RFC: HDF5 Tools Library Functions

Peter Cao
Jonathan Kim
Albert Cheng
Allen Byrne
Gerd Heber

This RFC discusses how to restructure the tools library and propose a list of public utility functions. The goal of the work is to set a consistent structure so that tool developers can more effectively reuse and maintain the code. The public tools functions provide a convenient way for general users to implement their applications.

The utility functions proposed in this RFC are intended to do simple and specific things. For comprehensive operations, we recommend using the HDF5 library functions or the HDF5 high level functions.

Table of Contents

1 Introduction

This RFC proposes a layout structure for the tool library and suggests a list of public utility functions. Currently there is no clear structure for developers to follow as implementing new functions either for updating current tools or developing a new tool. This causes repeated code and functions both in the tool library and tools. It significantly downgrades the quality of code and also reduces productivity for a developer not to be able to easily search and reuse functions already implemented before. The public functions will also help users developing their applications without making many HDF5 function calls for simple operations.

2 Structural layout design

The proposed code layout includes two sets of functions: public utility functions and tools internal functions. The public utility functions can be used by any applications while the internal functions are intended for HDF5 command-line tools only. See the figure below for the proposed layout.

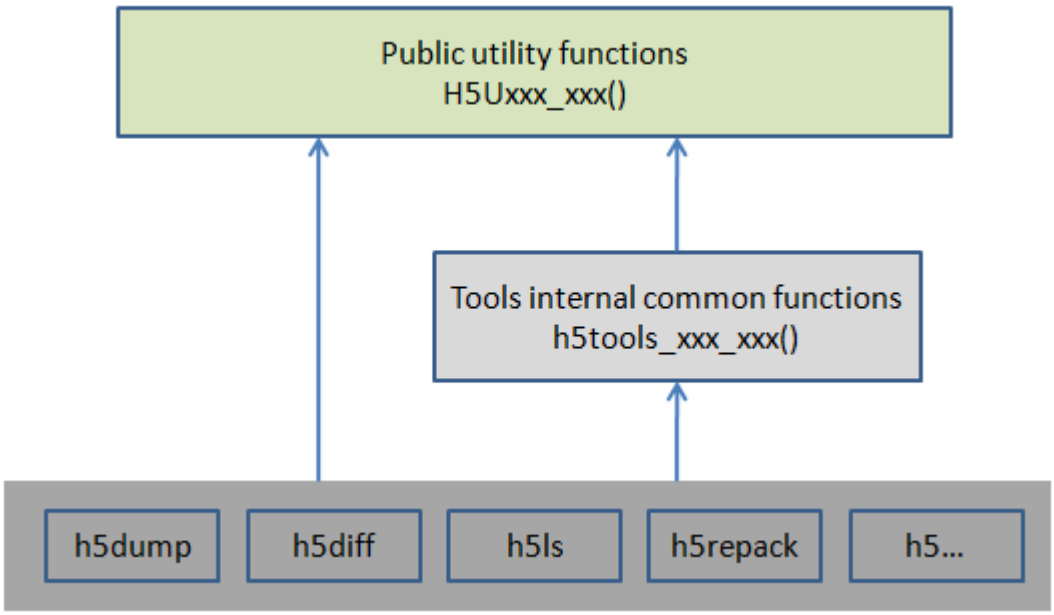

Figure 1 -- Structural layout design for HDF5 tools

2.1 Public utility functions

The public functions can be used both by the HDF5 tools and other applications. The main purpose of the public functions is to reduce the work for users (including our own tools and application developers) without writing a lot repeated codes especially HDF5 library function calls.

We propose the names of the public functions start with "*H5U*", as "*U*" stands for utility, useful, or user-friendly. This naming approach is also consistent with the rest of HDF5 public functions. The source file for public functions will be named H5U.c[h] and resided in the HDF5 src directory.

