent SWMR reader process (see coding below) at this point
   will fail because the file is not marked as SWMR-safe */

/* Enable SWMR writing mode */
H5Fstart_swmr_write(file_id);

/* Start a concurrent SWMR reader process (see coding below) at this point
   will succeed because the file is marked as SWMR-safe */


/* Perform SWMR-safe operations */
:
:
:
/* Close the file */
H5Fclose(file_id);

/* Close the property list */
H5Pclose(fapl_id);


/*
 *   The SWMR reader process
 */
read_file_id = H5Fopen(filename, H5F_ACC_RDONLY|H5F_ACC_SWMR_READ, fapl_id);

/* Perform reading operations */
:
:
:
/* Close the file */
H5Fclose(read_file_id);
```

3 Support for dataset append operation and object flush

The dataset append operation for SWMR write usually consists of extending the dataset's dataspace in a particular dimension and writes data elements to the newly extended region in the dataset. The high-level public routine *H5DOappend* condenses such dataspace and dataset write operations into a single function, thus eliminating much application code.

To provide flexibility for a user to manage the flush behavior of dataset elements during the append operation via *H5DOappend*, the following routines are provided to trigger actions on appends and flushes:

1) *H5Pget/set_append_flush()* for a dataset access property list
2) *H5Pget/set_object_flush_cb()* for a file access property list

Note that these routines will apply for both SWMR and non-SWMR access.

3.1 H5DOappend

Name:

H5DOappend

Signature:

```
herr_t H5DOappend(hid_t dset_id, hid_t dxpl_id, unsigned index, size_t
num_elem, hid_t memtype, const void *buffer)
```

Purpose:

Appends data to a dataset along a specified dimension.

Description:

The H5DOappend routine extends a dataset by num_elem number of elements along a dimension specified by a dimension index and writes buffer of elements to the dataset. Dimension index is 0-based. Elements' type is described by memtype.

This routine combines calling H5Dset_extent, H5Sselect_hyperslab and H5Dwrite into a single, routine that simplifies application development for the common case of appending elements to an existing dataset.

For multi-dimensional dataset, appending to one dimension will write a contiguous hyperslab over the other dimensions. For example, if a 3-D dataset has dimension sizes (3, 5, 8), extending the 0^{th} dimension (currently of size 3) by 3 will append 3*5*8 = 120 elements (which must be pointed to by the buffer parameter) to the dataset, making its final dimension sizes (6, 5, 8).

If a dataset has more than one unlimited dimension, any of those dimensions may be appended to, although only along one dimension per call to H5DOappend.

Parameters:

hid_t dset_id	IN: Dataset identifier.
hid_t dxpl_id	IN: Dataset transfer property list identifier.
unsigned index	IN: Dimension number (0-based)
size_t num_elem	IN: Number of elements to add along the dimension
hid_t memtype	IN: Memory type identifier
void *buffer	IN: Data buffer

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example Usage:

The example below for H5Pset_append_flush illustrates the usage of this public routine to append to a dataset.

3.2 H5Pset_append_flush

Name:

H5Pset_append_flush

Signature:

herr_t H5Pset_append_flush *(hid_t* dapl_id*,* int ndims*,* const hsize_t boundary[]*, H5D_append_cb_t* func*, void* *user_data*)*

Purpose:

Sets two actions to perform when the size of a dataset's dimension being appended reaches a specified boundary.

Description:

H5Pset_append_flush sets the following two actions to perform for a dataset associated with the dataset access property list dapl_id:

1) Call the callback function func set in the property list
2) Flush the dataset associated with the dataset access property list

While a user is appending data to a dataset via *H5DOappend* and the dataset's newly extended dimension size hits a specified boundary, the library will first perform action #1 listed above. Upon return from the callback function, the library will then perform the above action #2 and return to the user. If no boundary is hit or set, the two actions above are not invoked.

The specified boundary is indicated by the parameter boundary. It is a 1-dimensional array with ndims elements, which should be the same as the rank of the dataset's dataspace. While appending to a dataset along a particular dimension index via *H5DOappend*, the library determines a boundary is reached when the resulted dimension size is divisible by boundary[index]. A zero value for boundary[index] indicates no boundary is set for that dimension index.

The setting of this property will apply only for a chunked dataset with extendible dataspace. A dataspace is extendible when it is defined with either one of the following:

- Dataspace with fixed current and maximum dimension sizes
- Dataspace with at least one unlimited dimension for its maximum dimension sizes

When creating or opening a chunked dataset, the library will check whether the boundary as specified in the access property list is set up properly. The library will fail the dataset create or open when detecting the following conditions:

- ndims, the number of elements for boundary, is not the same as the rank of the dataset's dataspace.
- A non-zero boundary value is specified for a non-extendible dimension.

The callback function func must conform to the prototype defined below:

