RFC: Freeing Memory Allocated by the HDF5 Library

Dana Robinson

Several functions in the HDF5 C API return buffers allocated by the HDF5 library. When application code uses a different library for memory management than the HDF library, a corrupt heap or resource leaks can occur when these allocated buffers are freed. This is most commonly a problem on Windows since Microsoft implements C library functions in Visual Studio-specific libraries, which do not share heap state.

This RFC describes this problem and steps the user can take to mitigate the problem. It also introduces the new `H5free_memory()` function, which will appear in HDF5 1.8.13.

1 Introduction

In the HDF5 library, responsibility for the allocation and freeing of memory is usually the responsibility of the same component; either the library or the user's code. When data that would normally be stored in dynamically-allocated memory must be returned from the library, the user is usually asked to allocate a buffer, which is passed to the function and then filled by the library. The complication is that the user must be able to determine the buffer's size. The mechanism for this is for the user to make a preliminary call, passing a NULL pointer in for the buffer. The function will then return the appropriate number of bytes for the user to allocate.

Example:

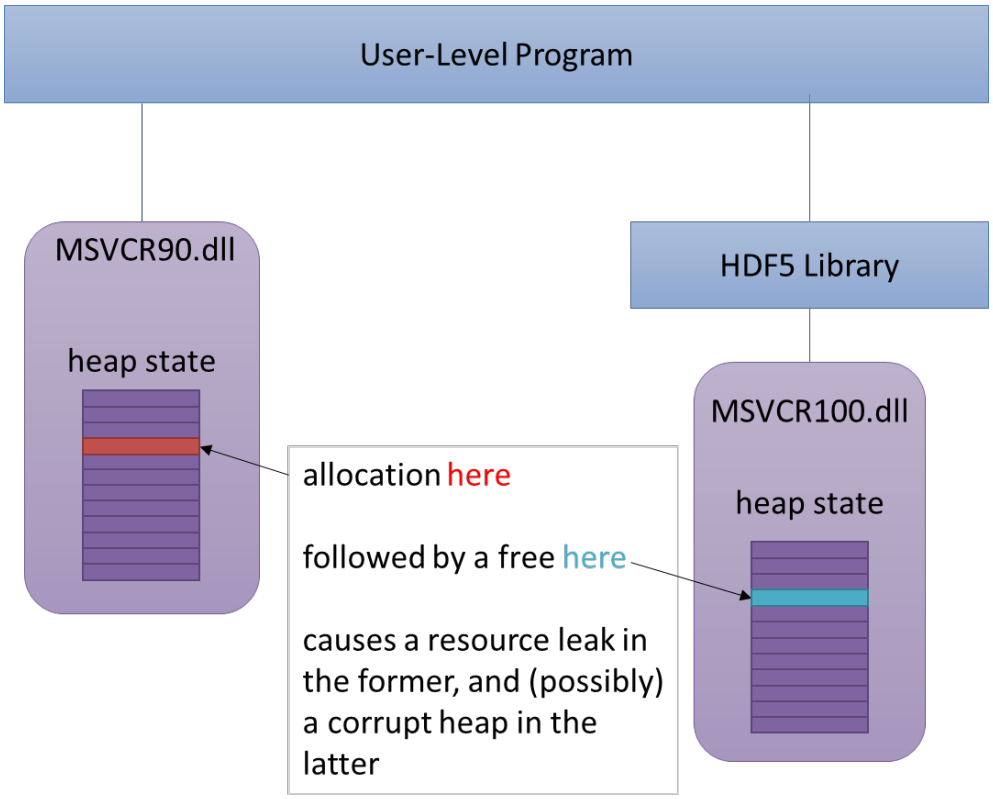
```
ssize_t    size;
size_t     bufsize;
hid_t      object_id;
char       *comment;
     …
size = H5Oget_comment(object_id, NULL, &bufsize);      /* determine size */
bufsize = size;
comment = (char *)malloc(bufsize * sizeof(char));
size = H5Oget_comment(object_id, comment, &bufsize);   /* fill buffer */
```

There are, however, several API calls in which the buffer is allocated by the HDF5 library and returned to the user, who is responsible for freeing it. This can be a problem when memory is managed via different libraries as it can result in resource leaks or a corrupted heap. This heap corruption can

result in subtle bugs that can be very difficult to reproduce and diagnose. In most cases, having the library allocate memory and the application free it is not a problem since memory operations will resolve down to the operating system's memory manager, however there are cases where this is not true. For example, a debug memory manager may be in use by the application code but not the library. A complication that is unique to Windows is that the C standard library functions are implemented in Visual-Studio-specific shared libraries. When different versions of Visual Studio are used to compile the library and user code, the allocate and free calls are made in different libraries, which do not share state, leading to the previously mentioned resource and corruption issues.

2 The Windows C Run-Time (CRT)

Microsoft implements the standard C library functions in debug and release libraries that are specific to each version of Visual Studio[1]. Each library is a separate entity and maintains its own internal CRT object state, including file handles and heap information. Creating an object in one CRT and destroying it in another CRT may appear to work but can cause corruption of one CRT and resource leaks in the other.



These problems are normally avoided on Windows by ensuring that all components that can return CRT resources are linked to the same CRT dll. Unfortunately, even debug and release CRTs are housed

[1] The names of these libraries are of the form MSVCR<#>.dll, where <#> is the Visual Studio version. e.g., MSVCR110.dll corresponds to Visual Studio 11.0 (2012).

in separate dlls, so this is not an easy solution to implement. Using static linkage does not avoid this problem since separate copies of the CRT are created in each statically linked component.

