Date: **June 30, 2014** **Delivered as part of Milestones 8.1, 8.2, 8.5**	***User's Guide to FastForward Features in HDF5*** **FOR EXTREME-SCALE COMPUTING RESEARCH AND DEVELOPMENT (FAST FORWARD) STORAGE AND I/O**

LLNS Subcontract No.	B599860
Subcontractor Name	Intel Federal LLC
Subcontractor Address	2200 Mission College Blvd. Santa Clara, CA 95052

Revision History

Date	Revision	Description	Authors
03/22/2013	1.0	Delivered to DOE stakeholders as part of Milestone 3.3 demonstration output.	Ruth Aydt, Mohamad Chaarawi, Quincey Koziol, Jerome Soumagne
06/18/2013	2.0	Delivered to DOE stakeholders as part of Milestones 4.2, 4.3, 4.4 demonstration output.	Ruth Aydt, Mohamad Chaarawi, Quincey Koziol, Jerome Soumagne
9/26/2013	3.0	• Updated Introduction section for Q5. • Changed placeholder ID to future ID. • Indicated that H5EQ and H5AO routines will be consolidated into H5ES; Added H5ES section and man pages. • Removed H5DO section. • Added H5RC and H5TR sections. • Added several new routines to H5P. • Updated RM section tables and man pages for Q5. See section table notes and History on man pages for details. • Updated instructions on how to build the EFF stack. • Updated test sections with the new testing framework. • Removed Appendix with Q4 example code. Did not include Q5 example code as it is much larger; see source distribution. • Delivered to DOE stakeholders as part of Milestones 5.6, 5.7 demonstration output.	Ruth Aydt, Mohamad Chaarawi, Quincey Koziol
12/20/2013	4.0.2	• Removed H5EQ and H5AO APIs – now consolidated into H5ES • Updated tables at beginning of each API section to reflect current status. • Updated list of example client programs and build instructions. • Fixed typos • Delivered to DOE stakeholders as part of Milestone 6.2, 6.3 demonstration output	Ruth Aydt, Mohamad Chaarawi
12/20/2013	4.0.3	• Added H5Qcreate/combine/close • Added H5ASexecute	Quincey Koziol, Jerome Soumagne
12/23/2013	4.0.4	• Final edit pass • Date on title page reconciled and highlight in section 1 removed without version change on 1/4/14.	Quincey Koziol
02/07/2104	5.1	• Update data markings and copyright year • Begin changes for Quarter 7 milestones • Added H5TRget_trans_num and H5TRget_version	Ruth Aydt
02/26/2014	5.2	• Added EFF API reference manual pages.	Ruth Aydt, Mohamad

		(EFF_finalize, EFF_init, EFF_start_server) • Add note to H5Dopen about using object id to access other CVs.	Chaarawi
03/10/2014	5.3	• Added persist flag to H5Fclose_ff	Mohamad Chaarawi
03/30/2014	5.4	• Added man pages for prefetch and evict routines and related properties. • Noted that H5V and H5X reference manual pages will be added in Q8, with pointer to draft versions in the design doc. • Updated list of client examples and build instructions. • Updated status in Section 4.x tables • Deliver to DOE as part of Q7 milestones.	Ruth Aydt, Mohamad Chaarawi
04/10/2014	6.0	• Base version for Q8.	Ruth Aydt
5/6/2014	6.1	• Added H5O*token man pages. • Add open issue to H5RCsnapshot • Add H5Aevict_ff man page; update H5*evict_ff to reflect Q8 work.	Mohamad Chaarawi, Ruth Aydt
5/22/2014	6.2	• Update H5RCacquire with Limitations and Open Issues.	Ruth Aydt
6/26/2014	6.3	• Update Introduction for final quarter. • Small corrections throughout document. • Added sections for H5V & H5X; Added missing entries in section tables throughout. • Added section for Python work. • Added H5Pset_ocpl_enable_checksum	Ruth Aydt
6/27/2014	6.4	• Add templates and some content for H5V*, H5Lget_info_ff, H5Lget_val_ff, H5P* entries completed.	Ruth Aydt, Mohamad Chaarawi
6/28/2014	6.5	• Added templates and some content for H5Q*. Misc updates elsewhere.	Ruth Aydt
6/30/2014	6.6	• Moved the index plugin section to this document (from the Design Doc), along with a lot of cleanup on its formatting. • Updated H5X*; reviewed H5L*	Quincey Koziol, Mohamad Chaarawi
6/30/2014	6.7	• Formatting updates;	Ruth Aydt
6/30/2014	6.8	• Updated H5V*; working on H5Q*	Quincey Koziol
6/30/2014	6.9	• Updated H5Q*; Added H5Dcreate_anon_ff and H5Dquery_ff; Added H5Aprefetch_ff; Alphabetized H5P* routines; Added missing H5P* routines; Added instructions for building the Python wrappers and test suite; Updated H5AS*; Final polishing pass; • Delivered to DOE stakeholders as part of Milestone 8.1/8.2/8.5 demonstration output.	Quincey Koziol

1 Introduction

This document is a deliverable for all of these Quarter 8 Milestones:

- Milestone 8.1: HDF5 Index Demonstration
- Milestone 8.2: Advanced Analysis Shipping Demonstration
- Milestone 8.5: End-to-End Demonstration with Final Design Documentation and Report

The document reflects the status of the software stack at the completion of the Extreme-Scale Computing Research and Development (Fast Forward) Storage and I/O prototype project.

This document provides reference manual pages for API routines, high-level descriptions of example programs, and instructions for building and running both the HDF5 and example packages in the demonstration environment. Separate documents provided over the course of the project contain additional program notes for the Milestone demo programs. Usage guidance is included in the reference manual section overviews, the reference manual pages, the design document, project presentations, and example program comments.

2 Definitions

ACG – Arbitrarily Connected Graphs

AXE – Asynchronous eXecution Engine

CN – Compute Node

EFF – Exascale Fast Forward

FS – Function Shipper, also known as Mercury

IOD – I/O Dispatcher

ION – I/O Node

Mercury – Name of Function Shipper

VOL – Virtual Object Layer

3 EFF API reference manual pages

This section contains the reference manual pages for new routines that are part of the Exascale Fast Forward I/O stack, but that are not specifically related to HDF5. These routines are used to start the Asynchronous eXecution Engine (AXE) and the function shipper (Mercury) server and clients – components that provide the foundation for the HDF5-IOD VOL plugin operations in the EFF stack.

3.1 Startup and Shutdown APIs

These routines start and stop components of the EFF stack such as the Asynchronous eXecution Engine (AXE) and the function shipper (Mercury).

Routine	Implemented	Notes
EFF_finalize	Quarter 4	
EFF_init	Quarter 4	
EFF_start_server	Quarter 4	

3.1.1 EFF_finalize

Name: EFF_finalize

Signature:

herr_t `EFF_finalize(` *void* `)`

Purpose:

Shut down the EFF stack.

Description:

`EFF_finalize` shuts down the EFF stack, stopping the Asynchronous eXecution Engine (AXE), finalizing IOD, and shutting down the function shipper (Mercury) that were started by a call to EFF_init.

The application must call `EFF_finalize` after all calls to the HDF5 routines had been completed. The normal sequence of calls at the end of an application running on the EFF stack is:

```
MPI_Barrier( MPI_COMM_WORLD );
EFF_finalize();
MPI_Finalize();
```

Parameters:

none

Returns:

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

History:

Routine added in Quarter 4.
Man page added in Quarter 7.

Man Page Status:

No known issues.

3.1.2 EFF_init

Name: EFF_init

Signature:

herr_t `EFF_init(` *MPI_Comm* `comm`, *MPI_Info* `info`)

Purpose:

Initialize the EFF stack.

Description:

`EFF_init` initializes the EFF stack by setting up the connection to the Mercury server running the HDF5 VOL server, so that the HDF5 IOD VOL plugin at the client side can transparently forward requests to it. The call also triggers the initialization of the IOD library.

The application must call `EFF_init` after the function shipping (Mercury) server has been started and before the application makes any calls to the HDF5 routines.

`comm` is the MPI communicator for the compute processes.

`info` passes hints to the MPI library. `MPI_INFO_NULL` can be used if no hints need to be passed. .

Parameters:

MPI_Comm `comm`	IN: MPI communicator
MPI_Info `info`	IN: MPI info.

Returns:

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

History:

Added in Quarter 4.
Man page added in Quarter 7.

Man Page Status:

No known issues.

3.1.3 EFF_start_server

Name: EFF_start_server

Signature:

herr_t `EFF_start_server(` *MPI_Comm* `comm`, *MPI_Info* `info`)

Purpose:

Start the function shipper server that is used with the EFF stack.

Description:

`EFF_start_server` starts the function shipper server that listens for incoming connections from clients. Once a connection has been established, the client will transparently forward calls to the server it has connected to.

`EFF_start_server` must be called in the server code before the user's application calls `EFF_init`.

`comm` is the MPI communicator for the compute processes.

`info` passes hints to the MPI library. `MPI_INFO_NULL` can be used if no hints need to be passed.

Parameters:

MPI_Comm `comm`	IN: MPI communicator
MPI_Info `info`	IN: MPI info.

Returns:

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

History:

Added in Quarter 4.
Man page added in Quarter 7.

Man Page Status:

No known issues.

4 HDF5 API reference manual pages

This section contains the reference manual pages for the modified and new HDF5 API routines that are part of the Fast Forward extensions to HDF5.

Tables at the beginning of each section also note routines that were implemented in the HDF5 IOD VOL client/server as part of the Fast Forward extensions to HDF5 for which the user interface is unchanged. Readers are directed to the standard HDF5 man pages, found at http://www.hdfgroup.org/HDF5/doc/RM/RM_H5Front.html, for those routines.

4.1 H5: General HDF5 APIs

These routines serve general-purpose needs of the HDF5 library and its users.

Routine	Implemented	Notes
H5checksum	Quarter 4	New for EFF.

4.1.1 H5checksum

Name: H5checksum

Signature:

uint32_t `H5checksum(` *const void* `*buf,` *hsize_t* `length`, *H5_checksum_seed_t* `*cseed )`

Purpose:

Generate a checksum.

Description:

`H5checksum` generates a checksum for the data in `buf` with size `length` bytes.

`H5checksum` will generate the same checksum for identical data regardless of whether the data is stored in a single contiguous buffer or in multiple non-continguous buffers. For non-contiguous buffers, `H5checksum` must be called once for each buffer to "accumulate" the checksum for the complete data.

The checksum seed structure is defined as follows:

```
typedef struct H5_checksum_seed_t {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    int32_t state;
    size_t total_length;
} H5_checksum_seed_t;
```

If the checksum is for a data in a contiguous buffer, call `H5checksum` with `NULL` for `cseed` parameter.

Otherwise, for checksumming a set of non-contiguous regions that will be passed in a single call to `H5Dwrite_ff`, `cseed` captures the internal state of the checksum generation algorithm, allowing a single checksum to be generated for the set of non-contiguous data that is identical to the checksum that would be generated if the same data was checksummed in one contiguous block.

When creating a checksum for a set of non-contiguous buffers, `H5checksum` should be called on each contiguous portion of the buffer with `length` set to that portion's corresponding size in bytes. The `a, b, c,` and `state` fields in the checksum seed structure should be initialized to `0` for the first call to `H5checksum`, and `total_length` should be set to the total size in bytes of all the data to be checksummed (over all the non-contiguous sections of the data to checksum). Each call to `H5checksum` updates fields in `cseed` to capture the current internal state of the parameters used to compute the checksum and returns the updated checksum of the entire data that have been passed in so far to the routine with the same `cseed` parameter. The return values of the intermediate calls to `H5checksum` for a non-contiguous buffer can be discarded; only the last returned value is used as the checksum for the non-contiguous data. By using the same `cseed` structure in subsequent calls to `H5checksum` for each of the non-continguous buffers, the overall checksum is computed step-by-step.

The generated checksum can be used with `H5Pset_dxpl_checksum` to attach a checksum to `H5Dwrite_ff` and `H5Mput` transfers and to verify the checksum returned by `H5Dread_ff` and `H5Mget` in the buffer specified by `H5Pset_dxpl_checksum_ptr.`

See *Design and Implementation of FastForward Features in HDF5* for information on the interaction between transformation operations, such as datatype conversion, and end-to-end integrity support with checksums.

Parameters:

const void `*buf`	IN: Buffer with data to be checksummed

hsize_t `length`	IN: Length, in bytes, of the data in `buf`
H5_checksum_seed_t `*cseed`	IN/OUT: Checksum state structure used to seed each invocation of `H5checksum`. Use `NULL` if data is in a contiguous buffer. If the data is in multiple non-contiguous buffers, use the same `cseed` structure for each call to `H5checksum,` which will be called once per buffer.

Returns:

Returns the checksum. In the case of checksumming data stored in non-contiguous buffers, the checksum returned for all calls but the last can be discarded

History:

Added in Quarter 4.
Quarter 5: Added note about interactions between checksums and data transformations.

Man Page Status:

No known issues.

4.2 H5A: Attribute APIs

These routines are used to operate on HDF5 Attribute Objects.

The routines ending in _ff have different signatures than the standard HDF5 library routines.

Man pages for routines whose user interface is unchanged from the standard HDF5 implementation can be found at: http://www.hdfgroup.org/HDF5/doc/RM/RM_H5A.html.

Routine	**Implemented**	**Notes**
H5Aclose_ff	Quarter 4	
H5Acreate_ff	Quarter 4	
H5Acreate_by_name_ff	Quarter 4	
H5Adelete_ff	Quarter 4	
H5Adelete_by_name_ff	Quarter 4	
H5Aevict_ff	Quarter 8	
H5Aexists_ff	Quarter 4	
H5Aexists_by_name_ff	Quarter 4	
H5Aopen_ff	Quarter 4	
H5Aopen_by_name_ff	Quarter 4	
H5Aprefetch_ff	Quarter 8	
H5Aread_ff	Quarter 4	
H5Arename_ff	Quarter 4	
H5Arename_by_name_ff	Quarter 4	
H5Awrite_ff	Quarter 4	
H5Aget_create_plist	Quarter 4	See standard HDF5 man page
H5Aget_name	Quarter 4	See standard HDF5 man page
H5Aget_space	Quarter 4	See standard HDF5 man page
H5Aget_type	Quarter 4	See standard HDF5 man page
H5Aget_info_ff		Not implemented in Prototype
H5Aget_info_by_name_ff		Not implemented in Prototype
H5Aiterate_ff		Not implemented in Prototype
H5Aiterate_by_name_ff		Not implemented in Prototype
H5Aget_storage_size_ff		Not implemented in Prototype
H5Adelete_by_idx		Not implemented in Prototype
H5Aget_info_by_idx		Not implemented in Prototype

H5Aget_name_by_idx		Not implemented in Prototype
H5Aopen_by_idx		Not implemented in Prototype

4.2.1 H5Aclose_ff

Name: H5Aclose_ff

Signature:

herr_t `H5Aclose_ff(` *hid_t* `attr_id`, *hid_t* `es_id )`

Purpose:

Close the specified attribute, possibly asynchronously.

Description:

`H5Aclose_ff` terminates access to the attribute specified by `attr_id` by releasing the identifier.

Further use of a released attribute identifier is illegal; a function using such an identifier will fail.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `attr_id`	IN: Identifier of the attribute to release access to.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from event queue to event stack.

Man Page Status:

No known issues.

4.2.2 H5Acreate_ff

Name: H5Acreate_ff

Signature:

hid_t `H5Acreate_ff(` *hid_t* `loc_id`, *const char* `*attr_name`, *hid_t* `type_id`, *hid_t* `space_id`, *hid_t* `acpl_id`, *hid_t* `aapl_id`, *hid_t* `trans_id`, *hid_t* `es_id )`

Purpose:

Create an attribute attached to the specified object, possibly asynchronously.

Description:

`H5Acreate_ff` creates an attribute, `attr_name`, which is attached to the object specified by the identifier `loc_id`. `loc_id` must be in scope for the transaction identified by `trans_id`.

The attribute name, `attr_name`, must be unique for the object.

The attribute is created with the specified datatype and dataspace, `type_id` and `space_id`, which are created with the H5T and H5S interfaces, respectively.

If `type_id` is either a fixed-length or variable-length string, it is important to set the string length when defining the datatype. String datatypes are derived from `H5T_C_S1`, which defaults to 1 character in size. See `H5Tset_size` and "Creating variable-length string datatypes."

The attribute creation and access property lists, `acpl_id` and `aapl_id`, are currently unused, but will be used in the future for optional attribute creation and access properties. These property lists should currently be `H5P_DEFAULT`.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The attribute identifier returned by this function must be released with `H5Aclose_ff` or resource leaks will develop.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier that is in scope for the transaction. If `loc_id` is a file identifier, the attribute will be attached to that file's root group.
const char `*attr_name`	IN: Attribute name
hid_t `type_id`	IN: Attribute datatype identifier
hid_t `space_id`	IN: Attribute dataspace identifier
hid_t `acpl_id`	IN: Attribute creation property list . *Currently not used; specify* ***`H5P_DEFAULT`.***
hid_t `aapl_id`	IN: Attribute access property list *Currently not used; specify* ***`H5P_DEFAULT`.***

hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns an attribute identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the new attribute is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a "normal" attribute identifier.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement and map object.

Man Page Status:

No known issues.

4.2.3 H5Acreate_by_name_ff

Name: H5Acreate_by_name_ff

Signature:

hid_t `H5Acreate_by_name_ff` (*hid_t* `loc_id`, *const char* *`obj_name`, *const char* *`attr_name`, *hid_t* `type_id`, *hid_t* `space_id`, *hid_t* `acpl_id`, *hid_t* `aapl_id`, *hid_t* `lapl_id`, *hid_t* `trans_id`, *hid_t* `es_id`)

Purpose:

Create an attribute attached to the specified object, possibly asynchronously.

Description:

`H5Acreate_by_name_ff` creates an attribute, `attr_name`, which is attached to the object specified by `loc_id and obj_name`.

`loc_id` is a location identifier; `obj_name` is the path to the object relative to `loc_id`. If `loc_id` fully specifies the object to which the attribute is to be attached, `obj_name` should be '.' (a dot). Both `loc_id and obj_name` must be in scope for the transaction identified by `trans_id`.

The attribute name, `attr_name`, must be unique for the object.

The attribute is created with the specified datatype and dataspace, `type_id` and `space_id`, which are created with the H5T and H5S interfaces, respectively.

The attribute creation and access property lists, `acpl_id` and `aapl_id`, are currently unused, but will be used in the future for optional attribute creation and access properties. These property lists should currently be `H5P_DEFAULT`.

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See "Link Access Properties" in the H5P APIs. The link access property list currently has no effect in the EFF stack and should be set to `H5P_DEFAULT`.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The attribute identifier returned by this function must be released with `H5Aclose_ff` or resource leaks will develop.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier that is in scope for the transaction.
const char *`obj_name`	IN: Object name The object name (path to the object) can be specified relative to loc_id, absolute from the file's root group, or '.' (a dot), and must be in scope for the transaction.
const char *`attr_name`	IN: Attribute name
hid_t `type_id`	IN: Attribute datatype identifier

hid_t `space_id`	IN: Attribute dataspace identifier
hid_t `acpl_id`	IN: Attribute creation property list *Currently not used; specify* *`H5P_DEFAULT.`*
hid_t `aapl_id`	IN: Attribute access property list *Currently not used; specify* *`H5P_DEFAULT.`*
hid_t `lapl_id`	IN: Link access property list *Currently not used in EFF; specify* *`H5P_DEFAULT.`*
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution

Returns:

Returns an attribute identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the new attribute is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a "normal" attribute identifier.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement and map object. Noted lapl not used in EFF.

Man Page Status:

No known issues.

4.2.4 H5Adelete_ff

Name: H5Adelete_ff

Signature:

herr_t `H5Adelete_ff(` *hid_t* `loc_id,` *const char* `*attr_name,` *hid_t* `trans_id,` *hid_t* `es_id )`

Purpose:

Delete an attribute from a specified location, possibly asynchronously.

Description:

`H5Adelete_ff` removes the attribute specified by its name, `attr_name`, from the object specified by `loc_id`. This function should not be used when attribute identifiers are open on `loc_id` as it may cause the internal indexes of the attributes to change and future writes to the open attributes to produce incorrect results. `loc_id` must be in scope for the transaction identified by `trans_id`.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `loc_id`	May be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier that is in scope for the transaction. If `loc_id` is a file identifier, the attribute will be deleted from that file's root group.
const char `*attr_name`	IN: Name of the attribute to delete
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement and map object.

Man Page Status:

No known issues.

4.2.5 H5Adelete_by_name_ff

Name: H5Adelete_by_name_ff

Signature:

herr_t `H5Adelete_by_name_ff(` *hid_t* `loc_id,` *const char* `*obj_name,` *const char* `*attr_name,` *hid_t* `lapl_id`, *hid_t* `trans_id,` *hid_t* `es_id )`

Purpose:

Delete an attribute from a specified location, possibly asynchronously.

Description:

`H5Adelete_by_name_ff` removes the attribute `attr_name` from an object specified by `loc_id` and `obj_name`.

`loc_id` is a location identifier; `obj_name` is the path to the object relative to `loc_id`. If `loc_id` fully specifies the object from which the attribute is to be deleted, `obj_name` should be '.' (a dot). Both `loc_id and obj_name` must be in scope for the transaction identified by `trans_id`.

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See "Link Access Properties" in the H5P APIs. The link access property list currently has no effect in the EFF stack and should be set to `H5P_DEFAULT`.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `loc_id`	IN: May be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier that is in scope for the transaction.
const char `*obj_name`	IN: Name of object from which attribute is to be removed. The object name (path to the object) can be specified relative to loc_id, absolute from the file's root group, or '.' (a dot), and must be in scope for the transaction.
const char `*attr_name`	In: Name of attribute to delete
hid_t `lapl_id`	IN: Link access property list *Currently not used in EFF; specify `H5P_DEFAULT`.*
hid_t `trans_id`	IN: Value used to indicate transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.

Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement and map object. Noted lapl not used in EFF.

Man Page Status:

No known issues.

4.2.6 H5Aevict_ff

Name: H5Aevict_ff

Signature:

herr_t `H5Aevict_ff(` *hid_t* `attr_id,` *uint64_t* `container_version,` *hid_t* `dxpl_id,` *hid_t* `es_id )`

Purpose:

Evict attribute from the burst buffer, possibly asynchronously.

Description:

`H5Aevict_ff` evicts data associated with an attribute from the burst buffer.

The data to be evicted may be resident in the burst buffer under two different scenarios

In the first scenario, the data is resident in the burst buffer as the result of updates to the attribute made via the EFF transaction model. For example, through a call to H5Awrite_ff. This call adds updates to a transaction that are atomically applied to the attribute when the transaction is committed, and we refer to this data as *transaction update data*.

When evicting transaction update data, the container version being evicted should first be persisted to permanent storage (DAOS), with the `H5RCpersist` command. The attribute's transaction update data for the specified container version, as well as the attribute's transaction update data for all lower-numbered container versions that has not yet been evicted from the burst buffer, will be evicted as the result of this call. If evicting the data would result in container versions with open read contexts becoming inaccessible, the evict will fail.

In the second scenario, the data is resident in the burst buffer as the result of a call to H5Aprefetch_ff. This call replicates data from persistent storage (DAOS) to the burst buffer, and we refer to this data as *replica data*. When replica data is evicted, only data in the burst buffer as a result of the exact replica specified is evicted – transaction update data and other replicas for the group remain in the burst buffer.

`The attr_id` parameter specifies the attribute whose data is to be evicted.

The version of the attribute to be evicted is specified by the `container_version` property.

The replica property in the transfer property list, `dxpl_id,` is used to specify that an attribute replica is to be evicted. `H5Pset_dxpl_replica()` sets the replica property. If this is set, then replia data will be evicted, otherwise transaction update data will be evicted.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Limitations / Future Considerations:

When evicting a replica, the `container_version` argument is redundant, as the replica identifier fully specifies the data to be evicted. Consider revisiting and possibly revising the API prior to production release. Possibly have separate evict commands for eviction of transaction update data and of replicas.

For other potential extensions that are beyond the scope of the EFF prototype project, refer to the document *Burst Buffer Space Management – Prototype to Production*.

Parameters:

hid_t `attr_id`	IN: Identifier of the attribute being evicted.

uint64_t `container_version`	IN: Container version specifying what version of the attribute to evict.
hid_t `aapl_id`	IN: Identifier of an access property list. If the access property list contains an evict replica property (set via `H5Pset_evict_replica()`), then the replica_id specified by that property will be evicted.
hid_t `es_id`	IN: The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.2.7 H5Aexists_ff

Name: H5Aexists_ff

Signature:

herr_t `H5Aexists_ff(` *hid_t* `loc_id,` *const char* `*attr_name`, *hbool_t* `*exists`, *hid_t* `rcntxt_id`, *hid_t* `es_id )`

Purpose:

Determine whether an attribute with a given name exists for an object, possibly asynchronously.

Description:

`H5Aexists_ff` determines whether the attribute, `attr_name`, exists for the object specified by `loc_id`. `loc_id` must be in scope for the read context identified by `rcntxt_id`.

The results of the existence test are put in `exists`. Note that this value will not be set until after the function completes, which may be later than when the call returns for asynchronous execution. Traditionally the existence results were indicated by the function call's return value, but with the support for asynchronous execution that is no longer possible and the FastForward version included the new `exists` parameter.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `loc_id`	IN: May be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier that is in scope for the read context.
const char `*attr_name`	IN: Attribute name
hbool_t `*exists`	OUT: Pointer to returned results indicating existence of attribute. When successful, will be a positive value to indicate the attribute exists and 0 (zero) if the attribute does not exist. The value pointed to will not be modified if the existence test failed for some reason.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. When the asynchronous execution completes successfully, `exists` will contain the existence test results.

History:

Added in Quarter 4.

Quarter 5: Changed from obj_id to loc_id, from transaction to read context id, and from event queue to event stack. Added scope requirement and map object. Changed from htri_t to hbool_t for exists parameter type and updated description of value returned on test failure.

Man Page Status:

No known issues.

4.2.8 H5Aexists_by_name_ff

Name: H5Aexists_by_name_ff

Signature:

herr_t `H5Aexists_by_name_ff(` *hid_t* `loc_id,` *const char* `*obj_name,` *const char* `*attr_name`, *hid_t* `lapl_id`, *hbool_t* `*exists`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Determine whether an attribute with a given name exists for an object, possibly asynchronously.

Description:

`H5Aexists_ff` determines whether the attribute, `attr_name`, exists for the object specified by `loc_id` and `obj_name`.

`loc_id` is a location identifier; `obj_name` is the path to the object relative to `loc_id`. If `loc_id` fully specifies the object the attribute may be attached to, `obj_name` should be `'.'` (a dot). Both `loc_id and obj_name` must be in scope for the read context identified by `rcntxt_id`.

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See "Link Access Properties" in the H5P APIs. The link access property list currently has no effect in the EFF stack and should be set to `H5P_DEFAULT`.

The results of the existence test are put in `exists`. Note that this value will not be set until after the function completes, which may be later than when the call returns for asynchronous execution. Traditionally the existence results were indicated by the function call's return value, but with the support for asynchronous execution that is no longer possible and the FastForward version included the new `exists` parameter.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier that is in scope for the read context.
const char `*obj_name`	IN: Object name The object name (path to the object) can be specified relative to loc_id, absolute from the file's root group, or '.' (a dot), and must be in scope for the read context.
const char `*attr_name`	IN: Attribute name
hid_t `lapl_id`	IN: Link property access list *Currently not used in EFF; specify* `H5P_DEFAULT.`
hbool_t `*exists`	OUT: Pointer to returned results indicating existence of attribute. When successful, will be a positive value to indicate the attribute exists and 0 (zero) if the attribute does not exist. The value pointed to will not be modified if the existence test failed for some reason.

hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. When the asynchronous execution completes successfully, `exists` will contain the existence test results.

History:

Added in Quarter 4.
Quarter 5: Changed from obj_id to loc_id, from transaction to read context id, and from event queue to event stack. Added scope requirement and map object. Noted lapl currently not used in EFF. Changed from htri_t to hbool_t for exists parameter type and updated description of value returned on test failure.

Man Page Status:

No known issues.

4.2.9 H5Aopen_ff

Name: H5Aopen_ff

Signature:

hid_t `H5Aopen_ff`(*hid_t* `loc_id`, *const char* `*attr_name`, *hid_t* `aapl_id`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Open an attribute for an object, possibly asynchronously.

Description:

`H5Aopen_ff` opens an attribute, `attr_name`, which is attached to the object specified by `loc_id`.

`loc_id` is a location identifier and must be in scope for the read context specified by `rcntxt_id`.

The attribute access property list, `aapl_id`, is currently unused and should be `H5P_DEFAULT`.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The attribute identifier returned by this function must be released with `H5Aclose_ff` or resource leaks will develop.

Parameters:

hid_t `loc_id`	IN: May be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier that is in scope for the read context.
const char `*attr_name`	IN: Attribute name
hit_t `aapl_id`	IN: Attribute access property list. Currently not used.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns an attribute identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the new attribute is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a “normal” attribute identifier.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to read context id and from event queue to event stack. Added scope requirement and map object.

Man Page Status:

No known issues.

4.2.10 H5Aopen_by_name_ff

Name: H5Aopen_by_name_ff

Signature:

hid_t `H5Aopen_by_name_ff(` *hid_t* `loc_id,` *const char* `*obj_name,` *const char* `*attr_name`, *hid_t* `aapl_id`, *hid_t* `lapl_id`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Open an attribute for an object, possibly asynchronously.

Description:

`H5Aopen_by_name_ff` opens an attribute, `attr_name`, that is attached to an object specified by `loc_id` and `obj_name`.

`loc_id` is a location identifier; `obj_name` is the path to the object relative to `loc_id`. If `loc_id` fully specifies the object to which the attribute is attached, `obj_name` should be `'.'` (a dot). Both `loc_id and obj_name` must be in scope for the read context identified by `rcntxt_id.`

The attribute access property list, `aapl_id`, is currently unused and should be `H5P_DEFAULT`.

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See "Link Access Properties" in the H5P APIs. The link access property list currently has no effect in the EFF stack and should be set to `H5P_DEFAULT`.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The attribute identifier returned by this function must be released with `H5Aclose_ff` or resource leaks will develop.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier that is in scope for the read context.
const char `*obj_name`	IN: Object name Either relative to loc_id, absolute from the file's root group, or '.' (a dot)
hid_t `aapl_id`	IN: Attribute access property list. *Currently not used; specify `H5P_DEFAULT.`*
hid_t `lapl_id`	IN: Link access property list. *Currently not used in EFF; specify `H5P_DEFAULT.`*
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns an attribute identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the new attribute is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a “normal” attribute identifier.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to read context id and from event queue to event stack. Added scope requirement and map object. Noted lapl not used in EFF.

Man Page Status:

No known issues.

4.2.11 H5Aprefetch_ff

Name: H5Aprefetch_ff

Signature:

herr_t `H5Aprefetch_ff(` *hid_t* `attr_id`, *hid_t* `rcntxt_id`, *hid_t* `*replica_id`, *hid_t* `dxpl_id`, *hid_t* `es_id )`

Purpose:

Prefetch an attribute from persistent storage to burst buffer storage, possibly asynchronously.

Description:

`H5Dprefetch_ff` prefetches am attribute, specified by its identifier `attr_id`, from persistent storage (DAOS) into burst buffer storage.

`rcntxt_id` indicates the read context for this operation.

`replica_id,` the replica identifier, is set to indicate where the pre-fetched data can be found in the burst buffer, and is passed to subsequent `H5Devict_ff` calls.

`dxpl_id,` a data transfer property list identifier, is not currently used and should be set to H5P_DEFAULT.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The `replica_id` is not valid until the operation has completed, if it is executing asynchronously.

Limitations / Future Considerations:

For the EFF prototype project, only the IOD Blob object associated with the HDF5 attribute will be prefetched; auxiliary IOD objects (including the Blob objects that hold variable-length data) remain on persistent storage (DAOS). For more information, and other potential extensions, refer to the document *Burst Buffer Space Management – Prototype to Production*.

This function was implemented for completeness, but since no access routines accept the attribute's `replica_id` in this phase of the project, it has no practical value other than to demonstrate the ability to prefetch and evict attributes.

Parameters:

hid_t `attr_id`	IN: Identifier of the attribute being prefetched. `attr_id` must be in scope for the read context.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `* replica_id`	IN: Identifier of the replicated data in the burst buffer.
hid_t `dapl_id`	IN: Identifier of an access property list for this I/O operation.
hid_t `es_id`	IN: The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 7.

Man Page Status:

No known issues.

4.2.12 H5Aread_ff

Name: H5Aread_ff

Signature:

herr_t `H5Aread_ff(` *hid_t* `attr_id`, *hid_t* `mem_type_id,` *void* `*buf,` *hid_t* `rcntxt_id,` *hid_t* `es_id )`

Purpose:

Read an attribute, possibly asynchronously.

Description:

`H5A`read_ff reads an attribute, specified by `attr_id`. `attr_id` must be in scope for the read context identified by `rcntxt_id.`

The attribute's memory datatype is specified with `mem_type_id`. The entire attribute is read into `buf` from the file.

Datatype conversion takes place at the time of a read and is automatic. See the Data Conversion section of *The Data Type Interface (H5T)* in the *HDF5 User's Guide* for a discussion of data conversion, including the range of conversions currently supported by the HDF5 libraries.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `attr_id`	IN: Identifier of the attribute to read. Must be in scope for the read context.
hid_t `mem_type_id`	IN: identifier of the attribute datatype (in memory)
void `*buf`	OUT: Buffer to receive data read from file
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `eq_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to read context id and from event queue to event stack. Added scope requirement.

Man Page Status:

No known issues.

4.2.13 H5Arename_ff

Name: H5Arename_ff

Signature:

herr_t `H5Arename_ff(` *hid_t* `loc_id`, *const char* `*old_attr_name`, *const char* `*new_attr_name`, *hid_t* `trans_id,` *hid_t* `es_id )`

Purpose:

Rename an attribute, possibly asynchronously.

Description:

`H5Arename_ff` changes the name of the attribute attached to the object specified by `loc_id`.

`loc_id` is a location identifier and must be in scope for the transaction identified by `trans_id`.

The old name, `old_attr_name`, is changed to the new name, `new_attr_name`.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `loc_id`	IN: Location identifier for the object whose attribute is to be renamed. May be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier that is in scope for the transaction.
const char `*old_attr_name`	IN: Name of the attribute to be changed.
const char `*new_attr_name`	IN: New name for the attribute.
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement and map object.

Man Page Status:

No known issues.

