RFC: Refactor h5dump to Improve Maintenance

Allen Byrne

The code base for h5dump has become bloated and unmanageable because of the duplication of functions and variables. This document is an overview of the current problems with the h5dump program and the tools library, followed by a proposal to refactor the existing h5dump and h5tools library code.

Introduction

The amount of duplicate functions and variables in the h5dump program versus the tools library, and the single program file packed with every function created, hinders the improvement of the h5dump program. These issues make the addition of new features very expensive in time and testing. The tools team has identified five areas of improvement to reduce the amount of duplicate functions and variables in the h5dump program versus the tools library. The first three issues can be handled immediately and the final two will need resources assigned. The use of bold names in this document is to help with cross-referencing of the names in the source code.

First, the h5dump source file should separate the existing functions into three files: ddl, xml, and program control functionality files. This will make it easier to focus on removing duplicate functions and variables.

Second, each main program function should create an independent context structure and initialize it from the global **indent** variable. This will improve the control over the output interaction with the **dump_function_table** functions by aligning the **context->indent** variable and the global **dump_indent** variable.

Third, remove functions in the h5dump code that already exist in the h5tools library. This will allow other functions to be moved into the h5tools library and simplify the h5dump code program to concentrate on function control flow. The tools library will concentrate on formatting the output.

Fourth, create and/or update the low-level **h5tools_str** functions into building block functions. This will allow more general dump content functionality to be built into the **h5tools_dump** functions, which can concentrate on how to display the content.

Finally, before making any changes, add unit testing of the low-level library code and integration testing of the library dump functions. This testing will help keep future library changes from breaking the h5dump design.

Motivation

It is necessary to understand why working on a feature or problem in h5dump has become a tedious

undertaking. While the tools library has moved forward with improvements, the h5dump program layer has remained a jumble of code with an inconsistent and almost indiscernible design. Some routines handle indentation manually while other routines use interfaces that align with the tools library, sometimes both in one function. Also the XML processing uses similar functions with completely different control and seems to be a different program. A function to display a datatype illustrates this problem; the **dump_datatype** function in the h5dump source file has a duplicated function in the tools library, **h5tools_dump_datatype**. Each calls one function; **dump_datatype** calls **print_datatype** in the h5dump source file and **h5tools_dump_datatype** calls **h5tools_print_datatype** in the tools library. Both print functions are used by other functions; however **h5tools_dump_datatype** is never referenced elsewhere, while **dump_datatype** is used by the **dump_function_table** callback function. Fixing a bug in printing a datatype requires knowing which program flow path produces the problem!

The following graphic is the current interaction between the h5dump program and the tools library. Given an hdf5 object (from an hdf5 file) the h5dump program processes it using the two control structures: format_info (**h5tool_format_t**) and context (**h5tools_context_t**).

The format structure, **h5tool_format_t**, members are strings that define the printf() format string used to print the variables and the element markup. These are set to default values in the **h5dump** file functions and can be changed depending on the process.

The **h5tools_context_t** structure is used to control where to place element rendering in a column defined output line or group of lines. The member **indent_level** is manipulated most often and is initialized by the h5dump functions from the global variable; **dump_indent**. The number of columns per indent level is controlled by the **h5tool_format_t** member; **line_indent**.

