Creating Missing Groups

Peter Cao
May 5, 2005

1. What it is

A missing group is a group that does not exist along an object (group, dataset or named datatype) path. For example, "/A/B/dset1" is a path to dataset "dset1", where group "B" does not exist in the file. Group "B" is a missing group. The current behavior of H5Dcreat(), H5Tcomit() and H5Gcreate() is like Unix. When there is a missing group, the create() call will fail. This is a normal behavior, not a bug. However, it will be very useful if the create() functions create the missing groups instead of failure

2. Use cases

Use case #1: missing group in an absolute path
H5Dcreate(file_id, "/A/B/dset1", ..), where group "B" is missing. H5Dcreate() should create group "B", and then create "dset1".

Use case #2: multiple missing groups along the path
H5Dcreate(file_id, "/A/B/C/dset1", ..), where groups "B" and "C" are missing. H5Dcreate() should create group "B" and "C", and then create "dset1".

Use case #3: missing group in relative path
H5Gcreate(loc_id, "A/B/C/g", ..), where groups "B" and "C" are missing, loc_id can be file id or any group id. H5Gcreate() should create group "B" and "C" and then create "g".

3. Approaches

Here I present two tentative ways to deal with this problem. Both have pros and cons. We will need more inputs to decide what is the best.

1) Change the current behavior of the H5Dcreate() and H5Gcreate() so that it will create missing groups instead of failure.
   Pros: no change to users' code.
   Cons: at some cases, failure is a normal behavior for missing groups.
2) Add new APIs such as H5Dcreate_expand() and H5Gcreate_expand(). The new APIs will create missing groups, while the old APIs stay the same.
   Pros: will not affect users who do not want the new feature
   Cons: users have to change their code to use the new API for missing groups.

4. Function prototype

hid_t H5Dcreate_expand(*hid_t loc_id, const char* name, size_t size_hint, hid_t create_id, hid_t access_id*)
hid_t H5Gcreate_expand(*hid_t loc_id, const char* name, size_t size_hint, hid_t create_id, hid_t access_id*)

herr_t H5Tcommit(*hid_t loc_id, const char* name, hid_t type, hid_t create_id, hid_t access_id*)

API to set/get property:
H5Pset_create_missing_group(*hid_t create_id, hid_t missing_gcpl*)
H5Pgset_create_missing_group(*hid_t create_id, hid_t missing_gcpl*)

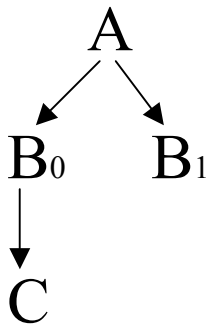
By setting the property list, users will be able to choose to have the new feature or stay with the old way. The property list can also be used for other purposes. For example, it can be used to indicate the character type of group/dataset names (ASCII or UTF-8), option to create index table when group/dataset is created.

5. Other concerns

What should we do for open() calls if there is a missing group in the path.
What should we do for soft links if there is a missing group in the soft link.

6. Extension

One useful extension of this work is to create a whole hierarchy with H5Gcreate(). For example, groups "A", "B_0", "B_1", and "C" are organized as the follow



H5Gcreate(… "[A [B0 [C] , B1]]", …) will create the whole tree shown above.

7. Quick solution

Giving the complexity and many variations of the problem, we may have to wait for a long time (six month or more) for the perfect solution. For immediate need, I propose the following quick solution.

1) Design the prototype for the future, such as H5Gcreate_expand(hid_t loc_id, const char* name, …., hid_t plist). Although we will not use the plist for now, we will reserve it for the future use

2) Implement the function that solves the problem quickly.

For example (pseudo code),

H5G_create_expand(*hid_t loc_id, const char* name, size_t size_hint, hid_t create_id, hid_t access_id*)

```
{
    hid_t gid;
    char *name1 = 1st_group_in_name;
    if (H5G_find(loc_id, name1, NULL, NULL)<0)
            gid = H5G_create(loc_id, name1) /* the current H5Gcreate() */
    else
            gid = H5G_open(loc_id, name1);

    H5G_create_expand(gid, name1, …., plist);
    H5G_close(gid);
}
```

8.Build test cases

Group:

Test case #1: single missing group
H5Gcreate_expand(file_id, "/A/B/grp", …), where B is missing
Test case #2: multiple missing group
H5Gcreate_expand(file_id, "/A/B/C/grp", …), where B and C are missing
Test case #3: relative path
H5Gcreate_expand(group_id, "A/B/grp/", …), where A and B are missing
Test case #4: absolute path with group_id
H5Gcreate_expand(group_id, "/A/B/grp/", …), where A and B are missing
Test case #5: loop
H5Gcreate_expand(file_id, "/A/B/C/A", …), where B and C are missing

Dataset:

Test case #6: single missing group
H5Dcreate_expand(file_id, "/A/B/grp", …), where B is missing
Test case #7: multiple missing group
H5Dcreate_expand(file_id, "/A/B/C/grp", …), where B and C are missing
Test case #8: relative path
H5Dcreate_expand(group_id, "A/B/grp/", …), where A and B are missing
Test case #9: absolute path with group_id
H5Dcreate_expand(group_id, "/A/B/grp/", …), where A and B are missing
Test case #10: loop
H5Dcreate_expand(file_id, "/A/B/C/A", …), where B and C are missing

Datatype:

Test case #6: single missing group
H5Tcommit_expand(file_id, "/A/B/grp", …), where B is missing
Test case #7: multiple missing group

H5Tcommit _expand(file_id, “/A/B/C/grp”, …), where B and C are missing

Test case #8: relative path

H5Tcommit _expand(group_id, “A/B/grp/”, …), where A and B are missing

Test case #9: absolute path with group_id

H5Tcommit _expand(group_id, “/A/B/grp/”, …), where A and B are missing

Test case #10: loop

H5Tcommit _expand(file_id, “/A/B/C/A”, …), where B and C are missing