HDF5 Debug & Trace Functionality

August 2011

Dana Robinson (derobins@hdfgroup.org)

The HDF5 library can be set up to emit verbose debugging and tracing information. Configuring the library to emit this information requires two things:

1) The library must have been built with particular configure settings or have appropriate preprocessor symbols defined.

2) The HDF5\_DEBUG environment variable must be set to allow debug/trace output.

The scope of this document is predominantly the use of the HDF5 debug and tracing framework. The use of several additional debug-oriented configure flags is also described. A final section of this document covers actions library maintainers need to take to ensure that the debug and trace functionality stays up-to-date.

# Configure Flags

This section describes the various debug-related configure flags that can be enabled. Some of them are required for debug/trace code functionality.

**WINDOWS/VMS USERS: You can set the preprocessor symbols directly in your platform's H5pubconf.h or in your build environment (*e.g.* the Visual Studio properties page for your project).**

--enable-debug=<package list | all | yes | no>

('Package list' is comma-separated list of packages without the preceding H5. e.g. 'A,B2,MM,Z')

Default:

Production: No packages have debug code enabled

Otherwise: Many packages have debug code enabled (see below for a list)

Library preprocessor symbols un/defined: **NDEBUG, H5<package>\_DEBUG**

Turns debug code on/off in specified HDF5 internal library packages. Depending on the package, this could enable extra error checking, assertions or verbose output. Verbose output is usually also controlled by the HDF5\_DEBUG environment variable (see below).

If any package has debug code enabled, the preprocessor symbol NDEBUG will be undefined. In addition, for each package selected, H5<package>\_DEBUG will be defined (e.g. H5Z\_DEBUG when the H5Z package is selected).

When no packages are selected, NDEBUG will be defined and no H5<package>\_DEBUG symbols will be defined.

Several package groups can be defined:

'yes' or default (production not enabled)

H5D, E, F, G, HG, I, MM, O, P, S, T, V and Z will have debug code enabled.

'all'

All the packages listed under 'yes' and H5A, B, B2, HL and MF will have debug code enabled.

'no', 'none' or default (production enabled)

No packages will have debug code enabled.

--enable-codestack

Default: NOT SET

Library preprocessor symbol defined: **H5\_HAVE\_CODESTACK**

Allows recording and displaying the function call stack in the HDF5 library without the use of a debugger.

--enable-metadata-trace-file

Default: NOT SET

Library preprocessor symbol defined: **H5\_METADATA\_TRACE\_FILE**

When this is enabled, verbose metadata cache trace information will be sent to an output file. The file name of the output file and many other parameters can be configured using the H5Pset\_mdc\_config() API call.

This is normally only of interest to HDF5 library developers.

--enable-trace

Default:

Any package debug flag set: SET

Otherwise: NOT SET

Library preprocessor symbol defined: **H5\_HAVE\_DEBUG\_API**

When set, this enables API tracing capability. The API calls which emit trace information can be set via the HDF5\_DEBUG environment variable.

Note that no trace output will be emitted unless the HDF5\_DEBUG environment variable is also set for trace output (see below).

--enable-instrument

Default:

Any package debug flag set: SET

Otherwise: NOT SET

Library preprocessor symbol defined: **H5\_HAVE\_INSTRUMENTED\_LIBRARY**

This enables extra property list settings that the HDF5 library tests use to track whether certain parallel optimizations were used correctly. There is no effect in non-parallel (non-MPI) builds.

--enable-using-memchecker

Default: NOT SET

Library preprocessor symbol defined: **H5\_USING\_MEMCHECKER**

Enable this option if a memory allocation and/or bounds checking tool will be used with the HDF5 library. Enabling this causes the library to be pickier about its memory operations and also disables the library's free space manager code.

NOTE: Some memory checkers and dynamic code analyzers will complain if --enable-clear-file-buffers is not also set (see below).

--enable-clear-file-buffers

Default: SET

Library preprocessor symbol defined: **H5\_CLEAR\_MEMORY**

This will ensure that unused areas of allocated memory are zeroed out before being written to disk. Note that this is not the same as a dataset's fill value as this applies to all allocated blocks (file metadata, etc.).

**WARNING: It is highly advisable that you leave this set, even though it is a slight performance hit. In the past, there have been several high-profile security holes that were caused by writing uncleared memory to disk[[1]](#footnote-1).**

NOTE: Some memory and dynamic code analyzers (Purify, etc.) will complain if this is not set as they will see the HDF5 library writing uncleared memory to disk.