```
typedef herr_t (H5D_append_cb_t)(hid_t dataset_id, hsize_t *cur_dims, void *user_data)
```

where

dataset_id is the dataset identifier
cur_dims is the dataset's current dimension sizes when a boundary is hit

user_data is the user-defined input data.

Parameters:

hid_t dapl_id	IN: Dataset access property list identifier.
int ndims	IN: The number of elements for boundary.
hsize_t *boundary	IN: The dimension sizes used to determine the boundary.
H5D_append_cb_t func	IN: The user-defined callback function.
void *user_data	IN: The user-defined input data.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example Usage:

The example below illustrates the usage of this public routine to manage the flush behavior while appending to a dataset.

```
hsize_t dims[2] = {0, 100};
hsize_t max_dims[2] = {H5S_UNLIMITED, 100};
hsize_t boundary_dims[2] = {5, 0};
unsigned counter;
void *buf;
hid_t file_id;
hid_t dataset_id, dapl_id, type;

/* Open the file */
file_id = H5Fopen(FILE, H5F_ACC_RDWR|H5F_ACC_SWMR_WRITE, H5P_DEFAULT);

/* Create a copy of the dataset access property list */
dapl_id = H5Pcreate(H5P_DATASET_ACCESS);

/* Set up the append property values */
/* boundary_dims[0]=5: to invoke callback and flush every 5 lines */
/* boundary_dims[1]=0: no boundary is set for the non-extendible dimension */
/* append_cb: callback function to invoke when hitting boundary (see below) */
/* counter: user data to pass along to the callback function */
H5Pset_append_flush(dapl_id, 2, boundary_dims, append_cb, &counter);

/* DATASET is a 2-dimensional chunked dataset with dataspace:
   dims[] and max_dims[] */
dataset_id = H5Dopen2(file_id, "dataset", dapl_id);

/* Get the dataset's datatype */
type = H5Dget_type(dataset_id);

/* Append 50 lines along the unlimited dimension (index = 0) to the dataset */
for(n = 0; n < 50; n++) {

      /* Append 1 line to the dataset */
      /* Whenever hitting the specified boundary i.e., every 5 lines,
         the library will invoke append_cb() and then flush the dataset. */
      H5DOappend(dataset_id, H5P_DEFAULT, 0, 1, type, buf);
}
:
```

```
:
:
/* counter will be equal to 10 */
:
:
:

/* The callback function */
static herr_t
append_cb(hid_t dset_id, hsize_t *cur_dims, void *_udata)
{
    unsigned *count = (unsigned *)_udata;
    ++(*count++);
    return 0;
} /* append_cb() */
```

3.3 H5Pget_append_flush

Name:

H5Pget_append_flush

Signature:

herr_t H5Pget_append_flush *(hid_t* dapl_id*, int* ndims*, hsize_t* boundary[]*, H5D_append_cb_t* *func*, void* **user_data*)*

Purpose:

Retrieves the values of the append property that is set up in the dataset access property list.

Description:

H5Pget_append_flush obtains the following information from the dataset access property list dapl_id:

- boundary[]—the sizes set up in the access property list that is used to determine when a dataset dimension size hits the boundary. Only at most ndims boundary sizes are retrieved, and ndims will not exceed the corresponding value that is set in the property list.
- func—the user-defined callback function to invoke when a dataset's appended dimension size reaches a boundary.
- user_data—the user-defined input data for the callback function.

Parameters:

hid_t dapl_id	IN: Dataset access property list identifier.
int ndims	IN: The number of elements for boundary.
hsize_t *boundary[]	IN: The dimension sizes used to determine the boundary .
H5D_append_cb_t *func	IN: The user-defined callback function.
void **user_data	IN: The user-defined input data.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example Usage:

The example below illustrates the usage of this public routine to obtain the append property values that is set up in the dataset access property list.

```
hid_t file_id;
hid_t dapl_id, dataset_id, dapl;
hsize_t dims[2] = {0, 100};
hsize_t max_dims[2] = {H5S_UNLIMITED, 100};
hsize_t boundary_dims[2] = {5, 0};
int counter;
hsize_t ret_boundary[1];
H5D_append_flush_cb_t ret_cb;
void *ret_udata;

/* Open the file */
file_id = H5Fopen(FILE, H5F_ACC_RDWR|H5F_ACC_SWMR_WRITE, H5P_DEFAULT);

/* Create a copy of the dataset access property list */
dapl_id = H5Pcreate(H5P_DATASET_ACCESS);

/* Set up the append property values */
/* boundary_dims[0]=5: to invoke callback and flush every 5 lines */
/* boundary_dims[1]=0: no boundary is set for the non-extendible dimension */
/* append_cb: callback function to invoke when hitting boundary (see below) */
/* counter: user data to pass along to the callback function */
H5Pset_append_flush(dapl_id, 2, boundary_dims, append_cb, &counter);

/* DATASET is a 2-dimensional chunked dataset with dataspace:
   dims[] and max_dims[] */
dataset_id = H5Dopen2(file_id, "dataset", dapl_id);

/* Get the dataset access property list for DATASET */
dapl = H5Dget_access_plist(dataset_id);

/* Retrieve the append property values for the dataset */
/* Only 1 boundary size is retrieved: ret_boundary[0] is 5 */
/* ret_cb will point to append_cb() */
/* ret_udata will point to counter */
H5Pget_append_flush(dapl, 1, ret_boundary, &ret_cb, &ret_udata);
:
:
:

/* The callback function */
static herr_t
append_cb(hid_t dset_id, hsize_t *cur_dims, void *_udata)
{
    unsigned *count = (unsigned *)_udata;
    ++(*count++);
    return 0;
} /* append_cb() */
```

3.4 H5Pset_object_flush_cb

Name:

H5Pset_object_flush_cb

Signature:

herr_t H5Pset_object_flush_cb *(hid_t* fapl_id, *H5F_flush_cb_t* func, void *user_data*)*

Purpose:

Sets a callback function to invoke when an object flush occurs in the file.

Description:

H5Pset_object_flush_cb sets the callback function to invoke in the file access property list *fapl_id* whenever an object flush occurs in the file. Library objects are *group*, *dataset*, and *committed datatype*. When a user flushes an object via *H5Gflush/H5Dflush/H5Tflush/H5Oflush*, the library will flush the object, invoke the specified callback function, and then return to the user.

The callback function func must conform to the prototype defined below:

typedef *herr_t* (*H5F_flush_cb_t)(*hid_t* object_id, *void* *user_data)
where

object_id is the identifier of the object which has just been flushed
user_data is the user-defined input data for the callback function

Parameters:

hid_t fapl_id	IN: Identifier for a file access property list.
H5F_flush_cb_t func	IN: The user-defined callback function.
void *user_data	IN: The user-defined input data for the callback function.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example Usage:

The example below illustrates the usage of this routine to set the callback function to invoke when an object flush occurs.

