RFC: Native Time Types in HDF5

Neil Fortner

Introduction

This document examines the current implementation of time datatypes in HDF5, and illustrates the additions and enhancements provided by Dan Anov's native time patch. The native time patch implements a native UNIX time type for the hdf5 library, as well as functions to convert between different time types. It is mostly complete, and even includes modifications to some of the tools to allow them to display time datatypes. This could be very useful to users who want to write portable code to save times obtained from time() or similar system calls, and to allow them to easily share hdf5 files with time datatypes between different machines.

1 Motivation

Time types in HDF5 could be useful for anyone that wants to save standard UNIX time data to an HDF5 file. The native time patch allows users to directly use time_t data obtained through standard C library calls with the introduction of the H5T_NATIVE_TIME_T predefined type. While it could be argued that it would be more appropriate to use integer (or floating point) types to represent time, with attributes describing the way it should be interpreted, the fact is that UNIX time types are already included in the library and people are using them.

2 Current Status

The library currently supports the UNIX time types H5T_UNIX_D32LE, H5T_UNIX_D32BE, H5T_UNIX_D64LE, and H5T_UNIX_D64BE. It however, is incapable of converting between these types and does not support a native time type. Therefore, anyone who wishes to use these must manually match the type with the current size and byte order of time_t, and must save to the file in the same type. If a user wishes to share such files between machines with different time_t, they must read the non-native data directly into memory and implement their own conversion function to retrieve native time_t.

3 Patch Details

The patch sets H5T_NATIVE_TIME_T based on the native byte order and sizeof(time_t). Times written by a patched version of the library can be read by an unpatched version, though the same limitations apply as above. Even times written as H5T_NATIVE_TIME_T can be read provided the application reading knows exactly which UNIX type H5T_NATIVE_TIME_T was mapped to when written, or takes the time to manually retrieve the size and byte order and use the appropriate UNIX time type. The patch as it currently stands in no way modifies the file format or breaks existing functionality. Conversions between the different UNIX types are handled in the same way as integers. Although it is not required by the ISO C standard (though it is required by POSIX), most

compilers will return time_t as UNIX time from calls to the time.h library, regardless of the system being run or its truly "native" time.

4 Limitations of the Patch

Unfortunately some non-POSIX compilers return time_t as an unsigned value, or as a signed value with negative values considered an error (as in MS Visual C++). As there is no distinction in the patch between signed and unsigned, this could cause ambiguity. The patch also does not include support for languages other than C; these would have to be added if desired. Finally, the output format of the tools would have to be decided upon. Currently the patch causes h5ls to output time according to the current locale (as found by setlocale(LC_ALL, "")). If the user wants to specify the locale at run time, the environmental variable LC_ALL can be set. H5dump does not display time data at all, though it displays information about the type. Fortunately, a test routine for time types is already mostly written (though disabled) and would only need minor modifications to test all of the features included in this patch., though additional tests for the tools should be written.

5 Outstanding Issues

At the developers meeting it was suggested that development of this time patch be placed in a branch. It was also more or less agreed upon that the patch should be feature complete before it is merged back into the trunk. We decided that we should only support UNIX (POSIX) time, and go ahead with the implementation of the remaining features. Outstanding issues include:

1) Should we implement a test to check if time_t is UNIX time before enabling use of H5T_NATIVE_TIME_T? If so should we check if time_t is signed and can have negative values?

Pro	Con
• **Maximum compatibility** • **Guarantees that H5T_NATIVE_TIME_T is truly "native"**	• Adds to the already substantial configure script • May not work on systems that don't use configure • In the second case this check could prevent some legitimate and correct uses of the native time type

2) Should we add support for native times to the C++ and Fortran API's?

Pro	Con
• **Allows more users to use the new features provided by this patch**	• The standard Fortran time type is a string, so Fortran support is unlikely unless we want to rely on system calls on UNIX systems • Similar situation with Java

	• Time required to implement

3) Should we define an "HDF5 time" type to be the same as UNIX time?

Pro	Con
• **Consistency in labeling (UNIX time is not only used on UNIX systems)**	• Potentially confusing • Would need to retain the old names in the code for compatibility

4) Should we leave the feature marked as "unsupported"?

Pro	Con
• **Makes it clear that this feature is not as platform independent as the rest of the HDF5 library** • **Helps resist any pressure to add support for many different time types, which is beyond the scope of the library**	• The feature would be otherwise complete and ready for use • Why go through all the effort of implementing an unsupported feature?