4.2.14 H5Arename_by_name_ff

Name: H5Arename_by_name_ff

Signature:

herr_t `H5Arename_by_name_ff`(*hid_t* `loc_id`, *const char** `obj_name`, *const char* `*old_attr_name`, *const char* `*new_attr_name`, *hid_t* `lapl_id`, *hid_t* `trans_id`, *hid_t* `es_id`)

Purpose:

Rename an attribute for an object, possibly asynchronously.

Description:

`H5Arename_by_name_ff` changes the name of the attribute that is attached to the object specified by `loc_id` and `obj_name`.

`loc_id` is a location identifier; `obj_name` is the path to the object relative to `loc_id`. If `loc_id` fully specifies the object to which the attribute is attached, `obj_name` should be `'.'` (a dot). Both `loc_id` and `obj_name` must be in scope for the transaction identified by `trans_id`.

The attribute named `old_attr_name` is renamed `new_attr_name`.

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access `obj_name`. See "Link Access Properties" in the H5P APIs. The link access property list currently has no effect in the EFF stack and should be set to `H5P_DEFAULT`.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `loc_id`	IN: Location identifier for the object whose attribute is to be renamed. May be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier that is in scope for the transaction.
const char `*obj_name`	IN: Name of the object whose attribute is to be renamed. The object name (path to the object) can be specified relative to `loc_id`, absolute from the file's root group, or `'.'` (a dot), and must be in scope for the transaction.
const char `*old_attr_name`	IN: Name of the attribute to be changed.
const char `*new_attr_name`	IN: New name for the attribute.
hid_t `lapl_id`	IN: Link access property list identifier. *Currently not used in EFF; specify* ***`H5P_DEFAULT`.***
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously.

Use `H5_EVENT_STACK_NULL` for synchronous execution

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement and map object. Noted LAPL not used in EFF.

Man Page Status:

No known issues.

4.2.15 H5Awrite_ff

Name: H5Awrite_ff

Signature:

herr_t `H5Awrite_ff`(*hid_t* `attr_id`, *hid_t* mem_type_id, *const void* *buf, *hid_t* trans_id, *hid_t* es_id)

Purpose:

Write data to an attribute, possibly asynchronously.

Description:

`H5Awrite`_ff writes an attribute specified with `attr_id`. `attr_id` must be in scope for the transaction identified by `trans_id.`

The attribute's memory datatype is specified with `mem_type_id`. The entire attribute is written from `buf` into the file.

If `mem_type_id` is either a fixed-length or variable-length string, it is important to set the string length when defining the datatype. String datatypes are derived from `H5T_C_S1`, which defaults to 1 character in size. See `H5Tset_size` and "Creating variable-length string datatypes."

Datatype conversion takes place at the time of a write and is automatic. See the Data Conversion section of *The Data Type Interface (H5T)* in the *HDF5 User's Guide* for a discussion of data conversion, including the range of conversions currently supported by the HDF5 libraries.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `attr_id`	IN: Identifier of the attribute to write. Must be in scope for the transaction.
hid_t `mem_type_id`	IN: identifier of the attribute datatype (in memory)
void *`buf`	IN: Buffer with data to be written
hid_t `trans`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement.

Man Page Status:

No known issues.

4.3 H5AS: Analysis Shipping APIs

These routines are used to perform an analysis shipping operation on data contained in HDF5 Files (containers) in the Exascale Fast Forward stack.

Analysis shipping operations apply Python operations on data *in situ*, restricted to the results matching an HDF5 query operation.

These routines did not exist prior to the Exascale FastForward project.

Routine	**Implemented**	**Notes**
H5ASinvoke	Quarter 6	Renamed from H5ASexecute to H5ASinvoke and expanded to support any query type in Q8

4.3.1 H5ASinvoke

Name: H5ASinvoke

Signature:

herr_t `H5ASinvoke` (*const char* *`file_name`, *hid_t* `query_id`, *const char* *`split_script`, *const char* *`combine_script`, *const char* *`integrate_script`, *hid_t* `estack_id`)

Purpose:

Perform an analysis shipping operation on a HDF5 dataset.

Description:

`H5ASinvoke` ships to a remote server node an analysis operation that executes on data locally stored and selected by the user-defined query `query_id`. The analysis operation is received on a master I/O or storage server node, which first generates a view for the supplied query. Then, for each dataset in the view, the master node queries the layout of the dataset in the container named `file_name`. The `split_script` is then farmed to the helper I/O or storage server nodes that locally own a portion of the dataset. The helper nodes read data that is local to the node, run the `split_script` and send the resulting data back to the master node. The master node executes `combine_script` on the data it has gathered from the helper nodes for each dataset. When all datasets have been farmed out and returned, the master node executes `integrate_script` over all the buffers from the `combine_script` executions and sends the final result back to the application that initiated the analysis operation.

Each script parameter is a string that contains a Python script, each of which accepts a Python buffer as a parameter and returns a Python buffer as its return value.

Note that the final data output result is not directly returned by this call; an application would need to either explicity request it (assuming that it can be easily transferred) or write it to a separate container on the server.

Parameters:

const char *`file_name`	IN: Name of the file.
hid_t `query_id`	IN: Identifier of the query that will be applied to create a view to apply the scripts to.
const char *`split_script`	IN: String defining a Python script. The script must define a Python function named `split`, which will be applied locally on the helper nodes that contain data matching the query.
const char *`combine_script`	IN: String defining a `combine` Python script. The script must define a Python function named `combine`, which will be applied on the master node once the data from the farmed split operations has been gathered.
const char *`integrate_script`	IN: String defining an `integrate` Python script. The script must define a Python function named `integate`, which will be applied on the master node once the data from all the combine operations is available.
hid_t `estack_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.4 H5D: Dataset APIs

These routines are used to operate on HDF5 Datasets Objects.

The routines ending in _ff have different signatures than the standard HDF5 library routines.

Man pages for routines whose user interface is unchanged from the standard HDF5 implementation can be found at: http://www.hdfgroup.org/HDF5/doc/RM/RM_H5D.html.

Routine	Implemented	Notes
H5Dclose_ff	Quarter 4	
H5Dcreate_ff	Quarter 3	
H5Dcreate_anon_ff	Quarter 8	
H5Devict_ff	Quarter 7	
H5Dopen_ff	Quarter 3	
H5Dprefetch_ff	Quarter 7	
H5Dread_ff	Quarter 3	
H5Dset_extent_ff	Quarter 4	
H5Dwrite_ff	Quarter 3	
H5Dquery_ff	Quarter 8	
H5Dget_access_plist	Quarter 3	See standard HDF5 man page
H5Dget_create_plist	Quarter 3	See standard HDF5 man page
H5Dget_space	Quarter 3	See standard HDF5 man page
H5Dget_type	Quarter 3	See standard HDF5 man page
H5Dget_storage_size_ff		Not implemented in prototype.
H5Dget_space_status		Not implemented in prototype.
H5Dget_offset		Doesn't make sense for FF storage; will not implement

4.4.1 H5Dclose_ff

Name: H5Dclose_ff

Signature:

hid_t `H5Dclose_ff(` *hid_t* `dset_id,` *hid_t* `es_id )`

Purpose:

Close the specified dataset, possibly asynchronously.

Description:

`H5Dclose_ff` ends access to a dataset specified by `dset_id` and releases resources used by it. Further use of the dataset identifier is illegal in calls to the dataset API.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `dset_id`	IN: Identifier of the dataset to close access to.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from event queue to event stack.

Man Page Status:

No known issues.

4.4.2 H5Dcreate_ff

Name: H5Dcreate_ff

Signature:

hid_t `H5Dcreate_ff` (*hid_t* `loc_id`, *const char* `*name`, *hid_t* `dtype_id`, *hid_t* `space_id`, *hid_t* `lcpl_id`, *hid_t* `dcpl_id`, *hid_t* `dapl_id`, *hid_t* `trans_id`, *hid_t* `es_id`)

Purpose:

Create a new dataset and link it into the file, possibly asynchronously.

Description:

`H5Dcreate_ff` creates a new dataset named `name` at the location specified by `loc_id`, and associates constant and initial persistent properties with that dataset, including `dtype_id`, the datatype of each data element as stored in the file; `space_id`, the dataspace of the dataset; and other initial properties as defined in the dataset creation property and access property lists, `dcpl_id` and `dapl_id`, respectively. Once created, the dataset is opened for access.

`loc_id` may be a file identifier or a group identifier. `name` may be either an absolute path in the file or a relative path from `loc_id` naming the dataset. Both `loc_id` and `name` must be in scope for the transaction identified by `trans_id`.

If `dtype_id` is either a fixed-length or variable-length string, it is important to set the string length when defining the datatype. String datatypes are derived from `H5T_C_S1`, which defaults to 1 character in size. See `H5Tset_size` and "Creating variable-length string datatypes."

Currently in the Exascale Fast Forward stack you cannot have nested variable-length datatypes, nor a complex or array datatype with variable-length elements.

In the Exascale Fast Forward stack, only the first dimension in the dataset can be unlimited. The maximum dimensions for the dataspace specified by `space_id` control this constraint.

The link creation property list, `lcpl_id`, governs creation of the link(s) by which the new dataset is accessed and the creation of any intermediate groups that may be missing. In the EFF stack, automatic creation of missing intermediate groups (controlled by `H5Pset_create_intermediate_group`) is not supported. The link access property list currently has no effect in the EFF stack and should be set to `H5P_DEFAULT`.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The datatype and dataspace properties and the dataset creation and access property lists are attached to the dataset, so the caller may derive new datatypes, dataspaces, and creation and access properties from the old ones, and reuse them in calls to create additional datasets.

Once created, the dataset is ready to receive raw data. Immediately attempting to read raw data from the dataset will return zeros. Note that this behavior differs from traditional HDF5 files, where the fill value will be returned.

To conserve and release resources, the dataset should be closed when access is no longer required.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 group identifier or file identifier that is in scope for the transaction

const char `*name`	IN: Dataset name The dataset name (path to the dataset) can be specified relative to `loc_id` or absolute from the file's root group, and must be in scope for the transaction.
hid_t `dtype_id`	IN: Datatype identifier
hid_t `space_id`	IN: Dataspace identifier
hid_t `lcpl_id`	IN: Link creation property list *Currently not used in EFF; specify* `H5P_DEFAULT`*.*
hid_t `dcpl_id`	IN: Dataset creation property list
hid_t `dapl_id`	IN: Dataset access property list
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a dataset identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the new dataset is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a "normal" dataset identifier.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 3.
Quarter 4: Updated to reflect switch from parameter *H5_request_t* `request_ptr` to *hid_t* `eq_id`.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement. Noted that lapl is currently not used in EFF. Noted current restrictions on support for variable-length types in EFF stack. Noted that only first dimension can be extensible in EFF.

Man Page Status:

No known issues.

4.4.3 H5Dcreate_anon_ff

Name: H5Dcreate_anon_ff

Signature:

hid_t `H5Dcreate_anon_ff` (*hid_t* `loc_id`, *hid_t* `dtype_id`, *hid_t* `space_id`, *hid_t* `dcpl_id`, *hid_t* `dapl_id`, *hid_t* `trans_id`, *hid_t* `es_id`)

Purpose:

Create a new anonymous dataset that is not attached to the group hierarchy, possibly asynchronously.

Description:

`H5Dcreate_anon_ff` creates a new dataset in the container specified by `loc_id`, and associates constant and initial persistent properties with that dataset, including `dtype_id`, the datatype of each data element as stored in the file; `space_id`, the dataspace of the dataset; and other initial properties as defined in the dataset creation property and access property lists, `dcpl_id` and `dapl_id`, respectively. Once created, the dataset is opened for access.

`loc_id` may be a file identifier or a group identifier and is only used to identify the container in which to create the dataset. `loc_id` must be in scope for the transaction identified by `trans_id`.

If `dtype_id` is either a fixed-length or variable-length string, it is important to set the string length when defining the datatype. String datatypes are derived from `H5T_C_S1`, which defaults to 1 character in size. See `H5Tset_size` and "Creating variable-length string datatypes."

Currently in the Exascale Fast Forward stack you cannot have nested variable-length datatypes, nor a complex or array datatype with variable-length elements.

In the Exascale Fast Forward stack, only the first dimension in the dataset can be unlimited. The maximum dimensions for the dataspace specified by `space_id` control this constraint.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The datatype and dataspace properties and the dataset creation and access property lists are attached to the dataset, so the caller may derive new datatypes, dataspaces, and creation and access properties from the old ones, and reuse them in calls to create additional datasets.

Once created, the dataset is ready to receive raw data. Immediately attempting to read raw data from the dataset will return zeros. Note that this behavior differs from traditional HDF5 files, where the fill value will be returned.

Anonymous datasets will be removed from the container when the dataset ID is closed, unless `H5Oincr_refcount` is used to increase the dataset's reference count in the file.

To conserve and release resources, the dataset should be closed when access is no longer required.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 group identifier or file identifier that is in scope for the transaction
hid_t `dtype_id`	IN: Datatype identifier

hid_t `space_id`	IN: Dataspace identifier
hid_t `dcpl_id`	IN: Dataset creation property list
hid_t `dapl_id`	IN: Dataset access property list
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a dataset identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the new dataset is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a “normal” dataset identifier.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.4.4 H5Devict_ff

Name: H5Devict_ff

Signature:

herr_t `H5Devict_ff(` *hid_t* `dset_id,` *uint64_t* `container_version,` *hid_t* `dxpl_id,`
hid_t `es_id )`

Purpose:

Evict dataset from the burst buffer, possibly asynchronously.

Description:

`H5Devict_ff` evicts data associated with a dataset from the burst buffer.

The data to be evicted may be resident in the burst buffer under two different scenarios

In the first scenario, the data is resident in the burst buffer as the result of updates to the dataset made via the EFF transaction model. For example, through calls to H5Dcreate_ff or H5Dwrite_ff. These calls add updates to a transaction that are atomically applied to the dataset when the transaction is committed, and we refer to this data as *transaction update data*.

When evicting transaction update data, the container version being evicted should first be persisted to permanent storage (DAOS), with the `H5RCpersist` command. The dataset's transaction update data for the specified container version, as well as the dataset's transaction update data for all lower-numbered container versions that has not yet been evicted from the burst buffer, will be evicted as the result of this call. If evicting the data would result in container versions with open read contexts becoming inaccessible, the evict will fail.

In the second scenario, the data is resident in the burst buffer as the result of a call to H5Dprefetch_ff. This call replicates data from persistent storage (DAOS) to the burst buffer, and we refer to this data as *replica data*. When replica data is evicted, only data in the burst buffer as a result of the exact replica specified is evicted – transaction update data and other replicas for the dataset remain in the burst buffer.

`The dset_id` parameter specifies the dataset whose data is to be evicted.

The version of the dataset to be evicted is specified by the `container_version` property.

The replica property in the transfer property list, `dxpl_id,` is used to specify that a dataset replica is to be evicted. `H5Pset_dxpl_replica()` sets the replica property. If this is set, then replia data will be evicted, otherwise transaction update data will be evicted.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Limitations / Future Considerations:

When evicting a replica, the `container_version` argument is redundant, as the replica identifier fully specifies the data to be evicted. Consider revisiting and possibly revising the API prior to production release. Possibly have separate evict commands for eviction of transaction update data and of replicas.

For other potential extensions that are beyond the scope of the EFF prototype project, refer to the document *Burst Buffer Space Management – Prototype to Production*.

Parameters:

hid_t `dset_id`	IN: Identifier of the dataset being evicted.

uint64_t `container_version`	IN: Container version specifying what version of the dataset to evict.
hid_t `dapl_id`	IN: Identifier of an access property list. If the access property list contains an evict replica property (set via `H5Pset_evict_replica()`), then the replica_id specified by that property will be evicted.
hid_t `es_id`	IN: The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 7.

Man Page Status:

No known issues.

4.4.5 H5Dopen_ff

Name: H5Dopen_ff

Signature:

hid_t `H5Dopen_ff`(*hid_t* `loc_id`, *const char* *`name`, *hid_t* `dapl_id`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Open an existing dataset, possibly asynchronously.

Description:

`H5Dopen_ff` opens the existing dataset specified by `loc_id` and `name`.

`loc_id` may be a file identifier or a group identifier. `name` may be either an absolute path in the file or a relative path from `loc_id` naming the dataset. Both `loc_id` and `name` must be in scope for the read context identified by `rcntxt_id`.

The dataset access property list, `dapl_id`, provides information regarding access to the dataset.

`rcntxt_id` indicates the read context for this operation. Note that the returned dataset identifier can be used in other calls that operate on different read contexts, as specified by the read context id passed to those calls.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

To conserve and release resources, the dataset should be closed with `H5Dclose_ff` when access is no longer required.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 group identifier or file identifier that is in scope for the read context.
const char *`name`	IN: Dataset name The dataset name (path to the dataset) can be specified relative to `loc_id` or absolute from the file's root group, and must be in scope for the read context.
hid_t `dapl_id`	IN: Dataset access property list
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `eq_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a dataset identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the dataset is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a "normal" dataset identifier.

The dataset identifier returned by this call can be used to access the dataset at different container versions, even though a specific read context (and associated container version) is passed in this routine.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 3.
Quarter 4: Updated to reflect switch from parameter *H5_request_t* `request_ptr` to *hid_t* `eq_id`.
Quarter 5: Changed from transaction to read context id and from event queue to event stack. Added scope requirement.
Quarter 7: Added note that the returned dataset ID can be used to access the dataset at other container versions – not just the one associated with the read context specified in this open call.

Man Page Status:

No known issues.

4.4.6 H5Dprefetch_ff

Name: H5Dprefetch_FF

Signature:

herr_t `H5Dprefetch_ff(` *hid_t* `dset_id`, *hid_t* `rcntxt_id`, *hid_t* `*replica_id`, *hid_t* `dxpl_id`, *hid_t* `es_id )`

Purpose:

Prefetch all or part of a dataset from persistent storage to burst buffer storage, possibly asynchronously.

Description:

`H5Dprefetch_ff` prefetches a (partial) dataset, specified by its identifier `dset_id`, from persistent storage (DAOS) into burst buffer storage.

`rcntxt_id` indicates the read context for this operation.

`replica_id,` the replica identifier, is set to indicate where the pre-fetched data can be found in the burst buffer, and is passed to subsequent `H5Dread_ff` and `H5Devict_ff` calls.

`dxpl_id,` a data transfer property list identifier, is used to specify partial datasets (hyperslab selection) and control layout of the fetched data on the burst buffers.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The `replica_id` is not valid until the operation has completed, if it is executing asynchronously.

Limitations / Future Considerations:

For the EFF prototype project, only the primay IOD Array object associated with the HDF5 dataset will be prefetched; auxiliary IOD objects (including the Blob objects that hold variable-length data) remain on persistent storage (DAOS). For more information, and other potential extensions, refer to the document *Burst Buffer Space Management – Prototype to Production*.

The current API takes a read context identifier rather than simply specifying the container version to be prefetched. Although this seems reasonable in the context of an application that will presumably do a read (on a read context) after a prefetch, in the case of a scheduler that might prefetch data on behalf of the application, a container version would be more appropriate. Revisit and possibly revise the API prior to production release.

In Quarter 7, only full datasets and the default layout are supported.

Parameters:

hid_t `dset_id`	IN: Identifier of the dataset being prefetched. `dset_id` must be in scope for the read context.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t * `replica_id`	IN: Identifier of the replicated data in the burst buffer.
hid_t `dapl_id`	IN: Identifier of an access property list for this I/O operation.
hid_t `es_id`	IN: The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 7.

Man Page Status:

Will need updates in Quarter 8 when additional features are implemented.

4.4.7 H5Dquery_ff

Name: H5Dquery_ff

Signature:

hid_t `H5Dquery_ff(` *hid_t* `dset_id,` *hid_t* `query_ id,` *hid_t* `dxpl_id,` *hid_t* `rcntxt_id,` *hid_t* `es_id )`

Purpose:

Apply a query operation to a dataset, possibly asynchronously.

Description:

`H5Dquery_ff` applies a dataset element query, given by `query_id`, to a dataset, given by `dset_id`, generating a dataspace ID with a selection defined for the elements that match the query as a return value. The dataset transfer property list, `dxpl_id`, allows for future expansion through properties that may influence the query operation.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The query may be a singleton or compound query, but must be of type `H5Q_TYPE_DATA_ELEM`.

If an index is defined for the dataset, that index will be used to speed up the query operation.

The dataspace returned from this routine must be released with `H5Sclose`, after the routine has completed, if it is executing asynchronously.

Parameters:

hid_t `dset_id`	IN: Identifier of the dataset query. `dset_id` must be in scope for the read context.
hid_t `query_id`	IN: Identifier of the query.
hid_t `dxpl_id`	IN: Identifier of a transfer property list for this I/O operation. If specified, the read replica property directs the read to access pre-fetched data.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Returns:

Returns a dataspace identifier with a selection for elements that match the query if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the dataspace is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a "normal" dataspace identifier.

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.4.8 H5Dread_ff

Name: H5Dread_ff

Signature:

herr_t `H5Dread_ff(` *hid_t* `dset_id`, *hid_t* `mem_type_id`, *hid_t* `mem_space_id`, *hid_t* `file_space_id`, *hid_t* `dxpl_id`, *void* `* buf`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Read raw data from a dataset into a buffer, possibly asynchronously.

Description:

`H5Dread_ff` reads a (partial) dataset, specified by its identifier `dset_id`, from the file into an application memory buffer `buf`. Data transfer properties are defined by the argument `dxpl_id`. The memory datatype of the (partial) dataset is identified by the identifier `mem_type_id`. The part of the dataset to read is defined by `mem_space_id` and `file_space_id`.

`file_space_id` is used to specify only the selection within the file dataset's dataspace. Any dataspace specified in `file_space_id` is ignored by the library and the dataset's dataspace is always used. `file_space_id` can be the constant `H5S_ALL`. which indicates that the entire file dataspace, as defined by the current dimensions of the dataset, is to be selected.

`mem_space_id` is used to specify both the memory dataspace and the selection within that dataspace. `mem_space_id` can be the constant `H5S_ALL`, in which case the file dataspace is used for the memory dataspace and the selection defined with `file_space_id` is used for the selection within that dataspace.

If raw data storage space has not been allocated for the dataset, the returned buffer `buf` is filled with zeros. Note that fill values are not supported with the Exascale FastForward storage stack.

The behavior of the library for the various combinations of valid dataspace identifiers and H5S_ALL for the `mem_space_id` and the `file_space_id` parameters is described below:

`mem_space_id`	**`file_space_id`**	**Behavior**
valid dataspace identifier	valid dataspace identifier	`mem_space_id` specifies the memory dataspace and the selection within it. `file_space_id` specifies the selection within the file dataset's dataspace.
`H5S_ALL`	valid dataspace identifier	The file dataset's dataspace is used for the memory dataspace and the selection specified with `file_space_id` specifies the selection within it. The combination of the file dataset's dataspace and the selection from `file_space_id` is used for memory also.
valid dataspace identifier	`H5S_ALL`	`mem_space_id` specifies the memory dataspace and the selection within it. The selection within the file dataset's dataspace is set to the "all" selection.
`H5S_ALL`	`H5S_ALL`	The file dataset's dataspace is used for the memory dataspace and the selection within the memory dataspace is set to the "all" selection. The selection within the file dataset's dataspace is set to the "all" selection.

Using `H5S_ALL` for the file dataspace selection indicates that the entire dataspace, as defined by the dimensions of the dataset's dataspace for the specified read context, will be selected. The number of elements selected in the memory dataspace must match the number of elements selected in the file dataspace.

`dxpl_id` can be the constant `H5P_DEFAULT`, in which case the default data transfer properties are used. To read from a replica that was previously brought into the burst buffer by a call to `H5Dprefetch_ff`, use the read replica property in the data transfer property list. The call `H5Pset_dxpl_replica()` sets the read replica property.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The selected elements in the buffer are not valid until the operation has completed, if it is executing asynchronously.

Data is automatically converted from the file datatype and dataspace to the memory datatype and dataspace at the time of the read. See the Data Conversion section of *The Data Type Interface (H5T)* in the *HDF5 User's Guide* for a discussion of data conversion, including the range of conversions currently supported by the HDF5 libraries.

See *Design and Implementation of FastForward Features in HDF5* for information on the interaction between transformation operations, such as datatype conversion, and end-to-end integrity support with checksums.

Parameters:

hid_t `dset_id`	IN: Identifier of the dataset read from. `dset_id` must be in scope for the read context.
hid_t `mem_type_id`	IN: Identifier of the memory datatype.
hid_t `mem_space_id`	IN: Identifier of the memory dataspace.
hid_t `file_space_id`	IN: Identifier of the dataset's dataspace in the file.
hid_t `dxpl_id`	IN: Identifier of a transfer property list for this I/O operation. If specified, the read replica property directs the read to access pre-fetched data.
*void ** `buf`	OUT: Buffer to receive data read from file.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 3.
Quarter 4: Updated to reflect switch from parameter *H5_request_t* `request_ptr` to *hid_t* `eq_id`.
Quarter 5: Changed from transaction to read context id and from event queue to event stack. Added scope requirement.
Quarter 7: Added information about reads from prefetched replicas.

Man Page Status:
No known issues.

4.4.9 H5Dset_extent_ff

Name: H5Dset_extent_ff

Signature:

hid_t `H5Dset_extent_ff`(*hid_t* `dset_id`, *const hsize_t* `size[]`, *hid_t* `trans_id`, *hid_t* `es_id`)

Purpose:

Change the sizes of a dataset's dimensions.

Description:

`H5Dset_extent_ff` sets the current dimensions of the dataset `dset_id` to the sizes specified in `size`.

`size` is a 1-dimensional array with *n* elements, where *n* is the rank of the dataset's dataspace.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Notes:

If the sizes specified in `size` are smaller than the dataset's current dimension sizes, `H5Dset_extent_ff` will reduce the dataset's dimension sizes to the specified values. In the EFF stack, with support for transactions and asynchronous operations, it is important to remember that the new sizes will not become part of the transaction until the asynchronous operation completes successfully, and will not be registered in the file until the transaction is committed.

It is the user application's responsibility to ensure that valuable data is not lost, as `H5Dset_extent_ff` does not check that the dataset size is growing.

Parameters:

hid_t `dset_id`	IN: Dataset identifier. `dset_id` must be in scope for the transaction.
const hsize_t `size[]`	IN: Array containing the new magnitude of each dimension of the dataset.
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement. Removed discussion of space allocation time and fill values as not relevant for EFF.

Man Page Status:

No known issues.

4.4.10 H5Dwrite_ff

Name: H5Dwrite_ff

Signature:

herr_t H5Dwrite_ff(*hid_t* `dset_id`, *hid_t* `mem_type_id`, *hid_t* `mem_space_id`, *hid_t* `file_space_id`, *hid_t* `dxpl_id`, *const void* * `buf`, *hid_t* `trans_id`, *hid_t* `es_id`)

Purpose:

Write raw data from a buffer to a dataset, possibly asynchronously.

Description:

H5Dwrite_ff writes a (partial) dataset, specified by its identifier `dset_id`, from the application memory buffer `buf` into the file. Data transfer properties are defined by the argument `dxpl_id`. The memory datatype of the (partial) dataset is identified by the identifier `mem_type_id`. The part of the dataset to write is defined by `mem_space_id` and `file_space_id`.

If `mem_type_id` is either a fixed-length or variable-length string, it is important to set the string length when defining the datatype. String datatypes are derived from `H5T_C_S1`, which defaults to 1 character in size. See H5Tset_size and "Creating variable-length string datatypes."

`file_space_id` is used to specify only the selection within the file dataset's dataspace. Any dataspace specified in `file_space_id` is ignored by the library and the dataset's dataspace is always used. `file_space_id` can be the constant `H5S_ALL`, which indicates that the entire file dataspace, as defined by the current dimensions of the dataset, is to be selected.

`mem_space_id` is used to specify both the memory dataspace and the selection within that dataspace. `mem_space_id` can be the constant `H5S_ALL`, in which case the file dataspace is used for the memory dataspace and the selection defined with `file_space_id` is used for the selection within that dataspace.

The behavior of the library for the various combinations of valid dataspace IDs and `H5S_ALL` for the `mem_space_id` and the `file_space_id` parameters is described below:

`mem_space_id`	`file_space_id`	Behavior
valid dataspace identifier	valid dataspace identifier	`mem_space_id` specifies the memory dataspace and the selection within it. `file_space_id` specifies the selection within the file dataset's dataspace.
`H5S_ALL`	valid dataspace identifier	The file dataset's dataspace is used for the memory dataspace and the selection specified with `file_space_id` specifies the selection within it. The combination of the file dataset's dataspace and the selection from `file_space_id` is used for memory also.
valid dataspace identifier	`H5S_ALL`	`mem_space_id` specifies the memory dataspace and the selection within it. The selection within the file dataset's dataspace is set to the "all" selection.
`H5S_ALL`	`H5S_ALL`	The file dataset's dataspace is used for the memory dataspace and the selection within the memory dataspace is set to the "all" selection. The selection within the file dataset's dataspace is set to

the "all" selection.

Using `H5S_ALL` for the file dataspace selection indicates that the entire dataspace, as defined by the dimensions of the dataset's dataspace for the read context associated with the transaction, will be selected. The number of elements selected in the memory dataspace must match the number of elements selected in the file dataspace.

`dxpl_id` can be the constant `H5P_DEFAULT`, in which case the default data transfer properties are used.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Writing to a dataset will fail if the HDF5 file was not opened with write access permissions.

The selected elements in the buffer must not be modified until the operation has completed, if it is executing asynchronously.

Data is automatically converted from the memory datatype and dataspace to the file datatype and dataspace at the time of the write. See the Data Conversion section of *The Data Type Interface (H5T)* in the *HDF5 User's Guide* for a discussion of data conversion, including the range of conversions currently supported by the HDF5 libraries.

See *Design and Implementation of FastForward Features in HDF5* for information on the interaction between transformation operations, such as datatype conversion, and end-to-end integrity support with checksums.

Parameters:

hid_t `dset_id`	IN: Identifier of the dataset to write to. `dset_id` must be in scope for the transaction.
hid_t `mem_type_id`	IN: Identifier of the memory datatype.
hid_t `mem_space_id`	IN: Identifier of the memory dataspace.
hid_t `file_space_id`	IN: Identifier of the dataset's dataspace in the file.
hid_t `dxpl_id`	IN: Identifier of a transfer property list for this I/O operation.
const void * `buf`	IN: Buffer with data to be written to the file.
hid_t `trans_id`	IN: IN: Transaction identifier specifying the transaction this operation is a part of
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 3.
Quarter 4: Updated to reflect switch from parameter *H5_request_t* `request_ptr` to *hid_t* `eq_id`.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement. Noted the interaction between transformations and checksums. Removed space allocation, fill values, and compact representation text. Noted that "all" is relative to the read context associated with the transaction.

Man Page Status:

No known issues.

4.5 H5ES: Event Stack APIs

These routines allow the application to create and work with event stacks. An event stack provides an organizing structure for managing and monitoring functions that have been called asynchronously.

Once an event stack is created, its identifier can be passed to other HDF5 APIs that will be run asynchronously. The event associated with an asynchronous operation will be pushed onto the event stack whose identifier was passed as a parameter to the function. The application can use the H5ES APIs to monitor the completion status of an individual event or all events in an event stack. One or more of the events in an event stack can be cancelled. Monitoring or cancelling an event is equivalent to monitoring or cancelling the asynchronous operation the event is associated with.

Event stacks are per-process and there may be multiple event stacks for any given process. The most recent event added to an event stack will always be at index 0 (useful for calls to H5EScancel, H5EStest, H5ESwait).

The order in which asynchronous functions execute and complete is independent of their associated event objects' location in any event stack.

These routines did not exist prior to the Exascale FastForward project.

Routine	**Implemented**	**Notes**
H5EScancel	Quarter 6	
H5EScancel_all	Quarter 6	
H5ESclear	Quarter 6	
H5ESclose	Quarter 6	
H5EScreate	Quarter 6	
H5ESget_count	Quarter 6	
H5ESget_event_info		Not implemented in prototype.
H5EStest	Quarter 6	
H5EStest_all	Quarter 6	
H5ESwait	Quarter 6	
H5ESwait_all	Quarter 6	

4.5.1 H5EScancel

Name: H5EScancel

Signature:

herr_t `H5EScancel(` *hid_t* `es_id`, *size_t* `event_idx`, *H5ES_status_t* `*status`)

Purpose:

Cancel a particular event in an event stack.

Description:

`H5EScancel` cancels the in-progress event specified by `event_idx` in the event stack specified by `es_id` and indicates the event's status.

`event_idx` is the index of the event to be cancelled. The most recent event pushed on to the event stack is at index 0.

`status` is set to indicate whether the event was cancelled or had already completed. Possible values for `status` and their meanings are:

`H5ES_STATUS_CANCEL`	Event was cancelled, either by this call or previously.
`H5ES_STATUS_SUCCEED`	Event completed successfully.
`H5ES_STATUS_FAIL`	Event completed unsuccessfully.

Cancelled (and completed) events remain on the event stack until the stack is cleared.

Parameters:

hid_t `es_id`	IN: Identifier of the event stack.
size_t `event_idx`	IN: Index of the event to be cancelled.
H5ES_status_t `*status`	OUT: Status of the event upon completion of the call.

Returns:

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

History:

Quarter 5: Designed and documented but not yet implemented.
Quarter 6: Implemented.

Man Page Status:

No known issues.

4.5.2 H5EScancel_all

Name: H5EScancel_all

Signature:

herr_t `H5EScancel_all(` *hid_t* `es_id`, *H5ES_status_t* `*status`)

Purpose:

Cancel all events in an event stack that have not yet completed.

Description:

`H5EScancel_all` cancels the in-progress events in the event stack specified by `es_id` and indicates a status.

`status` is set to indicate the overall status of the events in the event stack. Possible values for `status` and their meanings are:

`H5ES_STATUS_CANCEL`	All events were cancelled, either by this call or previously.
`H5ES_STATUS_SUCCEED`	Some or all events completed, and all completed events succeeded.
`H5ES_STATUS_FAIL`	Some or all events completed, and one or more completed event failed.

Cancelled (and completed) events remain on the event stack until the stack is cleared.

Parameters:

hid_t `es_id`	IN: Identifier of the event stack.
H5ES_status_t `*status`	OUT: Overall status of the events in the event stack upon completion of the call.

Returns:

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

History:

Quarter 5: Designed and documented but not yet implemented.
Quarter 6: Implemented.

Man Page Status:

No known issues.

4.5.3 H5ESclear

Name: H5ESclear

Signature:

herr_t `H5ESclear(` *hid_t* `es_id`)

Purpose:

Clear all events from an event stack.

Description:

`H5ESclear` clears all event objects from the event stack specified by `es_id`, after first confirming that no in-progress events remain in the stack. If the stack does contain one or more in-progress events, no events are cleared and the call fails (a negative value is returned).

