# RFC: Portable Lock Free Data Structures for HDF5

John Mainzer (john.mainzer@lifeboat.llc)
Aijun Hall (aijun.hall@lifeboat.llc)
Adam Ohlson (adam.ohlson@lifeboat.llc)
Anna Burton (anna.burton@lifeboat.llc)

The effort to retrofit multithread support on the HDF5 library has required the development of a variety of lock free data structures. These data structures were implemented without concern for reuse in other areas of the HDF5 library, or support for HDF5 error reporting.

While this is appropriate for prototype code, for production purposes, these data structures should be reworked as collections of C macros, allowing easy reuse in other sections of the HDF5 library, and integration into the HDF5 error reporting system.



## 1 Introduction

The ongoing effort to retrofit multi-thread support onto the HDF5 library has required the development of a variety of lock free data structures – most notably:

- Lock free singly linked lists,
- Lock free hash tables, and
- Lock free free lists for discarded dynamically allocated structures used in lock free data structures. These structures must be retained in the free lists until all references to them have been discarded – at which point they may be either re-allocated or released to the heap.

While these data structures have been essential in the prototype multi-thread implementations of the initial target HDF5 modules (most notably H5I and H5P), the implementations are specific to their target applications, are not easily portable, and are not integrated into the HDF5 library error reporting system.

This is appropriate for prototype code, but for production code, these deficits must be addressed.

The model we have selected for this effort is UTHash – a portable hash table that is implemented in a collection of C macros to allow easy re-use in any program written in C.

## 2 Conceptual Overview

The target lock free data structures are already implemented as a collection of structure definitions and regular C functions. Further there is an extensive test suite. Error checking and reporting is currently done via assertions.

To convert this code into an easily re-usable collection of C Macros, we must do the following:

- Convert the existing collection of functions into C macros. Note that in some cases, this requires structural modifications.
- Modify the macros to use configurable error reporting calls. Since assertions are very useful in debugging lock free multi-thread code, the option of using the existing assertion based system must be retained.
- Rework the existing test suite to meet the standards for HDF5 regression test code.
- Address limitations in the existing implementations. In particular:
    - Address the hard limit on the number of hash buckets in the lock free hash table.
    - The lock free free list implementations all use different heuristics to determine when an entry can be safely re-allocated or released to the heap. Abstract these heuristics out of the existing code so as facilitate easy re-use.
- Write user's manual.

The implementation and testing details are discussed in subsequent sections.

# 3 Implementation Details

Given the pre-existing implementations, the primary thrust of this RFC is document it to document the issues encountered, and how they were addressed.

As this is an ongoing process, this document will be updated as portions of the task are competed.

## 3.1 Lock Free Hash Table

### 3.1.1 Initial Conversion to Macros

In addition to the obvious name space issues, the following issues had to be addressed in the initial conversion of the lock free hash table code to a collection of macros:

Perhaps the simplest of these was the matter of returning values to the caller.

The notion of using a compound statement for this purpose was considered, as it would solve the return value problem quite neatly. Unfortunately, this is a gcc extension to C, and thus is not portable. Thus, to convert functions with return values, we elected to add pointers to variables of the appropriate type to the argument lists of the macro versions, and use these pointers to return the necessary values.

In a closely related issue, some of the original functions were used in Boolean expressions and/or argument lists. As the inability to return values directly made this impossible in the macro versions, we simply invoked the macros earlier in the code, stored the results in local variables, and replaced the function calls with these variables as required.

The other issue of significance was the use of recursion in one of the existing functions[1]. Here, recursion is used to insert any missing hash buckets that must exist prior to insertion of the target hash bucket. In the macro, the recursion is replaced with an iterative approach, in which a list of needed buckets is compiled, and then inserted in the correct order.

A slightly modified version of the original test code was used to verify the correctness of the initial macro conversion.

### 3.1.2 Error Reporting

TBD

### 3.1.3 Test Suite Conversion

TBD

### 3.1.4 Hash Buckets

TBD

## 3.2 Lock Free Singly Linked Lists

TBD

[1] Lfht_create_hash_bucket()

### 3.3 Lock Free Free Lists

TBD

## 4 Testing Details

TBD

## Acknowledgements

This work is supported by the U.S Department of Energy, Office of Science under award number DE-SC0022506 for Phase II SBIR project “Toward multi-threaded concurrency in HDF5 " and under award number DE-SC0023583 for Phase II SBIR project "Supporting Sparse Data in HDF5".

## References

1. The HDF Group. “HDF5 Documentation,” http://www.hdfgroup.org/HDF5/doc/doc-info.html (June 14, 2010).
2. (1) https://support.hdfgroup.org/documentation/hdf5/latest/_t_n_m_d_c.html (Metadata Caching in HDF5), insert formatted citation.
3. (2) - C11 Standard, insert formatted citation.
4. (3) - UTHash T. Hanson, insert formatted citation.

## Revision History

| | |
|---|---|
| *1/27/25* | Version 1 circulated for comment within Lifeboat, LLC. |
| *2/27/25* | Version 2 circulated for comment within Lifeboat, LLC. |