The set of the public utility functions, H5Uxxx, will be part of the HDF5 library release. It will be built and distributed in the same way as other HDF5 public function, e.g. H5Axxx.

2.2 Tools internal functions

The tools internal functions are intended to be used by the HDF5 tools. Most of the functions at tools/lib can be treated as internal functions. Some of the current tools functions at tools/lib can be moved to public function set.

The internal functions will named as h5tools_xxx_xxx(). The source files will stay at the current tools/lib directory.

Current HDF5 tools use the internal functions in the tools/lib directory. Those functions include:

- Traversing
- Comparing
- Searching
- Fetching
- Importing/Exporting
- Copying
- Objects manipulating (create, remove, rename)
- Status checking
- Debugging
- Print out the result various way (OUTPUT)
- Command parameter passing options scheme (INPUT from user)
- Error handling
- Debugging printouts
- Memory handling
- And so on.

3 Requirements for the public functions

Below are some basic requirements for the public utility functions:

- **Simple to use** – the main purpose for the utility functions is to make users easy to write program in HDF5. These functions do not provide comprehensive functionalities but they are simple and easy to use.
- **Consistent with other library API functions** – the naming, parameter list, and return values should be consistent with the rest of the HDF5 public functions.

- **Portable cross major programming languages** – the utility functions will avoid using function pointers or C-data structures so that they can easily supported by other programming languages such as Java, Fortran, .NET, and etc.
- **Well tested and documented** – the utility functions will be supported at the same level as other HDF5 API functions. They will be well tested and documented.

4 A list of proposed utility functions

In this section, we will describe a list of proposed public utility functions for open discussion. More functions will be added to the list as we work on code refactoring for the current tools code.

The functions are intended to do simple and specific things. For complex and complete functionalities, we recommend users to use HDF5 API functions or the HDF5 high level API functions.

4.1 *H5Uis_well_formed_path*

Name/ Signature:

hbool_t H5Uis_well_formed_path(const char path)*

Purpose:

Check if a path is well formed.

Description:

An absolute or relative HDF5 path name is well-formed, if and only if it is a syntactically correct HDF5 path name. (What does that mean? No trailing slashes, no occurrences of intermediate /\.*/, ... ?) The following function verifies whether a given path name is a well-formed HDF5 path name.

Parameters:

const char path* IN: path to be checked.

Returns:

Returns TRUE if the path is well formatted; otherwise, returns FALSE.

4.2 *H5Unormalize_path*

Name/ Signature:

herr_t H5Unormalize_path(const char path, char* normal_path)*

Purpose:

Normalize a path.

Description:

An absolute or relative HDF5 path name is normal, if and only if it is well-formed, it does not contain leading or trailing whitespace characters, and it does not contain consecutive occurrences of slash characters. On success, the following function returns a normalized HDF5 path name. It returns an error, if the path is not well formed. The normalized HDF5 path name of a normalized HDF5 path name is that HDF5 path name itself.

Parameters:

const char path*	IN: the input path.
char normal_path*	OUT: the normalized path.

Returns:

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

4.3 *H5Uparent_path*

Name/ Signature:

herr_t H5Uparent_path(const char path, char* parent_path)*

Purpose:

Get the parent path.

Description:

The following function returns the parent segment of a well-formed absolute or relative HDF5 path name. The parent path name of the HDF5 root path name /is the HDF5 root path name. The parent path name of a relative HDF5 path name referring to the current location (e.g., .or ./) is the HDF5 path name itself. This function returns an error, if the path is not well formed.

Parameters:

const char path*	IN:	the input path.
char parent_path*	OUT:	the parent path.

Returns:

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

4.4 *H5Uchild_path*

Name/ Signature:

herr_t H5Uchild_path(const char path, char* child_path)*

Purpose:

Get the child path.

Description:

The following function returns the child segment of a well-formed absolute or relative HDF5 path name. The child name of the HDF5 root path name is the empty string \0. The child name of a relative HDF5 path name referring to the current location (e.g., .or ./) is is the empty string \0. This function returns an error, if the path is not well formed.