h5dump Current Code Layout

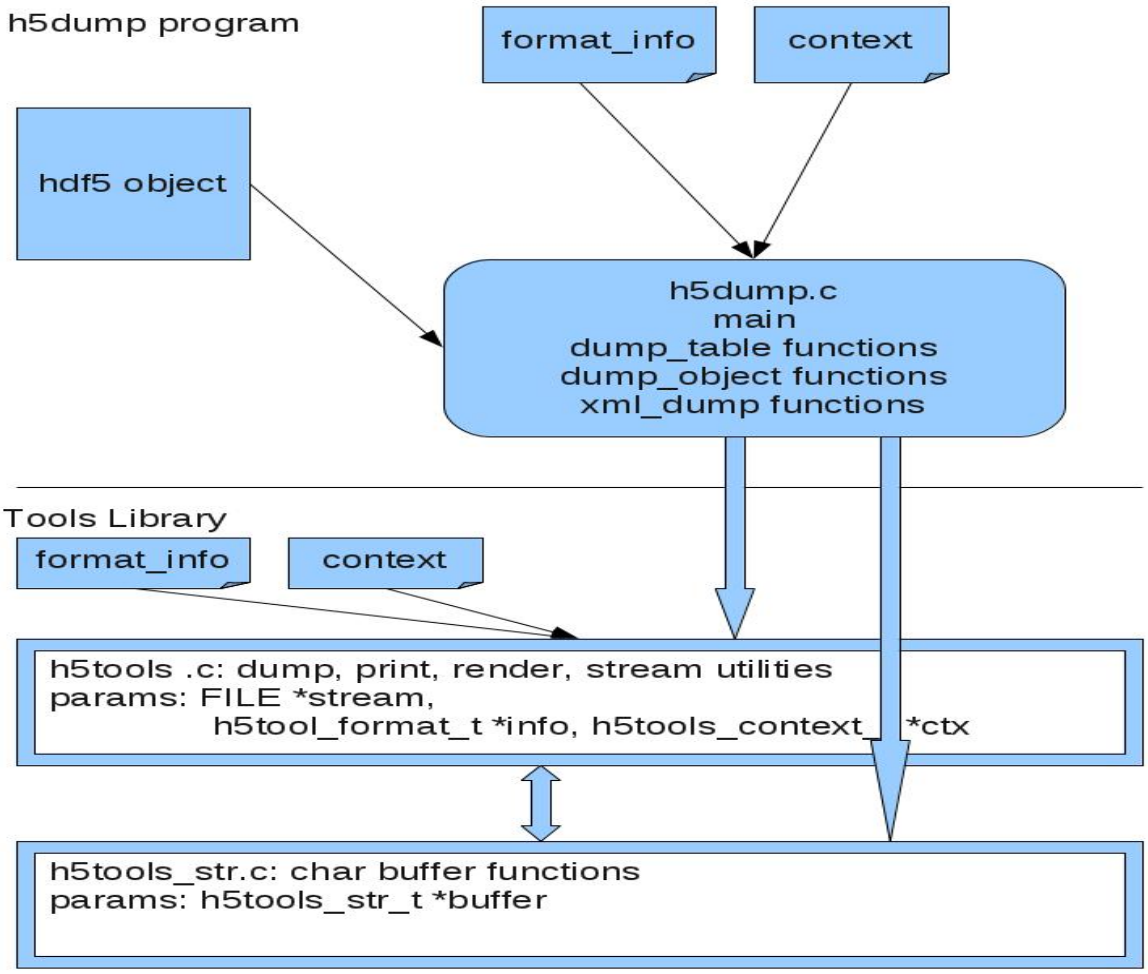

Figure 1 Current Code Layout

Currently, the h5dump functions are concentrated in two locations; h5dump.c and the library file h5tools.c. Both files have some duplication of functions, variables, defines and the format and context structures. Only the functions in h5dump.c use the global indentation variable. Also, the h5tools_str.c file provides string manipulation utility functions to h5dump by the IN/OUT variable: buffer.

