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 opened 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 the file as safe for SWMR writing. The marking will be removed upon file closing. Once the file is marked as SWMR-safe, the user cannot switch back to the previous mode.

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

- Check that the file is opened with write access (H5F_ACC_RDWR).
- Check that the file is opened with the latest library format to ensure data structures with check-summed metadata are used.
- 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.

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 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 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 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. To provide flexibility and convenience for a user to manage the flush behavior of dataset elements during the append operation, the library will add the following routines 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 or non-SWMR access.

3.1 H5Pset_append_flush

Name:

H5Pset_append_flush

Signature:

herr_t H5Pset_append_flush *(hid_t* dapl_id*,* int ndims*,* hsize_t flush_dims[]*,* *H5D_append_flush_cb_t* func*,* *void* *user_data*)*

Purpose:

Sets two actions to perform when the dataset's dimension sizes reach 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`:

- Call the callback function `func` set in the property list
- Flush the dataset associated with the dataset access property list

The library will invoke the above actions when the dataset's newly extended dimension sizes resulted from an append operation reach the boundary specified by `flush_dims`. `flush_dims` is a 1-dimensional array with `ndims` elements, which should be the same as the rank of the dataset's dataspace. The library determines a boundary is reached when the dataset's current dimension sizes are divisible by flush_dims.

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

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

```
typedef herr_t (H5D_append_flush_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 flush is going to occur
`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 dimensions for flush_dims.
hsize_t *flush_dims	IN: The dimension sizes used to determine the boundary for the flush.
H5D_append_flush_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 the dataset *DATASET*.

```
hsize_t dims[2] = {0, 100};
hsize_t max_dims[2] = {H5S_UNLIMITED, 100};
```

```
hsize_t boundary_dims[2] = {5, 100};
hid_t file_id;
hid_t dataset_id, dapl_id;
int counter;

/* 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 the callback and the flush to perform when hitting the boundary */
H5Pset_append_flush(dapl_id, 2, boundary_dims, UpdateAttCallBack, &counter);

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

/* Keep the metadata items that are relevant to the dataset in cache */
H5Ocork(dataset_id, TRUE);

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

      /*
         Whenever hitting the specified boundary boundary_dims[], (i.e., every
         5 lines), the library will invoke UpdateAttCallback() and then
         flush the dataset.
       */
      H5DOappend(dataset_id, lineN);

:
:
:
```

3.2 H5Pget_append_flush

Name:

H5Pget_append_flush

Signature:

herr_t H5Pget_append_flush *(hid_t* dapl_id*,* int ndims*,* hsize_t flush_dims[]*,* *H5D_append_flush_cb_t* *func*,* *void* **user_data*)*

Purpose:

Retrieves the flush boundary and the callback function from the dataset access property list.

Description:

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

- flush_dims[]—the flush boundary specified by the user that the library will use to determine when a dataset boundary is reached; it is a 1-dimensional array with ndims elements.

- `func`—the user-defined callback function to invoke when a dataset boundary is reached.
- `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 dimensions for `flush_dims`
`hsize_t` `*flush_dims[]`	IN: The dimension sizes used to determine the boundary for the flush.
`H5D_append_flush_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 information that is set in the property list when hitting the dataset boundary.

```
hid_t file_id;
hid_t dapl_id, dataset_id, dapl;
hsize_t boundary_dims[2] = {5, 100};
int counter;
hsize_t ret_boundary[2];
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 the callback and the flush to perform when hitting the boundary */
H5Pset_append_flush(dapl_id, 2, boundary_dims, UpdateAttCallBack, &counter);

/* DATASET is a 2-dimensional dataset */
dataset_id = H5Dopen2(file_id, DATASET, dapl_id);

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

/* Retrieve the callback and the flush boundary */
H5Pget_append_flush(dapl, 2, ret_boundary, &ret_cb, &ret_udata);
:
:
:
:
```

3.3 H5Pset_object_flush_cb

Name:

H5Pset_object_flush_cb

Signature:

herr_t H5Pset_object_flush_cb *(hid_t* fapl_id*, H5F_object_flush_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 *named datatype*. Presently, only the *dataset* object has defined its flush operation in the library.

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

typedef *herr_t* (*H5F_object_flush_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_object_flush_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 a dataset 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 the object flush callback in the file access property list */
/* See NotifyReaderCallback() below */
H5Pset_object_flush_cb(fapl_id, NotifyReaderCallback, &counter);

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


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

```
/* This will invoke NotifyReaderCallback() with input data counter */
H5Dflush(dataset_id);
:
:
:
:
:
:

int NotifyReaderCallback(hid_t obj_id, void *udata)
{
:
:
:
}
```

3.4 H5Pget_object_flush_cb

Name:

H5Pset_object_flush_cb

Signature:

herr_t H5Pset_object_flush_cb *(hid_t* fapl_id*, H5F_object_flush_t **func*, void **user_data*)*

Purpose:

Retrieves the user-defined callback function for an object flush.

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*. The callback is invoked whenever an object flush occurs in the file. This routine also obtains the user-defined input data for the callback in the parameter *user_data*.

Parameters:

hid_t fapl_id	IN: Identifier for a file access property list.
H5F_object_flush_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 user-defined callback function and input data.

```
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 the object flush callback in the file access property list */
/* See NotifyReaderCallback below */
H5Pset_object_flush_cb(fapl_id, NotifyReaderCallback, &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);

/* Get the object flush callback and user data from the file access
   property list:
      ret_cb will point to NotifyReaderCallback
      ret_counter will point to counter
*/
H5Pget_object_flush_cb(fapl, &ret_cb, &ret_counter);
:
:
:


int NotifyReaderCallback(hid_t obj_id, void *udata)
{
:
:
:
}
```

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.
November 26, 2013	Version 2 sent to DLS and posted on FTP

Appendix A:

References

1. The HDF Group. "RFC: SWMR Requirements and Use Cases," http://confluence.hdfgroup.uiuc.edu/pages/viewpage.action?pageId=25100365 (February 19, 2013).