RFC: H5LTget_hardlinks – High-level API to list all the hard links to an object

M. Scot Breitenfeld

This RFC introduces a new high-level API for returning all the hard links to an object.

Introduction

When developers truly want to delete an object–and not just delete the link to it from a particular group–then they need a complete list of all the hard links to the object since an object is deleted from the file by removing all the hard links to it. The new high-level API *H5LTget_hardlinks* traverses the file and returns a list of hard links to the user. Once the list of hard links is obtained, the user can, for example, use the list to delete all the links to an object. Deleting an object is just one use case for the new function; this function might also prove useful in HDFView.

Approach

The new high-level routine will take a user-provided file or group identifier and a user-allocated string. The new API will then use *H5Oget_info_by_name* to get the hard link information and *H5Lvisit* to then find all the hard links by using an operator function. The operator function will get the address of the hard link and then compare it to the object's address. If the hard link matches the object then the hard link is added to the list being returned to the user.

New high-level API routine *H5LTget_hardlinks*

The signature of *H5LTget_hardlinks* is defined as follows:

```
herr_t H5LTget_hardlinks(hid_t loc_id, const char *obj_name, size_t *num_links,
                         size_t *buf_size_required, char *buf)
```

The parameters of *H5LTget_hardlinks* are defined as follows:

- `loc_id` is an identifier of the file or group (IN).
- `obj_name` is the name of object, relative to `loc_id` to query (IN).
- `num_links is the number of hard links to the object (OUT).`
- `buf_size_required is the required size for buf (OUT).`
- `buf is a character string composed of a list of null-terminated strings for each of the hard links (OUT).`

buf can be NULL, meaning that *num_links* and *buf_size_required* will be returned so that the user can allocate the correct string size. The return value of the function is a negative value if an error occurred.

Fortran 2003 high-level API routine *H5LTget_hardlinks_f*

The signature of *H5LTget_hardlinks_f* is defined as follows:

```
SUBROUTINE H5LTget_hardlinks_f (loc_id, obj_name, error, buf, num_links,
max_string_size)
```

The parameters of *H5LTget_hardlinks_f* are defined as follows:

- `INTEGER, INTENT(IN) :: loc_id` ! an identifier of the file or group.
- `CHARACTER(len=*), INTENT(IN) :: obj_name !` the name of object, relative to `loc_id` to query.
- `INTEGER, INTENT(OUT)` :: error ! Error code: 0 on success and -1 on failure
- `CHARACTER(len=*), DIMENSION(*), INTENT(OUT), OPTIONAL:: buf !` list of all the hard links to the object.
- `INTEGER(size_t), INTENT(INOUT), OPTIONAL` :: `num_links ! Number of hard links`
- `INTEGER(size_t), INTENT(INOUT), OPTIONAL` :: `max_string_size`

The *max_string_size* is the minimum size needed in the character length parameter for *buf*. *Max_string_size* returns the maximum string length found in the hard links list, and includes in the length the space used for the C null character. This is because the C API returns a list of null character terminated strings and Fortran is passing the Fortran character buffer to the C API to fill directly (i.e. we are not creating a separate string buffer in C and then copying the C buffer to the Fortran buffer). Therefore, we need the Fortran buffer size to also include the C null characters. Note, the returned character buffer will not include the C null character in the Fortran strings. It is important to clarify the difference between the C and Fortran APIs; the C API is returning a character string containing a list of null character terminated strings, whereas the Fortran API is returning an array of character strings. This difference is mainly due to differences in programming preferences/conventions between Fortran and C.

Usage

Figure : Example of HDF5 file structure with groups and datasets [1].

The following example uses Figure 1 as the file's data structure. If a user wishes to delete the */group2/group2* object:

C -

```
/* Find the needed size of the buffer and the number of links */   H5LTget_hardlinks(file_id,
"/group2/group2", &num_links, &buf_size_required, NULL)


/* Allocate the string buffer */
hardlinks = HDmalloc(buf_size_required*sizeof(char));


/* Get all the hardlinks to the object */
H5LTget_hardlinks(file_id, "/group2/group2", &num_links, &buf_size_required, hardlinks)


/* Delete the hard links */
js = 0;
for(j = 0; j < buf_size_required ; j++) {
     if(hardlinks[j] == '\0') {
       if( H5Ldelete(file_id, &hardlinks[js], H5P_DEFAULT) < 0 )
       goto out;
       js = j + 1;
     }
   }
```

Fortran –

```
! Find the needed size of the buffer and the number of links
CALL h5ltget_hardlinks_f(file_id, "/group2/group2", errcode, num_links=num_links, &
                     max_string_size=max_string_size)

! Allocate the string array buffer
ALLOCATE(hardlinks(1:num_links))
ALLOCATE(CHARACTER(len=max_string_size) :: hardlinks)
```

```
! Get all the hardlinks to the object
CALL h5ltget_hardlinks_f(file_id, "/group2/group2", errcode, hardlinks)
```

Recommendation

Outlined is the high-level API routine *H5LTget_hardlinks*, which will help to insulate the user from the burdensome and possibly error prone chore of deleting an object. The API gives the user the necessary information to successfully delete an object.

Revision History

February 12, 2015:	Version 1 circulated for internal comment to interested parties within The HDF Group.
February 20, 2015	Version 2 circulated for internal comment within The HDF Group.

References

[1] The HDF5 User's Guide