Proposed h5dump Refactoring

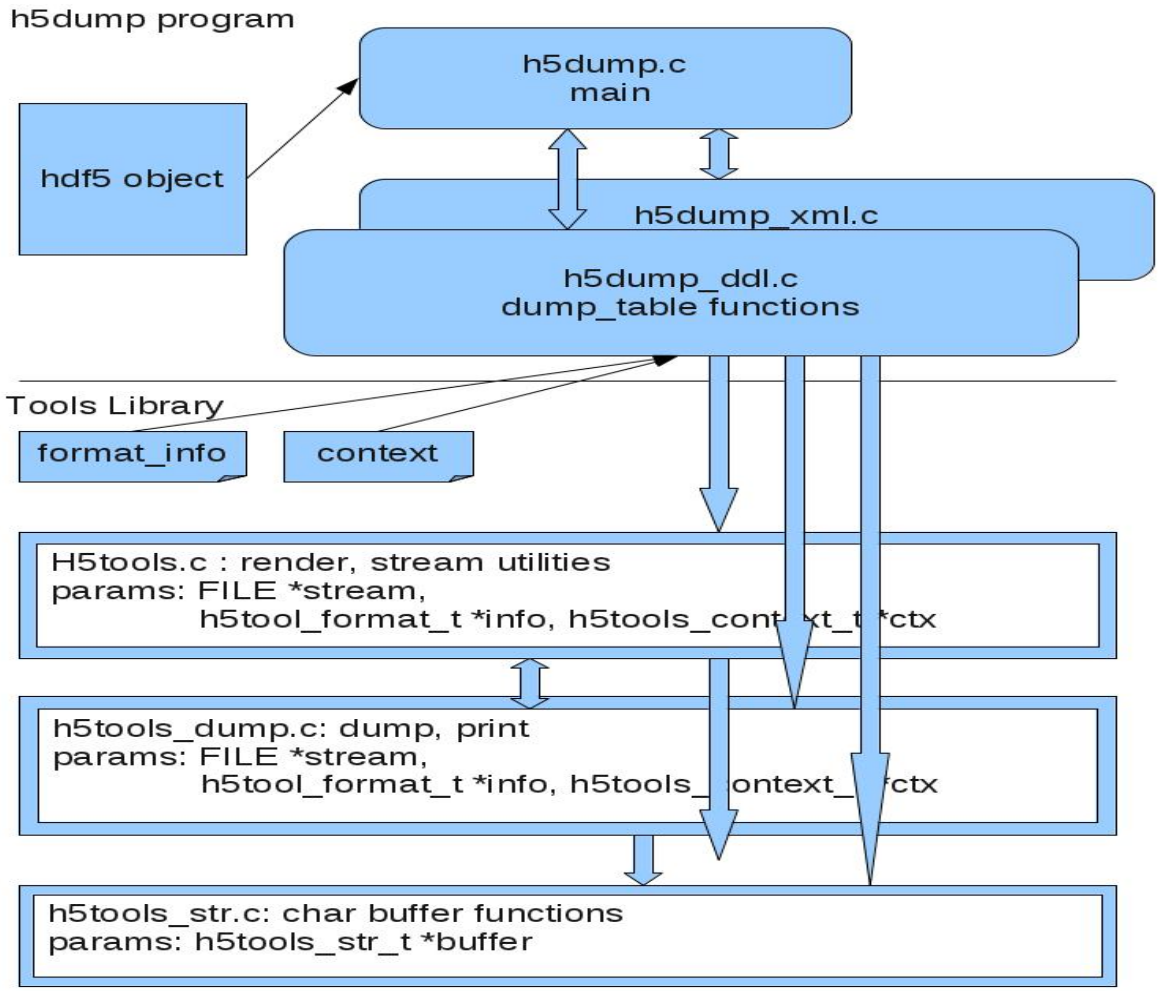

Figure 2 Refactored Code Layout

Split h5dump.c into three files

To expose the inconsistent use of functions and formatting and identify the dump program flow, group the common functionality of h5dump into three files; **h5dump_main**, **h5dump_xml**, and **h5dump_ddl**. The **h5dump_main** file will consist of the command-line parameter parsing and the main() function. The main() function, after initialization tasks, decides if ddl or xml formatted text is output to the screen. Also the main() function decides if just the file information is printed, or if individual groups/attributes/datasets are printed, or if the root group is printed by default. Splitting the h5dump file into three files separates the high level **dump_function_table** callback functions from the utility dump functions.

Add independent context structures to each high-level function.

The high level functions are those assigned to the callback function table, **dump_function_table**. These functions can safely initialize the format and context structures and pass these as parameters to the utility functions.

In the proposed refactored code, the two structures used to control the output of information from h5dump; **h5tool_format_t** and **h5tools_context_t**, and the #defines should exist only in the tools library. Each **dump_function_table** function will create and initialize the independent format and context structures using the global variable **dump_indent** (initialized by the **main** function) to configure the context indentation. The independent format and context structures can then be used by the tools library to control formatting of the data or information.

Remove duplicate functions from h5dump.

Moving the dump utility functions into the h5tools library will identify the dump process flow from the dump output control. Further refactoring out the dump content functions from the h5tools code should expose the lower level inconsistencies in the library that the dump content functions depend upon.

Each function moved into the tools library will depend on the formatting and context information provided by structure passed into the functions. This will require translation of indentation based on the global variable **dump_indent** to using the context indentation in the **h5tools_context** structure.

Create and/or update h5tools_str functions into building block functions.

There is a general data buffer output sequence followed by the dump functions in the h5dump program and the tools library:

> Insert an optional newline and prefix, then append text to the buffer, and conclude with the rendering of the buffer to the output stream.

This general sequence is usually contained within an element BEGIN/END pair that indicates the type of information and its bounds. For example, the GROUP object will display as follows with two pairs (the second pair BLOCK BEGIN/END is the brackets):

…
GROUP “groupname” {
…
}
…
	“BEGIN ELEM” “name” “BLOCK BEGIN”
	…
	“BLOCK END” “END ELEM”

The basic low-level operations in the **h5tools_str** file only operate on an output buffer using a supplied output format variable. These operations supply the foundation blocks for the functional operations to be in the **h5tools_dump** file. These mid-level functions should contain code blocks that involve the stream file and the output format passed in from a high-level function in the h5dump program. Utility functions in **h5tools** and **h5tools_utils** should provide functions that support the interface between these two files, **h5tools_str** and **h5tools_dump**. The h5tools_str.c file will still provide string manipulation utility functions to **h5tools_dump** by the IN/OUT variable: buffer.