Parameters:

const char path*	IN: the input path.
char child_path*	OUT: the child path.

Returns:

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

4.5 *H5Uis_valid_path*

Name/ Signature:

hbool_t H5Uis_valid_path(hid_t loc, const char path)*

Purpose:

Verify if a path points to a valid object.

Description:

A well-formed HDF5 path name is valid with respect to a handle loc, if and only if an HDF5 object or symbolic link exists at path relative to loc, i.e., can be successfully opened with H5Oopen. An absolute HDF5 path name is valid only with respect to the HDF5 root group. The following function verifies whether a given path name is a valid HDF5 path name.

A well-formed HDF5 path name is valid only with respect to a reference location. Nevertheless, to simplify the terminology, we will speak of "valid HDF5 path names" and assume that the location with respect to which the HDF5 path name is valid is clear from the context.

Parameters:

hid_t loc	IN: A file or group identifier.
const char path*	IN: the path.

Returns:

Returns TRUE if the path is valid; otherwise, returns FALSE.

4.6 *H5Uis_sym_link*

Name/ Signature:

hbool_t H5Uis_sym_link(hid_t loc, const char path)*

Purpose:

Check for symbolic link.

Description:

This function checks if the given path is a symbolic link (soft link or external link) or not.

Parameters:

hid_t loc	IN: A file or group identifier.
const char path*	IN: the path.

Returns:

Returns TRUE if the path is a symbolic link; otherwise, returns FALSE.

4.7 *H5Uis_external_link*

Name/ Signature:

hbool_t H5Uis_external_link(hid_t loc, const char path);*

Purpose:

Check for external link.

Description:

This function checks if the given path is an external link or not.

Parameters:

hid_t loc	IN: A file or group identifier.
const char path*	IN: the path.

Returns:

Returns TRUE if the path is an external link; otherwise, returns FALSE.

4.8 *H5Uis_soft_link*

Name/ Signature:

hbool_t H5Uis_soft_link(hid_t loc, const char path);*

Purpose:

Check for soft link.

Description:

This function checks if the given path is a soft link or not.

Parameters:

hid_t loc	IN: A file or group identifier.
const char path*	IN: the path.

Returns:

Returns TRUE if the path is a soft link; otherwise, returns FALSE.

4.9 *H5Uis_object*

Name/ Signature:

hbool_t H5Uis_object(hid_t loc, const char path);*

Purpose:

Checks for object.

Description:

The function checks if the given path points to an HDF5 object (a group, dataset, or named datatype).

Parameters:

hid_t loc	IN: A file or group identifier.
const char path*	IN: the path.

Returns:

Returns TRUE if the path points to an HDF5 object; otherwise, returns FALSE.

4.10 *H5Uhas_attributes*

Name/ Signature:

herr_t H5Uhas_attributes(hid_t loc, const char path)*

Purpose:

Checks for attributes.

Description:

For a given location and path, this function checks if there is any attribute attached the object for the given path.

Parameters:

hid_t loc	IN: A file or group identifier.
const char path*	IN: the path.

Returns:

Returns zero if no attribute, positive number for number of attributes, or negative value for failure. A failure can happen for different reasons, for example, the given path is invalid.

4.11 *H5Uis_group*

Name/ Signature:

herr_t H5Uis_group(hid_t loc, const char path)*

Purpose:

Checks for group.

Description:

Check if the given path points to a group.

Parameters:

hid_t loc	IN: A file or group identifier.
const char path*	IN: the path.

Returns:

Returns zero if not a group, positive number if it is a group, or negative value for failure. A failure can happen for different reasons, for example, the given path is invalid.

4.12 *H5Uis_dataset*

Name/ Signature:

herr_t H5Uis_dataset(hid_t loc, const char path)*

Purpose:

Checks for dataset.

