Shared Strings in Virtual Dataset Layout Encoding in HDF5

Neil Fortner

The HDF Group

This document proposes a method to improve the efficiency of storing virtual dataset mappings when a source dataset or source file name is repeated, both in the in-memory representation and as encoded in the file.

1 Introduction

The virtual dataset (VDS) feature in HDF5 is used to construct a dataset from any number of source datasets or parts thereof. This is accomplished by storing a list of VDS mappings, where each mapping contains the source dataset name, the file name the source dataset resides in, the selection of elements in the source dataset, and the selection of elements in the VDS. The VDS stores no actual data, only the mappings, and the data in the VDS is taken from the mapped part(s) of the source dataset(s).

For some use cases, the source file name and/or the source dataset name may be the same for multiple mappings. We can therefore save some space in memory and the file by, instead of storing many copies of the same name, storing the name once and sharing it with the other mappings that use it. For use cases involving many VDS mappings the savings will be substantial. This should also improve the speed of decoding and copying the layout message, since we will not need to allocate and copy the same string many times. Here we propose a method to accomplish this.

2 On Disk Format

2.1 Current Implementation

The current on disk format for virtual dataset mappings as stored in the global heap is:

byte	byte	byte	byte
Version			
Num Entries[L]			
Source Filename 0	*(variable size)*		
Source Dataset 0	*(variable size)*		
Source Selection 0	*(variable size)*		
Virtual Selection 0	*(variable size)*		
...			
Source Filename N-1	*(variable size)*		
Source Dataset N-1	*(variable size)*		
Source Selection N-1	*(variable size)*		
Virtual Selection N-1	*(variable size)*		
Checksum			

* Items marked with an 'L' in the above table are of the size specified in "Size of Lengths" field in the superblock.

The fields in the above layout are described here:

Field Name	Description
Version	The version number for the block; the value is `0`.
Num Entries[L]	The number of entries in the block.
Source Filename #n *(variable size)*	The source file name where the source dataset is located. Stored as a `NULL` terminated string.
Source Dataset #n *(variable size)*	The source dataset name that is mapped to the virtual dataset. Stored as a `NULL` terminated string.
Source Selection #n *(variable size)*	The dataspace selection in the source dataset that is mapped to the virtual selection.
Virtual Selection #n *(variable size)*	The dataspace selection in the source dataset that is mapped to the virtual selection.
Checksum	This is the checksum for the block.

2.2 Proposed Improvement

We plan to add a new field, flags, which will contain bit flags that indicate if the source file and/or dataset name is shared, and also if the source file is the same as the virtual file. We will also extend the meaning of the Source Filename and Source Dataset fields. We will add a new version number for this, and only upgrade to this version if the lower bound for the HDF5 library version bounds is set to 2.0 or later. The proposed format is therefore:

byte	byte	byte	byte
Version			
Num Entries[L]			

Flags 0	
Source Filename 0	*(variable size)*
Source Dataset 0	*(variable size)*
Source Selection 0	*(variable size)*
Virtual Selection 0	*(variable size)*
...	
Flags N-1	
Source Filename N-1	*(variable size)*
Source Dataset N-1	*(variable size)*
Source Selection N-1	*(variable size)*
Virtual Selection N-1	*(variable size)*
Checksum	

* Items marked with an 'L' in the above table are of the size specified in "Size of Lengths" field in the superblock.

The fields in the above layout are described here:

Field Name	Description
Version	The version number for the block; the value is `1`.
Num Entries^{L}	The number of entries in the block.
Flags #n	This field is a bit field containing additional information about the mapping. Bit: Description `0`: If set, the source filename is shared. `1`: If set, the source dataset name is shared `2`: If set, the source file is the same as the virtual file
Source Filename #n *(variable size)*	The source file name where the source dataset is located. If the source file name is shared, this is a "Size of Lengths" length unsigned integer containing the index into this array of mappings where the source filename is stored. For example, if the value stored here is `1`, then the actual string value for this field is stored in the Source Filename 1 field. If the source file name is not shared, this is stored as a `NULL` terminated string. If the source file is the same as the virtual file, this field is not present.
Source Dataset #n *(variable size)*	The source dataset name that is mapped to the virtual dataset. If the source file name is shared, this is a "Size of Lengths" length unsigned integer containing the index into this array of mappings where the source filename is stored. For example, if the value stored here is `1`, then the actual string value for this field is stored in the Source Dataset 1 field. If the source file name is not shared, this is stored as a `NULL` terminated string.
Source Selection #n *(variable size)*	The dataspace selection in the source dataset that is mapped to the virtual selection.

Virtual Selection #n *(variable size)*	The dataspace selection in the source dataset that is mapped to the virtual selection.
Checksum	This is the checksum for the block.

3 In Memory Format

3.1 Current Implementation

Currently the in-memory format for the the virtual dataset layout includes, within the `H5O_storage_virtual_t` struct stored directly in the `H5O_layout_t` struct, a simple array of `H5O_storage_virtual_ent_t` structs. Each of these structs represents a single mapping, with each including a simple, non-reference counted, `char *` for the source file name and another for the source dataset name. Each of these `char *`s points to a separately allocated buffer, and when the layout is copied, each must be duplicated (allocated and copied) individually.

3.2 Proposed Improvement

Since the implementation of the in-memory storage format for virtual dataset layouts does not affect the public API or file format it can be changed at will, and therefore is not as important to design ahead of time. Therefore we will only give a brief overview of the current design.

We plan to add two hash tables or skip lists to the `H5O_storage_virtual_t` struct, one for source file names and one for source dataset names. The source file and dataset names will be used as the indices into the data structures, and the record stored for each will be the index into the array of `H5O_storage_virtual_ent_t` structs where that name was first encountered (the lowest index into that array where that name occurs). When adding a new mapping with `H5Pset_virtual()`, we will check for the existence of the source file and dataset names in their respective data structures, and if not found, insert it with the current mapping index. If the name is found, then we will avoid copying the string into the newly created `H5O_storage_virtual_ent_t` struct, and simply copy the pointer value from the mapping located via the data structure. We will also store that index in the `H5O_storage_virtual_ent_t` struct, while making sure that it is unambiguously indicated that the string is shared. For further optimization, we may also give special status to the **“.”** string used to denote the source dataset is in the same file as the virtual dataset, possibly allowing the source file’s `char *` to be `NULL` in that case.

Since the data structure described above is only needed when adding new mappings, we will avoid creating it when copying or decoding the layout message, and instead only create and populate the data structure if a mapping is added later. We believe it is not necessary to fully reference count the strings, since there should be no situation where one VDS mapping is freed without all of the other VDS mappings being freed at the same time. We will need to make sure we don’t attempt to access the values of the shared strings while freeing the list of VDS mappings, but there should be no reason to do that anyways.

We believe this in-memory format will provide a natural mapping to the on-disk format, allowing for efficient encoding and decoding, while also being performant when performing operations purely in memory.