Add unit testing to h5dump functions and tools library functions h5dump uses.

Refactoring the h5dump program by eliminating duplication and moving dump functions to the tools library creates an urgent need to test the functions in the tools library. During a refactoring trial, verification of the output created by the refactored code exposed numerous indentation issues and incorrect data formatting. Had there been unit tests available, many of these issues would not have existed originally, and will save developer time trying to determine the correct formatting. This unit testing can be accomplished by beginning with the low-level individual functions in the tools library. Integration tests can check the parameters and logic that are at a scope above the functions being tested.

Unit testing should be limited to the functions that do not output information to the generic "FILE *stream" parameter (usually stdout/stderr). This requirement will simplify the verification of the functions operation. Later integration testing, similar to current regression tests, can handle the issues with streams and consider code coverage. The current test implementation has a large degree of duplication and doesn't present a cohesive test plan.

Recommendation

The h5dump tool can be made more efficient and allow new features to be easier to implement by refactoring the existing h5dump and tools library code base.

1. h5dump.c: Separate the ddl and xml dump functions of the h5dump file into separate files - this makes removing duplicate functions and variables easier for the developers. After the reduction of duplicate functions and variables, this separation of concerns will allow developers to better focus on future improvements.
2. h5dump_ddl.c/h5dump_xml.c: In each high-level function, add an independent context structure and initialize it with the global indentation variable. Control interaction with **dump_function_table** functions by aligning the **context->indent** variable and the global **dump_indent** variable. Also add an independent format structure and initialize it, usually mapping the default values supplied in the tools library.
3. h5dump_ddl.c/h5tools.c: Move the content display functionality into the new **h5tools_dump** file in the library. This will simplify the h5dump program to concentrate on function control flow and the tools library to concentrate on formatting. A future by-product could be the creation of hdf5 dump APIs in the tools library.

4. h5tools_str.c: Create and/or update the low-level **h5tools_str** functions into building block functions - this allows more general content dump functionality to be built into the **h5tools_dump** functions, which can concentrate on how to display the content.
5. Finally, add unit testing of the low-level library code and integration testing of the library dump functions. Unit testing will help keep future library changes from breaking the h5dump design. The h5tools_str.c functions can easily be tested because it only uses a memory buffer for I/O. The h5tools_dump.c functions will require the stream parameter to be redirected for test verification.

Function Locations (excluding XML named functions)

Original h5dump.c functions (NOTE: tools library functions use h5tools_ prefix)

function	File location	Refactored location
h5_fileaccess	h5dump.c	h5dump.c
dump_oid	h5dump.c	h5tools_dump.c
dump_packed_bits	h5dump.c	h5tools_dump.c(print_packed_bits)
print_enum	h5dump.c	----------
init_prefix	h5dump.c	h5dump.c
add_prefix	h5dump.c	h5dump.c
dump_all_cb	h5dump.c	h5dump_ddl.c
dump_extlink	h5dump.c	h5dump_ddl.c
dump_group	h5dump.c	h5dump_ddl.c
dump_named_datatype	h5dump.c	h5dump_ddl.c
dump_dataset	h5dump.c	h5dump_ddl.c
dump_dataspace	h5dump.c	h5dump_ddl.c
dump_datatype	h5dump.c	h5dump_ddl.c
dump_data	h5dump.c	h5dump_ddl.c
dump_dcpl	h5dump.c	h5tools_dump.c
dump_comment	h5dump.c	h5tools_dump.c
dump_fcpl	h5dump.c	h5dump_ddl.c
dump_fcontents	h5dump.c	h5dump_ddl.c
dump_attr_cb	h5dump.c	h5dump_ddl.c
leave	h5dump.c	h5dump.c
usage	h5dump.c	h5dump.c
table_list_add	h5dump.c	h5dump.c
table_list_visited	h5dump.c	h5dump.c