```
hid_t file_id, fapl_id;
hid_t dataset_id, dapl_id;
unsigned counter;

/* Create a copy of the file access property list *
fapl_id = H5Pcreate(H5P_FILE_ACCESS);

/* Set up the object flush property values */
/* flush_cb: callback function to invoke when an object flushes (see below) */
/* counter: user data to pass along to the callback function */
H5Pset_object_flush_cb(fapl_id, flush_cb, &counter);

/* Open the file */
file_id = H5Fopen(FILE, H5F_ACC_RDWR, H5P_DEFAULT);
```

```
/* Create a group */
gid = H5Gcreate2(fid, "group", H5P_DEFAULT, H5P_DEFAULT_H5P_DEFAULT);

/* Open a dataset */
dataset_id = H5Dopen2(file_id, DATASET, H5P_DEFAULT);

/* The flush will invoke flush_cb() with counter */
H5Dflush(dataset_id);
/* counter will be equal to 1 */
:
:
:
/* The flush will invoke flush_cb() with counter */
H5Gflush(gid);
/* counter will be equal to 2 */
:
:
:

/* The callback function for object flush property */
static herr_t
flush_cb(hid_t obj_id, void *_udata)
{
    unsigned *flush_ct = (unsigned*)_udata;
    ++(*flush_ct);
    return 0;
}
```

3.5 H5Pget_object_flush_cb

Name:

H5Pset_object_flush_cb

Signature:

herr_t H5Pset_object_flush_cb *(hid_t* fapl_id*, H5F_flush_cb_t **func, void **user_data*)*

Purpose:

Retrieves the object flush property values from the file access property list.

Description:

H5Pget_object_flush_cb gets the user-defined callback function that is set in the file access property list *fapl_id* and stores in the parameter *func*. It also obtains the user-defined input data that is passed along to the callback function in the parameter *user_data*.

Parameters:

hid_t fapl_id	IN: Identifier for a file access property list.
H5F_flush_cb_t *func	IN: The user-defined callback function.
void **user_data	IN: The user-defined input data for the callback function.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example Usage:

The example below illustrates the usage of this routine to obtain the object flush property values.

```
hid_t fapl_id;
unsigned counter;
H5F_object_flush_t *ret_cb;
unsigned *ret_counter;

/* Create a copy of the file access property list */
fapl_id = H5Pcreate(H5P_FILE_ACCESS);

/* Set up the object flush property values */
/* flush_cb: callback function to invoke when an object flushes (see below) */
/* counter: user data to pass along to the callback function */
H5Pset_object_flush_cb(fapl_id, flush_cb, &counter);

/* Open the file */
file_id = H5Fopen(FILE, H5F_ACC_RDWR, H5P_DEFAULT);

/* Get the file access property list for the file */
fapl = H5Fget_access_plist(file_id);

/* Retrieve the object flush property values for the file */
H5Pget_object_flush_cb(fapl, &ret_cb, &ret_counter);
/* ret_cb will point to flush_cb() */
/* ret_counter will point to counter */
:
:
:

/* The callback function for the object flush property */
static herr_t
flush_cb(hid_t obj_id, void *_udata)
{
    unsigned *flush_ct = (unsigned*)_udata;
    ++(*flush_ct);
    return 0;
}
```

Acknowledgements

This work was supported by a customer of The HDF Group, Dectris.

Revision History

November 18, 2013:	Version 1 circulated for comment within The HDF Group SWMR team.
Jan 2, 2014	Version 2 updated based on implementation.
Jan 7, 2014	Version 3 posted on the SWMR FTP site
Jan 29, 2014	Version 4 added RM entry for H5DOaappend function; posted on SWMR FTP site

Feb 26, 2014	Version 5 updated to reflect implementation.

References

1. The HDF Group. “RFC: SWMR Requirements and Use Cases,” RFC-THG-2013-02-06.v8, ftp://ftp.hdfgroup.uiuc.edu/pub/outgoing/SWMR/doc/SWMR%20Use%20Cases-2013-03-13.pdf March 13, 2013.