If there are in-progress events on the stack, `H5ESwait_all` can be used to wait for their completion before calling `H5ESclear.` Alternatively, `H5EScancel_all` can be used to cancel the in-progress events before calling `H5ESclear.`

Parameters:

hid_t `es_id`	IN: Identifier of the event stack to be cleared.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value. If the event contains one or more in-progress events a negative value is returned.

History:

Quarter 5: Designed and documented but not yet implemented.
Quarter 6: Implemented.

Man Page Status:

No known issues.

4.5.4 H5ESclose

Name: H5ESclose

Signature:

herr_t `H5ESclose`(*hid_t* `es_id`)

Purpose:

Close an event stack.

Description:

`H5ESclose` closes the event stack specified by `es_id`, after first confirming that no in-progress events remain in the stack. If the stack does contain one or more in-progress events, the event stack is not closed and the call fails (a negative value is returned).

If there are in-progress events on the stack, `H5ESwait_all` can be used to wait for their completion before calling `H5ESclose.` Alternatively, `H5EScancel_all` can be used to cancel the in-progress events before calling `H5ESclose.`

Upon successful completion`,` the event stack identifier, `es_id,` is no longer valid.

Parameters:

hid_t `es_id`	IN: Identifier of the event stack to be closed.

Returns:

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

History:

Quarter 5: Designed and documented but not yet implemented.
Quarter 6: Implemented.

Man Page Status:

No known issues.

4.5.5 H5EScreate

Name: H5EScreate

Signature:

hid_t `H5EScreate`(*void*)

Purpose:

Create an event stack.

Description:

`H5EScreate` creates an event stack to monitor and manage the events associated with asynchronous function execution. Multiple event stacks can be created and used concurrently in a program. Events added to the event stack are bundled and tracked together, although they may also be monitored individually.

The identifier for an event stack is used in all asynchronous function calls, which push the event associated with the function's execution onto the event stack. The order in which asynchronous functions execute and complete is independent of their associated event's location in any event stack.

Parameters:

none

Returns:

Returns an event stack identifier if successful; otherwise returns a negative value.

History:

Quarter 5: Designed and documented but not yet implemented.
Quarter 6: Implemented.

Man Page Status:

No known issues.

4.5.6 H5ESget_count

Name: H5ESget_count

Signature:

herr_t `H5ESget_count(` *hid_t* `es_id,` *size_t* `*count)`

Purpose:

Query the number of events in an event stack.

Description:

`H5ESget_count` retrieves the number of events in the event stack specified by `es_id`.

Parameters:

hid_t `es_id`	IN: Identifier of the event stack.
size_t `*count`	OUT: Number of events.

Returns:

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

History:

Quarter 5: Designed and documented but not yet implemented.
Quarter 6: Implemented.

Man Page Status:

No known issues.

4.5.7 H5ESget_event_info

Name: H5ESget_event_info

Signature:

herr_t `H5ESget_event_info(` *hid_t* `es_id`, *size_t* `start_idx`, *size_t* `count`,
const char `*ev_trace_str_arr[]`, *H5ES_status_t* `ev_status_arr[]`,
H5E_stack_id `ev_err_stack_id_arr[] )`

Purpose:

Retrieve information about events in an event stack.

Description:

`H5ESget_event_info` retrieves a variety of information about events in the event stack specified by `es_id`.

`start_idx` gives the index of the first event to retrieve information about. The most recent event pushed on the event stack is at index 0.

`count` gives the number of events to retrieve information about.

The array parameters `ev_trace_str_arr, ev_status_arr, and ev_err_stack_id_arr` must each be large enough to hold at least `count` entries. Passing a NULL pointer for any of the array parameters indicates the information normally returned in that array will not be retrieved.

`ev_trace_str_arr` is an array that holds tracing strings that document the API call and parameters for each event.

`ev_status_arr` is an array that holds the completion status (`H5ES_STATUS_SUCCEED, H5ES_STATUS_FAIL, H5ES_STATUS_IN_PROGRESS,` or `H5ES_STATUS_CANCEL`) for each event.

`ev_err_stack_id_arr` is an array that holds HDF5 error stack IDs for the operations associated with each event. Events that are in progress, have been cancelled, or have completed successfully will have a -1 value for the error stack ID. Error stacks retrieved from this routine may be queried with the `H5E` API calls.

This call may be made before or after all events in the event stack have completed.

Parameters:

hid_t `es_id`	IN: Identifier of the event stack.
size_t `*start_idx`	IN: Index of the first event to retrieve information about.
size_t `count`	IN: Number of events to retrieve information about.
const char `*ev_trace_str_arr[]`	OUT: Array containing tracing strings documenting API call and parameters of events
H5ES_status_t `ev_status_arr[]`	OUT: Array containing status of events
H5E_stack_id `ev_err_stack_id_arr[]`	OUT: Array containing error stack IDs of events.

Returns:

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

History:

Quarter 5: Designed and documented but not yet implemented.

Man Page Status:

No known issues.

4.5.8 H5EStest

Name: H5EStest

Signature:

herr_t `H5EStest(` *hid_t* `es_id`, *size_t* `event_idx`, *H5ES_status_t* `*status`)

Purpose:

Test to determine the status of a particular event in an event stack.

Description:

`H5EStest` reports the completion status of the event specified by `event_idx` in the event stack specified by `es_id`. The H5EStest call returns immediately.

`event_idx` is the index of the event to be tested. The most recent event pushed onto the event stack is always at index 0.

`status` is set to indicate the status of the event. Possible values for `status` and their meanings are:

`H5ES_STATUS_IN_PROGRESS`	Event is still active.
`H5ES_STATUS_SUCCEED`	Event completed successfully.
`H5ES_STATUS_FAIL`	Event completed unsuccessfully.
`H5ES_STATUS_CANCEL`	Event was cancelled.

Parameters:

hid_t `es_id`	IN: Identifier of the event stack.
size_t `event_idx`	IN: Index of the event to be tested.
H5ES_status_t `*status`	OUT: Status of the event.

Returns:

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

History:

Quarter 5: Designed and documented but not yet implemented.
Quarter 6: Implemented.

Man Page Status:

No known issues.

4.5.9 H5EStest_all

Name: H5EStest_all

Signature:

herr_t `H5EStest_all(` *hid_t* `es_id`, *H5ES_status_t* `*status`)

Purpose:

Test to determine the status of all events in an event stack.

Description:

`H5EStest_all` reports an overall completion status for all events in the event stack specified by `es_id`. The `H5EStest_all` call returns immediately.

`status` is set to indicate the overall status of the events in the event stack. Possible values for `status` and their meanings are:

`H5ES_STATUS_IN_PROGRESS`	At least one event is still active.
`H5ES_STATUS_SUCCEED`	Some or all events completed, and all completed events succeeded.
`H5ES_STATUS_FAIL`	Some or all events completed, and one or more completed event failed.
`H5ES_STATUS_CANCEL`	All events were cancelled.

Parameters:

hid_t `es_id`	IN: Identifier of the event stack.
H5ES_status_t `*status`	OUT: Overall status of the events in the event stack.

Returns:

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

History:

Quarter 5: Designed and documented but not yet implemented.
Quarter 6: Implemented.

Man Page Status:

No known issues.

4.5.10 H5ESwait

Name: H5ESwait

Signature:

herr_t `H5ESwait(` *hid_t* `es_id`, *size_t* `event_idx`, *H5ES_status_t* `*status`)

Purpose:

Wait for a particular event in an event stack to complete or be cancelled.

Description:

`H5ESwait` waits for the completion or cancellation of the event specified by `event_idx` in the event stack specified by `es_id` and reports the event's completion status. The `H5ESwait` call does not return until the event being waited on completes or is cancelled.

`event_idx` is the index of the event to be waited on The most recent event pushed on to the event stack is at index 0.

`status` is set to indicate the completion status of the event. Possible values for `status` and their meanings are:

`H5ES_STATUS_SUCCEED`	Event completed successfully.
`H5ES_STATUS_FAIL`	Event completed unsuccessfully.
`H5ES_STATUS_CANCEL`	Event was cancelled.

Parameters:

hid_t `es_id`	IN: Identifier of the event stack.
size_t `event_idx`	IN: Index of the event to be tested.
H5ES_status_t `*status`	OUT: Status of the event.

Returns:

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

History:

Quarter 5: Designed and documented but not yet implemented.
Quarter 6: Implemented.

Man Page Status:

No known issues.

4.5.11 H5ESwait_all

Name: H5ESwait_all

Signature:

herr_t `H5ESwait_all(` *hid_t* `es_id`, *H5ES_status_t* `*status )`

Purpose:

Wait for all events in an event stack to complete or be cancelled.

Description:

`H5ESwait_all` waits for the completion or cancellation of all events in the event stack specified by `es_id` and reports an overall completion status. The H5ESwait_all call does not return until all events in the event stack complete or are cancelled.

`status` is set to indicate the overall completion status of the events in the event stack. Possible values for `status` and their meanings are:

`H5ES_STATUS_SUCCEED`	Some or all events completed, and all completed events succeeded.
`H5ES_STATUS_FAIL`	Some or all events completed, and one or more completed event failed.
`H5ES_STATUS_CANCEL`	All events were cancelled.

Parameters:

hid_t `es_id`	IN: Identifier of the event stack.
H5ES_status_t `*status`	OUT: Overall completion status of the events in the event stack.

Returns:

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

History:

Quarter 5: Designed and documented but not yet implemented.
Quarter 6: Implemented

Man Page Status:

No known issues.

4.6 H5F: File (Container) APIs

These routines are used to operate on HDF5 File (Container) Objects.

The routines ending in _ff have different signatures than the standard HDF5 library routines.

Man pages for routines whose user interface is unchanged from the standard HDF5 implementation can be found at: http://www.hdfgroup.org/HDF5/doc/RM/RM_H5F.html.

Routine	**Implemented**	**Notes**
H5Fclose_ff	Quarter 4	
H5Fcreate_ff	Quarter 3	
H5Fopen_ff	Quarter 3	
H5Fget_access_plist	Quarter 3	Local operation, no need to do asynchronously. See standard HDF5 man page
H5Fget_create_plist	Quarter 3	Local operation, no need to do asynchronously. See standard HDF5 man page
H5Fget_intent	Quarter 3	Local operation, no need to do asynchronously. See standard HDF5 man page
H5Fget_name	Quarter 3	Local operation, no need to do asynchronously. See standard HDF5 man page
H5Fget_filesize		Not implemented in Prototype
H5Fget_info		Not implemented in Prototype
H5Fflush		Doesn't make sense for FF storage; will not implement *(decided in Q5)*
H5Fget_atomicity		Doesn't make sense for FF storage; will not implement
H5Fget_file_image		Doesn't make sense for FF storage; will not implement
H5Fget_free_space		Doesn't make sense for FF storage; will not implement
H5Fget_mdc_config		Doesn't make sense for FF storage; will not implement
H5Fget_mdc_hit_rate		Doesn't make sense for FF storage; will not implement
H5Fget_mdc_size		Doesn't make sense for FF storage; will not implement

H5Fget_vfd_handle		Doesn’t make sense for FF storage; will not implement
H5Fis_hdf5		Not implemented in prototype.
H5Fmount		Doesn’t make sense for FF storage; will not implement
H5Freopen		Doesn’t make sense for FF storage; will not implement
H5Freset_mdc_hit_rate_stats		Doesn’t make sense for FF storage; will not implement
H5Fset_mpi_atomicity		Doesn’t make sense for FF storage; will not implement
H5Fset_mdc_config		Doesn’t make sense for FF storage; will not implement
H5Funmount		Doesn’t make sense for FF storage; will not implement
H5Fget_obj_count		Not implemented in prototype
H5Fget_obj_ids		Not implemented in prototype

4.6.1 H5Fclose_ff

Name: H5Fclose_ff

Signature:

hid_t `H5Fclose_ff(` *hid_t* `file_id`, *hbool_t* `persist_flag,` *hid_t* `es_id`)

Purpose:

Terminate access to an HDF5 file (container), possibly asynchronously.

Description:

`H5Fclose_ff` terminates access to an HDF5 file on the Exascale FastForward storage system by terminating access to the container through `file_id`.

Applications are responsible for finishing transactions (`H5TRfinish`) before `H5Fclose_ff` is called. Applications should close all open objects in a container before calling `H5Fclose_ff`.

`H5Fclose_ff` will consume a transaction ID (the highest writable transaction) to write additional file metadata before closing the container. By default, `H5Fclose_ff` will call persist (`H5RCpersist`) on that last transaction. The user can opt not persist the final transaction created by `H5Fclose_ff` by setting `persist_flag` to `FALSE (0)`. The container will still be readable and correct, but there will be some performance loss if the additionl file metadata is not persisted.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `file_id`	IN: Identifier of container to terminate access to.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from event queue to event stack. Added text clarifying that call doesn't flush data to storage.
Quarter 7: Added persist flag to indicate whether the transaction consumed by H5Fclose should be persisted or not. Default is to persist.

Man Page Status:

No known issues.

4.6.2 H5Fcreate_ff

Name: H5Fcreate_ff

Signature:

hid_t `H5Fcreate_ff(` *const char* `*name`, *unsigned* `flags`, *hid_t* `fcpl_id`, *hid_t* `fapl_id`, *hid_t* `es_id`)

Purpose:

Create an HDF5 file (container), possibly asynchronously.

Description:

`H5Fcreate_ff` is the primary function for creating HDF5 files on the Exascale FastForward storage system. It creates a new HDF5 file (IOD container / DAOS container) with the specified name and property lists, and indicates whether an existing container of same name should be overwritten.

The `name` parameter specifies the name of the new container.

The `flags` parameter indicates whether an existing container is to be overwritten. It should be set to either `H5F_ACC_TRUNC` to overwrite an existing container or `H5F_ACC_EXCL`, instructing the function to fail if the container already exists.

New containers are always created in read-write mode, so the read-write and read-only flags, `H5F_ACC_RDWR` and `H5F_ACC_RDONLY`, respectively, are not relevant in this function. Further note that a specification of `H5F_ACC_RDONLY` will be ignored; the file will be created in read-write mode, regardless.

More complex behaviors of file creation and access are controlled through the file creation and file access property lists, `fcpl_id` and `fapl_id`, respectively. The value of `H5P_DEFAULT` for any property list value indicates that the library should use the default values for that appropriate property list.

Note: The `H5Pset_vol_iod()` routine must be invoked on the `fapl_id`, or the IOD VOL plugin will not be used.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The return value is a file identifier for the newly-created container; this file identifier should be closed by calling `H5Fclose_ff` when it is no longer needed.

Calling `H5Fcreate_ff` with an already opened file will fail with an error.

Parameters:

const char `*name`	IN: Name of the file to create.

uintn `flags`	IN: File access flags. Allowable values are:
	`H5F_ACC_TRUNC`
	Truncate file, if it already exists, erasing all data previously stored in the file.
	`H5F_ACC_EXCL`
	Fail if file already exists.
	
	• `H5F_ACC_TRUNC` and `H5F_ACC_EXCL` are mutually exclusive; use exactly one.

hid_t `fcpl_id`	IN: File creation property list identifier, used when modifying default file metadata. Use `H5P_DEFAULT` to specify default file creation properties.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a file identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the new file is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a “normal” file identifier.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 3.
Quarter 4: Updated to reflect switch from parameter *H5_request_t* `request_ptr` to *hid_t* `eq_id`.
Quarter 5: Changed from event queue to event stack.

Man Page Status:

No known issues.

4.6.3 H5Fopen_ff

Name: H5Fopen_ff

Signature:

hid_t `H5Fopen_ff(` *const char* `*name`, *unsigned* `flags`, *hid_t* `fapl_id`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Open an existing HDF5 file (container), possibly asynchronously.

Description:

`H5Fopen_ff` is the primary function for accessing existing HDF5 files (IOD containers / DAOS containers). This function opens the named file in the specified access mode and with the specified access property list.

Note that `H5Fopen` does not create a container if it does not already exist; see H5Fcreate_ff.

The `name` parameter specifies the name of the container to be opened.

The `flags` parameter specifies whether the container will be opened in read-write or read-only mode, `H5F_ACC_RDWR` or `H5F_ACC_RDONLY`, respectively. More complex behaviors of file access are controlled through the file-access property list.

The `fapl_id` parameter specifies the file access property list. Use of `H5P_DEFAULT` specifies that default I/O access properties will be used.

Note: the `H5Pset_vol_iod()` routine must be invoked on the `fapl_id`, or the IOD VOL plugin will not be used.

The `rcntxt_id` parameter allows the user to pass a pointer to an *hid_t* datatype signaling that the function should acquire a read handle for the last (highest) container version for the successfully opened container. In addition to acquiring the read handle, a read context is created and the identifier for that context is returned in `*rcntxt_id.` The read context identifier will not be available until after the `H5Fopen_ff` call completes successfully. `H5RCrelease` must be called to release the read handle and close the context when it is no longer needed.

If the `rcntxt_id` parameter is set to `NULL`, no read handle is acquired and no read context is created.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter

The return value is a file identifier for the newly-opened container; this file identifier should be closed by calling `H5Fclose_ff` when it is no longer needed.

Multiple opens:
A container can be opened with a new `H5Fopen_ff` call without closing an already-open identifier established in a previous `H5Fopen_ff` or `H5Fcreate_ff` call. Each `H5Fopen_ff` call will return a unique identifier and the container can be accessed through any of these identifiers as long as the identifier remains valid. A container can only have one open identifier with write privledges at any given time.

Parameters:

const char `*name`	IN: Name of the file to be opened.
unsigned `flags`	IN: File access flags. Allowable values are:
	`H5F_ACC_RDWR` Allow read and write access to file.

	`H5F_ACC_RDONLY` Allow read-only access to file. `H5F_ACC_RDWR` and `H5F_ACC_RDONLY` are mutually exclusive; use exactly one.
hid_t `fapl_id`	IN: Identifier for the file access properties list. If parallel file access is desired, this is a collective call according to the communicator stored in the `fapl_id`. Use `H5P_DEFAULT` for default file access properties.
hid_t `*rcntxt_id`	IN/OUT: Pointer to read context for the last (highest) container version for the successfully opened container. Pass in `NULL` if no read context is desired.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a file identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the file is returned initially; upon completion of the asynchronous operation, the future ID will be transparently modified to be a "normal" file identifier.

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event queue.

History:

Added in Quarter 3.
Quarter 4: Updated to reflect switch from parameter *H5_request_t* `request_ptr` to *hid_t* `eq_id`.
Quarter 5: Changed from event queue to event stack. Added rcntxt_id parameter.

Man Page Status:

No known issues.

4.7 H5G: Group APIs

These routines are used to operate on HDF5 Group Objects.

The routines ending in _ff have different signatures than the standard HDF5 library routines.

Man pages for routines whose user interface is unchanged from the standard HDF5 implementation can be found at: http://www.hdfgroup.org/HDF5/doc/RM/RM_H5G.html.

Routine	Implemented	Notes
H5Gclose_ff	Quarter 4	
H5Gcreate_ff	Quarter 3	
H5Gevict_ff	Quarter 7	
H5Gopen_ff	Quarter 3	
H5Gprefetch_ff	Quarter 7	
H5Gget_create_plist	Quarter 3	Local operation, no need to do asynchronously. See standard HDF5 man page.
H5Gget_info_ff		Not implemented in prototype.
H5Gget_info_by_name_ff		Not implemented in prototype.
H5Gcreate_anon		Not implemented in prototype.
H5Gget_info_by_idx		Not implemented in prototype.

4.7.1 H5Gclose_ff

Name: H5Gclose_ff

Signature:

hid_t `H5Gclose_ff(` *hid_t* `group_id,` *hid_t* `es_id )`

Purpose:

Close the specified group, possibly asynchronously.

Description:

`H5Gclose_ff` releases resources used by a group that was opened by `H5Gcreate_ff` or `H5Gopen_ff`. After closing a group, the `group_id` cannot be used again.

Failure to release a group with this call will result in resource leaks.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `group_id`	IN: Group identifier to release.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from event queue to event stack

Man Page Status:

No known issues.

4.7.2 H5Gcreate_ff

Name: H5Gcreate_ff

Signature:

hid_t `H5Gcreate_ff(` *hid_t* `loc_id,` *const char* `*name,` *hid_t* `lcpl_id,` *hid_t* `gcpl_id,` *hid_t* `gapl_id,` *hid_t* `trans_id,` *hid_t* `es_id )`

Purpose:

Create a new group and links it into the file, possibly asynchronously.

Description:

`H5Gcreate_ff` creates a new group in a file. After a group has been created, links to datasets and to other groups can be added.

The `loc_id` and `name` parameters specify where the group is located. `loc_id` may be a file identifier or a group identifier. `name` is the link to the group; `name` may be either an absolute path in the file (the links from the root group to the new group) or a relative path from `loc_id` (the link(s) from the group specified by `loc_id` to the new group). See the "Accessing objects by location and name" topic for more information. Both `loc_id` and `name` must be in scope for the transaction identified by `trans_id`.

`lcpl_id`, `gcpl_id`, and `gapl_id` are property list identifiers. These property lists govern how the link to the group is created, how the group is created, and how the group can be accessed in the future, respectively. `H5P_DEFAULT` can be passed in if the default properties are appropriate for these property lists. Currently, there are no APIs for the group access property list; use `H5P_DEFAULT`. See "H5P: Property List Interface" for the functions that can be used with each property list. `H5P_DEFAULT` must be used for all three of these parameters in the current EFF implementation.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The return value is the group identifier for the new group. Call `H5Gclose` with the group identifier to close the group and release resources when access is no longer required.

Parameters:

hid_t `loc_id`	IN: File or group identifier that is in scope for the transaction
const char `*name`	IN: Absolute or relative name of the link to the new group that is in scope for the transaction
hid_t `lcpl_id`	IN: Link creation property list identifier Must be `H5P_DEFAULT` in the current EFF implementation.
hid_t `gcpl_id`	IN: Group creation property list identifier Must be `H5P_DEFAULT` in the current EFF implementation.
hid_t `gapl_id`	IN: Group access property list identifier (No group access properties have been implemented at this time; use `H5P_DEFAULT`.)
uint64_t `trans`	IN: Value used to indicate transaction this operation is a part of.

hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a group identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the new group is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a “normal” group identifier.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 3.
Quarter 4: Updated to reflect switch from parameter *H5_request_t* `request_ptr` to *hid_t* `eq_id`.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement.

Man Page Status:

No known issues.

4.7.3 H5Gevict_ff

Name: H5G

Signature:

herr_t `H5Gevict_ff(` *hid_t* `grp_id`, *uint64_t* `container_version`, *hid_t* `dxpl_id`, *hid_t* `es_id`)

Purpose:

Evict group from the burst buffer, possibly asynchronously.

Description:

`H5Gevict_ff` evicts data associated with a group from the burst buffer.

The data to be evicted may be resident in the burst buffer under two different scenarios

In the first scenario, the data is resident in the burst buffer as the result of updates to the group made via the EFF transaction model. For example, through calls to H5Gcreate_ff or H5Dcreate_ff, where the new group is created or a new dataset is created in the group. These calls add updates to a transaction that are atomically applied to the group when the transaction is committed, and we refer to this data as *transaction update data*.

When evicting transaction update data, the container version being evicted should first be persisted to permanent storage (DAOS), with the `H5RCpersist` command. The group's transaction update data for the specified container version, as well as the group's transaction update data for all lower-numbered container versions that has not yet been evicted from the burst buffer, will be evicted as the result of this call. If evicting the data would result in container versions with open read contexts becoming inaccessible, the evict will fail.

In the second scenario, the data is resident in the burst buffer as the result of a call to H5Gprefetch_ff. This call replicates data from persistent storage (DAOS) to the burst buffer, and we refer to this data as *replica data*. When replica data is evicted, only data in the burst buffer as a result of the exact replica specified is evicted – transaction update data and other replicas for the group remain in the burst buffer.

`The grp_id` parameter specifies the group whose data is to be evicted.

The version of the group to be evicted is specified by the `container_version` property.

The replica property in the transfer property list, `dxpl_id,` is used to specify that a group replica is to be evicted. `H5Pset_dxpl_replica()` sets the replica property. If this is set, then replia data will be evicted, otherwise transaction update data will be evicted.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Limitations / Future Considerations:

When evicting a replica, the `container_version` argument is redundant, as the replica identifier fully specifies the data to be evicted. Consider revisiting and possibly revising the API prior to production release. Possibly have separate evict commands for eviction of transaction update data and of replicas.

For other potential extensions that are beyond the scope of the EFF prototype project, refer to the document *Burst Buffer Space Management – Prototype to Production*.

Parameters:

hid_t `grp_id`	IN: Identifier of the group being evicted.

uint64_t `container_version`	IN: Container version specifying what version of the group to evict.
hid_t `gapl_id`	IN: Identifier of an access property list. If the access property list contains an evict replica property (set via `H5Pset_evict_replica()`), then the replica_id specified by that property will be evicted.
hid_t `es_id`	IN: The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 7.

Man Page Status:

No known issues.

4.7.4 H5Gopen_ff

Name: H5Gopen_ff

Signature:

hid_t `H5Gopen_ff`(*hid_t* `loc_id`, *const char* * `name`, *hid_t* `gapl_id`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Open an existing group with a group access property list, possibly asynchronously.

Description:

`H5Gopen_ff` opens an existing group, `name`, at the location specified by `loc_id`.

`loc_id` may be a file identifier or a group identifier. `name` may be either an absolute path in the file or a relative path from `loc_id` to the group. Both `loc_id` and `name` must be in scope for the read context identified by `rcntxt_id`.

`gapl_id` is a group access property list identifier. No group access property lists have been implemented at this time; use `H5P_DEFAULT.`

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

`H5Gopen_ff` returns a group identifier for the group that was opened. This group identifier should be released by calling `H5Gclose` when it is no longer needed.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 group identifier or file identifier that is in scope for the read context.
const char *`name`	IN: Name of the group to open The group name (path to the group) can be specified relative to `loc_id` or absolute from the file's root group, and must be in scope for the read context.
hid_t `gapl_id`	IN: Group access property list identifier (No group access properties have been implemented at this time; use `H5P_DEFAULT`.)
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution

Returns:

Returns a group identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the group is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a "normal" group identifier.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 3.
Quarter 4: Updated to reflect switch from parameter *H5_request_t* `request_ptr` to *hid_t* `eq_id`.
Quarter 5: Changed from transaction to read context id and from event queue to event stack. Added scope requirement.

Man Page Status:

No known issues.

4.7.5 H5Gprefetch_ff

Name: H5Gprefetch_FF

Signature:

herr_t `H5Gprefetch_ff(` *hid_t* `grp_id`, *hid_t* `rcntxt_id`, *hid_t* `*replica_id`, *hid_t* `dxpl_id`, *hid_t* `es_id )`

Purpose:

Prefetch a group from persistent storage to burst buffer storage, possibly asynchronously.

Description:

`H5Gprefetch_ff` prefetches a group, specified by its identifier `grp_id`, from persistent storage (DAOS) into burst buffer storage.

`rcntxt_id` indicates the read context for this operation.

`replica_id,` the replica identifier, is set to indicate where the pre-fetched data can be found in the burst buffer, and is passed to subsequent `H5Gevict_ff` calls.

`dxpl_id,` a data transfer property list identifier, is not currently used and should be set to H5P_DEFAULT.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The `replica_id` is not valid until the operation has completed, if it is executing asynchronously.

Limitations / Future Considerations:

For the EFF prototype project, only the primary IOD KV object associated with the HDF5 group will be prefetched; auxiliary IOD objects remain on persistent storage (DAOS). For more information, and other potential extensions, refer to the document *Burst Buffer Space Management – Prototype to Production*.

This function was implemented for completeness, but since no access routines accept the group's `replica_id` in this phase of the project, it has no practical value other than to demonstrate the ability to prefetch and evict groups.

Parameters:

hid_t `grp_id`	IN: Identifier of the group being prefetched. `grp_id` must be in scope for the read context.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t * `replica_id`	IN: Identifier of the replicated data in the burst buffer.
hid_t `gapl_id`	IN: Identifier of an access property list for this I/O operation. Should be H5P_DEFAULT.
hid_t `es_id`	IN: The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 7.

Man Page Status:

No known issues.

4.8 H5L: Link APIs

These routines are used to create and manipulate Links in an HDF5 group, and are designed to be used in conjunction with the Object APIs (H5O).

The routines ending in _ff have different signatures than the standard HDF5 library routines.

Man pages for routines whose user interface is unchanged from the standard HDF5 implementation can be found at: http://www.hdfgroup.org/HDF5/doc/RM/RM_H5L.html.

Routine	**Implemented**	**Notes**
H5Lcopy_ff	Quarter 4	
H5Lcreate_hard_ff	Quarter 4	
H5Lcreate_soft_ff	Quarter 4	
H5Ldelete_ff	Quarter 4	
H5Lexists_ff	Quarter 4	
H5Lget_info_ff	Quarter 6	Man page added in Q8
H5Lget_val_ff	Quarter 6	Man page to be added in Q8
H5Lmove_ff	Quarter 4	
H5Literate_ff		Not implemented in prototype.
H5Literate_by_name_ff		Not implemented in prototype.
H5Lvisit_ff		Not implemented in prototype.
H5Lvisit_by_name_ff		Not implemented in prototype.
H5Lcreate_external		Not implemented in prototype.
H5Lcreate_ud		Not implemented in prototype.
H5Ldelete_by_idx		Not implemented in prototype.
H5Lget_info_by_idx		Not implemented in prototype.
H5Lget_name_by_idx		Not implemented in prototype.
H5Lget_val_by_idx		Not implemented in prototype.
H5Lis_registered		Not implemented in prototype.
H5Lregister		Not implemented in prototype.
H5Lunregister		Not implemented in prototype.

4.8.1 H5Lcopy_ff

Name: H5Lcopy_ff

Signature:

herr_t `H5Lcopy_FF(` *hid_t* `src_loc_id`, *const char* `*src_name`, *hid_t* `dest_loc_id`, *const char* `*dest_name`, *hid_t* `lcpl_id`, *hid_t* `lapl_id`, *hid_t* `trans_id`, *hid_t* `es_id`)

Purpose:

Copies a link from one location to another, possibly asynchronously.

Description:

`H5Lcopy_ff` copies the link specified by `src_name` from the file or group specified by `src_loc_id` to the file or group specified by `dest_loc_id`. The new copy of the link is created with the name `dest_name`.

`src_loc_id` and `dest_loc_id` must reference locations in the same file in the EFF stack. `src_loc_id, src_name, dest_loc_id,` and `dest_name` must all be in scope for the transaction identified by `trans_id`.

If `dest_loc_id` is a file identifier, `dest_name` will be interpreted relative to that file's root group.

The new link is created with the creation and access property lists specified by `lcpl_id` and `lapl_id`. The interpretation of `lcpl_id` is limited in the manner described in the next paragraph.

`H5Lcopy_ff` retains the creation time and the target of the original link. However, since the link may be renamed, the character encoding is that specified in `lcpl_id` rather than that of the original link. Other link creation properties are ignored.

If the link is a soft link, also known as a symbolic link, its target is interpreted relative to the location of the copy.

Several properties are available to govern the behavior of `H5Lcopy`. These properties are set in the link creation and access property lists, `lcpl_id` and `lapl_id`, respectively. In the EFF stack, neither the link creation property list nor the link access property list have any effect. Both should be set to `H5P_DEFAULT.`

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

`H5Lcopy_ff` does not affect the object that the link points to.

Parameters:

hid_t `src_loc_id`	IN: Location identifier of the source link May be any file or group identifier that is in scope for the transaction
const char `*src_name`	IN: Name of the link to be copied The name of the link can be specified relative to `src_loc_id`, absolute from the file's root group, or `'.'` (a dot), and must be in scope for the transaction.
hid_t `dest_loc_id`	IN: Location identifier specifying the destination of the copy May be any file or group identifier that is in scope for the transaction.
const char `*dest_name`	IN: Name to be assigned to the new copy The name of the new copy can be specified relative to `dest_loc_id` or

absolute from the file's root group, and must be in scope for the transaction.

hid_t `lcpl_id`	IN: Link creation property list identifier *Currently not used in EFF; specify* ***`H5P_DEFAULT.`***
hid_t `lapl_id`	IN: Link access property list identifier *Currently not used in EFF; specify* ***`H5P_DEFAULT.`***
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement and noted source and destination must be in same file. Noted lcpl and lapl have no effect in EFF.

Man Page Status:

No known issues.

4.8.2 H5Lcreate_hard_ff

Name: H5Lcreate_hard_ff

Signature:

herr_t H5Lcreate_hard_ff(*hid_t* obj_loc_id, *const char* *obj_name, *hid_t* link_loc_id, *const char* *link_name, *hid_t* lcpl_id, *hid_t* lapl_id,*hid_t* trans_id, *hid_t* es_id)

Purpose:

Creates a hard link to an object, possibly asynchronously.

Description:

`H5Lcreate_hard_ff` creates a new hard link to a pre-existing object in an HDF5 file. The new link may be one of many that point to that object.

The target object must already exist in the scope for the transaction indicated by `trans_id.`

`obj_loc_id` and `obj_name` specify the location and name, respectively, of the target object, i.e., the object that the new hard link points to. Both `obj_loc_id` and `obj_name` must be in scope for the transaction identified by `trans_id`.

`link_loc_id` and `link_name` specify the location and name, respectively, of the new hard link. Both `link_loc_id` and `link_name` must be in scope for the transaction identified by `trans_id`.

`obj_name` and `link_name` are interpreted relative to `obj_loc_id` and `link_loc_id`, respectively.

If `obj_loc_id` and `link_loc_id` are the same location, the HDF5 macro `H5L_SAME_LOC` can be used for either parameter (but not both).

`lcpl_id` and `lapl_id` are the link creation and access property lists associated with the new link. In the EFF stack, neither the link creation property list nor the link access property list have any effect. Both should be set to `H5P_DEFAULT.`

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Hard and soft links are for use only if the target object is in the current file. If the desired target object is in a different file from the new link, an external link may be created with `H5Lcreate_external`. External links are not currently supported in the EFF stack.

The HDF5 library keeps a count of all hard links pointing to an object; if the hard link count reaches zero (`0`), the object will be deleted from the file. Creating new hard links to an object will prevent it from being deleted if other links are removed. The library maintains no similar count for soft links and they can dangle.

Parameters:

hid_t `obj_loc_id`	IN: The file or group identifier for the target object. The identifier must be in scope for the transaction.
const char *`obj_name`	IN: Name of the target object, which must already exist. The object name (path to the object) can be specified relative to `obj_loc_id` or absolute from the file's root group and must be in scope for the transaction.

hid_t `link_loc_id`	IN: The file or group identifier for the new link. The identifier must be in scope for the transaction.
const char * `link_name`	IN: The name of the new link. The link name can be specified relative to `link_loc_id` or absolute from the file's root group and must be in scope for the transaction.
hid_t `lcpl_id`	IN: Link creation property list identifier. *Currently not used in EFF; specify `H5P_DEFAULT`.*
hid_t `lapl_id`	IN: Link access property list identifier. *Currently not used in EFF; specify `H5P_DEFAULT`.*
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution. .

