Character Encoding for Links in HDF5 Files

James Laird, Quincey Koziol, Elena Pourmal 9/30/05

We want to support UTF-8 to support European HDF5/NetCDF users. Currently, we know (by running tests) that HDF5 can handle UTF-8 strings as data and as object (group, dataset, etc.) names. There is a field in string datatypes to record which character encoding (ASCII or UTF-8) the string uses, but there is currently no way to tell whether an object name is in ASCII or UTF-8.

If a user attempts to display a UTF-8 string as ASCII or vice versa, they will usually notice no difference; as mentioned in the Unicode RFC, the first 128 characters of UTF-8 and ASCII are identical. However, "extended" ASCII values (some special characters like the degree sign) and non-English UTF-8 values will be displayed as garbage characters by the wrong encoding. Even when using the wrong encoding, users would be able to access these objects by entering their "garbage" names. Likewise, strings will be sorted by byte values, and will always be sorted in a consistent order no matter which encoding is used (or even if both are used together). We know that there are European NetCDF users, and at least one user uses extended ASCII object names.

The names of objects in an HDF5 file aren't stored on the object itself, but on the links that point to the object. Thus, it does not make straightforward sense to make character encoding a dataset (or datatype, or group) creation property, since it is not really a property of the group. Moreover, a single object can have multiple links with different names and (potentially) different character encodings.

As part of the group revisions currently in progress, links will have a field to store the character encoding of their names. However, links can be created and renamed by a number of HDF5 API calls (H5D/Gcreate, H5Tcommit, H5Gmove, and H5Grename). Thus, changing the API to allow this field to be set is not trivial.

Proposed methods for supporting the UTF-8 encoding include:

1) Do Nothing

As mentioned, HDF5 can accept both ASCII and UTF-8 strings. The character encoding field of links could be left blank, and users could be responsible for keeping track of which encoding the names use.

We can assume that users will usually know which encoding their own files use, but this approach may result in files received from other users being in an unknown format. As mentioned, this can result in object names displayed as "garbage" in the worst case, but will not result in users being unable to access data.

Doing nothing to address the problem may make NetCDF users (and developers!) unhappy, although the developers may well have bigger things to worry about.

More importantly, this violates HDF5's approach of being self-describing, since both UTF-8 and ASCII will be officially supported but there will be no way to tell which is being used for a given string.

This approach is the simplest and easiest, obviously; it requires no developer effort and no API change.

2) Put link encoding in Object Creation Property List and add Link Creation Property List for H5G functions

When creating an object--e.g., a Dataset--the user would specify the link encoding as a property in the Dataset Creation Property List.

When using H5Gmove or H5Grename, the user would supply a new "Link Creation" Property List that would contain this parameter.

This approach puts link-specific information in the Object Creation Property Lists, which isn't really appropriate. Unlike other information in the OCPLs, the link encoding will not be permanent (since the object could be re-linked later). It also puts one property in two different kinds of property lists (the Object Creation Property Lists, and the new link Creation Property Lists which would be used for H5Gmove and H5Grename).

This approach solves the problem with a relatively small amount of API change. It maintains the idea of a single API call creating both an object and a link, and of all necessary information being passed as part of the Creation Property List.

3) Put add Link Creation Property List for object creation and H5G functions

Like approach #2, we could create a new Link Creation Property List for H5Gmove and H5Grename which would specify the character encoding. We could require this property list to be passed in to object creation calls. Thus, creating a dataset would require a DCPL, a DAPL, and a LCPL.

This method keeps link information out of object creation property lists. However, it requires a more significant API change than approach #2.

4) Create H5L interface to handle link creation

The previous two approaches try to give link creation information to H5Dcreate, H5Gcreate, and H5Tcommit, since links now have parameters that need to be set. This approach instead extracts link creation into a separate step, with its own Link Creation Property List. Link creation (and some H5G functions that really affect links, not groups) would be included in a new "Link" interface, H5L. Thus, to create a new dataset, a user would:

dataset_id = H5Dcreate(dataspace, datatype, DCPL, DAPL, DXPL)
H5Lcreate("dataset name", dataset_id, LCPL)

The H5L interface would include link creation (currently H5G_insert), link moving/renaming (currently H5Gmove and H5Grename), and link deletion. It would also have functions for querying information on a link (its character encoding, whether it is a hard/soft/external link, etc.). H5Gobjinfo may be changed to return information on the object but not the link to the object.

The advantage to this approach is that it separates the steps of creating objects and creating links, which corresponds more accurately to what is actually happening in the library. This approach also provides a framework for future work on other features that affect links--access permissions, 16-bit character encodings, attributes on links, etc. Conceptually separating links from the objects they point to may help us to communicate our data model to users (the file is a graph, not a tree structure).

This requires as much API change as #3. It also treats links as separate objects, which may be too complicated for some users.

5) Set encoding for the entire file

Alternately, the character encoding could be set globally, either as part of the File Creation Property List (and stored in the superblock) or as an attribute convention. All links in a given file would have the same encoding.

This approach is unattractive because it requires a global setting, and global settings are a Bad Idea in principle (as a lesson learned from HDF4).

This would make it very difficult to "mix" encodings--copying data between two files with different encodings, or adding a group with a non-English name to an ASCII file.

However, this would require very little (or no) API change and would be relatively simple to implement.

6) Set default encoding for the entire file but allow exceptions

As above, set a "default" encoding for a file in File Creation Property List. This would affect the encoding of links created with a default property list (H5P_DEFAULT). Each link's encoding would still be stored with the link, so users could set the encodings of links individually (using approaches #2, #3, or #4 above).

This solves the problem with approach #5 of mixing files with different encodings. It would mean fewer API calls for UTF-8 users and could help applications to add objects with names that matched the encoding of other names in the file.

Setting a file-wide "default" could result in users being uncertain about which encoding a given file was using and having to set the character encoding for each object manually. Users who were entirely unaware of this feature might not understand why names could be written properly in one file but not another.

This would require no API change whatsoever, but would require a change to the file format. It would be paired with an approach #2-#4 above.

Conclusion

Any approach that requires an API change would also support the old functions for compatibility. No matter which approach is used, calling H5Dcreate with default properties would create a link with default encoding as it currently does. Calling H5Gmove with no LCPL supplied would still move the link with default properties.

The consensus of our group was that approach #6, that of having a file-wide default, was a good idea.

There was support for pairing Approach #6 with either Approach #2 (adding character encoding to the Object Creation Property Lists) or Approach #4 (creating a new H5L interface for links). Approach #2 was seen as simpler and requiring a smaller conceptual shift than Approach #4, which treats links as objects within an HDF5 file. Approach #4 was seen as being more consistent with how HDF5 represents links internally and as being more flexible for future work.