Description:

Check if the given path points to a dataset.

Parameters:

hid_t loc	IN: A file or group identifier.
const char path*	IN: the path.

Returns:

Returns zero if not a dataset, positive number if it is a dataset, or negative value for failure. A failure can happen for different reasons, for example, the given path is invalid.

4.13 *H5Uis_datatype*

Name/ Signature:

herr_t H5Uis_datatype(hid_t loc, const char path)*

Purpose:

Checks for named datatype.

Description:

Check if the given path points to a named datatype.

Parameters:

hid_t loc	IN: A file or group identifier.
const char path*	IN: the path.

Returns:

Returns zero if not a named datatype, positive number if it is a named datatype, or negative value for failure. A failure can happen for different reasons, for example, the given path is invalid.

4.14 *H5Uis_dimension_scale*

Name/ Signature:

herr_t H5Uis_dimension_scale(hid_t loc, const char path)*

Purpose:

Checks for dimension scales.

Description:

Check if a given path points to a dataset of dimension scales. A dataset of dimension scales has an attribute of name “CLASS” and value "DIMENSION_SCALE". For details, visit the webpage at http://www.hdfgroup.org/HDF5/hdf5_hl/doc/index.html.

Parameters:

hid_t loc	IN:	A file or group identifier.
const char path*	IN:	the path.

Returns:

Returns zero if not dimension scales, positive number if dimension scales, or negative value for failure. A failure can happen for different reasons, for example, the given path is invalid.

4.15 *H5Uis_image*

Name/ Signature:

herr_t H5Uis_image(hid_t loc, const char path)*

Purpose:

Checks for image.

Description:

Check if a given path points to a dataset of an image. A dataset of image has an attribute of name "CLASS" and value "IMAGE". See details at http://www.hdfgroup.org/HDF5/hdf5_hl/doc/index.html.

Parameters:

hid_t loc	IN:	A file or group identifier.
const char path*	IN:	the path.

Returns:

Returns zero if not an image, positive number if it is an image, or negative value for failure. A failure can happen for different reasons, for example, the given path is invalid.

4.16 *H5Uis_table*

Name/ Signature:

herr_t H5Uis_table(hid_t loc, const char path)*

Purpose:

Checks for table.

Description:

HDF5 compound datasets can be interpreted as tables. This functions checks if the given path points to a table.

Parameters:

hid_t loc	IN:	A file or group identifier.
const char path*	IN:	the path.

Returns:

Returns zero if not table, positive number if it is a table, or negative value for failure. A failure can happen for different reasons, for example, the given path is invalid.

4.17 *H5Uis_table*

Name/ Signature:

herr_t H5Uis_table(hid_t loc, const char path)*

Purpose:

Checks for packet table.

Description:

<< add description here>>.

Parameters:

hid_t loc	IN: A file or group identifier.
const char path*	IN: the path.

Returns:

Returns zero if not packet table, positive number if it is a packet table, or negative value for failure. A failure can happen for different reasons, for example, the given path is invalid.

4.18 *H5Ucan_create*

Name/ Signature:

hbool_t H5Ucan_create(hid_t loc, const char path, hbool_t crt_intermediate_group)*

Purpose:

Verify for new path.

Description:

An HDF5 object can be created at a well-formed HDF5 path name relative to a location, if and only if no HDF5 object already exists there. If the creation of intermediate HDF5 groups is required, the crt_intermediate_group parameter will contain TRUE on return. For example, if "/A" is a existing dataset, H5Ucan_create() will return false if the path is "/A/B".

Parameters:

hid_t loc	IN: A file or group identifier.
const char path*	IN: the path.
hbool_t crt_intermediate_group	IN: Flag to indicate if intermediate is required.

Returns:

Returns TRUE if a new object can be created with the new path; otherwise, returns FALSE.

4.19 H5Uis_ Predefined _type

Name/ Signature:

hbool_t H5Uis_ Predefined _type(hid_t type)

Purpose:

Check for pre-defined datatypes.