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement. Noted that lcpl and lapl are unused in EFF stack.

Man Page Status:

No known issues.

4.8.3 H5Lcreate_soft_ff

Name: H5Lcreate_soft_ff

Signature:

herr_t `H5Lcreate_soft_ff`(*const char* *`target_path`, *hid_t* `link_loc_id`, *const char* *`link_name`, *hid_t* `lcpl_id`, *hid_t* `lapl_id`, *hid_t* `trans_id`, *hid_t* `es_id`)

Purpose:

Creates a soft link to an object, possibly asynchronously.

Description:

`H5Lcreate_soft_ff` creates a new soft link to an object in an HDF5 file. The new link may be one of many that point to that object.

`target_path` specifies the path to the target object, i.e., the object that the new soft link points to. `target_path` can be anything and is interpreted at lookup time. This path may be absolute in the file or relative to `link_loc_id`.

`link_loc_id` and `link_name` specify the location and name, respectively, of the new soft link. `link_name` is interpreted relative to `link_loc_id` . Both `link_loc_id` and `link_name` must be in scope for the transaction identified by `trans_id.`

`lcpl_id` and `lapl_id` are the link creation and access property lists associated with the new link. In the EFF stack, neither the link creation property list nor the link access property list have any effect. Both should be set to `H5P_DEFAULT.`

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

`H5Lcreate_soft_ff` is for use only if the target object is in the current file. If the desired target object is in a different file from the new link, use `H5Lcreate_external` to create an external link. External links are not currently supported in the EFF stack.

For instance, if `target_path` is `./foo`, `link_loc_id` specifies `./x/y/bar`, and the name of the new link is `new_link`, then a subsequent request for `./x/y/bar/new_link` will return same the object as would be found at `./foo`.

Soft links and external links are also known as symbolic links as they use a name to point to an object; hard links employ an object's address in the file.

Unlike hard links, a soft link in an HDF5 file is allowed to *dangle*, meaning that the target object need not exist at the time that the link is created.

The HDF5 library does not keep a count of soft links as it does of hard links.

Parameters:

const char *`target_path`	IN: Path to the target object, which is not required to exist.
hid_t `link_loc_id`	IN: The file or group identifier for the new link. The identifier must be in scope for the transaction.

const char *`link_name`	IN: The name of the new link. The link name can be specified relative to `link_loc_id` or absolute from the file's root group and must be in scope for the transaction.
hid_t `lcpl_id`	IN: Link creation property list identifier. *Currently not used in EFF; specify `H5P_DEFAULT`.*
hid_t `lapl_id`	IN: Link access property list identifier. *Currently not used in EFF; specify `H5P_DEFAULT`.*
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement. Noted that lcpl and lapl are unused in EFF stack and that external links are unsupported.

Man Page Status:

No known issues.

4.8.4 H5Ldelete_ff

Name: H5Ldelete_ff

Signature:

herr_t `H5Ldelete_ff(` *hid_t* `loc_id`, *const char* `*name`, *hid_t* `lapl_id`, *hid_t* `trans_id`, *hid_t* `es_id )`

Purpose:

Removes a link from a group.

Description:

`H5Ldelete_ff` removes the link specified by `name` from the location `loc_id`. Both `name` and `loc_id` must be in scope for the transaction indicated by `trans_id.`

`lapl_id` specifies the link access property list. The link access property list currently has no effect in the EFF stack and should be set to `H5P_DEFAULT`.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

If the link being removed is a hard link, `H5Ldelete_ff` also decrements the link count for the object to which `name` points. Unless there is a duplicate hard link in that group, this action removes the object to which `name` points from the group that previously contained it.

A reference count keeps track of how many hard links refer to an object; when the reference count reaches zero, the object can be removed from the file. Objects that are open are not removed until all identifiers to the object are closed.

Warning:

Exercise caution in the use of H5Ldelete_ff; if the link being removed is on the only path leading to an HDF5 object, that object may become permanently inaccessible in the file.

Parameters:

hid_t `loc_id`	IN: Identifier of the file or group containing the link object. Must be in scope for the transaction.
const char `*name`	IN: Name of the link to delete. Must be in scope for the transaction.
hid_t `lapl_id`	IN: Link access property list identifier. *Currently not used in EFF; specify* ***`H5P_DEFAULT.`***
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement. Noted that lapl is not used in EFF.

Man Page Status:

No known issues.

4.8.5 H5Lexists_ff

Name: H5Lexists_ff

Signature:

herr_t `H5Lexists_ff`(*hid_t* `loc_id`, *const char* `*name`, *hid_t* `lapl_id`, *hbool_t* `*exists`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Determine whether a link with the specified name exists in a group.

Description:

`H5Lexists_ff` allows an application to determine whether the link `name` exists in the group or file specified with `loc_id`. The link may be of any type; only the presence of a link with that name is checked. Both `name` and `loc_id` must be in scope for the read context indicated by `rcntxt_id`.

The results of the existence test are put in `exists`. Note that this value will not be set until after the function completes, which may be later than when the call returns for asynchronous execution. Traditionally the existence results were indicated by the function call's return value, but with the support for asynchronous execution that is no longer possible and the FastForward version included the new `exists` parameter.

The link access property list, `lapl_id`, may provide information regarding the properties of links. The link access property list currently has no effect in the EFF stack and should be set to `H5P_DEFAULT`.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Note that `H5Lexists_ff` verifies only that the target link exists. If `name` includes either a relative path or an absolute path to the target link, intermediate steps along the path must be verified before the existence of the target link can be safely checked. If the path is not verified and an intermediate element of the path does not exist, `H5Lexists_ff` will fail. The example in the next paragraph illustrates one step-by-step method for verifying the existence of a link with a relative or absolute path.

Example: Use the following steps to verify the existence of the link `datasetD` in the group `group1/group2/softlink_to_group3/`, where `group1` is a member of the group specified by `loc_id`:

- First use `H5Lexists_ff` to verify that `group1` exists.
- If `group1` exists, use `H5Lexists_ff` again, this time with `name` set to `group1/group2`, to verify that `group2` exists.
- If `group2` exists, use `H5Lexists_ff` with `name` set to `group1/group2/softlink_to_group3` to verify that `softlink_to_group3` exists.
- If `softlink_to_group3` exists, you can now safely use `H5Lexists_ff` with `name` set to `group1/group2/softlink_to_group3/datasetD` to verify that the target link, `datasetD`, exists.

If the link to be verified is specified with an absolute path, the same approach should be used, but starting with the first link in the file's root group. For instance, if `datasetD` were in `/group1/group2/softlink_to_group3`, the first call to `H5Lexists_ff` would have `name` set to `/group1`.

Note that this is an outline and does not include all necessary details. Depending on circumstances, for example, you may need to verify that an intermediate link points to a group and that a soft link points to an existing target.

Parameters:

hid_t `loc_id`	IN: Identifier of the file or group to query.
const char *`name`	IN: The name of the link to check.
hid_t `lapl_id`	IN: Link access property list identifier. *Currently not used in EFF; specify* ***`H5P_DEFAULT`.***
hbool_t *`exists`	OUT: Pointer to returned results indicating existence of link. When successful, will be a positive value to indicate the link exists and 0 (zero) if the link does not exist. The value pointed to will not be modified if the existence test failed for some reason.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. When the asynchronous execution completes successfully, `exists` will contain the existence test results.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to read context id and from event queue to event stack. Added scope requirement. Noted that link access property list is not used.

Man Page Status:

No known issues.

4.8.6 H5Lget_info_ff

Name: H5Lget_info_ff

Signature:

herr_t `H5Lget_info_ff(` *hid_t* `loc_id`, *const char* `*name`, *H5L_ff_info_t* `*link_info`, *hid_t* `lapl_id`, *hid_t* `rcntxt_id`, *hid_t* `es_id )`

Purpose:

Retrieve the metadata for a link.

Description:

`H5Lget_info_ff` retrieves the metadata for the specified link and puts it into the struct *H5L_ff_info_t*.

An *H5L_ff_info_t* struct is defined (in H5FFpublic.h) as follows :

```
typedef struct H5L_ff_info_t{
    H5L_type_t     type;
    H5T_cset_t     cset;
    union {
        haddr_ff_t  address;
        size_t      val_size;
    } u;
} H5L_ff_info_t;
```

`lapl_id` indicates the link access property list for this operation.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `loc_id`	IN: Location identifier of the link May be any file or group identifier that is in scope for the read context.
const char `*name`	IN: Name of the link The name of the link can be specified relative to `loc_id`, absolute from the file's root group, or `'.'` (a dot), and must be in scope for the read context.
H5L_ff_info_t `*link_info`	OUT: Buffer in which to return link information.
hid_t `lapl_id`	IN: Link access property list.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. When the asynchronous execution completes successfully, `link_info` will contain all the metadata values for that link.

History:

Implemented in Q6; Man page added in Q8.

Man Page Status:

No known issues.

4.8.7 H5Lget_val_ff

Name: H5Lget_val_ff

Signature:

herr_t `H5Lget_val_ff`(*hid_t* `loc_id`, *const char* `*name`, *void* `*buf`, *size_t* `size`, *hid_t* `lapl_id`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Retrieve the link value for the named link.

Description:

`H5Lget_val_ff` returns the link value of a link specified by the combination of `loc_id` and `name.` For symbolic links, the link value is the path to which the link points, including the null terminator. For user-defined links, it is the link buffer.

At most `size` bytes are copied into the `buf` result buffer.

`lapl_id` indicates the link access property list for this operation.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

When asynchronous execution is used, wait for the completion of the operation before reading the results from `buf`.

Parameters:

hid_t `loc_id`	IN: Location identifier of the link May be any file or group identifier that is in scope for the read context.
const char `*name`	IN: Name of the link The name of the link can be specified relative to `loc_id`, absolute from the file's root group, or `'.'` (a dot), and must be in scope for the read context.
void `*buf`	OUT: Buffer in which to return link value
size_t `size`	IN: Maximum number of bytes copies into the result buffer.
hid_t `lapl_id`	IN: Link access property list.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution

Returns:

Returns a non-negative value if successful; otherwise returns a negative value. Also updates `*buf` if successful.

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. When the asynchronous execution completes successfully, `*buf` will contain the value for that link.

History:

Implemented in Q6; Man page added in Q8.

Man Page Status:

No known issues.

4.8.8 H5Lmove_ff

Name: H5Lmove_ff

Signature:

herr_t `H5Lmove_ff(` *hid_t* `src_loc_id`, *const char* `*src_name`, *hid_t* `dest_loc_id`, *const char* `*dest_name`, *hid_t* `lcpl_id`, *hid_t* `lapl_id`, *hid_t* `trans_id`, *hid_t* `es_id )`

Purpose:

Moves a link within an HDF5 file, possibly asynchronously.

Description:

`H5Lmove_tt` moves a link within an HDF5 file. The original link, `src_name`, is removed from `src_loc_id` and the new link, `dest_name`, is inserted at `dest_loc_id`. This change is accomplished as an atomic operation.

`src_loc_id` and `src_name` identify the original link. `src_loc_id` is either a file or group identifier; `src_name` is the path to the link and is interpreted relative to `src_loc_id`. `dest_loc_id` and `dest_name` identify the new link. `dest_loc_id` is either a file or group identifier; `dest_name` is the path to the link and is interpreted relative to `dest_loc_id.`

`src_loc_id` and `dest_loc_id` must reference locations in the same file in the EFF stack. `src_loc_id, src_name, dest_loc_id,` and `dest_name` must all be in scope for the transaction identified by `trans_id`.

Note that `H5Lmove_tt` does not modify the value of the link; the new link points to the same object as the original link pointed to. Furthermore, if the object pointed to by the original link was already open with a valid object identifier, that identifier will remain valid after the call to `H5Lmove_tt`.

`lcpl_id` and `lapl_id` are the link creation and link access property lists, respectively, associated with the new link, `dest_name`. Neither of these property lists currently have any effect in the EFF stack and

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Warning:

Exercise care in moving links, as it is possible to render data in a file inaccessible with `H5Lmove`. If the link being moved is on the only path leading to an HDF5 object, that object may become permanently inaccessible in the file.

Parameters:

hid_t `src_loc_id`	IN: Original file or group identifier.
const char `*src_name`	IN: Original link name.
hid_t `dest_loc_id`	IN: Destination file or group identifier.
const char `*dest_name`	IN: New link name.
hid_t `lcpl_id`	IN: Link creation property list identifier to be associated with the new link. *Currently not used in EFF; specify* ***`H5P_DEFAULT.`***

hid_t `lapl_id`	IN: Link access property list identifier to be associated with the new link. *Currently not used in EFF; specify* ***`H5P_DEFAULT.`***
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement and noted source and destination must be in same file. Updated text in property list description to match what EFF stack supports.

Man Page Status:

No known issues.

4.9 H5M: Map APIs

These routines are used to read or write HDF5 Map objects, which were added to support ACG use cases. Map objects, which offer a key-value store data structure, should also have wide applicability to other application.

Map objects in HDF5 are similar to a typical "map" data structure in computer science, and are implemented in the EFF stack using IOD Key-Value objects.

HDF5 maps set/get a value in the object, according to the key provided, with a 1-1 mapping of keys to values. All keys for a given map object must be of the same HDF5 datatype. All values for a given map object must be of the same HDF5 datatype.

HDF5 maps are leaf objects in the group hierarchy within a container. Attributes can be attached to map objects.

Map APIs were designed for the EFF stack and also for the traditional HDF5 Binary HDF5 file format storage. EFF-specific versions have the "_ff" suffix and include one or more of (1) a read context id, (2) a transaction id, and/or (3) an event stack id. The Map APIs for the traditional HDF5 format do not have the _ff suffix or the parameters listed, and have not been implemented or explicitly included in this section.

These routines did not exist prior to the Exascale FastForward project.

Routine	**Implemented**	**Notes**
H5Mclose_ff	Quarter 5	
H5Mcreate_ff	Quarter 5	
H5Mdelete_ff	Quarter 5	
H5Mevict_ff	Quarter 7	
H5Mexists_ff	Quarter 5	
H5Mget_ff	Quarter 5	
H5Mget_count_ff	Quarter 5	
H5Mget_types_ff	Quarter 5	
H5Miterate_ff		Not implemented in prototype.
H5Mopen_ff	Quarter 5	
H5Mprefetch_ff	Quarter 7	
H5Mset_ff	Quarter 5	

4.9.1 H5Mclose_ff

Name: H5Mclose_ff

Signature:

herr_t `H5Mclose_ff(` *hid_t* `map_id`, *hid_t* `es_id )`

Purpose:

Close the specified map object, possibly asynchronously.

Description:

`H5Mclose_ff` ends access to the map object specified by `map_id` and releases resources used by it. Further use of the map identifier is illegal in calls to the map API.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `map_id`	IN: Identifier of the map object to close access to.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.9.2 H5Mcreate_ff

Name: H5Mcreate_ff

Signature:

hid_t `H5Mcreate_ff` (*hid_t* `loc_id`, *const char* `*name`, *hid_t* `keytype_id`, *hid_t* `valtype_id`, *hid_t* `lcpl_id`, *hid_t* `mcpl_id`, *hid_t* `mapl_id`, *hid_t* `trans_id`, *hid_t* `es_id`)

Purpose:

Create a new map object and link it into the file, possibly asynchronously.

Description:

`H5Mcreate_ff` creates a new map object named `name` at the location specified by `loc_id.`

`loc_id` may be a file identifier or a group identifier. `name` may be either an absolute path in the file or a relative path from `loc_id` naming the map object. Both `loc_id` and `name` must be in scope for the transaction identified by `trans_id`.

`keytype_id` specifies the datatype for all keys in the map object and `valtype_id` specifies the datatype for all values in the object.

If `keytype_id` or `valtype_id` are either fixed-length or variable-length strings, it is important to set the string length when defining the datatype. String datatypes are derived from `H5T_C_S1`, which defaults to 1 character in size. See `H5Tset_size` and "Creating variable-length string datatypes."

The EFF stack currently does not support variable-length datatypes for keys.

The link creation property list, `lcpl_id`, governs creation of the link(s) by which the new map object is accessed and the creation of any intermediate groups that may be missing. In the EFF stack, automatic creation of missing intermediate groups (controlled by `H5Pset_create_intermediate_group`) is not supported and none of the properties in the link creation property list currently have any effect.

The map creation property list, `mcpl_id`, and map access property list, `mapl_id`, modify the new map object's behavior.

`H5P_DEFAULT` must be used for all three of the property list parameters in the current EFF implementation.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

To conserve and release resources, the map object id returned from this routine must be closed with `H5Mclose` when access is no longer required.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 group identifier or file identifier that is in scope for the transaction
const char `*name`	IN: name The map object name (path to the map object) can be specified relative to `loc_id` or absolute from the file's root group, and must be in scope for the transaction.
hid_t `keytype_id`	IN: Datatype identifier for the map keys

hid_t `valtype_id`	IN: Datatype identifier for the map values
hid_t `lcpl_id`	IN: Link creation property list *Currently not used in EFF; specify* ***`H5P_DEFAULT.`***
hid_t `mcpl_id`	IN: Map creation property list *Currently not used; specify* ***`H5P_DEFAULT.`***
hid_t `mapl_id`	IN: Map access property list *Currently not used; specify* ***`H5P_DEFAULT.`***
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a map object identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the new map object is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a "normal" map object identifier.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.9.3 H5Mdelete_ff

Name: H5Mdelete_ff

Signature:

herr_t `H5Mdelete_ff(` *hid_t* `map_id`, *hid_t* `key_mem_type_id`, *const void* `*key`, *hid_t* `trans_id`, *hid_t* `es_id )`

Purpose:

Delete a key/value pair from a map object.

Description:

`H5Mdelete_ff` removes a key/value pair from the map object given by `map_id`. `map_id` must be in scope for the transaction indicated by `trans_id.`

The `key` parameter points to the key of the map entry (the key/value pair) that will be deleted, and `key_mem_type_id` specifies the datatype of the `key` parameter.

If `key_mem_type_id` is not the same as the datatype of the keys in the map object, the `key` parameter will automatically undergo datatype conversion in order to locate the correct key in the map object.

If the key is not in the map file, a negative value will be returned if the routine is called synchronously (`es_id` is `H5_EVENT_STACK_NULL`). If the routine is called asynchronously, the completion status of the asynchronous operation will be `H5ES_STATUS_FAIL.`

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `map_id`	IN: Identifier of the map object. Must be in scope for the transaction.
hid_t `key_mem_type_id`	IN: Datatype of the `key` parameter.
const void `*key`	IN: Pointer to key of the map key/value pair that will be deleted.
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.9.4 H5Mevict_ff

Name: H5Mevict_ff

Signature:

herr_t `H5Mevict_ff(` *hid_t* `map_id,` *uint64_t* `container_version,` *hid_t* `dxpl_id,` *hid_t* `es_id )`

Purpose:

Evict map object from the burst buffer, possibly asynchronously.

Description:

`H5Mevict_ff` evicts data associated with a map object from the burst buffer.

The data to be evicted may be resident in the burst buffer under two different scenarios

In the first scenario, the data is resident in the burst buffer as the result of updates to the map object made via the EFF transaction model. For example, through calls to H5Mcreate_ff or H5Mset_ff. These calls add updates to a transaction that are atomically applied to the map object when the transaction is committed, and we refer to this data as *transaction update data*.

When evicting transaction update data, the container version being evicted should first be persisted to permanent storage (DAOS), with the `H5RCpersist` command. The map object's transaction update data for the specified container version, as well as the map object's transaction update data for all lower-numbered container versions that has not yet been evicted from the burst buffer, will be evicted as the result of this call. If evicting the data would result in container versions with open read contexts becoming inaccessible, the evict will fail.

In the second scenario, the data is resident in the burst buffer as the result of a call to H5Mprefetch_ff. This call replicates data from persistent storage (DAOS) to the burst buffer, and we refer to this data as *replica data*. When replica data is evicted, only data in the burst buffer as a result of the exact replica specified is evicted – transaction update data and other replicas for the dataset remain in the burst buffer.

`The map_id` parameter specifies the map object whose data is to be evicted.

The version of the map object to be evicted is specified by the `container_version` property.

The replica property in the transfer property list, `dxpl_id,` is used to specify that a map replica is to be evicted. `H5Pset_dxpl_replica()` sets the replica property. If this is set, then replia data will be evicted, otherwise transaction update data will be evicted.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Limitations / Future Considerations:

When evicting a replica, the `container_version` argument is redundant, as the replica identifier fully specifies the data to be evicted. Consider revisiting and possibly revising the API prior to production release. Possibly have separate evict commands for eviction of transaction update data and of replicas.

For other potential extensions that are beyond the scope of the EFF prototype project, refer to the document *Burst Buffer Space Management – Prototype to Production*.

Parameters:

hid_t `map_id`	IN: Identifier of the map object being evicted.

uint64_t `container_version`	IN: Container version specifying what version of the map object to evict.
hid_t `mapl_id`	IN: Identifier of an access property list. If the access property list contains an evict replica property (set via `H5Pset_evict_replica())`, then the replica_id specified by that property will be evicted.
hid_t `es_id`	IN: The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 7.

Man Page Status:

No known issues.

4.9.5 H5Mexists_ff

Name: H5Mexists_ff

Signature:

herr_t `H5Mexists_ff(` *hid_t* `map_id,` *hid_t* `key_mem_type_id`, *const void* `*key`, *hbool_t* `*exists`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Determine whether a key exists in a map object, possibly asynchronously.

Description:

`H5Mexists_ff` determines whether the key, `key`, exists in the map object specified by `map_id`. `map_id` must be in scope for the read context identified by `rcntxt_id`.

The `key` parameter points to the key whose existence is being checked, and `key_mem_type_id` specifies the datatype of the `key` parameter.

If `key_mem_type_id` is not the same as the datatype of the keys in the map object, the `key` parameter will automatically undergo datatype conversion in order to locate the correct key in the map object.

The results of the existence test are put in `exists`. Note that this value will not be set until after the function completes, which may be later than when the call returns for asynchronous execution.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `map_id`	IN: Identifier of the map object. Must be in scope for the read context.
hid_t `key_mem_type_id`	IN: Datatype of the `key` parameter.
const void `*key`	IN: Pointer to key whose existence in the map object is being checked.
hbool_t `*exists`	OUT: Pointer to returned results indicating existence of key. When successful, will be a positive value to indicate the key exists in the map object and 0 (zero) if the key does not exist. The value pointed to will not be modified if the existence test fails with an error.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. When the asynchronous execution completes successfully, `exists` will contain the existence test results.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.9.6 H5Mget_ff

Name: H5Mget_ff

Signature:

herr_t `H5Mget_ff(` *hid_t* `map_id,` *hid_t* `key_mem_type_id`, *const void* `*key`, *hid_t* `val_mem_type_id`, *void* `*value,` *hid_t* `dxpl_id,` *hid_t* `rcntxt_id`, *hid_t* `es_id )`

Purpose:

Retrieve the value for a given key from a map object, possibly asynchronously.

Description:

`H5Mget_ff` retrieves the value for the key, `key`, from the map object specified by `map_id`. `map_id` must be in scope for the read context identified by `rcntxt_id`.

The `key` parameter points to the memory buffer holding the key whose value is being retrieved; `key_mem_type_id` specifies the datatype of the `key` parameter.

The `value` parameter points to the memory buffer where the value should be stored; `val_mem_type_id` specifies the datatype of the `value` parameter.

If `key_mem_type_id` is not the same as the datatype of the keys in the map object, the `key` parameter will automatically undergo datatype conversion in order to locate the correct key in the map object. If `val_mem_type_id` is not the same as the datatype of the values in the map object, the value retrieved will automatically undergo datatype conversion when placed in the `value` buffer.

The data transfer property list, `dxpl_id`, may modify the operation's behaviour. When the user sets a property (via a call to `H5Pset_dxpl_checksum_ptr`) to pass a checksum pointer to the library, the checksum for `value` (not `key`) is placed at the pointer location when the operation completes. To get from a replica that was previously brought into the burst buffer by a call to `H5Mprefetch_ff`, use the read replica property in the data transfer property list. The call `H5Pset_dxpl_replica()` sets the read replica property.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `map_id`	IN: Identifier of the map object. Must be in scope for the read context.
hid_t `key_mem_type_id`	IN: Datatype of the `key` parameter.
const void `*key`	IN: Pointer to key whose value is being retrieved.
hid_t `val_mem_type_id`	IN: Datatype of the `value` parameter.
void `*value`	OUT: Pointer to buffer for the retrieved value.
hid_t `dxpl_id`	IN: Data transfer property list. If specified, the read replica property directs the get to access pre-fetched data.

hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.
Quarter 7: Added information about gets from prefetched replicas.

Man Page Status:

No known issues.

4.9.7 H5Mget_count_ff

Name: H5Mget_count_ff

Signature:

herr_t `H5Mget_count_ff`(*hid_t* `map_id`, *hsize_t* `*count`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Retrieve the number of key/value pairs in a map object, possibly asynchronously.

Description:

`H5Mget_count_ff` retrieves the number of key/value pairs in the map object specified by `map_id`. `map_id` must be in scope for the read context identified by `rcntxt_id`.

The number of pairs is returned in `count.`

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `map_id`	IN: Identifier of the map object. Must be in scope for the read context.
hsize_t `*count`	OUT: The number of key/value pairs in the map object.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.9.8 H5Mget_types_ff

Name: H5Mget_types_ff

Signature:

herr_t `H5Mget_types_ff`(*hid_t* `map_id`, *hid_t* *`key_type_id`, *hid_t* *`val_type_id`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Retrieve the datatypes for the keys and values of a map object, possibly asynchronously.

Description:

`H5Mget_types_ff` retrieves the datatypes for the keys and values of the map object specified by `map_id`. `map_id` must be in scope for the read context identified by `rcntxt_id`.

The datatype of the keys is returned in `key_type_id` and the datatype of the values is returned in `val_type_id`.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `map_id`	IN: Identifier of the map object. Must be in scope for the read context.
hid_t *`key_type_id`	OUT: Datatype of the keys in the map object.
hid_t *`val_type_id`	OUT: Datatype of the values in the map object.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.9.9 H5Miterate_ff

Name: H5Miterate_ff

Signature:

herr_t `H5Miterate_ff(` *hid_t* `map_id,` *hid_t* `key_mem_type_id`, *hid_t* `val_mem_type_id`, *H5M_iterate_func_t* `callback_func`, *void* `*func_data,` *hid_t* `rcntxt_id )`

Purpose:

Iterate over the key/value pairs in a map object, possibly asynchronously.

Description:

`H5Miterate_ff` iterates over the key/values pairs in the specified map object and invokes a user-defined callback routine for each pair in the map.

The map object to be iterated over is specified by `map_id`. `map_id` must be in scope for the read context identified by `rcntxt_id`.

`key_mem_type_id` and `val_mem_type_id` specify the datatypes for the keys and values that will be presented to the callback routine.

The user-defined callback function, specified by `callback_func`, is invoked for each pair in the map and passed the key, value, and user-provided `func_data`. Note that `func_data` can include a transaction id or read context, allowing the user's callback function to do updates to or reads from an HDF5 container as part of its operation.

The prototype for `H5M_iterate_func_t` is:

```
typedef herr_t (*H5M_iterate_func_t)(const void *key, const void *value,
void *func_data);
```

The iteration callback routine should obey the same rules as other HDF5 iteration callbacks: return `H5_ITER_ERROR` for an error condition (which will stop iteration), `H5_ITER_CONT` for success (with continued iteration) and `H5_ITER_STOP` for success (and stop iteration).

`rcntxt_id` indicates the read context for this operation.

There is no asynchronous execution option for this function because the user callback function cannot be invoked asynchronously.

Parameters:

hid_t `map_id`	IN: Identifier of the map object. Must be in scope for the read context.
hid_t `key_mem_type_id`	IN: Datatype for the key that is presented to the callback routine.
hid_t `val_mem_type_id`	IN: Datatype for the value that is presented to the callback routine.
H5M_iterate_func_t `callback_func`	IN: User's function to pass each attribute to
void `*func_data`	IN/OUT: User's data to pass through to callback function
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.

Returns:

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

History:

Quarter 5: Man page added but not yet implemented.

Man Page Status:

No known issues.
Future consideration: Should this have a dxpl so that values can be checksummed?
Future consideration: Should you be able to specify a replica id so you can read from prefetched data?

4.9.10 H5Mopen_ff

Name: H5Mopen_ff

Signature:

hid_t `H5Mopen_ff`(*hid_t* `loc_id`, *const char* *`name`, *hid_t* `mapl_id`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Open an existing map object possibly asynchronously.

Description:

`H5Mopen_ff` opens the existing map object specified by `loc_id` and `name`.

`loc_id` may be a file identifier or a group identifier. `name` may be either an absolute path in the file or a relative path from `loc_id` naming the map object. Both `loc_id` and `name` must be in scope for the read context identified by `rcntxt_id`.

The map access property list, `mapl_id`, provides information regarding access to the map object. The map access property list currently has no effect and should be set to `H5P_DEFAULT`.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

To conserve and release resources, the map object should be closed with `H5Mclose_ff` when access is no longer required.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 group identifier or file identifier that is in scope for the read context.
const char *`name`	IN: Map object name The map object name (path to the map object) can be specified relative to `loc_id` or absolute from the file's root group, and must be in scope for the read context.
hid_t `mapl_id`	IN: Map object access property list *Currently not used; specify `H5P_DEFAULT`.*
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `eq_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a map object identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the map object is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a "normal" map object identifier.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.9.11 H5Mprefetch_ff

Name: H5Mprefetch_FF

Signature:

herr_t `H5Mprefetch_ff(` *hid_t* `map_id,` *hid_t* `rcntxt_id,` *hid_t* `*replica_id,` *hid_t* `dxpl_id,` *hid_t* `es_id )`

Purpose:

Prefetch all or part of a map object from persistent storage to burst buffer storage, possibly asynchronously.

Description:

`H5Mprefetch_ff` prefetches all or part of a map object, specified by its identifier `map_id`, from persistent storage (DAOS) into burst buffer storage.

`rcntxt_id` indicates the read context for this operation.

`replica_id,` the replica identifier, is set to indicate where the pre-fetched key-value data can be found in the burst buffer, and is passed to subsequent `H5Mget_ff` and `H5Mevict_ff` calls.

`dxpl_id,` a data transfer property list identifier, is used to specify partial maps (key range selection) and control layout of the fetched data on the burst buffers.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The `replica_id` is not valid until the operation has completed, if it is executing asynchronously.

Limitations / Future Considerations:

For the EFF prototype project, only the primay IOD KV object associated with the HDF5 map will be prefetched; auxiliary IOD objects (including the Blob objects that hold variable-length data) remain on persistent storage (DAOS). For more information, and other potential extensions, refer to the document *Burst Buffer Space Management – Prototype to Production*.

In the EFF prototypeproject, only the H5Mget_ff routine accepts the replica_id and can read from prefetched map objects. Other routines, such as H5Mexists_ff, should also be able to access the prefetched data in a production release.

In Quarter 7, only complete maps and the default layout are supported.

Parameters:

hid_t `map_id`	IN: Identifier of the map being prefetched. `map_id` must be in scope for the read context.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t * `replica_id`	IN: Identifier of the replicated data in the burst buffer.
hid_t `mapl_id`	IN: Identifier of an access property list for this I/O operation.
hid_t `es_id`	IN: The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 7.

Man Page Status:

Will need updates in Quarter 8 when additional features are implemented.

4.9.12 H5Mset_ff

Name: H5Mset_ff

Signature:

herr_t `H5Mset_ff(` *hid_t* `map_id,` *hid_t* `key_mem_type_id`, *const void* `*key`, *hid_t* `val_mem_type_id`, *const void* `*value,` *hid_t* `dxpl_id,` *hid_t* `trans_id`, *hid_t* `es_id )`

Purpose:

Set the value for a given key in a map object, possibly asynchronously.

Description:

`H5Mset_ff` sets the value for a key in a map object. If the key did not exist in the map object previously, this function creates a new key/value pair in the map. If the key already existed in the map object, this function overwites the previous value in the existing key/value pair.

The map object is specified by `map_id`, which must be in scope for the transaction identified by `trans_id`.

The `key` parameter points to the memory buffer holding the key whose value is being set; `key_mem_type_id` specifies the datatype of the `key` parameter.

The `value` parameter points to the memory buffer of the value that will be stored; `val_mem_type_id` specifies the datatype of the `value` parameter.

If `key_mem_type_id` is not the same as the datatype of the keys in the map object, the `key` parameter will automatically undergo datatype conversion in order to locate the correct key in the map object. If `val_mem_type_id` is not the same as the datatype of the values in the map object, the `value` parameter will automatically undergo datatype conversion when stored in the map object.

The data transfer property list, `dxpl_id,` may modify the operation's behaviour. When the user sets a property (via a call to `H5Pset_dxpl_checksum)` to pass a checksum, the checksum given should be for `value` (not `key`).

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `map_id`	IN: Identifier of the map object. Must be in scope for the transaction.
hid_t `key_mem_type_id`	IN: Datatype of the `key` parameter.
const void `*key`	IN: Pointer to key whose value is being set.
hid_t `val_mem_type_id`	IN: Datatype of the `value` parameter.
const void `*value`	IN: Pointer to buffer holding the new value.
hid_t `dxpl_id`	IN: Data transfer property list.
hid_t `trans_id`	IN: Transaction identifier indicating the transaction this operation is part of.

hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.10 H5O: Object APIs

These routines are used to operate on HDF5 Objects, and are designed to be used in conjunction with the Link APIs (H5L).

The routines ending in _ff have different signatures than the standard HDF5 library routines.

Man pages for routines whose user interface is unchanged from the standard HDF5 implementation can be found at: http://www.hdfgroup.org/HDF5/doc/RM/RM_H5O.html.

Routine	**Implemented**	**Notes**
H5Oclose_ff	Quarter 4	
H5Oexists_by_name_ff	Quarter 4	
H5Oget_comment_ff	Quarter 4	
H5Oget_comment_by_name_ff	Quarter 4	
H5Oget_info_ff	Quarter 6	
H5Oget_info_by_name_ff	Quarter 6	
H5Oget_token	Quarter 6	
H5Olink_ff	Quarter 4	
H5Oopen_ff	Quarter 5	Q5: Changed from standard HDF5 H5Oopen, which was implemented in Q4.
H5Oopen_by_token	Quarter 6	
H5Oset_comment_ff	Quarter 4	
H5Oset_comment_by_name_ff	Quarter 4	
H5Ovisit_ff		Not implemented in prototype.
H5Ovisit_by_name_ff		Not implemented in prototype.
H5Oincr_ref_count		Not implemented in prototype.
H5Odecr_ref_count		Not implemented in prototype.
H5Ocopy_ff		Q5: Wjill not implement for prototype. Early version done in Q4, but very limited functionality or value without support for object copies from lower layers in the stack. Cross-container copies with transactions raise additional issues so removed.
H5Oopen_by_addr		Doesn't make sense for FF storage. Will not implement
H5Oget_info_by_idx		Not implemented in prototype.
H5Oopen_by_idx		Not implemented in prototype.