table_list_free	h5dump.c	h5dump.c
print_datatype	h5dump.c	----------
dump_selected_attr	h5dump.c	h5dump_ddl.c
dump_dims	h5dump.c	h5tools_dump.c(print_dims)
dump_subsetting_header	h5dump.c	h5tools_dump.c
dump_fill_value	h5dump.c	h5tools_dump.c (print_fill_value)
set_output_file	h5dump.c	h5dump.c
set_binary_form	h5dump.c	h5dump.c
set_sort_by	h5dump.c	h5dump.c
set_sort_order	h5dump.c	h5dump.c
handle_attributes	h5dump.c	h5dump_ddl.c
parse_hsize_list	h5dump.c	h5dump.c
parse_subset_params	h5dump.c	h5dump.c
parse_mask_list	h5dump.c	h5dump.c
handle_datasets	h5dump.c	h5dump_ddl.c
handle_groups	h5dump.c	h5dump_ddl.c
handle_links	h5dump.c	h5dump_ddl.c
handle_datatypes	h5dump.c	h5dump_ddl.c
parse_command_line	h5dump.c	h5dump.c

Original h5tools.c functions

function	File location	Refactored location
h5tools_init	h5tools.c	h5tools.c
h5tools_close	h5tools.c	h5tools.c
h5tools_fopen	h5tools.c	h5tools_dump.c
h5tools_dump_dset	h5tools.c	h5tools_dump.c
h5tools_dump_mem	h5tools.c	h5tools_dump.c
h5tools_get_native_type	h5tools.c	h5tools.c
h5tools_get_little_endian_type	h5tools.c	h5tools.c
h5tools_get_big_endian_type	h5tools.c	h5tools.c
h5tools_detect_vlen	h5tools.c	h5tools.c
h5tools_detect_vlen_str	h5tools.c	h5tools_dump.c
h5tools_is_obj_same	h5tools.c	h5tools.c
h5tools_dump_simple_data	h5tools.c	h5tools_dump.c
h5tools_canreadf	h5tools.c	h5tools.c

h5tools_can_encode	h5tools.c	h5tools.c
init_acc_pos	h5tools.c	h5tools.c
h5tools_dump_datatype	h5tools.c	h5tools_dump.c
h5tools_print_dataspace	h5tools.c	h5tools_dump.c
h5tools_print_datatype	h5tools.c	h5tools_dump.c
h5tools_print_enum	h5tools.c	h5tools_dump.c
h5tools_dump_init	----------	h5tools_dump.c
h5tools_dispaly_simple_subset	h5tools.c	h5tools_dump.c
h5tools_dump_region_data_blocks	h5tools.c	h5tools_dump.c
h5tools_dump_region_data_points	h5tools.c	h5tools_dump.c
h5tools_dump_simple_dset	h5tools.c	h5tools_dump.c
h5tools_dump_simple_mem	h5tools.c	h5tools_dump.c
h5tools_dump_simple_subset	h5tools.c	h5tools_dump.c
h5tools_is_zero	h5tools.c	h5tools.c
h5tools_ncols	h5tools.c	h5tools.c (count_ncols)
h5tools_print_region_data_blocks	h5tools.c	h5tools_dump.c
h5tools_print_region_data_points	h5tools.c	h5tools_dump.c
h5tools_print_simple_subset	h5tools.c	h5tools_dump.c
h5tools_region_simple_prefix	h5tools.c	h5tools.c
h5tools_render_element	h5tools.c	h5tools.c
h5tools_render_region_element	h5tools.c	h5tools.c
h5tools_simple_prefix	h5tools.c	h5tools.c
render_bin_output	h5tools.c	h5tools.c
render_bin_output_region_blocks	h5tools.c	h5tools.c
render_bin_output_region_data_blocks	h5tools.c	h5tools.c
render_bin_output_region_data_points	h5tools.c	h5tools.c
render_bin_output_region_points	h5tools.c	h5tools.c

1)

Reference

- RFC: Code Refactoring for h5dump at https://www.hdfgroup.uiuc.edu/RFC/HDF5/tools/h5dump/h5dump_code_refactoring_v2.pdf

- BNF: DDL in BNF for HDF5 at
  http://www.hdfgroup.org/HDF5/doc/ddl.html
- XML: Document Type Definition (DTD) for HDF5 at
  http://www.hdfgroup.org/HDF5/XML/DTD/HDF5-File.dtd

Revision History

June 22, 2011:	Version 1 reviewed for comment.
August 31, 2011:	Version 1 reordered sections.
September 13, 2011:	Version 2 split into two documents
September 19, 2011:	Version 2 renumbering of sections to identify five major points.
October 5, 2011	Version 3 added structure appendix and list of functions targeted for refactoring
October 7, 2011	Removed appendix, best to refer to companion document (in process)