Description:

The functions checks if the given datatype is one of the predefined datatypes as listed at http://www.hdfgroup.org/HDF5/doc/UG/UG_frame11Datatypes.html.

Parameters:

hid_t type	IN: the datatype identifier.

Returns:

Returns TRUE if the datatype is a pre-defined type; otherwise, returns FALSE.

4.20 H5Uis_atomic_type

Name/ Signature:

hbool_t H5Uis_atomic_type(hid_t type)

Purpose:

Check for atomic datatype.

Description:

Check if a given datatype is atomic. See details of HDF5 datatypes at http://www.hdfgroup.org/HDF5/doc/UG/UG_frame11Datatypes.html.

Parameters:

hid_t type	IN: the datatype identifier.

Returns:

Returns TRUE if the datatype is an atomic type; otherwise, returns FALSE.

4.21 H5Uis_string_type

Name/ Signature:

hbool_t H5Uis_string_type(hid_t type)

Purpose:

Check for string datatype.

Description:

Check if a given datatype is an atomic type. See details of HDF5 datatypes at http://www.hdfgroup.org/HDF5/doc/UG/UG_frame11Datatypes.html.

Parameters:

hid_t type	IN: the datatype identifier.

Returns:

Returns TRUE if the datatype is a string type; otherwise, returns FALSE.

4.22 H5Uis_array_type

Name/ Signature:

hbool_t H5Uis_array_type(hid_t type)

Purpose:

Check for array datatype.

Description:

Check if a given datatype is an array type. See details of HDF5 datatypes at http://www.hdfgroup.org/HDF5/doc/UG/UG_frame11Datatypes.html.

Parameters:

hid_t type	IN: the datatype identifier.

Returns:

Returns TRUE if the datatype is an array type; otherwise, returns FALSE.

4.23 H5Uis_compound_type

Name/ Signature:

hbool_t H5Uis_compound_type(hid_t type)

Purpose:

Check for compound datatype.

Description:

Check if a given datatype is a compound type. See details of HDF5 datatypes at http://www.hdfgroup.org/HDF5/doc/UG/UG_frame11Datatypes.html.

Parameters:

hid_t type	IN: the datatype identifier.

Returns:

Returns TRUE if the datatype is a compound type; otherwise, returns FALSE.

4.24 H5Uis_vlen_type

Name/ Signature:

hbool_t H5Uis_vlen_type(hid_t type)

Purpose:

Check for variable length datatype.

Description:

Check if a given datatype is a variable length type. See details of HDF5 datatypes at http://www.hdfgroup.org/HDF5/doc/UG/UG_frame11Datatypes.html.

Parameters:

hid_t type	IN: the datatype identifier.

Returns:

Returns TRUE if the datatype is a variable length type; otherwise, returns FALSE.

4.25 *H5Uhas_contiguous_layout*

Name/ Signature:

hbool_t H5Uhas_contiguous_layout(hid_t dset)

Purpose:

Check for contiguous layout.

Description:

Check if the given dataset has contiguous storage layout.

Parameters:

hid_t dset	IN: the dataset identifier.

Returns:

Returns TRUE if the dataset has contiguous storage layout; otherwise, returns FALSE.

4.26 *H5Uhas_chunked_layout*

Name/ Signature:

hbool_t H5Uhas_chunked_layout(hid_t dset)

Purpose:

Check for chunked layout.

Description:

Check if the given dataset has chunked storage layout.

Parameters:

hid_t dset	IN: the dataset identifier.

Returns:

Returns TRUE if the dataset has chunked storage layout; otherwise, returns FALSE.

4.27 *H5Uhas_compact_layout*

Name/ Signature:

hbool_t H5Uhas_compact_layout(hid_t dset)

Purpose:

Check for compact layout.

Description:

Check if the given dataset has compact storage layout.