4.10.1 H5Oclose_ff

Name: H5Oclose_ff

Signature:

herr_t `H5Oclose_ff(` *hid_t* `object_id,` *hid_t* `es_id )`

Purpose:

Close an object in an HDF5 file, possibly asynchronously.

Description:

`H5Oclose_ff` closes the group, dataset, map, or named datatype specified by `object_id`.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

This function is the companion to `H5Oopen_ff`, and has the same effect as calling `H5Gclose_ff`, `H5Dclose_ff`, `H5Fclose_ff`, or `H5Tclose_ff`.

`H5Oclose_ff` is *not* used to close a dataspace, attribute, property list, or file.

Parameters:

hid_t `object_id`	IN: Object identifier
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from from event queue to event stack. Added map object.

Man Page Status:

No known issues.

4.10.2 H5Oexists_by_name_ff

Name: H5Oexists_by_name_ff

Signature:

herr_t `H5Oexists_by_name_ff`(*hid_t* `loc_id`, *const char* * `name`, *hbool_t* `*exists`, *hid_t* `lapl_id`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Determine whether a link resolves to an actual object, possibly asynchronously.

Description:

`H5Oexists_by_name_ff` allows an application to determine whether the link `name` in the group or file specified by `loc_id` resolves to an HDF5 object or if the link dangles. The link may be of any type, but hard links will always resolve to objects and do not need to be verified. The EFF stack currently does not support external or user-defined links. `loc_id and name` must both be in scope for the read context identified by `rcntxt_id`.

The results of the existence test are put in `exists`. Note that this value will not be set until after the function completes, which may be later than when the call returns for asynchronous execution. Traditionally the existence results were indicated by the function call's return value, but with the support for asynchronous execution that is no longer possible and the FastForward version included the new `exists` parameter.

`rcntxt_id` indicates the read context for this operation.

The link access property list, `lapl_id`, may provide information regarding the properties of links. The link access property list currently has no effect in the EFF stack and should be set to `H5P_DEFAULT`.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Note that `H5Oexists_by_name_ff` only verifies that the target object exists. If `name` includes either a relative path or an absolute path to the target link, intermediate steps along the path must be verified before the existence of the target link can be safely checked. If the path is not verified and an intermediate element of the path does not exist, `H5Oexists_by_name_ff` will fail. The example in the next paragraph illustrates one step-by-step method for verifying the existence of a link with a relative or absolute path.

Example: Use the following steps to verify the existence of the link `datasetD` in the group `group1/group2/softlink_to_group3/`, where `group1` is a member of the group specified by `loc_id`:

- First use `H5Lexists_ff` to verify that a link named `group1` exists.
- If `group1` exists, use `H5Oexists_by_name_ff` to verify that the link `group1` resolves to an object.
- If `group1` exists, use `H5Lexists_ff` again, this time with `name` set to `group1/group2`, to verify that the link `group2` exists in `group1`.
- If the `group2` link exists, use `H5Oexists_by_name_ff` to verify that `group1/group2` resolves to an object.
- If `group2` exists, use `H5Lexists_ff` again, this time with `name` set to `group1/group2/softlink_to_group3`, to verify that the link `softlink_to_group3` exists in `group2`.

- If the softlink_to_group3 link exists, use H5Oexists_by_name_ff to verify that group1/group2/softlink_to_group3 resolves to an object.
- If softlink_to_group3 exists, you can now safely use H5Lexists_ff with name set to group1/group2/softlink_to_group3/datasetD to verify that the target link, datasetD, exists.
- And finally, if the link datasetD exists, use H5Oexists_by_name_ff to verify that group1/group2/softlink_to_group3/datasetD resolves to an object.

If the link to be verified is specified with an absolute path, the same approach should be used, but starting with the first link in the file's root group. For instance, if datasetD were in /group1/group2/softlink_to_group3, the first call to H5Lexists_ff would have name set to /group1.

Note that this is an outline and does not include all necessary details. Depending on circumstances, for example, an application may need to verify the type of an object also.

Parameters:

hid_t loc_id	IN: Identifier of the file or group to query. Must be in scope for the read context.
const char *name	IN: The name of the link to check. Must be in scope for the read context.
hbool_t *exists	IN: Pointer to returned results indicating existence of link. When successful, will be a positive value to indicate the link exists and 0 (zero) if the link does not exist. The value pointed to will not be modified if the existence test failed for some reason.
hid_t lapl_id	IN: Link access property list identifier. *Currently not used in EFF; specify H5P_DEFAULT.*
hid_t es_id	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use H5_EVENT_STACK_NULL for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. When the asynchronous execution completes successfully, exists will contain the existence test results.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to read context id and from event queue to event stack. Added scope requirement and note that property list currently not used in EFF.

Future Considerations:

Order of parameters exists and lapl_id was reversed in initial design. Man page reflects implementation. May want to swap back prior to production release, or modify in H5Aexists_by_name_ff for consistency.

Man Page Status:

No known issues.

4.10.3 H5Oget_comment_ff

Name: H5Oget_comment_ff

Signature:

herr_t `H5Oget_comment_ff(` *hid_t* `object_id`, *char* *`comment`, *size_t* `bufsize`, *ssize_t* *`comment_size`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Retrieve comment for specified object, possibly asynchronously.

Description:

`H5Oget_comment_ff` retrieves the comment for the specified object and puts it into the buffer `comment`.

The object with the comment is specified by the identifier `object_id`. `object_id` must be in scope for the read context specified by `rcntxt_id.`

The size in bytes of the buffer `comment`, including the `NULL` terminator, is specified in `bufsize`. If `bufsize` is unknown, a preliminary `H5Oget_comment_ff` call with the pointer `comment` set to `NULL` will report the size of the comment *without* the `NULL` terminator.

If `bufsize` is set to a smaller value than described above, only `bufsize` bytes of the comment, without a `NULL` terminator, are returned in `comment`.

If an object does not have a comment, the empty string is returned in `comment`.

Upon success, the number of characters in the comment, not including the NULL terminator, or zero (0) if the comment has no comment, is put in `comment_size`. The value returned in `comment_size` may be larger than `bufsize.` On failure, `comment_size` will contain a negative value. Note that this value will not be set until after the function completes, which may be later than when the call returns for asynchronous execution. Traditionally the comment size was indicated by the function call's return value, but with the support for asynchronous execution that is no longer possible and the FastForward version included the new `comment_size` parameter.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `object_id`	IN: Identifier for the object whose comment is to be retrieved. Must be in scope for the read context.
char *`comment`	OUT: The comment.
size_t `bufsize`	IN: Anticipated required size of the `comment` buffer.
ssize_t *`comment_size`	OUT: Pointer to returned results indicating comment size. When successful, will contain the number of characters in the comment, not including the NULL

	terminator, or zero (0) if the comment has no comment. On failure, will contain a negative value.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. When the asynchronous execution completes successfully, `comment_size` will contain the number of characters in the comment, not including the NULL terminator, or zero (0) if the comment has no comment. On failure, comment_size will be a negative number.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to read context id and from event queue to event stack. Added scope requirement.

Man Page Status:

No known issues.

4.10.4 H5Oget_comment_by_name_ff

Name: H5Oget_comment_by_name_ff

Signature:

herr_t `H5Oget_comment_by_name_ff`(*hid_t* `loc_id`, *const char* *`name`, *char* *`comment`, *size_t* `bufsize`, *hid_t* `lapl_id`, *ssize_t* *`comment_size`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Retrieve comment for specified object, possibly asynchronously.

Description:

`H5Oget_comment_by_name_ff` `H5Oget_comment_ff` retrieves the comment for the specified object and puts it into the buffer `comment`.

The object with the comment is specified by `loc_id` and `name`. `loc_id` can specify any object in the file. `name` can be one of the following:
— The name of the object relative to `loc_id`
— An absolute name of the object, starting from `/`, the file's root group
— A dot (`.`), if `loc_id` fully specifies the object
Both `loc_id` and `name` must be in scope for the read context specified by `rcntxt_id.`

The size in bytes of the buffer `comment`, including the `NULL` terminator, is specified in `bufsize`. If `bufsize` is unknown, a preliminary `H5Oget_comment_by_name_ff` call with the pointer `comment` set to `NULL` will report the size of the comment *without* the `NULL` terminator.

If `bufsize` is set to a smaller value than described above, only `bufsize` bytes of the comment, without a `NULL` terminator, are returned in `comment`.

If an object does not have a comment, the empty string is returned in `comment`.

Upon success, the number of characters in the comment, not including the NULL terminator, or zero (0) if the comment has no comment, is put in `comment_size`. The value returned in `comment_size` may be larger than `bufsize.` On failure, `comment_size` will contain a negative value. Note that this value will not be set until after the function completes, which may be later than when the call returns for asynchronous execution. Traditionally the comment size was indicated by the function call's return value, but with the support for asynchronous execution that is no longer possible and the FastForward version included the new `comment_size` parameter.

The link access property list, `lapl_id`, may provide information regarding the properties of links. The link access property list currently has no effect in the EFF stack and should be set to `H5P_DEFAULT`.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier that is in scope for the read context

const char *`name`	IN: Object name The object name (path to the object) can be specified relative to `loc_id,` absolute from the file's root group, or '.' (a dot), and must be in scope for the read context.
char *`comment`	OUT: The comment.
size_t `bufsize`	IN: Anticipated required size of the `comment` buffer.
hid_t `lapl_id`	IN: Link access property list identifier. *Currently not used in EFF; specify `H5P_DEFAULT`.*
ssize_t *`comment_size`	OUT: Pointer to returned results indicating comment size. When successful, will contain the number of characters in the comment, not including the NULL terminator, or zero (0) if the comment has no comment. On failure, will contain a negative value.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. When the asynchronous execution completes successfully, `comment_size` will contain the number of characters in the comment, not including the NULL terminator, or zero (0) if the comment has no comment. On failure, comment_size will be a negative number.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to read context id and from event queue to event stack. Added scope requirement and note about link access property list under EFF.

Man Page Status:

No known issues.

4.10.5 H5Oget_info_ff

Name: H5Oget_info_ff

Signature:

herr_t `H5Oget_info_ff(` *hid_t* `object_id`, *H5O_ff_info_t* *`object_info`, *hid_t* `rcntxt_id`, *hid_t* `es_id )`

Purpose:

Retrieve the metadata for an object specified by an identifier, possibly asynchronously.

Description:

`H5Oget_info_ff` retrieves the metadata for the specified object and puts it into the struct *H5O_ff_info_t*.

An *H5O_ff_info_t* struct is defined (in H5FFpublic.h) as follows :

```
typedef struct H5O_ff_info_t {
    haddr_ff_t          addr;       /* Object address in file               */
    H5O_type_t          type;       /* Basic object type                    */
    unsigned            rc;         /* Reference count of object            */
    hsize_t             num_attrs;  /* # of attributes attached to object   */
} H5O_ff_info_t;
```

The object with the info is specified by the identifier `object_id`. `object_id` must be in scope for the read context specified by `rcntxt_id.`

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `object_id`	IN: Identifier for the object whose metadata is to be retrieved. Must be in scope for the read context.
H5O_ff_info_t `*object_info`	OUT: Buffer in which to return object information.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. When the asynchronous execution completes successfully, `object_info` will contain all the metadata values for that object.

History:

Implemented in Q6; Man page added in Q8.

Man Page Status:

No known issues.

4.10.6 H5Oget_info_by_name_ff

Name: H5Oget_info_by_name_ff

Signature:

herr_t `H5Oget_info_by_name_ff`(*hid_t* `loc_id`, *const char* *`object_name`, *H5O_ff_info_t* **object_info*, *hid_t* `lapl_id`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Retrieve the metadata for an object specified by a location and a pathname, possibly asynchronously.

Description:

`H5Oget_info_by_name_ff` retrieves the metadata for the object specified by the `loc_id` and `object_name` and puts it into the struct *H5O_ff_info_t*.

An *H5O_ff_info_t* struct is defined (in H5FFpublic.h) as follows :

```
typedef struct H5O_ff_info_t {
    haddr_ff_t          addr;       /* Object address in file              */
    H5O_type_t          type;       /* Basic object type                   */
    unsigned            rc;         /* Reference count of object           */
    hsize_t             num_attrs;  /* # of attributes attached to object  */
} H5O_ff_info_t;
```

The link access property list, `lapl_id`, is not currently used; it should be passed in as `H5P_DEFAULT`

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `loc_id`	IN: File or group identifier specifying location of group in which object is located. Must be in scope for the read context.
const char *`name`	IN: Name of object, relative to `loc_id`.
H5O_ff_info_t **object_info*	OUT: Buffer in which to return object information.
hid_t `lapl_id`	IN: Link access property list. (Not currently used; pass as H5P_DEFAULT.)
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. When the asynchronous execution completes successfully, `object_info` will contain all the metadata values for that object.

History:

Implemented in Q6; Man page added in Q8.

Man Page Status:

No known issues.

4.10.7 H5Oget_token

Name: H5Oget_token

Signature:

herr_t `H5Oget_token` (*hid_t* `obj_id`, *void* *`token` , *size_t* *`token_size`)

Purpose:

Retrieve an object token buffer containing all necessary information to open the object from any rank.

Description:

`H5Oget_token` retrieves the object token containing all the object metadata needed to open the object from any rank in the application, even in the same transaction that the object was created in.

The token must be a user allocated buffer with at least a size of the object token. When the function is called with a NULL token buffer, the size of the token buffer is returned in `token_size.`

The actions performed by this function are all local to the compute node so an asynchronous execution option is not provided.

Parameters:

hid_t `obj_id`	IN: Object identifier indicating the object that the token needs to be retrieved for.
void *`token`	OUT: The pointer to buffer where the object token is retrieved.
size_t `token_size`	OUT: The token size in bytes.

Returns:

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

History:

Implemented in Q6; Man page added in Q8.

Man Page Status:

No known issues.

4.10.8 H5Olink_ff

Name: H5Olink_ff

Signature:

herr_t `H5Olink_ff`(*hid_t* `object_id`, *hid_t* `new_loc_id`, *const char* *`new_link_name`, *hid_t* `lcpl`, *hid_t* `lapl`, *hid_t* `trans_id`, *hid_t* `es_id`)

Purpose:

Create a hard link to an object in an HDF5 file, possibly asynchronously.

Description:

`H5Olink_ff` creates a new hard link to an object in an HDF5 file.

`new_loc_id` and `new_link_name` specify the location and name of the new link, while `object_id` identifies the object that the link points to. `new_loc_id` and `object_id` must both be in scope for the transaction identified by `trans_id.`

`H5Olink_ff` is designed for two purposes:

- To create the first hard link to an object that has just been created with one of the `H5*create_anon` functions or with `H5Tcommit_anon`. (The prototype EFF stack does not support the _anon functions.)
- To add additional structure to an existing file so that, for example, an object can be shared among multiple groups.

`lcpl` and `lapl` are the link creation and access property lists associated with the new link. Neither of these property lists currently have any effect in the EFF stack and both shouldbe set to `H5P_DEFAULT`.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `object_id`	IN: Object to be linked. May be any HDF5 object identifier (group, dataset, map, or named datatype) that is in scope for the transaction.
hid_t `new_loc_id`	IN: File or group identifier specifying location at which object is to be linked. Must be in scope for the transaction.
const char `*new_link_name`	IN: Name of link to be created, relative to `new_loc_id`.
hid_t `lcpl_id`	IN: Link creation property list identifier. *Currently not used in EFF; specify `H5P_DEFAULT.`*
hid_t `lapl_id`	IN: Link access property list identifier. *Currently not used in EFF; specify `H5P_DEFAULT.`*
hid_t *`trans_id`*	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function

call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement. Noted what is not supported in EFF stack and removed example that demonstrated functionality that isn't available in EFF.

Man Page Status:

No known issues.

4.10.9 H5Oopen_ff

Name: H5Oopen_ff

Signature:

hid_t `H5Oopen_ff`(*hid_t* `loc_id`, *const char* *`name`, *hid_t* `lapl_id`, *hid_t* `rcntxt_id`)

Purpose:

Open an existing object.

Description:

`H5Oopen_ff` opens the existing HDF5 object specified by `loc_id` and `name`.

`loc_id` is a location identifier and may be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier.

`name` is the path to the object relative to loc_id. If `loc_id` fully specifies the object that is to be opened, `name` should be `'.'` (a dot).

Both `loc_id` and `name` must be in scope for the read context identified by `rcntxt_id`.

The link access property list, `lapl_id`, currently has no effect in the EFF stack and should be set to `H5P_DEFAULT`.

`rcntxt_id` indicates the read context for this operation.

There is no asynchronous execution option for this function because the object type must be known before the object id (even a future object id) can be returned. Since the object type is not known until the operation completes, executing the operation asynchronously is not possible.

To conserve and release resources, the object should be closed with `H5Oclose_ff` when access is no longer required.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier that is in scope for the read context.
const char *`name`	IN: Object name The object name (path to the map object) can be specified relative to `loc_id` or absolute from the file's root group, and must be in scope for the read context.
hid_t `lapl_id`	IN: Link object access property list *Currently not used in EFF; specify `H5P_DEFAULT`.*
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.

Returns:

Returns an object identifier if successful; otherwise returns a negative value.

History:

Quarter 4: Extended H5Open to support EFF stack.
Quarter 5: New signature, H5Oopen_ff, to include the read context. Man page created.

Man Page Status:

No known issues.

4.10.10 H5Oopen_by_token

Name: H5Oopen_by_token

Signature:

hid_t `H5Oopen_by_token`(*const void* *`token`, *hid_t* `trans_id`, *hid_t* `es_id`)

Purpose:

Open an existing object using a token buffer retrieved by `H5Oget_token`.

Description:

`H5Oopen_by_token` opens the existing HDF5 object described by the token buffer `token`.

`token` is a buffer obtained by using the function `H5Oget_token`. It contains all required data to open the object and add updates to it.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

`H5Oopen_by_token` allows the user to open an object in the same transaction it was created in. This would allow users to create the object independently from one rank, retrieve the token for that object, and send the token to other ranks that can open the object using `H5Oopen_by_token` and add updates to that object without having to start another transaction based on the read context of the transaction that the object was created in.

To conserve and release resources, the object should be closed with `H5Oclose_ff` when access is no longer required.

Parameters:

const void *`token`	IN: object token pointer to a token buffer retrieved by `H5Oopen_by_token.`
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns an object identifier if successful; otherwise returns a negative value.

History:

Implemented in Q6; Man page added in Q8.

Man Page Status:

No known issues.

4.10.11 H5Oset_comment_ff

Name: H5Oset_comment_ff

Signature:

herr_t `H5Oset_comment_ff`(*hid_t* `object_id`, *const char* *`comment`, *hid_t* `trans_id`, *hid_t* `es_id`)

Purpose:

Set comment for specified object, possibly asynchronously.

Deprecated Function:

This function is deprecated in favor of object attributes.

Description:

`H5Oset_comment_ff` sets the comment for the specified object to the contents of `comment`. Any previously existing comment is overwritten. Comments should be relatively short, null-terminated, ASCII strings.

The target object is specified by an identifier, `object_id`, which must be in scope for the transaction indicated by `trans_id.`

If `comment` is the empty string or a null pointer, any existing comment message is removed from the object.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Comments can be attached to any object that has an object header. Datasets, groups, maps, and named datatypes have object headers. Symbolic links do not have object headers. In the EFF stack, comments cannot be attached to attributes.

Parameters:

hid_t `object_id`	IN: Identifier of the target object.
const char *`comment`	IN: The new comment.
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `eq_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Noted that you can't add comments to attributes in EFF.

Man Page Status:

No known issues.

4.10.12 H5Oset_comment_by_name_ff

Name: H5Oset_comment_by_name_ff

Signature:

herr_t `H5Oset_comment_by_name_ff`(*hid_t* `loc_id`, *const char* *`name`, *const char* *`comment`, *hid_t* `lapl_id,` uint64_t `trans`, hid_t `eq_id`)

Purpose:

Set comment for specified object, possibly asynchronously.

Deprecated Function:

This function is deprecated in favor of object attributes.

Description:

`H5Oset_comment_by_name_ff` sets the comment for the specified object to the contents of `comment`. Any previously existing comment is overwritten. Comments should be relatively short, null-terminated, ASCII strings.

The target object is specified by `loc_id` and `name`. `loc_id` can specify any object in the file. `name` can be one of the following:
— The name of the object relative to `loc_id`
— An absolute name of the object, starting from `/`, the file's root group
— A dot (`.`), if `loc_id` fully specifies the object
Both `loc_id` and `name` must be in scope for the read context specified by `rcntxt_id.`

If `comment` is the empty string or a null pointer, any existing comment message is removed from the object.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Comments can be attached to any object that has an object header. Datasets, groups, maps, and named datatypes have object headers. Symbolic links do not have object headers.

Comments can be attached to any object that has an object header. Datasets, groups, maps, and named datatypes have object headers. Symbolic links do not have object headers. In the EFF stack, comments cannot be attached to attributes. In the EFF stack, comments cannot be attached to attributes.

`lapl_id` contains a link access property list identifier. A link access property list can come into play when traversing links to access an object. The link access property list currently has no effect in the EFF stack and should be set to `H5P_DEFAULT`.

Parameters:

hid_t `loc_id`	IN: Identifier of a file, group, dataset, map, or named datatype. Must be in scope for the transaction.
const char *`name`	IN: Name of the object whose comment is to be set or reset, specified as a path relative to `loc_id`. `name` can be '.' (a dot) if `loc_id` fully specifies the object for which the comment is to be set. Must be in scope for the transaction.

const char *`comment`	IN: The new comment.
hid_t `lapl_id`	IN: Link access property list identifier. *Currently not used in EFF; specify `H5P_DEFAULT`.*
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement and noted that link access property lists not supported in EFF. Noted that you can't add comments to attributes in EFF.

Man Page Status:

No known issues.

4.11 H5P: Property APIs

These routines allow applications to set values in HDF5 Property Lists. The routines appear in the table below grouped by the property list they apply to; the reference manual pages are organized alphabetically.

With the exception of H5Pset_fapl_iod, H5Pset_prefetch_selection, and H5Pset_prefetch_range, each of the routines has a H5Pget_* variant that can be used to get the value of the property. If the type of the last parameter in the **set** call is **type***,* then the type of the last parameter in the **get** call is ***type***.

For example:

herr_t H5P**set**_dxpl_replica(hid_t dxpl_id, **hrpl_t replica_id**);

and

herr_t H5P**get**_dxpl_replica(hid_t dxpl_id, **hrpl_t *replica_id**);

The H5Pget_ variants for the H5Pset_ routines do not appear in this section of the reference manual, but they are available in the library.

The H5Pget_get_[xapl | xxpl]_transaction and H5Pget_[xapl | xxpl]_read_context routines are included, as they do not have H5Xset_ counterparts.

These routines did not exist prior to the Exascale FastForward project.

Routine	**Implemented**	**Notes**
Data Creation		
H5Pset_dcpl_dim_layout	Quarter 8	
H5Pset_dcpl_stripe_count	Quarter 8	
H5Pset_dcpl_stripe_size	Quarter 8	
Data Transfer		
H5Pset_dxpl_checksum	Quarter 4	
H5Pset_dxpl_checksum_ptr	Quarter 4	
H5Pset_dxpl_inject_corruption	Quarter 4	
H5Pset_prefetch_layout	Quarter 8	
H5Pset_prefetch_range	Quarter 8	
H5Pset_prefetch_selection	Quarter 8	
H5Pset_rawdata_integrity_scope	Quarter 5	
H5Pset_dxpl_replica	Quarter 7	
Index Access		
H5Pget_xapl_transaction	Quarter 8	
H5Pget_xapl_read_context	Quarter 8	
Index Transfer		

H5Pget_xxpl_transaction	Quarter 8	
H5Pget_xxpl_read_context	Quarter 8	
Object Creation		
H5Pset_ocpl_enable_checksum	Quarter 7	
View Creation		
H5Pset_view_elmt_scope	Quarter 8	
File Access		
H5Pset_fapl_iod	Quarter 3	
H5Pset_metadata_integrity_scope	Quarter 5	
Read Context Acquire		
H5Pset_rcapl_version_request	Quarter 5	
Transaction Start		
H5Pset_trspl_num_peers	Quarter 5	

4.11.1 H5Pset_dcpl_dim_layout

Name: H5Pset_dcpl_dim_layout

Signature:

herr_t `H5Pset_dcpl_dim_layout` (*hid_t dcpl_id, H5FF_dset_dim_layout_t dims_layout*)

Purpose:

Set the physical dimension layout property of a dataset on a dataset creation property list.

Description:

`H5Pset_dcpl_dim_layout` sets a property in the `dcpl_id` dataset creation property list to indicate the physical dimension order of how the dataset to be layout on disk. The property layout can be set to H5D_ROW_MAJOR or H5D_COL_MAJOR.

Parameters:

hid_ t dcpl_id	IN: property list identifier
H5FF_dset_dim_layout_t dims_layout	IN: Layout dimension order.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.11.2 H5Pset_dcpl_stripe_count

Name: H5Pset_dcpl_stripe_count

Signature:
herr_t `H5Pset_dcpl_stripe_count` (*hid_t dcpl_id, size_t stripe_count*)

Purpose:
Set the number of storage targets for the physical location of a dataset in the dataset creation property list.

Description:
`H5Pset_dcpl_stripe_count` sets a property in the `dcpl_id` dataset creation property list to indicate the number of storage targets the dataset will be stripped on. The default count of 0 indicates that all storage targets can be used.

Parameters:

hid_ t dcpl_id	IN: property list identifier
size_t stripe_count	IN: Number of storage targets.

Returns:
Returns a non-negative value if successful. Otherwise returns a negative value.

History:
Added in Quarter 8.

Man Page Status:
No known issues.

4.11.3 H5Pset_dcpl_stripe_size

Name: H5Pset_dcpl_stripe_size

Signature:

herr_t `H5Pset_dcpl_stripe_size` (*hid_t dcpl_id, size_t stripe_size*)

Purpose:

Set the stripe size in terms of dataset elements for the physical location of a dataset in the dataset creation property list.

Description:

`H5Pset_dcpl_stripe_size` sets a property in the `dcpl_id` dataset creation property list to indicate the number of dataset elements to stripe the dataset on across the storage targets.

Parameters:

hid_ t dcpl_id	IN: property list identifier
size_t stripe_size	IN: stripe size.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.11.4 H5Pset_dxpl_checksum

Name: H5Pset_dxpl_checksum

Signature:

herr_t `H5Pset_dxpl_checksum(` *hid_t* `dxpl_id`, *uint32_t* `value`)

Purpose:

Specify a user-supplied checksum for a write data transfer.

Description:

`H5Pset_dxpl_checksum` sets a property in the `dxpl_id` data transfer property list specifying `value` as a user-supplied checksum for data written with a call to `H5Dwrite_ff` or `H5Mset` using `dxpl_id.`

When this is set, the HDF5 IOD VOL client will create a checksum for the data and verify that checksum matches the value supplied by the user before sending the data on to the HDF5 VOL IOD server.

Regardless of whether the user supplies a checksum value, the HDF5 IOD VOL client will create a checksum that is compared with the value generated on the HDF5 IOD VOL server.

The user must call `H5checksum` to obtain the value used in the call to `H5Pset_dxpl_checksum.` This ensures that all levels of the stack are using compatible checksum algorithms.

See *Design and Implementation of FastForward Features in HDF5* for information on the interaction between transformation operations, such as datatype conversion, and end-to-end integrity support with checksums.

Parameters:

hid_t `dxpl_id`	IN: Data transfer property list identifier
uint32_t `value`	IN: User-supplied checksum obtained via call to `H5checksum`

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 4.
Quarter 5: Added `H5Mset` to routines that are affected by the dxpl setting. Added reference for interaction between transformations and checksums.

Man Page Status:

No known issues.

4.11.5 H5Pset_dxpl_checksum_ptr

Name: H5Pset_dxpl_checksum_ptr

Signature:

herr_t `H5Pset_dxpl_checksum_ptr(` *hid_t* `dxpl_id`, *uint32_t* `*value`)

Purpose:

Specify a memory location to receive the checksum from a read data transfer.

Description:

`H5Pset_dxpl_checksum_ptr` sets a property in the `dxpl_id` data transfer property list specifying `value` as a memory location to receive a checksum for data read with a call to `H5Dread_ff` or `H5Mget` using `dxpl_id.`

When this is set, the HDF5 IOD VOL client will put the checksum for the data into the memory location (`*value`) supplied by the user, allowing the user to compare the client's checksum to a value it generates using the `H5checksum` routine.

Regardless of whether the user supplies a memory location for the checksum, the HDF5 IOD VOL client will create a checksum and compare it with the value generated on the HDF5 IOD VOL server before the operation completes.

The user must call `H5checksum` to obtain the value for its comparison with the received value. This ensures that all levels of the stack are using compatible checksum algorithms.

See *Design and Implementation of FastForward Features in HDF5* for information on the interaction between transformation operations, such as datatype conversion, and end-to-end integrity support with checksums.

Parameters:

hid_t `dxpl_id`	IN: Data transfer property list identifier
uint32_t `*value`	OUT: User-supplied memory location to receive checksum value

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 4.
Quarter 5: Added `H5Mget` to routines that are affected by the dxpl setting. Added reference for interaction between transformations and checksums.

Man Page Status:

No known issues.

4.11.6 H5Pset_dxpl_inject_corruption

Name: H5Pset_dxpl_inject_corruption

Signature:

herr_t `H5Pset_dxpl_inject_corruption(` *hid_t* `dxpl_id`, *hbool_t* `flag`)

Purpose:

Specify that data should be corrupted prior to transfer so that data integrity pipeline can be tested. This routine is for testing purposes only and should not be used in a real application.

Description:

`H5Pset_dxpl_inject_corruption` sets a property in the `dxpl_id` data transfer property list that can tell the HDF5 IOD VOL client (for writes) and HDF5 IOD VOL server (for reads) to corrupt the data being transferred with `H5Dwrite_ff` or `H5Dread_ff` using `dxpl_id.` Corruption will occur when when `flag` is `TRUE`.

Parameters:

hid_t `dxpl_id`	IN: Data transfer property list identifier
hbool_t `flag`	IN: Boolean (TRUE/FALSE), indicating if corruption should be injected. TRUE means inject corruption.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 4.

Man Page Status:

No known issues.

4.11.7 H5Pset_dxpl_replica

Name: H5Pset_dxpl_replica

Signature:

herr_t `H5Pset_dxpl_replica(` *hid_t* `dxpl_id`, *hrpl_t* `replica_id`)

Purpose:

Specify a property that identifies the object replica to be used.

Description:

`H5Pset_dxpl_replica` sets the replica property in the `dxpl_id` transfer property list, specifying an object replica identifier.

When this property is set, evict or read operations on the object will evict or read the specified replica rather than the transaction update data for the object.

Replica IDs are returned by the `H5Dprefetch_ff, H5Gprefetch_ff, H5Mprefetch_ff`, and `H5Tprefetch_ff` routines.

Replica properties are recognized by `H5Devict_ff, H5Mevict_ff, H5Gevict_ff`, `H5Tevict_ff, H5Dread_ff` and `H5Mget_ff` routines.

Limitations / Future Considerations:

For other potential extensions that are beyond the scope of the EFF prototype project, refer to the document *Burst Buffer Space Management – Prototype to Production*.

Parameters:

hid_t `dxpl_id`	IN: data transfer property list identifier
hrpl_t `replica_id`	IN: Identifier for an object replica that was previously prefeched into the Burst Buffer via a call to `H5?prefetch_ff,` where `?` may be "`D`", "`M`", "`G`", or "`T`", depending on the object type that was prefetched.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 7 as H5Pset_read_replica.
Quarter 8: Change name to H5Pset_dxpl_replica.

Man Page Status:

No known issues.

4.11.8 H5Pset_fapl_iod

Name: H5Pset_fapl_iod

Signature:

herr_t `H5Pset_fapl_iod(` *hid_t* `fapl_id`, *MPI_Comm* `comm`, *MPI_Info* `info`)

Purpose:

Specify that the IOD VOL plugin should be used to access the HDF5 container.

Description:

`H5Pset_fapl_iod` sets a property in the `fapl_id` file access property list specifying that the IOD VOL plugin should be use to perform I/O. The IOD VOL plugin is used to interact with the Exascale Fast Forward I/O stack, where HDF5 files are stored as IOD and DAOS containers and objects rather than in the native binary HDF5 File Format or another storage/access mechanism.

Calling this routine is *mandatory* to use the HDF5 FastForward (H5*_ff) APIs described in this document.

`H5Pset_fapl_iod` also stores the user-supplied MPI parameters `comm`, for communicator, and `info`, for information, in the file access property list `fapl_id`. That property list can then be used to create and/or open a file. The communicator and info parameters are used to set up communication channels for collective operations on the HDF5 container.

`comm` is the MPI communicator to be used for accessing the file. This function makes a duplicate of the communicator, so modifications to `comm` after this function call returns have no effect on the file access property list.

`info` is the MPI Info object use to pass MPI info parameters to the MPI library. This function makes a duplicate copy of the Info object, so modifications to the Info object after this function call returns will have no effect on the file access property list.

If the file access property list already contains previously-set communicator and Info values, those values will be replaced and the old communicator and Info object will be freed.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier
MPI_Comm `comm`	IN: MPI communicator
MPI_Info `info`	IN: MPI info object

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 3.

Man Page Status:

No known issues.

4.11.9 H5Pset_metadata_integrity_scope

Name: H5Pset_metadata_integrity_scope

Signature:

herr_t `H5Pset_metadata_integrity_scope(` *hid_t* `fapl_id`, *uint32_t* `scope )`

Purpose:

Specify the scope of checksum generation and verification for metadata transfer.

Description:

`H5Pset_metadata_integrity_scope` sets a property in the `fapl_id` file access property list specifying the scope of checksum generation and verification for metadata transfer in the Exascale Fast Forward stack. The specified scope will be used in all metadata transfer operations for containers opened with `fapl_id.` Note that different scopes may be specified for different containers, or for different opens of the same container.

The `scope` is a bitflag parameter that specifies which stages of the metadata transfer process will have checksums generated and verified. The possible values are:

`H5_CHECKSUM_NONE`	No checksums are computed or verified at any part of the stack.
`H5_CHECKSUM_TRANSFER`	Metadata is verified after transfer through mercury (the function shipper).
`H5_CHECKSUM_IOD`	Checksums are computed for the metadata, given to IOD when written, and verified when read.
`H5_CHECKSUM_MEMORY`	Metadata is verified when moved into memory.
`H5_CHECKSUM_ALL`	Metadata is checksummed and verified at all levels.