--enable-strict-format-checks

Defaults:

Any package debug flag set: SET

Otherwise: NOT SET

#define: H5\_STRICT\_FORMAT\_CHECKS

This enables some stricter file format checks than the library normally performs. The effects of this flag are minimal, mainly affecting very small parts of the H5B, H5G and H5O packages. A few tests have sections which are skipped when this flag is set in order to avoid spurious failure results.

Setting this flag may cause some backward-compatibility issues when reading older HDF5 files[[2]](#footnote-2).

# Debug Environment Variable

The HDF5 library uses an environment variable named HDF5\_DEBUG to set up the debug output streams on a per-package basis. This also implicitly turns on/off verbose debug output; without this variable being set the debug code will only perform extra sanity checks.

The HDF5\_DEBUG environment variable is a string which is parsed from left to right. The string contains a list of packages (not including the preceding H5) for which verbose debug output is desired. Each package can be preceded by a '+' or '-' to enable or disable output, respectively ('+' is the default and is redundant). Package names must be lower-case. The packages can be separated by spaces or commas.

The list of packages for which verbose output can be selected is:

a, ac, b, e, f, g, hg, hl, I, mf, mm, o, p, s, t, v, z

Several special entries can be added to the package list:

all – Turns verbose debugging output for all selectable packages on or off. +/- can be used with this.

trace – Enables API call trace output.

ttimes – Adds elapsed times to the API call trace output. This will also enable tracing.

ttop – Limits trace output to the top-level API calls. This will also enable tracing.

Additionally, the output files/streams to which the output will be directed can be set in the HDF5\_DEBUG string environment variable. A number (including 1 (stdout) and 2 (stderr)) can appear in the string and all subsequently listed packages will have their output directed to that file descriptor or stream[[3]](#footnote-3). The default output stream is stderr.

The default value of HDF5\_DEBUG when it is not set is "-all" *so no debug or trace output will be emitted by default*.

Example: "all -ac 1 trace ttimes"

This turns on debug output for all packages specified during library configuration except the H5AC package. Tracing and API call durations will also be emitted, though stdout will be used instead of the default stderr.

# HDF5 Library Developers

It is important that HDF5 library developers both maintain the existing debug functionality, work to add new debugging features and extend debug code to new packages.

## Debugging

* Make sure that the package is listed in the --enable-debug section of configure.in and that an H5<pkg>\_DEBUG symbol will be created. Add it to the defaults unless the debug code is extremely time-consuming.
* Surround package-specific debug-only code with #ifdef H5<pkg>\_DEBUG
* #ifndef NDEBUG blocks can be used to surround code that should always be executed in debug mode.
* Be sure to update this document when you are finished.

## Verbose Debug Output

* Make sure that the package is listed in configure.in as described in section 3.1.
* Add an entry to the H5\_pkg\_t enum in H5private.h.
* Initialize the package stream in H5\_init\_library() in H5.c.
* Surround any verbose debug output code with if(H5DEBUG(<pkg>)) *e.g.* if(H5DEBUG(P)). All verbose output code MUST go inside a package-specific debug block (see section 3.1, above).
* Be sure to update this document when finished.

## Trace Output

* When a new API function has been added to the library, run bin/trace to add the H5TRACE macros (this is done automatically from bin/reconfigure) and monitor the output for errors. Errors usually occur when the script encounters types it doesn't understand.
* When a new type is added to the library, it must be added to the bin/trace Perl script along with a new format code. 'x' can be used if the type can't be printed (*e.g.* function pointers).

NOTE: Only use the 'x' output format for API parameters that can't be printed. Do not use it as a shortcut during development.

* If a new type has been added to the bin/trace script or if a type has been modified (*e.g.* a new enum value has been added), the H5\_trace() function in H5trace.c will have to be modified so that the new/changed type will display properly.
* If your new API call uses a large number of parameters (more than 11 at the time of this document), you will need to add a new trace macro (example: H5TRACE12) to H5private.h.

1. We haven't seen this in HDF5 – In particular, this refers to the famous tar bug in Solaris 2.0 where newly created tarballs contained fragments of the /etc/passwd file. See p.15 of *Secure Coding: Principles and Practices* by Graff and van Wyk for details. [↑](#footnote-ref-1)
2. A previous version of the library could create erroneous object header messages. When –enable-strict-format-checks is unset, these are ignored or corrected, when it is set they will be reported as errors. [↑](#footnote-ref-2)
3. The intended use of this feature is to set the HDF5\_DEBUG environment variable from inside the program: open your output files using POSIX I/O calls and then call setenv() (or its equivalent on non-POSIX systems, SetEnvironmentVariable() on Windows, for example) to create the environment variable. [↑](#footnote-ref-3)