Parameters:

hid_t dset	IN: the dataset identifier.

Returns:

Returns TRUE if the dataset has compact storage layout; otherwise, returns FALSE.

4.28 *H5Uis_simple_space*

Name/ Signature:

hbool_t H5Uis_simple_space(hid_t space)

Purpose:

Check for simple dataspace.

Description:

Check if the given dataspace is simple.

Parameters:

hid_t space	IN: the dataspace identifier.

Returns:

Returns TRUE if the dataspace is simple; otherwise, returns FALSE.

4.29 *H5Uis_scalar_space*

Name/ Signature:

hbool_t H5Uis_scalar_space(hid_t space)

Purpose:

Check for scalar dataspace.

Description:

Check if the given dataspace is scalar.

Parameters:

hid_t space	IN: the dataspace identifier.

Returns:

Returns TRUE if the dataspace is scalar; otherwise, returns FALSE.

4.30 *H5Uis_null_space*

Name/ Signature:

hbool_t H5Uis_null_space(hid_t space)

Purpose:

Check for null dataspace.

Description:

Check if the given dataspace is null.

Parameters:

hid_t space	IN: the dataspace identifier.

Returns:

Returns TRUE if the dataspace is null; otherwise, returns FALSE.

4.31 *H5Uis_obj_same*

Name/ Signature:

*hbool_t H5Uis_obj_same(hid_t loc_id1, const char *name1, hid_t loc_id2, const char *name2)*

Purpose:

Check if two given object IDs or link names point to the same object.

Description:

Check if two given object IDs or link names point to the same object.

Parameters:

hid_t loc_id1	IN: the location of the first object.
*char *name1*	IN: the link name of the first object. Use "." if loc_id1 is the object to be compared.
hid_t loc_id2	IN: the location of the second object.
*char *name2*	IN: the link name of the secondt object. Use "." if loc_id2 is the object to be compared.

Returns:

Returns TRUE if the two IDs or paths point to the same object; otherwise, returns FALSE.

4.32 *H5Uis_obj_same*

Name/ Signature:

*hbool_t H5Uis_obj_same(hid_t loc_id1, const char *name1, hid_t loc_id2, const char *name2)*

Purpose:

Check if two given object IDs or link names point to the same object.

Description:

Check if two given object IDs or link names point to the same object.

Parameters:

hid_t loc_id1	IN: the location of the first object.
*char *name1*	IN: the link name of the first object. Use "." if loc_id1 is the object to be compared.
hid_t loc_id2	IN: the location of the second object.
*char *name2*	IN: the link name of the secondt object. Use "." if loc_id2 is the object to be compared.

Returns:

Returns TRUE if the two IDs or paths point to the same object; otherwise, returns FALSE.

4.33 *H5Udetect_vlen*

Name/ Signature:

herr_t H5Udetect_vlen (hid_t type)

Purpose:

Check for any variable length type in a given type.

Description:

Recursively check if there is any variable length type in a given type.

Parameters:

hid_t type	IN: the datatype.

Returns:

Returns TRUE or FALSE if successful; otherwise, returns a negative value..

4.34 *H5Udetect_vlen_str*

Name/ Signature:

herr_t H5Udetect_vlen_str (hid_t type)

Purpose:

Check for any variable length string in a given type.

Description:

Recursively check if there is any variable length string in a given type.

Parameters:

hid_t type	IN: the datatype.

Returns:

Returns TRUE or FALSE if successful; otherwise, returns a negative value..

4.35 *H5Ufind_obj*

Name/ Signature:

*hsize_t H5Ufind_obj(hid_t loc, const char *name_pattern, char **matched_paths)*

Purpose:

Find all the objects that their name matches the given name pattern.

Description:

The name cannot contain "/". The matched paths will be returned through the array of strings, *matched_paths*. For example, *H5Ufind_obj(fid, "N2O*",matched_paths)* will return all the objects that their name starts with "N2O".