Multiple individual selections can be combined using the OR operator. For example:

```
status = H5_CHECKSUM_IOD || H5_CHECKSUM_MEMORY
```

Parameters:

hid_t `fapl_id`	IN: File access property list identifier
uint32_t `scope`	IN: Bitfield specifying the scope of checksums for raw data transfers

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 5.

Man Page Status:

To do: Update with description of what is considered metadata. Improve descriptions of scope options, maybe adding an ASCII picture of data flow.

4.11.10 H5Pset_ocpl_enable_checksum

Name: H5Pset_ocpl_enable_checksum

Signature:

herr_t `H5Pset_ocpl_enable_checksum(` *hid_t* `ocpl_id,` *hbool_t* `flag )`

Purpose:

Indicates whether lower layer in stack should enable checksums for an object.

Description:

`H5Pset_ocpl_enable_checksum` sets a Boolean flag on the object creation property list to indicate whether the VOL plugin (IOD for the Exascale Fast Forward project) should enable checksums on the object that is being created.

The `flag` is a boolean parameter that specifies whether checksums are to be enabled. The possible values are:

`0`	Checksums are not enabled.
`1`	Checksums are enabled.

The default for the Exascale Fast Forward project for objects crated without this creation propert list is to enable checksums at the IOD level.

Parameters:

ocpl_t `fapl_id`	IN: Object creation property list identifier
hbool_t `flag`	IN: Boolean flag specifying if checksums should be enabled by lower layer

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Implemented in Quarter 7; Man page added in Quarter 8.

Man Page Status:

No known issues.

4.11.11 H5Pset_prefetch_layout

Name: H5Pset_ prefetch_layout

Signature:

herr_t `H5Pset_prefetch_layout` (*hid_t dxpl_id, H5FF_layout_t layout*)

Purpose:

Set the prefetch layout property on the data transfer property list.

Description:

`H5Pset_prefetch_layout` sets a property in the `dxpl_id` to prefetch an object with a particular layout indicated by *layout*. The layout property value could be either H5_DEFAULT_LAYOUT, H5_LOCAL_NODE, or H5_CONTROLLED_LAYOUT. H5_DEFAULT_LAYOUT uses the default properties of the underlying library. H5_LOCAL_NODE prefetches the entire object to the node that the prefetch is executed on instead of striping over several nodes. H5_CONTROLLED_LAYOUT is not supported for now.

Parameters:

hid_t `dxpl_id`	IN: data transfer property list identifier
H5FF_layout_t layout	IN: layout type

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.11.12 H5Pset_prefetch_range

Name: H5Pset_ prefetch_range

Signature:

herr_t `H5Pset_prefetch_range` (*hid_t dxpl_id, hid_t keymem_type, const void *low_key, const void *high_key*)

Purpose:

Set the prefetch range property on the data transfer property list.

Description:

`H5Pset_prefetch_range` sets a property in the `dxpl_id` to prefetch a particular range of a Map object starting with *low_key* and ending with *high_key*. The default property is to prefetch the entire Map.

Parameters:

hid_t `dxpl_id`	IN: data transfer property list identifier
hid_t keymem_type	IN: datatype of the passed in keys
*const void * low_key*	IN: The starting range of the keys/values to prefetch
*const void *high_key*	IN: The ending range of the keys/values to prefetch

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.11.13 H5Pset_prefetch_selection

Name: H5Pset_ prefetch_selection

Signature:

herr_t `H5Pset_prefetch_selection` (*hid_t dxpl_id, hid_t dataspace*)

Purpose:

Set the prefetch selection for a dataset to be prefetched on the data transfer property list.

Description:

`H5Pset_prefetch_selection` sets a property in the `dxpl_id` to prefetch a portion of the dataset described by the selection in *dataspace*. The default property is to prefetch the entire selection.

Parameters:

hid_t `dxpl_id`	IN: data transfer property list identifier
hid_t dataspace	IN: dataspace with a selection on it.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.11.14 H5Pset_rawdata_integrity_scope

Name: H5Pset_rawdata_integrity_scope

Signature:

herr_t `H5Pset_rawdata_integrity_scope(` *hid_t* `dxpl_id`, *uint32_t* `scope`)

Purpose:

Specify the scope of checksum generation and verification for raw data transfer.

Description:

`H5Pset_rawdata_integrity_scope` sets a property in the `dxpl_id` data transfer property list specifying the scope of checksum generation and verification for raw data transfer in the Exascale Fast Forward stack. The specified scope will be used in all data transfer operations that are called with `dxpl_id.` In particular, `H5Dwrite_ff`, `H5Mset`, `H5Dread_ff`, and `H5Mget`. Note that different scopes may be specified for each data transfer operations.

The `scope` is a bitflag parameter that specifies which stages of the raw data transfer process will have checksums generated and verified. The possible values are:

`H5_CHECKSUM_NONE`	No checksums are computed or verified at any part of the stack.
`H5_CHECKSUM_TRANSFER`	Raw data is verified after transfer through mercury (the function shipper).
`H5_CHECKSUM_IOD`	Checksums are computed for raw data, given to IOD when written, and verified when read.
`H5_CHECKSUM_MEMORY`	Raw data is verified when moved into memory.
`H5_CHECKSUM_ALL`	Raw data is checksummed and verified at all levels.

Multiple individual selections can be combined using the OR operator. For example:

```
status = H5_CHECKSUM_IOD || H5_CHECKSUM_MEMORY
```

Parameters:

hid_t `dxpl_id`	IN: Data transfer property list identifier
uint32_t `scope`	IN: Bitfield specifying the scope of checksums for raw data transfers

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 5.

Man Page Status:

To do: Update with list of functions that are affected by the property's setting. Improve descriptions of scope options, maybe adding an ASCII picture of data flow.

4.11.15 H5Pset_rcapl_version_request

Name: H5Pset_rcapl_version_request

Signature:

herr_t `H5Pset_rcapl_version_request(` *hid_t* `rcapl_id`, *H5RC_request_t* `acquire_req`)

Purpose:

Specify a version request modifier in a read context acquire property list.

Description:

`H5Pset_rcapl_version_request` sets a property in the `rcapl_id` read context acquire property list specifying `acquire_req` as a container version request modifier for a read handle acquired with a call to `H5RCacquire` using `rcapl_id.`

When this is set, the container version specified in the call to `H5RCacquire` will be interpreted based on the value of the property. The possible values and interpretations are:

`H5RC_EXACT`	Acquire a read handle for the exact container version specified. (Default)
`H5RC_PREV`	Acquire a read handle for the container version specified, or the highest previous version if the specified version is not available.
`H5RC_NEXT`	Acquire a read handle for the container version specified, or the lowest next version if the specified version is not available. With this setting, and `*container_version` set to 0, the handle will be acquired for the first (lowest) readable version of the container.
`H5RC_LAST`	Acquire a read handle for the last (highest) container version possible. With this setting, the input value pointed to by `container_version` is ignored.

`H5Pcreate(H5P_RC_ACQUIRE)` will create a read context acquire property list whose identifier can be passed to this function.

Parameters:

hid_t `rcapl_id`	IN: Read context acquire property list identifier
H5RC_request_t `acquire_req`	IN: Modifier for container version specified in H5RCacquire

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.11.16 H5Pset_ trspl_num_peers

Name: H5Pset_trspl_num_peers

Signature:

herr_t `H5Pset_trspl_num_peers(` *hid_t* `trspl_id,` *unsigned* `num_peers)`

Purpose:

Set the leader count property in a transaction start property list.

Description:

`H5Pset_trspl_version_request` sets a property in the `trspl_id` transaction start property list to `num_peers.` The property is the number of leaders who will call `H5TRstart` for a given transaction using `trspl_id.`

`H5Pcreate(H5P_TR_START)` will create a transaction start property list whose identifier can be passed to this function.

Parameters:

hid_t `trspl_id`	IN: Transaction start property list identifier
unsigned `num_peers`	IN: Number of leader processes that will call `H5TRstart` for a given transaction.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.11.17 H5Pset_view_elmt_scope

Name: H5Pset_view_elmt_scope

Signature:

herr_t `H5Pset_view_elmt_scope`(*hid_t* `vcpl_id`, *hid_t* `dataspace_id`)

Purpose:

Set a selection that constrains dataset element query results for view creation.

Description:

`H5Pset_view_elmt_scope` sets a property in the `vcpl_id` view creation property list to `dataspace_id.` The property is the selection that limits the data elements queries to just the selection region specified in the dataspace.

For example, if the selection in `dataspace_id` is a 2x3 region of elements, calling this routine and then passing the modified `vcpl_id` property list to the `H5Vcreate` routine will constrain the elements of the dataset(s) examined during the execution of the view's query to just that 2x3 region.

NOTE: Other properties that constrain query behavior, such as restricting the names of link and attributes or map keys examined could be added in the future.

`H5Pcreate(H5P_VIEW_CREATE)` will create a view creation property list whose identifier can be passed to this function.

Parameters:

hid_t `vcpl_id`	IN: View creation property list identifier
hid_t `dataspace_id`	IN: Dataspace containing selection that constrains the elements queried when creating a view.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.11.18 H5Pget_ xapl_read_context

Name: H5Pget_xapl_read_context

Signature:

herr_t `H5Pget_xapl_read_context(` *hid_t* `xapl_id`, *hid_t* `*rcxt_id`)

Purpose:

Get the current read context property from an index access property list.

Description:

`H5Pget_xapl_read_context` gets the read context property from a `xapl_id` index access property list and stores it in `*rcxt_id.` The property is the current read context, when an index plugin callback is invoked from within the library. This property is set internally to the HDF5 library and is a read-only value for the index plugin.

The read context ID retrieved via this call must be closed with `H5RCclose`.

`H5Pcreate(H5P_INDEX_ACCESS)` will create a index access property list whose identifier can be passed to this function.

Parameters:

hid_t `xapl_id`	IN: Index access property list identifier
hid_t * `rcxt_id`	OUT: An ID for the current read context.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.11.19 H5Pget_ xapl_transaction

Name: H5Pget_xapl_transaction

Signature:

herr_t `H5Pget_xapl_transaction(` *hid_t* `xapl_id`, *hid_t* `*trans_id`)

Purpose:

Get the current transaction property from an index access property list.

Description:

`H5Pget_xapl_transaction` gets the transaction property from a `xapl_id` index access property list and stores it in `*trans_id.` The property is the current transaction, when an index plugin callback is invoked from within the library. This property is set internally to the HDF5 library and is a read-only value for the index plugin.

The transaction ID retrieved via this call must be closed with `H5TRclose`.

`H5Pcreate(H5P_INDEX_ACCESS)` will create a index access property list whose identifier can be passed to this function.

Parameters:

hid_t `xapl_id`	IN: Index access property list identifier
hid_t * `trans_id`	OUT: An ID for the current transaction.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.11.20 H5Pget_ xxpl_read_context

Name: H5Pget_xxpl_read_context

Signature:

herr_t `H5Pget_xxpl_read_context(` *hid_t* `xxpl_id`, *hid_t* `*rcxt_id`)

Purpose:

Get the current read context property from an index transfer property list.

Description:

`H5Pget_xxpl_read_context` gets the read context property from a `xxpl_id` index transfer property list and stores it in `*rcxt_id.` The property is the current read context, when an index plugin callback is invoked from within the library. This property is set internally to the HDF5 library and is a read-only value for the index plugin.

The read context ID retrieved via this call must be closed with `H5RCclose`.

`H5Pcreate(H5P_INDEX_TRANSFER)` will create a index transfer property list whose identifier can be passed to this function.

Parameters:

hid_t `xxpl_id`	IN: Index transfer property list identifier
hid_t * `rcxt_id`	OUT: An ID for the current read context.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.11.21 H5Pget_ xxpl_transaction

Name: H5Pget_xxpl_transaction

Signature:

herr_t `H5Pget_xxpl_transaction`(*hid_t* `xxpl_id`, *hid_t* *`trans_id`)

Purpose:

Get the current transaction property from an index transfer property list.

Description:

`H5Pget_xxpl_transaction` gets the transaction property from a `xxpl_id` index transfer property list and stores it in *`trans_id.` The property is the current transaction, when an index plugin callback is invoked from within the library. This property is set internally to the HDF5 library and is a read-only value for the index plugin.

The transaction ID retrieved via this call must be closed with `H5TRclose`.

`H5Pcreate(H5P_INDEX_TRANSFER)` will create a index transfer property list whose identifier can be passed to this function.

Parameters:

hid_t `xxpl_id`	IN: Index transfer property list identifier
hid_t * `trans_id`	OUT: An ID for the current transaction.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.12 H5Q: Query APIs

These routines are used to specify a query for generating views on HDF5 containers, or for analysis shipping operations on HDF5 Files (containers) in the Exascale Fast Forward stack.

Query objects specify constraints on HDF5 objects that restrict generating a view or an analysis shipping operation to certain objects or portions of objects in an HDF5 File.

The actions performed by all these functions are local to the compute node so asynchronous execution options are not provided.

These routines did not exist prior to the Exascale FastForward project.

Routine	**Implemented**	**Notes**
H5Qcreate	Quarter 6	
H5Qcombine	Quarter 6	
H5Qget_type	Quarter 8	
H5Qget_components	Quarter 8	
H5Qget_combine_op	Quarter 8	
H5Qget_match_op	Quarter 8	
H5Qencode	Quarter 8	
H5Qdecode	Quarter 8	
H5Qclose	Quarter 6	

4.12.1 H5Qclose

Name: H5Qclose

Signature:

herr_t `H5Qclose` (*hid_t* `query_id`)

Purpose:

Release the resources for a query object.

Description:

`H5Qclose` closes the query specified by `query_id` and releases resources used by it. Further use of the query identifier is illegal in HDF5 API calls.

Parameters:

hid_t `query_id`	IN: Identifier of the query to close.

Returns:

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

History:

Added in Quarter 6.

Man Page Status:

No known issues.

4.12.2 H5Qcombine

Name: H5Qcombine

Signature:

hid_t `H5Qcombine` (*hid_t* `query_id_1`, *H5Q_combine_op_t* `combine_op`, *hid_t* `query_id_2`)

Purpose:

Create a new query object by combining two existing query objects.

Description:

`H5Qcombine` creates a new query object by combining the conditions specified for two existing query objects. Query objects created with this routine can be used for view creation (with `H5Vcreate_ff`), analysis shipping operations (with `H5ASexecute`) or combined again into more complicated queries (with `H5Qcombine`).

The `query_id_1 and query_id_2` parameters indicate the two query objects to be combined with the `combine_op` parameter.

The `combine_op` parameter indicates how the two query objects are combined. The possible values and meanings are:

`H5Q_COMBINE_AND`	Both of the conditions specified in the `query_id_1 and query_id_2` parameters must be fulfilled for the resulting query.
`H5Q_COMBINE_OR`	Either, or both, of the conditions specified in the `query_id_1 and query_id_2` parameters must be fulfilled for the resulting query.

For example, to create a query that specifies matching on dataset elements that are greater than 53.4 and less than 93.6, the following example code could be used:

```
float f1 = 53.4;

hid_t query_id_1 = H5Qcreate(H5Q_TYPE_DATA_ELEM,

      H5Q_MATCH_GREATER_THAN, H5T_NATIVE_FLOAT, &f1);

float f2 = 93.6;

hid_t query_id_2 = H5Qcreate(H5Q_TYPE_DATA_ELEM,

      H5Q_MATCH_LESS_THAN, H5T_NATIVE_FLOAT, &f2);

hid_t query_id_3 = H5Qcombine(query_id_1, H5Q_COMBINE_AND, query_id_2);
```

The following table describes the result types for atomic queries and combining queries of different types:

Query	Result Type
H5Q_TYPE_DATA_ELEM	Dataset Element
H5Q_TYPE_ATTR_VALUE	Attribute
H5Q_TYPE_ATTR_NAME	Object
H5Q_TYPE_LINK_NAME	Object
<Dataset Element> AND <Dataset Element>	Dataset Element

<Dataset Element> AND <Attribute>	<none>
<Dataset Element> AND <Object>	Dataset Element
<Attribute> AND <Attribute>	Attribute
<Attribute>AND <Object>	Attribute
<Object> AND <Object>	Object
<Dataset Element> OR <Dataset Element>	Dataset Element
<Dataset Element> OR <Attribute>	Combination
<Dataset Element> OR <Object>	Combination
<Dataset Element> OR <Combination>	Combination
<Attribute> OR <Attribute>	Attribute
<Attribute> OR <Object>	Combination
<Attribute> OR <Combination>	Combination
<Object> OR <Object>	Object
<Object> OR <Combination>	Combination
<Combination> OR <Combination>	Combination
<Combination> AND <Dataset Element>	<none>
<Combination> AND <Attribute>	<none>
<Combination> AND <Object>	<none>
<Combination> AND <Combination>	<none>

Query results of <none> type are rejected when `H5Qcombine` is called, causing it to return failure.

NOTE: Query results of <none> type may be implemented with another result type in the future, once experience with the query framework is acquired and a meaningful grammar for those results are defined.

Parameters:

hid_t `query_id_1`	IN: Specifies one of the queries to be combined.
H5Q_combine_op_t `combine_op`	IN: Specifies the how the two queries are to be combined.
hid_t `query_id_2`	IN: Specifies one of the queries to be combined.

Returns:

Returns a query identifier if successful; otherwise returns a negative value. Query identifiers returned from this routine must be released with the H5Qclose routine.

History:

Added in Quarter 6.

Man Page Status:

No known issues.

4.12.3 H5Qcreate

Name: H5Qcreate

Signature:

hid_t `H5Qcreate` (*H5Q_type_t* `query_type`, *H5Q_match_op_t* `match_op`, …)

Purpose:

Create a new query object, specifying a type of object to apply the query to and the condition for matching.

Description:

`H5Qcreate` creates a new query object, and specifies the type of object in the HDF5 file that the query applies to, along with the condition of matching and the match value. Query objects created with this routine can be used for creating views (with `H5Vcreate_ff`), analysis shipping operations (with `H5ASexecute`) or combined into more complicated queries (with `H5Qcombine`).

The `query_type` parameter indicates the type of HDF5 object that the query will apply to. This parameter will determine the interpretation of the varargs parameters. The possible values and meanings are:

`H5Q_TYPE_DATA_ELEM`	Query will be applied to dataset elements.
`H5Q_TYPE_ATTR_VALUE`	Query will be applied to attribute values.
`H5Q_TYPE_ATTR_NAME`	Query will be applied to attribute names.
`H5Q_TYPE_LINK_NAME`	Query will be applied to link names.

The `match_op` parameter indicates the conditions for query matching. The possible values and meanings are:

`H5Q_MATCH_EQUAL`	Match values for the HDF5 object that are equal to the value specified for the query.
`H5Q_MATCH_NOT_EQUAL`	Match values for the HDF5 object that are not equal to the value specified for the query.
`H5Q_MATCH_LESS_THAN`	Match values for the HDF5 object that are less than the value specified for the query.
`H5Q_MATCH_GREATER_THAN`	Match values for the HDF5 object that are greater than the value specified for the query.

The values for the varargs parameters (…) are interpreted according to the query_type parameter. The possible types and meanings for each value of query_type are:

`H5Q_TYPE_DATA_ELEM`	*hid_t* `value_datatype_id`, *const void* * `value`	`value_datatype_id` specifies the datatype of the query value and `value` is a pointer to the query value itself.
`H5Q_TYPE_ATTR_VALUE`	*hid_t* `value_datatype_id`, *const void* * `value`	`value_datatype_id` specifies the datatype of the query value and `value` is a pointer to the query value itself.
`H5Q_TYPE_ATTR_NAME`	*const char* * `attr_name`	`attr_name` is a pointer to the query value.

`H5Q_TYPE_LINK_NAME`	*const char* * `link_name`	`link_name` is a pointer to the query value.

For example, to create a query that specifies matching on dataset elements that are greater than 53.4, the following example code could be used:

```
float f = 53.4;
hid_t query_id = H5Qcreate(H5Q_TYPE_DATA_ELEM, H5Q_MATCH_GREATER_THAN,
      H5T_NATIVE_FLOAT, &f);
```

The following query shows how to select objects with link names equal to "Pressure":

```
hid_t q2 = H5Qcreate(H5Q_TYPE_LINK_NAME, H5Q_MATCH_EQUAL, "Pressure");
```

Many more query types are possible, including types that select on keys or values of a map object, types that match objects based on their metadata (such as datasets with an integer datatype or with three dimensions) or types that restrict the query to a region of a dataset, but the types above represent a starting point and more can always be added over time. The same could be said for the match conditions, with additions of regular expressions for attribute or link names, etc. possible in the future.

Query objects created by this routine must be closed by H5Qclose.

Parameters:

H5Q_type_t `query_type`	IN: Specifies the type of HDF5 object that the query will apply to. This parameter will determine interpretation of the varargs parameters.
H5Q_match_op_t `match_op`	IN: Specifies the conditions for matching a query value.
`...`	IN: The varargs parameters are interpreted according to the query type.

Returns:

Returns a query identifier if successful; otherwise returns a negative value. Query identifiers returned from this routine must be released with the H5Qclose routine.

History:

Added in Quarter 6.
Quarter 8: Added support for attribute values and names, and link names.

Man Page Status:

No known issues.

4.12.4 H5Qdecode

Name: H5Qdecode

Signature:

hid_t `H5Qdecode` (*const void* `*buf`)

Purpose:

Deserialize a buffer containing a serialized query and return a new query handle.

Description:

`H5Qdecode` deserializes the buffer, `buf`, and return a new query handle. The handle must later be closed using `H5Qclose`.

Parameters:

const void `*buf`	IN: Buffer to be deserialized.

Returns:

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

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.12.5 H5Qencode

Name: H5Qencode

Signature:

herr_t `H5Qencode` (*hid_t* `query_id,` *void* `*buf,` *size_t* `*nalloc`)

Purpose:

Serialize a query into a buffer.

Description:

`H5Qencode` serializes a query, given by `query_id`, into a used-provided buffer, `buf`, and sets the `*nalloc` parameter to the number bytes used to encode the query. If `buf` is NULL, then the query is not encoded, but the number of bytes required to encode the query is still set. The query is unaffected by the serialization operation.

Parameters:

hid_t `query_id`	IN: Identifier for query that is to be serialized.
void `*buf`	OUT: Buffer where serialized query is written.
size_t `*nalloc`	OUT: Number of bytes required to serialize the query.

Returns:

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

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.12.6 H5Qget_combine_op

Name: H5Qget_combine_op

Signature:

herr_t `H5Qget_combine_op(` *hid_t* `query_id`, *H5Q_combine_op_t* `*op_type )`

Purpose:

`Get` the operator type for a query

Description:

`H5Qget_combine_op` retrieves the operator type for a query, given by `query_id`, setting the `*op_type` value with the type. Possible values returned are:

- `H5Q_COMBINE_AND`
- `H5Q_COMBINE_OR`
- `H5Q_SINGLETON`

`H5Q_COMBINE_AND` and `H5Q_COMBINE_OR` are only returned for query objects produced with `H5Qcombine` and `H5Q_SINGLETON` is returned for query objects produced with `H5Qcreate`.

Parameters:

hid_t `query_id`	IN: Specifies the query to probe
H5Q_combine_op_t `*op_type`	OUT: The operator type for the query

Returns:

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

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.12.7 H5Qget_components

Name: H5Qget_components

Signature:

herr_t `H5Qget_components(` *hid_t* `query_id`, *hid_t* `*sub_query1_id`, *hid_t* `*sub_query2_id )`

Purpose:

Get the component queries of a combined query

Description:

`H5Qget_components` retrieves the component queries from a compound query object, given by query_id. The component queries are returned in `*sub_query1_id`, and `*sub_query2_id`, both of which must be closed with `H5Qclose`. It is an error to apply `H5Qget_components` to a singleton query (one which was created with `H5Qcreate`).

Note that the component queries may be compound or singleton, depending on how the initial query was constructed.

Parameters:

hid_t `query_id`	IN: Specifies the query to probe
hid_t `*sub_query1_id`	OUT: Query ID for one of the components of the probed query
hid_t `*sub_query2_id`	OUT: Query ID for the other component of the probed query

Returns:

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

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.12.8 H5Qget_match_op

Name: H5Qget_match_op

Signature:

herr_t `H5Qget_match_op(` *hid_t* `query_id`, *H5Q_match_op_t* `*match_op`)

Purpose:

Retrieve the match operation for a singleton query

Description:

`H5Qget_match_op` retrieves the match operation for a query, given by `query_id`, into the `*match_op` parameter. The match operation was originally given to the `H5Qcreate` call, in its `match_op` parameter. See `H5Qcreate` for a table listing the complete set of values that may be returned for `*match_op`. It is an error to perform this call on a compound query object (one which was created with `H5Qcombine`).

Parameters:

hid_t `query_type`	IN: Specifies the singleton query to probe
H5Q_match_op_t `*match_op`	OUT: The match operation for the singleton query

Returns:

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

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.12.9 H5Qget_type

Name: H5Qget_type

Signature:

herr_t `H5Qget_type(` *hid_t* `query_id`, *H5Q_type_t* `*query_type`)

Purpose:

Retrieve the type of a query

Description:

`H5Qget_type` retrieves the type for a query, given by `query_id`, originally provided to `H5Qcreate` in its `query_type` parameter, or set after a call to `H5Qcombine`, putting the query's type into `*query_type`. See `H5Qcreate` for a table listing the complete set of values that may be returned for the query type.

Parameters:

hid_t `query_id`	IN: Specifies the query to probe
H5Q_type_t `*query_type`	OUT: The query type for the probed query

Returns:

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

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.13 H5RC: Read Context APIs

These routines are used to establish the read context for operations on HDF5 Files (containers) in the Exascale Fast Forward stack.

In the Exascale Fast Forward stack it is possible that multiple versions of a container may be readable at any given time, where a container version results when a transaction is committed, and earlier versions may be 'flattened' by later versions as the data is reorganized for optimized access.

The application must indicate what container version it wants to read from, not simply what container. A read context is used to open a handle on a particular version, guaranteeing that the version will remain available until the context is closed. A read context ID is passed to HDF5 operations to indicate what version of the container should be read from. All reads made using the same read context will see a consistent version of the container data. Multiple read contexts can be open on a single container at any given time.

These routines did not exist prior to the Exascale FastForward project.

Routine	**Implemented**	**Notes**
H5RCacquire	Quarter 5	
H5RCacquire_wait		Added in response to Q5 feedback; May be implemented in future quarter if time allows.
H5RCclose	Quarter 5	
H5RCcreate	Quarter 5	
H5RCget_version	Quarter 5	
H5RCpersist	Quarter 5	
H5RCrelease	Quarter 5	
H5RCsnapshot	Quarter 5	

4.13.1 H5RCacquire

Name: H5RCacquire

Signature:

hid_t `H5RCacquire` (*hid_t* `file_id`, *uint64_t* `*container_version`, *hid_t* `rcapl_id`, *hid_t* `es_id`)

Purpose:

Acquire a read handle for a container at a given version and create a read context associated with the container and version.

Description:

`H5RCacquire` acquires a read handle on a container version and creates a read context associated with that container version. The container version is guaranteed to remain readable and consistent until the context is closed and the handle released (with `H5RCrelease`).

The `file_id` parameter indicates the container the read context will be associated with.

The `container_version` parameter indicates the container version that the read handle will be acquired for and the read context associated with. Parameters in the read context acquire property list control how the value pointed to by `container_version` will be interpreted by the function, since it is not always possible to acquire a particular container version. Upon successful completion of the function, the value pointed to by the `container_version` parameter contains the actual version that the read handle was acquired for and the read context associated with.

The version request property in the read context acquire property list, `rcapl_id,` controls how the function interprets the value pointed to by `container_version. H5Pset_rcapl_version_request()` sets the version request property, whose possible values and meanings are:

`H5RC_EXACT`	Acquire a read handle for the exact container version specified. (Default)
`H5RC_PREV`	Acquire a read handle for the container version specified, or the highest previous version if the specified version is not available.
`H5RC_NEXT`	Acquire a read handle for the container version specified, or the lowest next version if the specified version is not available. With this setting, and `*container_version` set to 0, the handle will be acquired for the first (lowest) readable version of the container.
`H5RC_LAST`	Acquire a read handle for the last (highest) container version possible. With this setting, the input value pointed to by `container_version` is ignored.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter, but the returned read context should not be used until the asynchronous function completes.

This function must be called by one process in a parallel application to ensure that the container version remains readable. That process can notify other processes in the application when the read handle is acquired, and the other processes can each call `H5RCcreate` after notification is received to create local read context identifiers for the container version.

In the current prototype implementation, only one process can call `H5RCacquire` for a given container version. Coordination amoung processes wanting to read the same container version must be done at the application level and multiple processes cannot independently call `H5RCacquire` to indicate that they each are interested in reading from the container version.

Parameters:

hid_t `file_id`	IN: File identifier indicating the container that the read handle is being acquired for and read context associated with.
uint64_t `*container_version`	IN/OUT: On input, pointer to value that, in conjunction with the read context acquire property list, specifies the container version to open the handle for. Upon successful completion of the function, the value pointed to by this parameter will contain the actual container_version that the read handle was acquired. The value is unchanged if the function does not complete successfully.
hid_t `rcapl_id`	IN: Read context acquire property list. Can contain the version request property, set with `H5Pset_rcapl_version_request()`, that controls how `container_version` is interpreted.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a read context identifier if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the new read context is returned initially. Upon completion of the asynchronous operation, the future ID will be transparently modified to be a “normal” read context identifier. Since the success of the read handle acquisition, and the actual container version acquired, is not known until completion, the application should not use the future ID for the read context in any other functions as a read context ID parameter.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

Limitations and Open Issues:

As designed, `H5RCacquire` could be called by multiple processes independently. Although this usage is not optimal from a performance perspective, it could be useful (and even necessary) in cases where multiple reading processes are not tightly coupled. In the prototype phase of the project, IOD does not support the multi-caller use case, so a single process must call `H5RCacquire`, notify other processes who may want to read from the version at the same time, and have those processes call `H5RCcreate`.

History:

Added in Quarter 5.
Quarter 8: Added section on Limitations and Open Issues and revised text in Description accordingly.

Man Page Status:

No known issues.

4.13.2 H5RCclose

Name: H5RCclose

Signature:

herr_t `H5RCclose` (*hid_t* `rcntxt_id`)

Purpose:

Close a read context.

Description:

`H5RCclose` closes the read context specified by `rcntxt_id` and releases resources used by it. Further use of the read context identifier is illegal in HDF5 API calls. `H5RCclose` does not release the read handle for the container version associated with the read context that is being closed. Typically the read context specified by `rcntxt_id` was obtained by a call to `H5RCcreate`, but in some circumstances `H5RCacquire` may have returned the read context.

`H5RCclose` should be called before the handle for the container version is released via a call to `H5RCrelease`. Typically the process that called `H5RCacquire` will call `H5RCrelease`, but in some circumstances, such as premature process termination, another process with a read context for the container version obtained via a call to `H5RCcreate` may make the call to `H5RCrelease`. Regardless of which process calls `H5RCrelease`, it closes the read context and releases the read handle on the associated container version.

For a given `rcntxt_id`, either `H5RCclose` or `H5RCrelease` should be called, but not both.

The actions performed by this function are all local to the compute node so an asynchronous execution option is not provided.

Parameters:

hid_t `rcntxt_id`	IN: Identifier of the read context to close.

Returns:

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

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.13.3 H5RCcreate

Name: H5RCcreate

Signature:

hid_t `H5RCcreate` (*hid_t* `file_id`, *uint64_t* `container_version`)

Purpose:

Create a read context associated with a container and version.

Description:

`H5RCcreate` creates a read context associated with a specified container and version after a read handle has already been acquired on the container version by another process in the application. The created read context is passed to other HDF5 functions that perform reads on the container.

The `file_id` parameter indicates the container the read context is associated with.

The `container_version` parameter indicates the container version that the read context is associated with.

Prior to this call, a read handle must be acquired on the container version by another process in the parallel application via a call to `H5RCacquire`**.** The container version is guaranteed to remain readable and consistent until the handle released (with `H5RCrelease`).

To conserve and release resources, the read context created by this function should be closed, when access is no longer required, usually with `H5RCclose`. The read context acquired by this function cannot be used after the handle for the container version has been released via a call to `H5RCrelease`. Typically the process that called `H5RCacquire` will call `H5RCrelease,` but in some circumstances (such as premature process termination), another process with a read context for the container version may make the call to `H5RCrelease`, closing the read context and releasing the read handle.

The actions performed by this function are all local to the compute node so an asynchronous execution option is not provided.

Parameters:

hid_t `file_id`	IN: File identifier indicating the container that the read context is associated with.
uint64_t `container_version`	IN: The container version that the read context is associated with..

Returns:

Returns a read context identifier if successful; otherwise returns a negative value.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.13.4 H5RCget_version

Name: H5RCget_version

Signature:

herr_t `H5RCget_version` (*hid_t* `rcntxt_id`, *uint64_t* `*container_version`)

Purpose:

Retrieve the container version associated with a read context.

Description:

The `rcntxt_id` specifies the read context whose associated container version is to be retrieved.

The value pointed to by the `container_version` parameter will be set to the container version that the read context is associated with.

The actions performed by this function are all local to the compute node so an asynchronous execution option is not provided.

Parameters:

hid_t `rcntxt_id`	IN: Read context identifier whose associated container version is being retrieved.
uint64_t `*container_version`	OUT: Container version associated with the read context. Value is unchanged if operation is unsuccessful.

Returns:

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

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.13.5 H5RCpersist

Name: H5RCpersist

Signature:

herr_t `H5RCpersist` (*hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Copy data for a container from IOD to DAOS, bringing DAOS up to specified container version.

Description:

`H5RCpersist` requests that IOD update a container on DAOS to a specific version.

The `rcntxt_id` indicates both the container that is to be updated and the version that is to be persisted. The indicated container must be open for write.

Updates to the container between the last persisted version and the version currently being persisted will be copied to DAOS as part of this persist request.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `rcntxt_id`	IN: Read context identifier indicating the container (that is open for write) and the version that is to be persisted.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Quarter 4: Documented as H5Fpersist but not yet implemented.
Quarter 5: Renamed from H5Fpersist to H5RCpersist, using rcntxt_id instead of file_id and container_version. Implemented.

Man Page Status:

No known issues.