3 Potentially Affected API Calls

This is a list of all API calls that are potentially affected. Note that these have not all been validated as having allocate/free issues. Most will require reference manual annotations about resource disposal/management, however.

3.1 Functions that return a pointer

- `H5Iobject_verify`
- `H5Iremove_verify`
- `H5Isearch`
- `H5Pget_class_name`
- `H5Pget_driver_issue`
- `H5Tget_member_name`
- `H5Tget_tag`

3.2 Functions that have a ** OUT parameter

- `H5Eget_auto (both 1 & 2)`
- `H5Fget_vfd_handle`
- `H5Lunpack_elink_val`
- `H5Pget_buffer`
- `H5Pget_elink_cb`
- `H5Pget_fapl_multi`
- `H5Pget_file_image`
- `H5Pget_mdct_search_cb`
- `H5Pget_type_conv_cb`
- `H5Pget_vlen_mem_manager`

3.3 Other

- `H5Fget_mdc_config (trace_file_name struct member)`
- `H5Pget_mdc_config (trace_file_name struct member)`

4 Mitigation

There are several potential solutions to the problem of freeing memory allocated by the HDF5 library.

4.1 Use the Same Memory Manager/Correct C Run-Time Everywhere

Both application code and the HDF5 library must use the same memory allocator. When using Visual Studio, both the Visual Studio version and release/debug state must be identical. As of HDF5 1.8.12, this is the only available solution.

4.2 Change Function Semantics

Optionally, the small number of HDF5 library API calls that return memory to the user could be modified to use the preliminary call scheme. Although this is not an option for HDF5 1.8 due to our policies concerning API stability, we could rework these functions for HDF5 1.10, perhaps as part of a larger API overhaul that modifies all functions to uniformly return a *herr_t* type, remove the non-ANSI-C *ssize_t* type, etc.

4.3 Create an H5free_memory() Function

A new function could be created that is essentially a thin wrapper for the run-time's `free()` call. This function would be used to free any memory allocated by the library. This solution has the advantages of being extremely easy to implement and intuitive to use. It can also be used as a solution with legacy API calls, so it would be necessary even if we modify the HDF5 API. This function will also be extremely useful when HDF5 is wrapped for use with managed languages (Java, .NET, Python, etc.) so the wrappers can properly clean up resources.

A tentative reference manual page for this function is available as an appendix to this RFC.

Note that the creation of this function does not imply that it will be acceptable for new API calls to be created that return library-allocated memory. The preferred mechanism will still be to use the "preliminary call" scheme where the user allocates the buffer.

5 Recommendation

The easiest, least painful solution is to add the H5free_memory() function to the API, starting with HDF5 1.8.13 (release date: May 2014). This solution has been requested by our Windows users for a long time and is the most straightforward solution to the "multiple CRT dlls" problem.

It might also be worth considering reworking the function calls that return data via a library-allocated buffer for the upcoming HDF5 1.10.0 release.

Additionally, the reference manual should be updated to indicate where the user is responsible for freeing returned buffers. This should be done in all instances where an API call returns a pointer or has a ** OUT parameter.

Acknowledgements

This work is being internally funded by The HDF Group.

Revision History

March 18, 2014:	Version 1 circulated for comment within The HDF Group.
March 24, 2014:	Version 2 incorporates the group's changes. Circulated for comment on the forum.

References

1. The HDF Group. "HDF5 Documentation", http://www.hdfgroup.org/HDF5/doc/doc-info.html (March 18, 2014).
2. Microsoft MDSN. "Potential Errors Passing CRT Objects Across DLL Boundaries (Visual Studio 2012)", http://msdn.microsoft.com/en-us/library/ms235460%28v=vs.110%29.aspx (March 18, 2014).
3. Microsoft MDSN. "C Run-Time Libraries (Visual Studio 2012)", http://msdn.microsoft.com/en-us/library/abx4dbyh%28v=vs.110%29.aspx (March 18, 2014).
4. Microsoft MDSN. "How to link with the correct C Run-Time (CRT) library", http://support.microsoft.com/kb/140584 (March 18, 2014).

Appendix: Proposed H5free_memory() Reference Manual Page

Name: H5free_memory

Signature:

```
herr_t H5free_memory(void *buf)
```

Purpose:

Frees memory allocated by the HDF5 library.

Description:

In order to avoid heap corruption, allocated memory should be freed using the same library that initially allocated it. In most cases, the HDF5 API uses resources that are allocated and freed entirely by either the user or the library so this is not a problem. In some cases, however, HDF5 API calls will allocate memory that the user must free (*e.g.*, H5Tget_member_name). This function allows the user to safely free this memory.

Note:

It is especially important to use this function to free memory allocated by the library on Windows. The C standard library is implemented in dynamic link libraries (dlls) known as the C run-time (CRT). Each version of Visual Studio comes with two CRT dlls (debug and release) and allocating and freeing across dll boundaries can cause resource leaks and subtle bugs due to heap corruption.

Only use this function to free memory allocated by the library. It will generally not be safe to use this function to free memory allocated by any other means.

Even when using this function, it is still best to ensure that all components of a C application are built with the same version of Visual Studio and build (debug or release) and thus linked against the same CRT.

Parameters:

`void * mem`	IN: Buffer to be freed. Can be NULL.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.