Parameters:

hid_t loc	IN: A file or group identifier.
*const char *name_patthern*	IN: the link name.
*Char **matched_paths*	OUT: the list of matched paths (array of strings)

Returns:

Returns the number of matched objects.

4.36 *H5Uhas_attribute_by_name*

Name/ Signature:

*herr_t H5Uhas_attribute_by_name(hid_t loc, const char *path, const char *attr_name)*

Purpose:

Check if there is any attributes that their name matches the given name.

Description:

This function checks for attributes by name. For example, *H5Uhas_attribute_by_name (fid "/dset", "IMAGE")* will check if the dataset, "/dset", has attribute of name "IMAGE".

Parameters:

hid_t loc	IN: A file or group identifier.
*const char *path*	IN: the object path.
*const char *attr_name_pattern*	OUT: the list of matched paths (array of strings)

Returns:

Returns zero if no attribute with the given name, positive number if attribute found, or negative value for failure. A failure can happen for different reasons, for example, the given path is invalid.

4.37 *H5Udetect_userblock*

Name/ Signature:

*hsize_t H5Udetect_userblock (const char *filename)*

Purpose:

Check for user block.

Description:

This function checks it there is any user block for a given file.

Parameters:

*const char *filename*	IN: the file name.

Returns:

Returns the size of user block, zero indicates there is no user block.

4.38 *H5Uremove_userblock*

Name/ Signature:

*herr_t H5Uremove_userblock (const char *h5_file, const char *ub_file)*

Purpose:

Removes the user block.

Description:

This function removes the user block from a file and saves the content of user block to a file.

Parameters:

*const char *h5_file*	IN: the file name of hdf5 file.
*Const char *ub_file*	IN: the file name of user block

Returns:

Returns the size of user block if successful; otherwise returns a negative value.

4.39 *H5Ureplace_userblock*

Name/ Signature:

*herr_t H5Ureplace_userblock (const char *h5_file, const char *ub_file)*

Purpose:

Replace the user block.

Description:

This function replaces the user block with the content from a given file.

Parameters:

*const char *h5_file*	IN: the file name of hdf5 file.
*Const char *ub_file*	IN: the file name of user block

Returns:

Returns the size of user block if successful; otherwise returns a negative value.

4.40 *H5Uread_data_in_bytes*

Name/ Signature:

*herr_t H5Uread_data_in_bytes(hid_t loc, const char *path, char *byte_buf, hsize_t buf_size)*

Purpose:

Read the value of a dataset to a byte buffer.

Description:

This function reads the value of a dataset into a byte buffer. If size is zero, nothing is read. Otherwise, up to buf_size bytes will be read into the byte buffer.

Parameters:

hid_t loc	IN: A file or group identifier.
*const char *path*	IN: the path of the dataset.
*Char *byte_buf*	OUT: the byte buffer containing the data values
hsize_t buf_size	IN: the size of the buffer

Returns:

Returns a positive value if successful; otherwise returns a negative value.

4.41 *H5Uread_attribute*

Name/ Signature:

*herr_t H5Uread_ attribute(hid_t loc, const char *path, const char *attr_name, void *attr_value)*

Purpose:

Read value(s) of a simple attribute.

Description:

This function reads the values of a simple attribute to a 1D array. The default buffer size is 1. The actual buffer size will be returned from the function call. Only the following types will be supported. The function will fail for other types.

H5T_NATIVE_INT8	*H5T_NATIVE_INT32*	*H5T_NATIVE_FLOAT*
H5T_NATIVE_UINT8	*H5T_NATIVE_UINT32*	*H5T_NATIVE_DOUBLE*
H5T_NATIVE_INT16	*H5T_NATIVE_INT64*	*H5T_C_S1*
H5T_NATIVE_UINT16	*H5T_NATIVE_UINT64*	*H5T_STD_REF_OBJ*

Parameters:

hid_t loc	IN: A file or group identifier.
*const char *path*	IN: the path of the object.
*const char *attr_name*	IN: the attribute name
*Void *attr_value*	OUT: the attribute value

Returns:

Returns the actual buffer size if successful; otherwise returns a negative value.

