RFC: Code Refactoring for h5dump

Peter Cao

This RFC describes the work of code refactoring for h5dump.

# Introduction

Maintainable and extendable code is a main part of our work toward to high quality software (or tools). We started an RFC on structural layout design for tool library[[1]](#footnote-1) a year ago. This work will focus on improving the source code of h5dump.

Better code structure and design has many benefits. Below we list a few:

* Better maintainable: easier to understand and test
* More extendable: easier to modify and add new features

We start with h5dump for three reasons: 1) h5dump is one of the most used tools; 2) h5dump source code is complicated and unorganized; 3) we have an urgent need to merge the source code for showing packed-bits to the current source repository. Before adding the code to h5dump.c, we need a better structure for the current h5dump source code.

Restructuring the code means

* moving codes to separate files based on features;
* breaking up large functions into small functions;
* renaming functions and variables for better consistency;
* fixing defects such as dead code, redundant code, warnings, and other defects;
* replacing algorithms or conditions for better performance;
* modifying or adding tests, and
* improving documentation.

However, this work will not

* change the functionalities and user interface for h5dump,
* rewrite the h5dump tools, which is an accumulation of a lot work over ten years.

# Code assessment

h5dump is one of the command-line utilities developed with the first release of the HDF5 library in 1998. Over the years more and more features and options were added to h5dump and the source code became larger and poorly organized. debug defects or adding any new features become very difficulty since the code is not well designed and organized.

The following table is a list of the source files that are used by h5dump (in tools/h5dump directory). h5dump also uses functions in the tools library at tools/lib directory. This work does not include reconstructing of the tools library, which has about 20,000 lines of code.

Table 1 -- h5dump source files

|  |  |  |
| --- | --- | --- |
| File | Lines of Code | Description |
| h5dump.c | 7,175 | Main source code for h5dump |
| h5dumpgentest.c | 7,161 | Generating testing files |
| testh5dump.sh.in | 646 | Testing scripts |
| testh5dumpxml.sh.in | 201 | Scripts for testing xml output |
| h5dump.h | 160 | Header file for h5dump |
| Total | 15,343 |  |

# Refactoring techniques and tools

The goal of this work is to make h5dump easy to maintain. To accomplish this, below are of the techniques that will be used in the code refactoring:

* Using abstraction: creating more general types, functions, and libraries to allow for more code sharing
* Breaking code into more logic pieces: breaking a large code into small logical pieces for more easily understandable. Independent features will be moved into separate files for more configurable
* Improving names and documentation: following a common name standard, e.g. the “HDF5 Coding Standards”[[2]](#footnote-2) and Tools naming scheme.

Some software editors and IDEs, such as Eclipse, NetBeans, and Visual Studio, include features for automated code refactoring.

# Tasks and work estimation

The first priority is to reconstruct the main h5dump code, h5dump.c, which have more 7,000 lines of code. We also continue to fix defects in h5dump tool as part of Coverity session effort according to the coverity assessment information

## Revise feature functions (32 hours)

The first task is to make independent features as easily maintainable components. The main code frame will just hold a list of features functions and the implementation of individual features will be moved to separate files. Some of the separate features in h5dump include:

* Write to xml format,
* Print header information only,
* Dump groups,
* Dump datasets,
* Dump named datatypes,
* Show packed bits,
* Print content list (a list of objects in a file),
* and etc.

Using an easy maintainable template is not new. There is already a global dispatch table for the dump functions. We will revise the list and move the implementation to separate files as needed.

typedef struct dump\_functions\_t {

void (\*dump\_group\_function) (hid\_t, const char \*);

void (\*dump\_named\_datatype\_function) (hid\_t, const char \*);

void (\*dump\_dataset\_function) (hid\_t, const char \*, struct subset\_t \*);

void (\*dump\_dataspace\_function) (hid\_t);

void (\*dump\_datatype\_function) (hid\_t);

herr\_t (\*dump\_attribute\_function) (hid\_t, const char \*, const H5A\_info\_t \*, void \*);

void (\*dump\_data\_function) (hid\_t, int, struct subset\_t \*, int);

} dump\_functions;

## Add code for packed bits (16 hours)

The feature for printing values of packed bits was added to the HDF5 1.8 branch by using a macro, H5\_HAVE\_H5DUMP\_PACKED\_BITS. The code is not in the HDF5 trunk. This task is to have this feature in both HDF5 trunk and 1.8 branches using better organized structure. Since this part of the code is relatively independent and small, it should not be so hard to reorganize the code using the feature template discussed above.

## Separate the XML feature (16 hours)

The part for handling xml has about 2000 lines of code. This task is to move the xml feature to a separate file. The major functions for writing xml include:

* xml\_dump\_group,
* xml\_dump\_named\_datatype,
* xml\_dump\_dataset,
* xml\_dump\_dataspace,
* xml\_dump\_datatype,
* xml\_dump\_attr,
* xml\_dump\_data

## Other work

Other work listed section 4.2 will be evaluated as we move forward. We will also need to examine the code that generates the test files, h5dumpgentest.c

# Revision History

|  |  |
| --- | --- |
| *March 25, 210:* | Version 1 circulated for comment within The HDF Group. |
|  |  |
|  |  |
|  |  |

1. <https://www.hdfgroup.uiuc.edu/RFC/HDF5/tools/general/RFC_Tools_Lib_v1.pdf> [↑](#footnote-ref-1)
2. https://thehdfgroup-import.pbworks.com/w/page/26711525/HDF5-Coding-Standards [↑](#footnote-ref-2)