4.13.6 H5RCrelease

Name: H5RCrelease

Signature:

herr_t `H5RCrelease` (*hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Close a specified read context and release a read handle for the associated container version

Description:

`H5RCrelease` closes the read context specified by `rcntxt_id` and releases resources used by it. It also releases the read handle for the container version associated with the read context. Further use of the read context identifier is illegal in HDF5 API calls. Typically the read context specified by `rcntxt_id` was obtained by a call to `H5RCacquire, H5Fopen_ff, or H5TRfinish`, but in some circumstances `H5RCcreate` may have returned the read context**.**

`H5RCrelease` should be called after all other read contexts for the container version have been released via calls to `H5RCclose`. Typically the process that called `H5RCacquire, H5Fopen_FF,` or `H5TRfinish` will call `H5RCrelease,` but in some circumstances, such as premature process termination, another process with a read context for the container version obtained via a call to `H5RCcreate` may make the call to `H5RCrelease.` Regardless of which process calls `H5RCrelease`, the read context is closed and the read handle on the associated container version is released.

For a given `rcntxt_id`, either `H5RCrelease` or `H5RCclose` should be called, but not both.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `rcntxt_id`	IN: Identifier of the read context that will be closed; the read handle for the associated container version will also be released.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.13.7 H5RCsnapshot

Name: H5RCsnapshot

Signature:

herr_t `H5RCsnapshot(` *hid_t* `rcntxt_id`, *const char** `snapshot_name`, *hid_t* `es_id )`

Purpose:

Make a snapshot of a container on DAOS.

Description:

`H5RCsnapshot` requests that DAOS make a copy of the container and version specified by the read context, and give it the indicated container name. The snapshot is a permanently visible copy of the container's state at the given container version.

The `rcntxt_id` indicates both the container and the container version that the new copy will duplicate. If the requested version is not readable on DAOS for the container, the snapshot request will fail.

`snapshot_name` is the name given to the newly created snapshot.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `rcntxt_id`	IN: Read context identifier indicating the container and the version that is to be persisted for with the snapshot should be taken.
*const char** `snapshot_name`	IN: Name of the snapshot.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Quarter 4: Documented but not yet implemented.
Quarter 5: Renamed from H5Fsnapshot to H5RCsnapshot, using rcntxt_id instead of file_id and container_version. Implemented.

Open Issues:

The IOD API to support this functionality is missing in the prototype, so this feature cannot be accessed from the HDF5 API. The ability to snapshot containers is implemented at the DAOS level.

Man Page Status:

No known issues.

4.14 H5T: Datatype APIs

These routines are used to operate on HDF5 Datatype Objects.

The routines ending in _ff have different signatures than the standard HDF5 library routines.

Man pages for routines whose user interface is unchanged from the standard HDF5 implementation can be found at: http://www.hdfgroup.org/HDF5/doc/RM/RM_H5T.html.

Routine	**Implemented**	**Notes**
H5Tclose_ff	Quarter 4	
H5Tcommit_ff	Quarter 4	
H5Tevict_ff	Quarter 7	
H5Topen_ff	Quarter 4	This is implemented synchronously for now.
H5Tprefetch_ff	Quarter 7	
H5Tget_create_plist	Quarter 4	Local operation, no need to do asynchronously. See standard HDF5 man page.
H5Tcommit_anon		Will not implement

4.14.1 H5Tclose_ff

Name: H5Tclose_ff

Signature:

hid_t `H5Tclose_ff(` *hid_t* `dtype_id`, *hid_t* `es_id )`

Purpose:

Releases a datatype, possibly asynchronously.

Description:

`H5Tclose_ff` releases the datatype specified by `dtype_id`. Further access through the datatype identifier is illegal. Failure to release a datatype with this call will result in resource leaks.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `dtype_id`	IN: Identifier of the datatype to release.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from event queue to event stack.

Man Page Status:

No known issues.

4.14.2 H5Tcommit_ff

Name: H5Tcommit_ff

Signature:

herr_t `H5Tcommit_ff(` *hid_t* `loc_id`, *const char* *`name`, *hid_t* `dtype_id`, *hid_t* `lcpl_id`, *hid_t* `tcpl_id`, *hid_t* `tapl_id`, *hid_t* `trans_id`, *hid_t* `es_id`)

Purpose:

Commit a transient datatype, linking it into the file and creating a new named datatype, possibly asynchronously.

Description:

`H5Tcommit_ff` saves a transient datatype as an immutable named datatype in a file. The datatype specified by `dtype_id` is committed to the file with the name `name` at the location specified by `loc_id` and with the datatype creation and access property lists `tcpl_id` and `tapl_id`, respectively.

`loc_id` may be a file identifier, or a group identifier within that file. `loc_id` must be in scope for the transaction identified by `trans_id`.

`name` may be either an absolute path in the file or a relative path from `loc_id` naming the newly-commited datatype. `name` must be in scope for the transaction identified by `trans_id`.

The link creation property list, `lcpl_id`, governs creation of the link(s) by which the new named datatype is accessed and the creation of any intermediate groups that may be missing. The link creation property list currently has no effect in the EFF stack and should be set to `H5P_DEFAULT`.

The datatype creation property list, `tcpl_id`, and dataset access property list, `tapl_id`, are currently not used and should be set to `H5P_DEFAULT`.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Once commited, this datatype may be used to define the datatype of any other dataset or attribute in the file.

This function will not accept a datatype that cannot actually hold data. This currently includes compound datatypes with no fields and enumerated datatypes with no members.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 group identifier or file identifier file identifier that is in scope for the transaction.
const char *`name`	IN: Name given to named datatype The name can be specified relative to `loc_id`, or absolute from the file's root group, and must be in scope for the transaction.
hid_t `dtype_id`	IN: Identifier of datatype to be committed and, upon function's return, identifier for the named datatype
hid_t `lcpl_id`	IN: Link creation property list

Currently not used in EFF; specify `H5P_DEFAULT`.

hid_t `tcpl_id`	IN: Datatype creation property list *Currently not used; specify `H5P_DEFAULT`.*
hid_t `tapl_id`	IN: Datatype access property list *Currently not used; specify `H5P_DEFAULT`.*
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 4.
Quarter 5: Changed from transaction to transaction id and from event queue to event stack. Added scope requirement and noted property lists that are not used.

Man Page Status:

No known issues.

4.14.3 H5Tevict_ff

Name: H5Tevict_ff

Signature:

herr_t `H5Tevict_ff(` *hid_t* `dtype_id,` *uint64_t* `container_version,` *hid_t* `dxpl_id,` *hid_t* `es_id )`

Purpose:

Evict named datatype from the burst buffer, possibly asynchronously.

Description:

`H5Tevict_ff` evicts data associated with a named datatype from the burst buffer.

The data to be evicted may be resident in the burst buffer under two different scenarios

In the first scenario, the data is resident in the burst buffer as the result of updates to the datatype made via the EFF transaction model. For example, through a call to H5Tcommit_ff. This calls add updates to a transaction that are atomically applied when the transaction is committed, and we refer to this data as *transaction update data*.

When evicting transaction update data, the container version being evicted should first be persisted to permanent storage (DAOS), with the `H5RCpersist` command. The named datatype's transaction update data for the specified container version, as well as the named datatype's transaction update data for all lower-numbered container versions that has not yet been evicted from the burst buffer, will be evicted as the result of this call. If evicting the data would result in container versions with open read contexts becoming inaccessible, the evict will fail.

In the second scenario, the data is resident in the burst buffer as the result of a call to H5Tprefetch_ff. This call replicates data from persistent storage (DAOS) to the burst buffer, and we refer to this data as *replica data*. When replica data is evicted, only data in the burst buffer as a result of the exact replica specified is evicted – transaction update data and other replicas for the named datatype remain in the burst buffer.

`The dtype_id` parameter specifies the named datatype whose data is to be evicted.

The version of the named datatype to be evicted is specified by the `container_version` property.

The replica property in the transfer property list, `dxpl_id,` is used to specify that a named datatype replica is to be evicted. `H5Pset_dxpl_replica()` sets the replica property. If this is set, then replia data will be evicted, otherwise transaction update data will be evicted.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Limitations / Future Considerations:

In Quarter 7, only the primary IOD Blob object associated with the named datatype is evicted. Auxilary IOD objects will also be evicted in Quarter 8.

When evicting a replica, the `container_version` argument is redundant, as the replica identifier fully specifies the data to be evicted. Consider revisiting and possibly revising the API prior to production release. Possibly have separate evict commands for eviction of transaction update data and of replicas.

For other potential extensions that are beyond the scope of the EFF prototype project, refer to the document *Burst Buffer Space Management – Prototype to Production*.

Parameters:

hid_t `dtype_id`	IN: Identifier of the named datatype being evicted.
uint64_t `container_version`	IN: Container version specifying what version of the named datatype to evict.
hid_t `tapl_id`	IN: Identifier of an access property list. If the access property list contains an evict replica property (set via `H5Pset_evict_replica()`), then the replica_id specified by that property will be evicted.
hid_t `es_id`	IN: The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 7.

Man Page Status:

Will need updates in Quarter 8 when additional features are implemented.

4.14.4 H5Topen_ff

Name: H5Topen_ff

Signature:

hid_t `H5Topen_ff(` *hid_t* `loc_id`, *const char ** `name`, *hid_t* `tapl_id,` *hid_t* `rcntxt_id,` *hid_t* `es_id )`

Purpose:

Opens a named datatype, possibly asynchronously.

Description:

`H5Topen_ff` opens a named datatype at the location specified by `loc_id` and returns an identifier for the datatype. `loc_id` is either a file or group identifier. `name` is the path to the named datatype object relative to `loc_id`. Both `loc_id and name` must be in scope for for the read context identified by `rcntxt_id`.

The identifier should eventually be closed by calling `H5Tclose_ff` to release resources.

The named datatype is opened with the datatype access property list `tapl_id`. The datatype access property list currently has no effect and should be set to `H5P_DEFAULT`.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter. *NOTE: Currently, only synchronous execution is supported. We will continue to evaluate whether asynchronous execution makes sense for this function, as the user needs the information returned in order to proceed.*

Parameters:

hid_t `loc_id`	IN: A file or group identifier that is in scope for the read context.
*const char ** `name`	IN: A datatype name, defined within the file or group identified by `loc_id`. Must be in scope for the read context.
hid_t `tapl_id`	IN: Datatype access property list identifier. *Currently not used; specify `H5P_DEFAULT.`*
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution. *Currently will be treated as H5_EVENT_STACK_NULL regardless of what value is passed in.*

Returns:

Returns a named datatype identifier if successful; otherwise returns a negative value.

History:

Added in Quarter 4.

4.14.5 H5Tprefetch_ff

Name: H5Tprefetch_FF

Signature:

herr_t `H5Tprefetch_ff(` *hid_t* `dtype_id`, *hid_t* `rcntxt_id`, *hid_t* `*replica_id`, *hid_t* `dxpl_id`, *hid_t* `es_id )`

Purpose:

Prefetch a named datatype from persistent storage to burst buffer storage, possibly asynchronously.

Description:

`H5Tprefetch_ff` prefetches a named datatype, specified by its identifier `dtype_id`, from persistent storage (DAOS) into burst buffer storage.

`rcntxt_id` indicates the read context for this operation.

`replica_id,` the replica identifier, is set to indicate where the pre-fetched data can be found in the burst buffer, and is passed to subsequent `H5Tevict` calls.

`dxpl_id,` a data transfer property list identifier, is not currently used and should be set to H5P_DEFAULT.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The `replica_id` is not valid until the operation has completed, if it is executing asynchronously.

Limitations / Future Considerations:

For the EFF prototype project, only the primay IOD Blob object associated with the HDF5 named datatype will be prefetched; auxiliary IOD objects remain on persistent storage (DAOS). For more information, and other potential extensions, refer to the document *Burst Buffer Space Management – Prototype to Production*.

This function was implemented for completeness, and to demonstrate that IOD Blob objects can be prefetched, but since no access routines accept the named datatype's `replica_id` in this phase of the project, it has no practical value other than to demonstrate the ability to prefetch and evict HDF5 named datatypes and IOD Blobs.

Parameters:

hid_t `dtype_id`	IN: Identifier of the named datatype being prefetched. `grp_id` must be in scope for the read context.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `* replica_id`	IN: Identifier of the replicated data in the burst buffer.
hid_t `tapl_id`	IN: Identifier of an access property list for this I/O operation. Should be H5P_DEFAULT.
hid_t `es_id`	IN: The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Returns:

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

Note that when this routine is executed asynchronously, the return value from the routine only indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 7.

Man Page Status:

No known issues.

4.15 H5TR: Transaction Operation APIs

These routines are used to create and manage HDF5 transactions in the Exascale Fast Forward stack.

An HDF5 transaction consists of a set of updates that modify an HDF5 file (container). Transactions are applied atomically - either all of the transaction’s updates are applied and become readable (the transaction is committed) or none of them are applied (the transaction is discarded). Transactions are numbered and are committed in strict numerical sequence. Multiple transactions may be in progress concurrently, and multiple processes may participate in a single transaction.

A transaction is associated with a particular container and transaction number. The transaction ID returned when a transaction is created is used to start the transaction, and is passed to HDF5 functions that update the container. This allows every update to be added to the correct transaction, and all of the updates for a transaction to be applied atomically when the transaction is committed.

These routines did not exist prior to the Exascale FastForward project.

Routine	**Implemented**	**Notes**
H5TRabort	Quarter 5	
H5TRclose	Quarter 5	
H5TRcreate	Quarter 5	
H5TRfinish	Quarter 5	
H5TRget_trans_num	Quarter 7	
H5TRget_version	Quarter 7	
H5TRset_dependency	Quarter 5	
H5TRskip	Quarter 5	
H5TRstart	Quarter 5	

4.15.1 H5TRabort

Name: H5TRabort

Signature:

herr_t `H5TRabort(` *hid_t* `trans_id,` *hid_t* `es_id )`

Purpose:

Abort a transaction.

Description:

`H5TRabort` signals that all updates in the transaction identified by `trans_id` should be discarded.

`H5TRabort` can be called by any process that has a transaction identifier for the transaction to be aborted. All past and future updates in the transaction identified by `trans_id` will be discarded. The aborted transaction will never be committed to the container and the transaction number associated with the aborted transaction cannot be reused. An aborted transaction is classified as *discarded*. Discarded transactions do not block the commitment of higher-numbered transactions.

If a higher-numbered transaction registered a dependency on the aborted transaction, the dependent transaction will also be aborted. The abort of the dependent transaction may occur at any time before it commits.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `trans_id`	IN: Transaction identifier for the transaction that is being aborted.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.15.2 H5TRclose

Name: H5TRclose

Signature:

herr_t `H5TRclose` (*hid_t* `trans_id`)

Purpose:

Close the specified transaction.

Description:

`H5TRclose` closes the transaction specified by `trans_id` and releases resources used by it. Further use of the transaction identifier is illegal in HDF5 API calls. This function does not finish the transaction and allow it to be committed – that is done by `H5TRfinish.`

All processes that called `H5TRcreate` to create a transaction and get a transaction id must call `H5TRclose` to release the resources used by the transaction. For a transaction that is to be committed, the function `H5TRfinish` must be called `L` times, where `L` is the number of leader processes that called H5TRstart to start the transaction. Typically the leader processes each call `H5TRfinish`, but in some circumstances – such as if a process dies – other processes may call `H5TRfinish.` Any process with a transaction id for a given transaction may abort the transaction with a call to `H5TRabort`. On a given process for a given transaction, calls to `H5TRfinish` or `H5TRabort` should be made – and completed – before `H5TRclose` is called.

The actions performed by this function are all local to the compute node so an asynchronous execution option is not provided.

Parameters:

hid_t `trans_id`	IN: Identifier of the transaction to close.

Returns:

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

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.15.3 H5TRcreate

Name: H5TRcreate

Signature:

hid_t `H5TRcreate` (*hid_t* `file_id,` *hid_t* `rcntxt_id` , *uint64_t* `trans_num`)

Purpose:

Create a transaction associated with a specified container, read context, and number.

Description:

`H5TRcreate` creates a transaction associated with a specified container, read context, and transaction number. Every process that will add updates to the transaction must call `H5TRcreate` and use the transaction identifier returned by `H5TRcreate` in subsequent calls related to the transaction.

After a transaction has been created, one or more transaction leaders must start the transaction by calling `H5TRstart`, effectively notifying lower layers of the I/O stack that the transaction is in-progress. After the leader processes start the transaction, it is their responsibility to notify the processes that called `H5TRcreate` but did not call `H5TRstart` (the delegate processes) that updates to the transaction may begin.

The `file_id` parameter indicates the container the transaction is associated with, and must be open for write. Updates added to the transaction will be applied atomically to the container identified by `file_id` when the transaction is committed.

The `rcntxt_id` parameter indicates read context for the transaction. Any read operations that occur in conjunction with updates to the transaction will be made on the container version associated with this read context.

The `trans_num` parameter indicates the number that is associated with the transaction. A single container can have multiple transactions in progress at any given time. Transactions are committed in strict numerical order, and it it the application's responsibility to manage transaction numbers. Transaction numbers uniquely identify a transaction and once a transaction is started or skipped the number can never be used again.

Note that although reads will be made from the container version associated with the read context, the transaction's updates will be atomically applied to the latest container version at the time the transaction is committed.

The actions performed by this function are all local to the compute node so an asynchronous execution option is not provided.

Parameters:

hid_t `file_id`	IN: File identifier indicating the container that the transaction is associated with.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for the transaction.
uint64_t `trans_num`	IN: The transaction number for the transaction.

Returns:

Returns a transaction identifier if successful; otherwise returns a negative value.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.15.4 H5TRfinish

Name: H5TRfinish

Signature:

herr_t `H5TRfinish(` *hid_t* `trans_id,` *hid_t* `trfpl_id,` *hid_t* `*rcntxt_id,` *hid_t* `es_id )`

Purpose:

Finish a transaction that was started with `H5TRstart`.

Description:

`H5TRfinish` signals that no more updates will be added to the transaction identified by `trans_id.` If updates to the transaction have been made with asynchronous function calls, then all of the operations must complete before the last call to `H5TRfinish` is made for the transaction.

For a given transaction, `H5TRfinish` must be called the same number of times that `H5TRstart` was called to start the transaction (assuming that the transaction was not aborted with `H5TRabort).` Typically, the leader processes that called `H5TRstart` to start the transaction each call `H5TRfinish`; in some circumstances – such as when a process dies – other processes may call `H5TRfinish.`

The transaction is considered finished when `H5TRfinish` has been called the required number of times.

After the transaction is finished and all lower-numbered transactions are finished or discarded, the transaction is *committed* and all updates that are part of the transaction will be applied atomically to the container and become readable in a new container version with the same number as the committed transaction.

The transaction finish property list, `trfpl_id`, is currently not used and should be set to `H5P_DEFAULT.`

The `rcntxt_id` parameter allows the user to pass a pointer to an *hid_t* datatype signaling that the function should acquire a read handle and create a read context immediately after the transaction is committed. This ensures that a read handle on the container version resulting from the committed transaction can be acquired before a subsequent transaction commit occurs.

In addition to acquiring the read handle, a read context is created and the identifier for that context is returned in `*rcntxt_id.` The read context identifier will not be available until after the `H5TRfinish` call completes successfully, but a future id is returned immediately and can be used in subsequent HDF5 calls. `H5RCrelease` must be called to release the read handle and close the context when it is no longer needed.

If the `rcntxt_id` parameter is set to `NULL`, no read handle is acquired and no read context is created.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

When executed asynchronously, the function does not complete until after the finished transaction has been committed. If `rcntxt_id` is not `NULL,` acquisition of the read handle, creation of the read context, and updating of `*rcntxt_id` will also occur before the function completes.

After `H5TRfinish` completes, the transaction should be closed by all processes that called `H5TRcreate.`

Parameters:

hid_t `trans_id`	IN: Transaction identifier for the transaction being finished.
hid_t `trfpl_id`	IN: Transaction finish property list.

Currently not used; specify `H5P_DEFAULT`.

hid_t `*rcntxt_id`	IN/OUT: Pointer to read context for container version resulting from successfully committed transaction. Pass in `NULL` if no read context is desired.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the read context is returned initially in `*rcntxt_id`; upon completion of the asynchronous operation, the future ID will be transparently modified to be a “normal” read context identifier.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.15.5 H5TRget_trans_num

Name: H5TRget_trans_num

Signature:

herr_t `H5TRget_trans_num` (*hid_t* `trans_id`, *uint64_t* `*trans_num`)

Purpose:

Retrieve the transaction number associated with a transaction.

Description:

The `trans_id` specifies the transaction whose associated transaction number is to be retrieved.

The value pointed to by the `trans_num` parameter will be set to the transaction number that the transaction is associated with.

The actions performed by this function are all local to the compute node so an asynchronous execution option is not provided.

Parameters:

hid_t `trans_id`	IN: Identifier of the transaction whose associated transaction number is being retrieved.
uint64_t `*trans_num`	OUT: Transaction number associated with the transaction. Value is unchanged if operation is unsuccessful.

Returns:

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

History:

Added in Quarter 7.

Man Page Status:

No known issues.

4.15.6 H5TRget_version

Name: H5TRget_version

Signature:

herr_t `H5TRget_version` (*hid_t* `trans_id`, *uint64_t* `*container_version`)

Purpose:

Retrieve the container version associated with a transaction.

Description:

The `trans_id` specifies the transaction whose associated container version is to be retrieved.

When a transaction is started via the `H5TRcreate` call, a read context is associated with the transaction. The container version returned by this H5TRget_version call is the one associated with the transaction is the container version associated with that read context.

The value pointed to by the `container_version` parameter will be set to the container version that transaction is associated with.

The actions performed by this function are all local to the compute node so an asynchronous execution option is not provided.

Parameters:

hid_t `trans_id`	IN: Identifier of the transaction whose associated container version is being retrieved.
uint64_t `*container_version`	OUT: Container version associated with the transaction. Value is unchanged if operation is unsuccessful.

Returns:

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

History:

Added in Quarter 7.

Man Page Status:

No known issues.

4.15.7 H5TRset_dependency

Name: H5TRset_dependency

Signature:

herr_t `H5TRset_dependency(` *hid_t* `trans_id,` *uint64_t* `trans_num,` *hid_t* `es_id )`

Purpose:

Register the dependency of a transaction on a lower-numbered transaction.

Description:

`H5TRset_dependency` registers a dependency of the transaction identified by `trans_id` on a lower-numbered transaction whose transaction number is given by `trans_num.` The transaction with the dependency is referred to as *dependent transaction* (identified by `trans_id`) and the lower-numbered transaction it depends on is the *prerequisite transaction* (identified by `trans_num`).

A dependency must be registered when a commit of the *dependent transaction* will leave the container in an inconsistent state unless the *prerequisite transaction* commits.

If a prerequisite transaction is discarded (either aborted or skipped) the dependent transaction will be aborted. The abort of the dependent transaction may occur at any time before it commits.

A given transaction may depend on multiple prerequisite transactions. It is an error to register a dependency on a future (higher-numbered) transaction.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `trans_id`	IN: Transaction identifier for the transaction with the dependency – the dependent transaction.
uint64_t `trans_num`	IN: Transaction number of the lower-numbered transaction that must commit successfully -- the prerequisite transaction.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.15.8 H5TRskip

Name: H5TRskip

Signature:

herr_t `H5TRskip(` *hid_t* `file_id,` *uint64_t* `start_trans_num,` *uint64_t* `count,` *hid_t* `es_id )`

Purpose:

Explicitly skip one or more transaction numbers for a container.

Description:

`H5TRskip` signals that the application will not be using the identified transaction numbers for the specified container. Since transactions are numbered and are committed in strict numeric order, having unused transaction numbers that are not explicitly skipped will prevent higher-numbered transactions from committing.

The `file_id` is the file identifier for a container that is open for write.

`start_trans_num` is the first transaction number that should be skipped.

`count` specifies how many transaction numbers should be skipped.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

`H5TRskip` should always be called by a single process for any given container and transaction number.

`H5TRskip` should never be called with the same `file_id` and `transaction_num` parameters used in a call to `H5TRcreate.` The (uncreated) transactions with the skipped transaction numbers are classified as *discarded*. Discarded transactions do not block the commitment of higher-numbered transactions.

Parameters:

hid_t `file_id`	IN: File identifier for container open for write.
uint64_t *`start_trans_num`*	IN: The first transaction number to skip.
uint64_t *`count`*	IN: How many transaction numbers to skip.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.15.9 H5TRstart

Name: H5TRstart

Signature:

herr_t `H5TRstart(` *hid_t* `trans_id,` *hid_t* `trspl,` *hid_t* `es_id )`

Purpose:

Start a created transaction.

Description:

`H5TRstart` starts the previously created transaction identified by `trans_id`.

After a transaction has been created with `H5TRcreate`, one or more transaction leaders must start the transaction by calling `H5TRstart`, effectively notifying lower layers of the I/O stack that the transaction is in-progress. The transaction is considered in-progress when the first `H5TRstart` for the transaction completes successfully.

Updates of the container associated with the transaction may be added to the transaction after the transaction has been started. The transaction leaders may issue updates at any time after `H5TRstart` had been called. The transaction leader(s) must notify the delegates (processes that called `H5TRcreate` but did not call `H5TRstart`) when they can begin adding updates to the transaction. Unlike the leaders, the delegates cannot add updates until after at least one of the asynchronous `H5TRstart` operations for the transaction completes successfully.

The `trspl` parameter identifies the transaction start property list for the call. Currently the list can contain only the leader count property that specifies the total number of leader processes that will call `H5TRstart` for the transaction identified by `trans_id.` If there is a single transaction leader, `trspl` should be set to `H5P_DEFAULT`. If there are multiple transaction leaders, then each must pass in a trasanction start property list specifying the same number of leaders. `H5Pset_trspl_num_peers()` is used to set the leader count property.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

For a given transaction, `H5TRfinish` must be called the same number of times that `H5TRstart` was called to start the transaction (assuming that the transaction was not aborted with `H5TRabort).` Typically, the leader processes that called `H5TRstart` to start the transaction each call `H5TRfinish`; in some circumstances – such as when a process dies – other processes may call `H5TRfinish.`

The transaction is considered finished when `H5TRfinish` has been called the required number of times.

After the transaction is finished and all lower-numbered transactions are finished or discarded (by `H5TRabort` or `H5TRskip`), the transaction is *committed* and all updates that are part of the transaction will be applied atomically to the container and become readable in a new container version with the same number as the committed transaction.

Parameters:

hid_t `trans_id`	IN: Transaction identifier indicating the transaction to start.
hid_t `trspl`	IN: Transaction start property list. Should be set to `H5P_DEFAULT` if only one process will call `H5TRstart` for the transaction. If multiple processes will make the call, the transaction start property list must container the leader count property, set with

`H5Pset_trspl_num_peers()`.

hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 5.

Man Page Status:

No known issues.

4.16 H5V: View Operation APIs

These routines are used to construct a view or portion of the HDF5 container matching a specific query. The view could potentially contain HDF5 objects (datasets, groups, maps, named datatypes), dataspace selections on HDF5 datasets, and attributes on the HDF5 objects depending on the query type that is passed to the view create. The H5V API also provides a set of routines to retrieve the view components from the view object.

These routines did not exist prior to the Exascale FastForward project.

Routine	**Implemented**	**Notes**
H5Vcreate_ff	Quarter 7	
H5Vget_location_ff	Quarter 7	
H5Vget_query	Quarter 7	
H5Vget_counts	Quarter 8	
H5Vget_attrs_ff	Quarter 8	
H5Vget_objs_ff	Quarter 8	
H5Vget_elem_regions_ff	Quarter 7	
H5Vclose	Quarter 7	

4.16.1 H5Vcreate_ff

Name: H5Vcreate_ff

Signature:

hid_t `H5Vcreate_ff(` *hid_t* `loc_id,` *hid_t* `query_id,` *hid_t* `vcpl_id,` *hid_t* `rcntxt_id,` *hid_t* `es_id )`

Purpose:

Create a new view object from a query on a container.

Description:

H5Vcreate_ff creates a new view of the file of all object, attributes, and/or dataset regions that satisfy the query in `query_id`. The file is traversed starting from the location specified in `loc_id`. To retrieve information and the view contents of the created view, the H5Vget_* routines can be used with the hid_t returned from this call.

The `vcpl_id` property list is used to pass in properties (described in the H5P* section) that influence the behavior of the view created. `vcpl_id` can also be the constant `H5P_DEFAULT`, in which case the default view creation properties are used.

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

The location ID can be an HDF5 File ID (indicating that the entire container is used to construct the view), an HDF5 group ID (indicating that just the group and objects recursively linked to from it are used to construct the view), or an HDF5 dataset ID (indicating that just the dataset and its elements are used to construct the view). For example, passing a file ID for the location ID in combination with a query on attribute names would result in a view object that contained attribute references on all the objects in the container that had an attribute of a given name. Or, passing a group ID for the location ID and a query on dataset elements would result in a view containing dataset region references for all datasets in the group and its sub-groups that had elements that match the query parameters. Some combinations of location and query IDs may result in creating an empty view object (such as passing a query on link names when using a dataset ID for a container ID, etc.).

The following table lists the query types and combinations for H5Q_COMBINE_AND operations, and the results that are put into the view object for each:

Query	Result Type	Result in View object
H5Q_TYPE_DATA_ELEM	Dataset Element	Dataset element regions in any dataset that matches the data element query
H5Q_TYPE_ATTR_VALUE	Attribute	Attributes on any object that matches the attribute value query
H5Q_TYPE_ATTR_NAME	Object	Objects which have an attribute that matches the attribute name query
H5Q_TYPE_LINK_NAME	Object	Objects which are reached through a link that matches the link name query
H5Q_TYPE_DATA_ELEM	Dataset	Dataset element regions in datasets that match both of the data

&& H5Q_TYPE_DATA_ELEM	Element	element queries
H5Q_TYPE_DATA_ELEM && H5Q_TYPE_ATTR_VALUE	<none>	<nothing> (Can't query regions in datasets and attribute values simultaneously)
H5Q_TYPE_DATA_ELEM && H5Q_TYPE_ATTR_NAME	Dataset Element	Dataset element regions in datasets that also have an attribute that matches the attribute name query
H5Q_TYPE_DATA_ELEM && H5Q_TYPE_LINK_NAME	Dataset Element	Dataset element regions in datasets that are reached through a link that matches the link name query
H5Q_TYPE_ATTR_VALUE && H5Q_TYPE_ATTR_VALUE	Attribute	Attributes whose values match both of the attribute value queries
H5Q_TYPE_ATTR_VALUE && H5Q_TYPE_ATTR_NAME	Attribute	Attributes whose values match the attribute value query and are on objects that match the attribute name query
H5Q_TYPE_ATTR_VALUE && H5Q_TYPE_LINK_NAME	Attribute	Attributes whose values match the attribute value query and are on objects reached through links that match the link name query
H5Q_TYPE_ATTR_NAME && H5Q_TYPE_ATTR_NAME	Object	Objects which have an attribute that matches both of the attribute name queries
H5Q_TYPE_ATTR_NAME && H5Q_TYPE_LINK_NAME	Object	Objects which have an attribute that matches the attribute name query and are reached through links that match the link name query
H5Q_TYPE_LINK_NAME && H5Q_TYPE_LINK_NAME	Object	Objects which are reached through links that match both of the link name queries

Further nesting of compound queries created with the H5Q_COMBINE_AND operator are allowed and refine the set of objects passed to the attribute value or dataset element queries.

Views created from compound queries that use the H5Q_COMBINE_OR operator contain the results for both of the sub-queries provided to that query.

View IDs returned from this routine must be released with H5Vclose.

Parameters:

hid_t `loc_id`	IN: Location identifier May be any HDF5 object identifier (group, dataset, map, or named datatype) or file identifier that is in scope for the read contxt. If `loc_id` is a file identifier, the file's root group will be the basis for the query.

hid_t `query_id`	IN: Query identifier for the query that is used to create the view.
hid_t `vcpl_id`	IN: View creation property list.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value. When executed asynchronously, a future ID for the view object is returned; upon completion of the asynchronous operation, the future ID will be transparently modified to be a "normal" view identifier.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 7. Man page added in Quarter 8.

Man Page Status:

No known issues.

4.16.2 H5Vget_location_ff

Name: H5Vget_location_ff

Signature:

herr_t `H5Vget_location_ff(` *hid_t* `view_id,` *hid_t* `*loc_id,` *hid_t* `es_id )`

Purpose:

Get the location that was used to construct a view.

Description:

`H5Vget_location_ff` gets the HDF5 object used as the base location to construct the view indicated by `view_id` and returns it in `*loc_id.`

The object identifier returned in `*loc_id` must be later be closed with `H5Oclose` (or the corresponding close operation) when it is no longer needed.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

When executed asynchronously, the value in `*loc_id` should not be used until after the asynchronous operation completes

Parameters:

hid_t `view_id`	IN: View identifier
hid_t `*loc_id`	IN/OUT: The HDF5 object used as the base location when the view was constructed.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value. In addition, *loc`_id` is set.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. Value in *`loc_id` should not be used until after the operation has completed.

History:

Added in Quarter 7. Man page added in Quarter 8.

Man Page Status:

No known issues.

4.16.3 H5Vget_query

Name: H5Vget_query

Signature:

herr_t `H5Vget_query(` *hid_t* `view_id`, *hid_t* `*query_id )`

Purpose:

Get the query that was used to construct a view.

Description:

`REVIEW FOR CORRECTNESS:`

`H5Vget_query` gets the query object used to construct the view indicated by `view_id` and returns a copy of it in `*query_id.`

The object handle returned in `*query_id` must be later be closed with `H5Qclose` when it is no longer needed.

Parameters:

hid_t `view_id`	IN: View identifier
hid_t `*query_id`	IN/OUT: The HDF5 query object used to construct the view.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value. In addition, `*fquery_id` is set.

History:

Added in Quarter 7. Man page added in Quarter 8.

Man Page Status:

No known issues.

4.16.4 H5Vget_counts

Name: H5Vget_counts

Signature:

herr_t `H5Vget_counts(` *hid_t* `view_id,` *hsize_t* `*attr_count,` *hsize_t* `*obj_count,` *hsize_t* `*elem_region_count`)

Purpose:

Retrieve the number of objects contained in the HDF5 view object.

Description:

`H5Vget_counts` retrieves various aspects of a view object, given by `view_id`. The number of attributes, objects and dataset element regions in the view is returned in the `attr_count`, `obj_count` and `elem_region_count` parameters, respectively.

Parameters:

hid_t `view_id`	IN: View identifier for the view that is being queried
hsize_t `*attr_count`	OUT: Number of attributes contained in the view.
hsize_t `*obj_count`	OUT: Number of HDF5 objects (maps, groups, datasets, named datatypes) contained in the view.
hsize_t `*elem_region_count`	OUT: Number of Datasets & Dataset regions contained in the view object.

Returns:

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

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.16.5 H5Vget_attrs_ff

Name: H5Vget_attrs_ff

Signature:

herr_t `H5Vget_attrs_ff(` *hid_t* `view_id,` *hsize_t* `start,` *hsize_t* `count,` *hid_t* `attr_id[],` *hid_t* `es_id )`

Purpose:

Retrieve attributes from an HDF5 View object.