4.42 *H5Uwrite_attribute*

Name/ Signature:

*herr_t H5Uwrite_ attribute(hid_t loc, const char *path, const char *attr_name, void *attr_value, hid_t type)*

Purpose:

Write a simple attribute to an HDF5 object.

Description:

This function writes a simple attribute to the object of the given path. If the attribute does not exist, it creates the attribute with the given name and writes the value to the attribute. If the specified attribute already exists but does not have a datatype or dataspace consistent with attr_value, *H5Uwrite_simple_attribute* deletes the attribute and recreates it; otherwise, it updates the attribute value.

This function will only create an attribute of single point (scalar value) or 1D array. Only the following types will be supported. The function will fail for other types.

H5T_NATIVE_INT8	*H5T_NATIVE_INT32*	*H5T_NATIVE_FLOAT*
H5T_NATIVE_UINT8	*H5T_NATIVE_UINT32*	*H5T_NATIVE_DOUBLE*
H5T_NATIVE_INT16	*H5T_NATIVE_INT64*	*H5T_C_S1*
H5T_NATIVE_UINT16	*H5T_NATIVE_UINT64*	*H5T_STD_REF_OBJ*

For example,

- H5Uwrite_ attribute(fid, "/dset", "IMAGE_MINMAXRANG", (int[]){0, 255}, H5T_NATIVE_UINT8) – create an attribute of two values (0, 255).
- H5Uwrite_ attribute(fid, "/dset", "CLASS", "IMAGE", H5T_C_S1) creates an attribute of one string with the name "CLASS" and value "IMAGE"
- H5Uwrite_ attribute(fid, "/dset", "PALETTE", 251736, H5T_STD_REF_OBJ) creates an attribute of object reference.

Parameters:

hid_t loc	IN: A file or group identifier.
*const char *path*	IN: the path of the object.
*const char *attr_name*	IN: the attribute name
*Void *attr_value*	OUT: the attribute value
hid_t type	IN: the attribute type

Returns:

Returns a positive value if successful; otherwise returns a negative value.

4.43 *H5Uwrite_to_ddl*

Name/ Signature:

*herr_t H5Uwrite_ to_ddl(hid_t loc, const char *path, const char *ddl_name)*

Purpose:

Write a HDF5 file or object to a DDL text file.

Description:

This function writes a HDF5 file or object to text file. The text file will be in the format of DDL. For details of the HDF5 DDL format, visit the "*DLL in BNF for HDF5*" webpage at http://www.hdfgroup.org/HDF5/doc/ddl.html.

Parameters:

hid_t loc	IN: A file or group identifier.
*const char *path*	IN: the path of the object.
*const char *ddl_name*	IN: the name of the DDL file

Returns:

Returns a positive value if successful; otherwise returns a negative value.

4.44 *H5Ucreate_from_ddl*

Name/ Signature:

*herr_t H5Ucreate_ from_ddl(const char *ddl_name)*

Purpose:

Create a HDF5 file from a DDL text file.

Description:

This function creates a HDF5 file from a text file. The text file will be in the format of DDL. For details of the HDF5 DDL format, visit the "*DLL in BNF for HDF5*" webpage at http://www.hdfgroup.org/HDF5/doc/ddl.html.

Parameters:

*const char *ddl_name*	IN: the name of the DDL file

Returns:

Returns a positive value if successful; otherwise returns a negative value.

5 Development process and project plan

<< Need to add details after we indentify the complete list and staff resources>>

Revision History

September 7, 2009:	Version 1 circulated for comment within The HDF Group.
May 9, 2011:	Version 2 changed the proposed layout from version 1 and added a list of utility functions.