Description:

`H5Vget_attrs_ff` retrieves attributes contained in a view object, given by `view_id`. Attributes contained in the view are uniquely enumerated internally to the view object, and the set of `count` attributes returned from this routine begin at offset `start` in that enumeration and are placed in the array of IDs given by `attr_id`, which must be preallocated by the user. The user is responsible for calling `H5Aclose` on all the attributes that have been returned in `attr_id[]` to free resources.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `view_id`	IN: View identifier
hsize_t `start`	IN: Starting index of the attributes in the view.
hsize_t `count`	IN: Number of attributes to return.
hid_t `attr_id[]`	OUT: preallocated array where the attribute identifiers will be returned.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. Values in `attr_id[]` should not be used until after the operation has completed.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.16.6 H5Vget_objs_ff

Name: H5Vget_objs_ff

Signature:

herr_t `H5Vget_objs_ff(` *hid_t* `view_id,` *hsize_t* `start,` *hsize_t* `count,` *hid_t* `obj_id[],` *hid_t* `es_id )`

Purpose:

Retrieve objects that are contained in the view.

Description:

`H5Vget_objs_ff` gets *count* objects starting from index *start* that are contained in the view object. Identifiers for those objects are return in the array *obj_id*[].

`H5Vget_objs_ff` retrieves objects contained in a view object, given by `view_id`. Objects contained in the view are uniquely enumerated internally to the view object, and the set of `count` objects returned from this routine begin at offset `start` in that enumeration and are placed in the array of IDs given by `obj_id`, which must be preallocated by the user. The user is responsible for closing those objects with their corresponding close operations or with `H5Oclose`.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `view_id`	IN: View identifier
hsize_t `start`	IN: Starting index of the objects in the view.
hsize_t `count`	IN: Number of objects to return.
hid_t `obj_id[]`	OUT: Array where the identifiers of the returned objects will be placed.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value. In addition, `obj_id[]` values are set.

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. Values in `obj_id[]` should not be used until after the operation has completed.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.16.7 H5Vget_elem_regions_ff

Name: H5Vget_elem_regions_ff

Signature:

herr_t `H5Vget_elem_regions_ff(` *hid_t* `view_id,` *hsize_t* `start,` *hsize_t* `count,` *hid_t* `dataset_id[],` *hid_t* `dataspace_id[],` *hid_t* `es_id )`

Purpose:

Retrieve datasets and dataspace element regions from the HDF5 view object.

Description:

`H5Vget_elem_regions_ff` retrieves dataset and dataspace (with selection) pairs contained in a view object, given by `view_id`. Data element regions referenced by the view are uniquely enumerated internally to the view object, and the set of `count` regions returned from this routine begin at offset `start` in that enumeration and are placed in the array of IDs given by `dataset_id` and `dataspace_id,` which must be preallocated by the user. Both `dataset_id` and `dataspace_id` must be large enough to hold at least `count` IDs.

Each dataspace ID returned from this routine corresponds to the dataset ID at the same offset as the dataspace ID. Each dataspace returned by this routine has a selection defined, which corresponds to the elements from the dataset that are included in the view. A dataspace returned from this routine can be used as a file dataspace parameter for calls to H5Dread or H5Dwrite on the matching dataset (or any other dataset with identical dimensions).

The user is responsible for closing the datasets and dataspaces with `H5Dclose` and `H5Sclose` respectively to free resources.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `view_id`	IN: View identifier
hsize_t `start`	IN: Starting index in the view.
hsize_t `count`	IN: Number of dataset/dataspace regions to return.
hid_t `dataset_id[]`	OUT: array with the datasets identifiers that are returned.
hid_t `dataspace_id[]`	OUT: array with the dataspace selection for the datasets in dataset_id[] array.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. Values in `dataset_id[]` and `dataspace_id[]` should not be used until after the operation has completed.

History:

Added in Quarter 7. Man page added in Quarter 8.

Man Page Status:

No known issues.

4.16.8 H5Vclose

Name: H5Vclose

Signature:

herr_t `H5Vclose(` *hid_t* `view_id )`

Purpose:

Terminate access to a view.

Description:

`H5Vclose` ends access to a view specified by `view_id` and releases resources used by it. Further use of the view identifier is illegal.

Parameters:

hid_t `view_id`	IN: View identifier for the view that is being closed.

Returns:

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

History:

Added in Quarter 7. Man page added in Quarter 8.

Man Page Status:

No known issues.

4.17 H5X: Index Operation APIs

These routines are used to register and manage index plugins, as well as create and remove indices on datasets.

These routines did not exist prior to the Exascale FastForward project.

Routine	Implemented	Notes
H5Xcreate_ff	Quarter 8	
H5Xget_count_ff	Quarter 8	
H5Xremove_ff	Quarter 8	
H5Xregister	Quarter 8	
H5Xunregister	Quarter 8	

4.17.1 H5Xcreate_ff

Name: H5Xcreate_ff

Signature:

herr_t `H5Xcreate_ff(` *hid_t* `file_id,` *unsigned* `plugin_id`, *hid_t* `scope_id,` *hid_t* `xcpl_id,` *hid_t* `trans_id,` *hid_t* `es_id )`

Purpose:

Create a new index on a container.

Description:

H5Xcreate_ff creates a new index object of type `plugin_id` (from the list of index plugins registered with the HDF5 library) in a container, given by `file_id`, over a set of objects in the container, given by `scope_id`.

The set of objects that an index applies to is determined by the `scope_id` passed to H5Xcreate_ff. Three types of scope are currently defined, determined by the type of ID passed in for the `scope_id`:

- H5File ID – Creates indices that include information about the contents of the whole container
- H5Group ID – Creates indices that include information about a group and all its descendants
- H5Dataset ID – Creates indices that include information about a dataset

Currently, only the dataset ID scope is supported for the `scope_id`.

Note that some combinations, such as creating a link name index on a dataset, are invalid and will fail with an error. Also, indices are created immediately on the objects in file and group scopes and future new objects added to the scope won't be included in the index (although they could have an index applied to them individually).

The indices created on the container are stored in the container's metadata and are not visible in the group hierarchy.

`xcpl_id` can be the constant `H5P_DEFAULT`, in which case the default index creation properties are used.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `file_id`	IN: File identifier indicating the container that the index will be created for.
unsigned `plugin_id`	IN: ID of index plugin to use for this operation.
hid_t `scope_id`	IN: Identifier indicating the scope that the index creation applies to.
hid_t `xcpl_id`	IN: Index creation property list.
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. The index will not be usable until the operation completes.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.17.2 H5Xget_count_ff

Name: H5Xget_count_ff

Signature:

herr_t `H5Xget_count_ff`(*hid_t* `object_id`, *hsize_t* * `idx_count`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

Purpose:

Determine the number of index objects on an object, possibly asynchronously.

Description:

`H5Xget_count_ff` retrieves the number of index object that exist for the object specified by `object_id`. `scope_id` must be in scope for the read context identified by `rcntxt_id`.

The number of index objects is returned in `idx_count.`

`rcntxt_id` indicates the read context for this operation.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `object_id`	IN: Identifier of the object. Must be in scope for the read context.
hsize_t *`idx_count`	OUT: The number of index objects for the object.
hid_t `rcntxt_id`	IN: Read context identifier indicating the read context for this operation.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.17.3 H5Xremove_ff

Name: H5Xremove_ff

Signature:

herr_t `H5Xremove_ff(` *hid_t* `file_id,` *unsigned* `plugin_id`, *hid_t* `scope_id,` *hid_t* `trans_id,` *hid_t* `es_id )`

Purpose:

Remove an index from objects in a container.

Description:

H5Xremove_ff removes index objects of type `plugin_id` (which doesn't have to be from the set of index plugins currently registered with the HDF5 library) in a container, given by `file_id`, over a set of objects in the container, given by `scope_id`.

The set of objects that an index applies to is determined by the `scope_id` passed to H5Xremove_ff. Three types of scope are currently defined, determined by the type of ID passed in for the `scope_id`:

- H5File ID – Creates indices that include information about the contents of the whole container
- H5Group ID – Creates indices that include information about a group and all its descendants
- H5Dataset ID – Creates indices that include information about a dataset

Currently, only the dataset ID scope is supported for the `scope_id`.

`trans_id` indicates the transaction this operation is part of.

The `es_id` parameter indicates the event stack the event object for this call should be pushed onto when the function is executed asynchronously. The function may be executed synchronously by passing in `H5_EVENT_STACK_NULL` for the `es_id` parameter.

Parameters:

hid_t `file_id`	IN: File identifier indicating the container that the index will be created for.
unsigned `plugin_id`	IN: ID of index plugin to use for this operation.
hid_t `scope_id`	IN: Identifier indicating the scope that the index creation applies to.
hid_t `trans_id`	IN: Transaction identifier specifying the transaction this operation is a part of.
hid_t `es_id`	IN: Event stack identifier specifying the event stack that will be used to monitor the status of the event associated with this function call when executed asynchronously. Use `H5_EVENT_STACK_NULL` for synchronous execution.

Returns:

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

When this routine is executed asynchronously, the return value indicates whether the operation has been successfully scheduled for asynchronous execution. The actual success or failure of the asynchronous operation must be checked separately through the event stack. The index will not be removed until the operation completes.

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.17.4 H5Xregister

Name: H5Xregister

Signature:

herr_t `H5Xregister(` *const H5X_class_t* `*index_plugin`)

Purpose:

Register a new index plugin.

Description:

H5Xregister registers an index plugin, given by `index_plugin` for use with H5Xcreate_ff. The plugin registered is immediately available.

The H5X_class_t structure is defined in the Index Plugin Interface section.

Parameters:

const H5X_class_t `*index_class`	IN: A pointer to the index plugin class structure for the plugin to register.

Returns:

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

History:

Added in Quarter 8.

Man Page Status:

No known issues.

4.17.5 H5Xunregister

Name: H5Xunregister

Signature:

herr_t `H5Xunregister(` *unsigned* `plugin_id`)

Purpose:

Unregister an index plugin.

Description:

H5Xunregister removes an index plugin, given by `plugin_id`, from the library.

Parameters:

unsigned `plugin_id`	IN: The index plugin ID for the plugin to unregister

Returns:

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

History:

Added in Quarter 8.

Man Page Status:

No known issues.

5 Index Plugin Interface

This interface allows index plugins to define callback routines that the HDF5 library can invoke, to perform various operations with indices on HDF5 datasets. Index plugins are registered for application use with the H5Xregister API routine.

5.1 Index Plugin Callbacks

Each index plugin must implement the following callbacks:

1. "create"

   Purpose: Creates a new index for a dataset. Called on dataset creation or later, after data has been written to the dataset, to create an index on an existing dataset.

   Parameters:

   IN:

   - container/file ID
   - dataset ID
   - index creation property list (XCPL)
     - Used for setting any permanent properties on the index
   - index access property list (XAPL)
     - Used for setting any transient properties on the index

   OUT:

   - metadata and metadata size, for future index opens
     - Allocated/managed by plugin
     - Stored by HDF5 library in object header for dataset

   Returns:

   NULL on failure, pointer to index plugin's structure for future use on success

   Prototype:

   void * (*create)(hid_t file_id, hid_t dataset_id, hid_t xcpl_id, hid_t xapl_id, size_t *metadata_size, void **metadata);

   Notes:

   - Index can query datatype & dataspace of dataset, if needed.
   - Internally, the index plugin may call public HDF5 API routines for storing its information. The H5[D|G|T]create_anon() + H5Oincr_refcount() combination is ideal for creating an object that isn't attached to the file's group hierarchy, and whose existence can be managed by the index plugin (through the ref. count on the object). (The anonymous object's address in the file will queried with H5Oget_info, so that it can be stored in the index plugin metadata for re-opening later)

2. "open"

   Purpose: Opens an existing index for a dataset. Called on dataset open.

Parameters:

IN:

- container/file ID
- dataset ID
- index access property list (XAPL)
  - Used for setting any transient properties on the index
- metadata and metadata size
  - Used to locate index within container

OUT:

- <none>

Returns:

NULL on failure, pointer to index plugin's structure for future use on success

Prototype:

void * (*open)(hid_t file_id, hid_t dataset_id, hid_t xapl_id, size_t metadata_size, void *metadata);

Notes:

- Index can query datatype & dataspace of dataset, if needed.
- Internally, the index plugin may decode its metadata and use any information contained within to access its index information. (Object addresses can be used to open objects with H5Oopen_by_addr)

3. "pre-update"

Purpose: Notifies index plugin that data elements will be overwritten. Called immediately before data elements are written (or overwritten).

Parameters:

IN:

- index info pointer (from 'create' or 'open' callback)
- selection for data elements to be written in dataset
- index transfer property list (XXPL)
  - Used for setting any transfer properties on the index update

OUT:

- <none>

Returns:

SUCCEED / FAIL

Prototype:

herr_t (*pre_update)(void *idx_handle, hid_t dataspace_id, hid_t xxpl_id);

Notes:

- This callback is designed to enable data element overwrites, by possibly allowing the plugin to remove elements that will be overwritten from its index.
- This callback may be ignored by a plugin, if the 'post_update' callback is sufficient.

4. "post-update"

   Purpose: Updates index information with new data elements. Called after data elements are written to the dataset (or overwritten).

   Parameters:

   IN:

   - index info pointer (from 'create' or 'open' callback)
   - buffer with data elements written to dataset
   - selection for data written to dataset (same as for 'pre_update' callback)
   - index transfer property list (XXPL)
     - Used for setting any transfer properties on the index update

   OUT:

   - <none>

   Returns:

   SUCCEED / FAIL

   Prototype:

   herr_t (*post_update)(void *idx_handle, const void *buffer, hid_t dataspace_id, hid_t xxpl_id);

   Notes:

   - Even with the 'pre_update/post_update' callbacks, overwrites are probably going to be painful, since the previous element location/value will need to be removed from the index. Deferring actual updates to the index until the 'close' callback may be possible, if the plugin tracks all the regions overwritten.

5. "query"

   Purpose: Retrieves selection that matches query from index. Called when an application calls H5Vcreate or H5Dquery.

   Parameters:

   IN:

   - index info pointer (from 'create' or 'open' callback)
   - query
   - index transfer property list (XXPL)

        - Used for setting any transfer properties on the index update

    OUT:

    - dataspace with selection of elements that match query

Returns:

SUCCEED / FAIL

Prototype:

herr_t (*query)(void *idx_handle, hid_t query_id, hid_t xxpl_id, hid_t *dataspace_id);

Notes:

- Any restriction on the region of the dataset to query (the "array slice") will be passed in as a property in the XXPL. The index plugin can query it through the H5P*() interface.

6. "refresh"

Purpose: Refreshes the index metadata, if it has changed as a result of an update. Called after the ‘post_update’ callback when data elements are written to the dataset (or overwritten).

Parameters:

IN:

- index info pointer (from 'create' or 'open' callback)

OUT:

- metadata and metadata size, for future index opens
    - Stored by HDF5 library in object header for dataset, overwriting previous metadata

Returns:

SUCCEED / FAIL

Prototype:

herr_t (*refresh)(void *idx_handle, size_t *metadata_size, void **metadata);

Notes:

- The previous metadata is presented to the plugin, which may update it, including reallocating it and/or changing the size.
- If this routine is not needed, the callback value may be set to NULL.

7. "close"

Purpose: Close the in-memory data structures for an index plugin. Called when the dataset is closed.

Parameters:

IN:

- index info pointer (from 'create' or 'open' callback)

OUT:

- <none>

Returns:

SUCCEED / FAIL

Prototype:

herr_t (*close)(void *idx_handle);

Notes:

- <none>

8. "remove"

Purpose: Delete the index's information from the container. Called when the dataset is deleted from the container, or the index is removed from the dataset.

Parameters:

IN:

- container/file ID
- dataset ID
- metadata and metadata size
  - Used to locate index within container

OUT:

- <none>

Returns:

SUCCEED / FAIL

Prototype:

herr_t (*remove)(void *idx_handle, hid_t dataset_id, size_t metadata_size, void *metadata);

Notes:

- The plugin will need to delete any objects in the container that it is using to store the index information.

5.2 Index Plugin Types

The following types are defined to support the index plugin extensions:

5.2.1 H5X_type_t

```
typedef enum {
   H5X_TYPE_LINK_NAME,           // Link name index
   H5X_TYPE_ATTR_NAME,           // Attribute name index
   H5X_TYPE_DATA_ELEM,           // Dataset element index
   H5X_TYPE_MAP_VALUES           // Map value index
} H5X_type_t;
```

This enum is used to show the type of data that an index plugin can handle. Currently, only plugins that handle dataset elements (H5X_TYPE_DATA_ELEMENTS) are supported.

5.2.2 H5X_class_t

```
typedef struct {
   unsigned version;   /* Version number of the index plugin class struct */
                       /* (Should always be set to H5X_CLASS_VERSION, which
                        *      may vary between releases of HDF5 library)
                        */
   unsigned id;        /* Index ID (assigned by The HDF Group, for now) */
   const char *idx_name; /* Index name (for debugging only, currently) */
   H5X_type_t type;    /* Type of data indexed by this plugin */

   /* Callbacks, described above */
   void * (*create)(hid_t file_id, hid_t dataset_id, hid_t xcpl_id,
            hid_t xapl_id, size_t *metadata_size, void **metadata);
   herr_t (*remove)(hid_t file_id, hid_t dataset_id, size_t metadata_size,
            void *metadata);
   void * (*open)(hid_t file_id, hid_t dataset_id, hid_t iapl_id,
            size_t metadata_size, void *metadata);
   herr_t (*close)(void *idx_hand);
   herr_t (*pre_update)(void *idx_hand, hid_t buffer_dataspace,
            hid_t xxpl_id);
   herr_t (*post_update)(void *idx_hand, const void *buffer,
            hid_t buffer_dataspace, hid_t xxpl_id);
   herr_t (*query)(void *idx_hand, hid_t query_id, hid_t xxpl_id,
            hid_t *dataspace_selection_id);
   herr_t (*refresh)(void *idx_hand, size_t *metadata_size, void **metadata);
} H5X_class_t;
```

This struct holds all the information about an index plugin, and is passed to the HDF5 library to register the plugin, with H5Xregister.

5.3 Sample Codeflow

The following pseudo-code outlines what calls occur when an index is created and updated:

Application Code:

```
   .
   .
   .

H5X_class_t idx_plugin_struct = {
   H5X_CLASS_VERSION,        /* (From the HDF5 H5Xpublic.h header file) */
   32,                       /* (Or whatever number is assigned) */
   "sample index plugin",    /* Whatever name desired */
   H5X_TYPE_DATA_ELEM,       /* This plugin operates on dataset elements */

   /* Plugin callbacks, defined elsewhere in application/library code */
   sample_create,
   sample_remove
   sample_open,
   sample_close,
   sample_pre_update,
   sample_post_update,
   sample_query,
   sample_refresh,
};

   .
   .
   .

/* Register the index plugin */
     H5Xregister(&index_plugin_struct);

     ==> Interally, registers the index plugin, but doesn't invoke any callbacks in it.

/* Create a dataset '/A' */
dsid = H5Dcreate(fid, "A", ...);

/* Add a data element index using our plugin to dataset '/A' */
/* (Uses a default index creation property list, for now) */
     H5Xcreate(fid, 32, dsid, H5P_DEFAULT);

      ==> Internally, the HDF5 library calls the index plugin's 'create'
         callback, which allows plugin to create its metadata for indexing
         the '/A' dataset.  The metadata returned from the index plugin is
         stored by the HDF5 library with other object metadata for '/A'.

      ==> The application could have also set up the index in a dataset creation property
         list that was passed in to the H5Dcreate() call, which would create
         the index at the same time as the dataset was created.

   .
   .
   .

/* Write data to '/A' */
     H5Dwrite(dsid, ...);

      ==> Internally, the HDF5 library calls the plugin's 'pre_update'
         callback, writes the data to the dataset, then calls the plugin's
         'post_update' callback (allowing the plugin to update the index
         information for the dataset), then calls the plugin's 'refresh' callback.
         [The plugin could also store the
         region updated in the dataset, and defer the index update until
         later (possibly the index 'close' callback).]

   .
   .
   .

/* Close the dataset */
     H5Dclose(dsid);

      ==> Internally, the HDF5 library calls the plugin's 'close' callback,
         allowing it to release in-memory data structures and to close
         objects it may have open in the HDF5 file.
```

```
<< Later, in another section of code or a different application>>

/* Open the dataset */
dsid = H5Dopen(fid, "A", ...);

        ==> Internally, the HDF5 library reads the index metadata stored for
            the index plugin, looks up and opens a DLL that contains the
            plugin (if it isn't already opened) and calls the plugin's 'open'
            callback.
    .
    .
    .

/* Define a query that will search for element values equal to 15 */
/* (Note that this is strictly "local" and doesn't actually invoke plugin) */
int query_val = 15;
qid = H5Qcreate(H5Q_TYPE_DATA_ELEM, H5Q_MATCH_EQUAL, H5T_NATIVE_INT,
&query_val);

/* Create a view that holds the results of applying the query to a dataset */
/* (Uses a default view creation property list, for now) */
/* (A non-default view creation property list could contain an array slice, to
 *      limit the query to a portion of the dataset)
 */
vid = H5Vcreate(dsid, qid, H5P_DEFAULT);

        ==> Internally, the HDF5 library calls the plugin's 'query' callback,
            which will perform the query against the index and build a dataspace
            selection to return to the library for later use.

/* Retrieve the region that matched the query from the view */
    H5Vget_elem_regions(vid, ..., &dataspace_id);

/* Use the region to read the elements from the dataset that matched the query */
/* (Doesn't invoke plugin) */
   H5Dread(dsid, H5T_NATIVE_INT, mem_space_id, dataspace_id, H5P_DEFAULT, buf);
   .
   .
   .

/* Close the dataset */
     H5Dclose(dsid);

      ==> Internally, the HDF5 library calls the plugin's 'close' callback,
         allowing it to release in-memory data structures and to close
         objects it may have open in the HDF5 file.


<< Later, in another section of code or a different application>>

/* Remove an index from a dataset */
     H5Xremove(file_id, 32, dataset_id);

      ==> Internally, the HDF5 library removes the metadata from the object's
         metadata and calls the index plugin's 'remove' callback, so the plugin can
         delete any objects it is using to store the index information in the container.  (The
         plugin's 'remove' callback would also get called when the dataset
         was deleted in the file)
```

6 Description of example programs

All the example programs that are included in the source code are heavily commented to describe the functionality being demonstrated. A higher-level description of the example programs is included in this section.

Figure 1 shows the relationship of the client and server example programs to the layers of the EFF stack being demonstrated in this milestone.

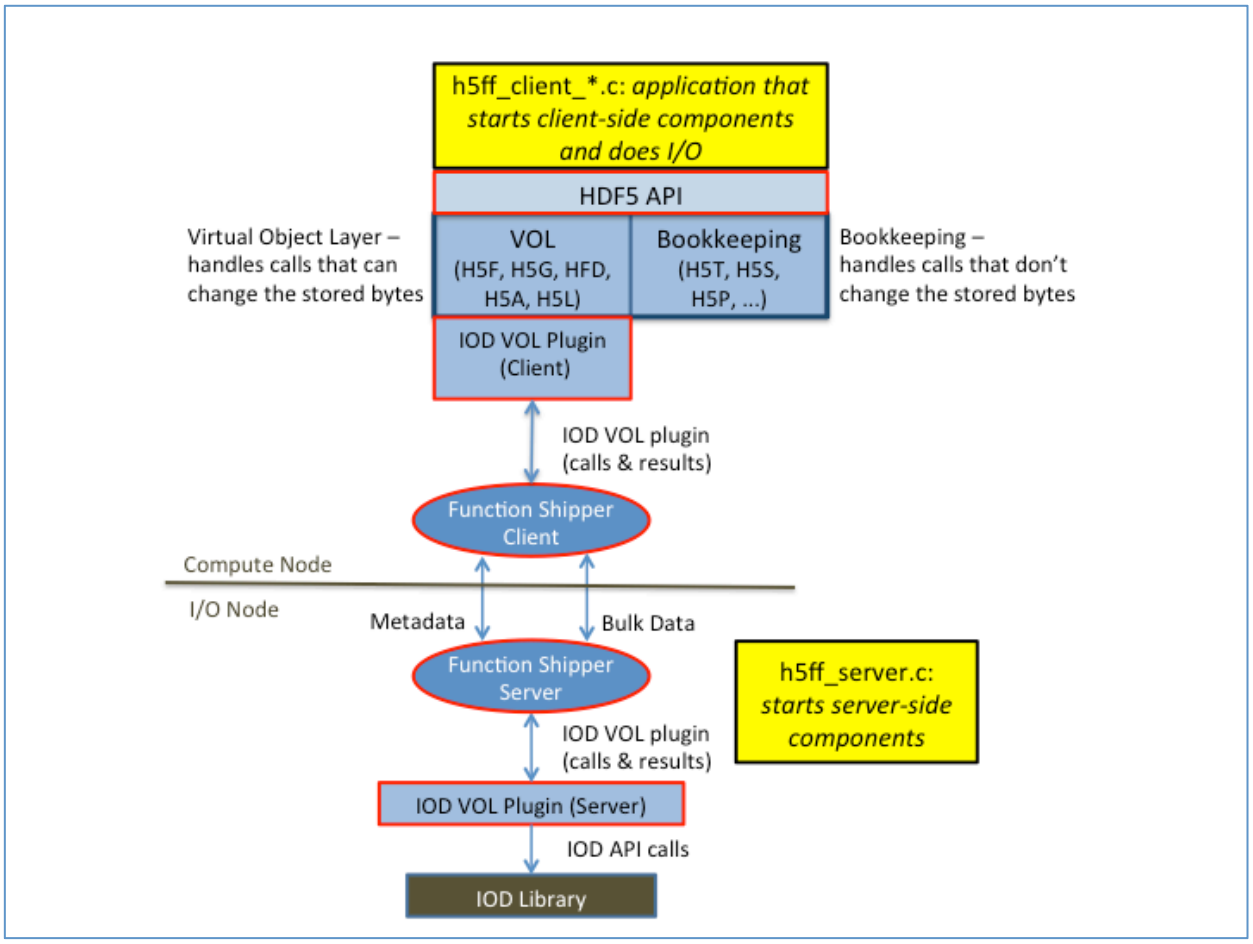
h5ff_client_*.c: application that starts client-side components and does I/O
HDF5 API
Virtual Object Layer – handles calls that can change the stored bytes
VOL (H5F, H5G, HFD, H5A, H5L)
Bookkeeping (H5T, H5S, H5P, ...)
Bookkeeping – handles calls that don't change the stored bytes
IOD VOL Plugin (Client)
IOD VOL plugin (calls & results)
Function Shipper Client
Compute Node
I/O Node
Metadata
Bulk Data
Function Shipper Server
h5ff_server.c: starts server-side components
IOD VOL plugin (calls & results)
IOD VOL Plugin (Server)
IOD API calls
IOD Library

Figure 1: Client and server programs demonstration components

6.1 h5ff_server.c

The h5ff_server.c program launches an MPI application that acts as a server. This program will run on the IONs in the EFF architecture, but is being run on the same multi-processor system as clients for Q5 Milestones.

h5ff_server.c initializes the function shipper interfaces for metadata and bulk data, registers all the required callbacks to handle HDF5 VOL operations, and listens to incoming requests from clients. The first request received from clients is an initialization call that starts up the IOD library for the client and creates an AXE engine to insert future operations into. The last call received from clients is a terminate call that will eventually shut down all the initialized libraries, once all clients have issued this call.

Note that the server program is independent from the client program, meaning that any client that uses HDF5 with the HDF5 IOD VOL plugin can connect to the server.

6.2 h5ff_client_*.c

Each of the client programs in the examples directory of the hdf5_ff source tree launches an MPI application that acts as a client. These programs run on the CNs in the EFF architecture. The client programs demonstrate how an application would use the EFF I/O stack. This is a list of the primary client programs included with a note to indicate what each one is testing (note that the comments inside each program explain in much more detail the flow of execution and usage of the HDF5 routines and semantics):

- h5ff_client_analysis.c: This tests the analysis shipping functionality (H5AS).
- h5ff_client_attr.c : This tests attribute routines (H5A).
- h5ff_client_dset.c : This tests dataset routines (H5D).
- h5ff_client_evict_deltas.c: This tests eviction of transaction updates (deltas).
- h5ff_client_links.c : This tests Links routines (H5L).
- h5ff_client_map.c : This tests the new Map routines (H5M) added to support Dynamic Data Structures.
- h5ff_client_multiple_cont.c : This tests access to multiple containers.
- h5ff_client_obj.c : This tests generic object routines (H5O).
- h5ff_client_prefetch.c: This tests prefetch and eviction of replicas.
- h5ff_client_view.c: This tests query and view routines (H5Q and H5V)
- H5ff_client_vl_data.c: This tests variable length data.
- h5ff_client_M6.2_demo.c: HDF5 and I/O Dispatcher Container Versioning Demonstration.
- h5ff_client_M7.2-pep_demo.c: Persist, Evict, and Prefetch data movement demonstration.

7 Instructions for building and running demo code

These instructions can also be found in the file "EFF_INSTALL.txt" in the top directory of the distributed tarball.

```
System pre-requisites:
        An MPI3 implementation that supports MPI_THREAD_MULTIPLE
            [i.e. MPICH3 or later - we built with MPICH 3.0.4 in Q6]
        Pthread
        BOOST

build opa:
        It is located in the tarball in the 'openpa' subdirectory, OR get it
        from here: git clone git://git.mcs.anl.gov/radix/openpa.git

        ./configure --prefix=/path/to/opa/install/directory
        make
        make check
        make install


build AXE:
        It is located in the tarball in the 'axe' subdirectory, OR get it from
        here: svn checkout https://svn.hdfgroup.uiuc.edu/axe/trunk

        ./configure --prefix=/path/to/axe/install/directory --with-
opa=/path/to/opa/install/directory
        make
        make check
        make install


build DAOS, PLFS, and IOD:
        Please refer to the IOD tarball for instruction on how to build and setup the
three libraries.


build Mercury (Function Shipper)
        The code is located in tarball in the 'mercury' directory.

        refer to the mercury build recipe in:
        mercury/README

build HDF5 IOD VOL plugin:
        The code is located in the tarball in the 'hdf5_ff' subdirectory, OR
        get it from here:
            svn checkout http://svn.hdfgroup.uiuc.edu/hdf5/features/hdf5_ff

       ./configure --with-daos=/path/to/daos/posix --with-plfs=/path/to/plfs --with-
       iod=/path/to/iod/ --with-axe=/path/to/axe/install/directory
       PKG_CONFIG_PATH=/path/to/mercury/install/directory/lib/pkgconfig/:/path/to/mchecksu
       m/install/dir --enable-parallel --enable-debug --enable-trace --enable-threadsafe -
       -enable-unsupported --with-pthread=/usr --enable-eff --enable-shared --enable-
       python

        If you want indexing to be built in add --enable-indexing.

       Note in that case all 3rd party libraries have to be build shared or with -fPIC.
       You should also have devel python devel libraries and numpy installed on your
       system.
```

You should see in the configure summary at the end if the EFF plugin in HDF5 was successfully configured.

```
make
make check
make install
```

build the example programs:

```
        The examples are located in hdf5_ff/examples/.
        The server is h5ff_server.
        The client programs are
        - h5ff_client_attr.c : This tests attribute routines (H5A).
        - h5ff_client_dset.c : This tests dataset routines (H5D).
        - h5ff_client_links.c : This tests Links routines (H5L).
        - h5ff_client_map.c : This tests the new Map routines (H5M) added this quarter to
support Dynamic Data Structures.
        - h5ff_client_multiple_cont.c : This tests access to multiple containers.
        - h5ff_client_obj.c : This tests generic object routines (H5O).
        - h5ff_client_analysis.c : This tests the analysis shipping functionality (H5AS).
        - h5ff_client_M6.2_demo.c: HDF5 and I/O Dispatcher Container Versioning
Demonstration
       - h5ff_client_M7.2-pep_demo.c: Prefetch, evict, persist data movement demo.

        cd path/where/hdf5_ff/is/built/examples/
        make
        chmod 775 run_ff_tests.sh
        ./run_ff_tests.sh num_server_procs num_client_procs

        Or you can run each test manually:
                The client and server need to be launched from the same directory for now.
                Launch the server first:
                mpiexec -n <x> ./h5ff_server
                then launch one of the clients
                mpiexec -n <x> ./h5ff_client_xx
```

Note, for now, the number of clients must be greater than or equal to the number of servers.

END

8 Instructions for Building Python Wrappers & Regression Test Suite

All the API functions described above were integrated with Python using the framework utilized by the h5py[1] module. H5py is an open-source Python module is based on one-to-one mapping to the standard HDF5 C API using the Cython[2] programming language. Cython allows creating C language extensions for Python through language syntax very similar to Python's own.

Once the new API functions were added into appropriate Cython extension types (acting like Python built-in types), h5py's Python interface was extended to reflect the new capabilities of the HDF5 library. A testing suite based on the Python `unittest` framework was developed to test the integrated API and its Python interface.

The source code for the modified h5py module can be obtained from the HDF Group's Subversion repository[3]. The prerequisites for building the module are: an installed FastForward version of the HDF5 library, and the `NumPy` and `setuptools` Python modules. If the files are checked out from the repository, another requirement is to have Cython installed.

Once all the requirements are satisfied, the following command from the top-level directory (where the setup.py file is) builds the module:

```
CC={path to h5pcc} python setup.py build --hdf5={path to HDF5 lib} --eff
```

If the build step was successful, the next step is to test the module. The prerequisites for testing are similar as for using the module.

- The following Python modules must be installed prior:
    - `mpi4py`
    - `unittest` (`unittest2` in case of Python 2.6)
- Several environment variables must also be set:
    - $EFF_MPI_IONS – contains cluster ION hostnames, separated by commas
    - $EFF_MPI_CNS – contains cluster CN hostnames, separated by commas
    - $H5FF_SERVER – path to the h5ff_server executable

The command to run the testing suite is:

```
mpiexec -np 1 -hosts {CN} python setup.py test
```

where CN is hostname of one of the compute nodes. This command must be executed on an ION.

Once the testing suite finishes, and assuming no problems are encountered, the final step is to install the module. The command to install the module in a system-wide location is:

```
python setup.py install
```

To install the modified h5py module for a specific user:

```
python setup.py install --user
```

[1] http://www.h5py.org

[2] http://cython.org

[3] http://svn.hdfgroup.uiuc.edu/fastforward/h5py_ff

User-specific installation requires an environment variable $PYTHONUSERBASE to point to a directory under which the modified h5py module will be installed.

Incorporating the modified h5py module in Python programs is the same as the original h5py module: an “import h5py” statement should be added to the code that wants to use the module.