Overall, the minimum implementation of this patch would make some changes to the tools, but provide no checking to see if the native time type is POSIX compliant. The responsibility would then be placed in the hands of the users to ensure that the times they are saving will be interpreted correctly when read.

6 Examples

Here we show a few simplified examples to help illustrate the current support of time types, and show how this time patch improves the handling of time data.

6.1 Example 1

Example 1 shows how to write time data using the current library. Notice that both the file and memory type must be explicitly specified, and must be identical. If the file type (specified in the call to H5Dcreate) were different from the memory type (from H5Dwrite), H5Dwrite would fail. This code would only produce correct results on a little endian machine with 32 bit time_t.

```
time_t wdata = time (NULL);
dset = H5Dcreate (file, name, H5T_UNIX_D32LE, ...);
H5Dwrite (dset, H5T_UNIX_D32LE, ..., &wdata);
```

6.2 Example 2

Example 2 shows how to read a file using the current library. In order for this code to work, the dataset must have a type of H5T_UNIX_D32LE and the machine must again be little endian with 32 bit time_t.

```
time_t rdata;
H5Dread (dset, H5T_UNIX_D32LE, ..., &rdata);
printf ("time read is: %s\n", ctime (&rdata));
```

6.3 Example 3

Example 3 shows how to write time data with the patched library. This code is completely portable across all compilers which use UNIX time and will always produce correct datasets with type H5T_UNIX_D32LE regardless of the native type. It is also possible to save as a native type or to convert between two standard types.

```
time_t wdata = time (NULL);
dset = H5Dcreate (file, name, H5T_UNIX_D32LE, ...);
H5Dwrite (dset, H5T_NATIVE_TIME_T, ..., &wdata);
```

6.4 Example 4

Example 4 shows how to read data with the patched library. This will always produce correct results regardless of the file type or the processor architecture.

```
time_t rdata;
H5Dread (dset, H5T_NATIVE_TIME_T, ..., &rdata);
printf ("time read is: %s\n", ctime (&rdata));
```

6.5 Example 5

Example 5 shows how to save data as the native type. The dataset produced will be different depending on the machine, but will always be readable by patched versions of the library using code similar to example 4. Unpatched versions will need to specify the exact standard type saved to the file.

```
time_t wdata = time (NULL);
dset = H5Dcreate (file, name, H5T_NATIVE_TIME_T, ...);
H5Dwrite (dset, H5T_NATIVE_TIME_T, ..., &wdata);
```

7 Resolved Issues

These issues were previously listed under "outstanding issues", but have been decided upon. They are listed here for reference and to help explain the exact direction we plan to take with this patch.

1) Should we apply the patch at all? Though the necessity of having a time type in the library may be questionable, the fact that it is already in the library prevents us from removing it. Should we put the effort into finishing the patch or should we simply leave it as an unsupported feature?

Pro	Con
• **Finishes a half-implemented feature** • **Allows users to more easily store and retrieve time data**	• Development time • Specifies a method of interpretation for data

We have decided to implement the patch.

2) Should we implement checks when converting time types to a lower precision to ensure that overflow does not occur or should we leave the responsibility in the hands of the users? Should this be the default behavior?

Pro	Con
• **Will catch some errors related to the "Y2k38" problem**	• Adds to conversion overhead

The user can implement such checks using the already existing function H5Pset_type_conv_cb(). No new functionality will be added here.

3) Should we make a distinction between signed and unsigned time types? While this would not affect any conversions (unless we implemented overflow checking as above), it would allow users to specify how to interpret times beginning with a (binary) 1. This would require another test for the native type, and would require an update to the file format. It would, however not require a new datatype version as old versions of the library could still read signed and unsigned times – there is plenty of reserved space in the time datatype message currently.

Pro	Con
• **Prevents potential ambiguity of times before 1/1/1970**	• Requires update to file format • Unsigned times are not truly "UNIX" (POSIX) times

We have decided there will be no new updates to the file type, and the time type represented will be POSIX time. Therefore, the time type is assumed to be signed.

4) Similarly, should we support the "signed but only positive" type used by MS Visual C++?

Pro	Con
• **When paired with overflow checking would prevent erroneous times from being used by MSVC**	• Requires update to file format • Not a standard time type

See above, this type will not be explicitly supported.

5) Should h5dump display time data? If so, how should it be displayed?

Pro	Con
• **Allows users to view time data using h5dump**	• ASCII time strings could take up too much space in the output

Yes, h5dump will display time data.

Acknowledgements

The native time patch was written by Dan Anov of the Danish Technological Institute (dan.anov.dk@gmail.com).

Revision History

June 30, 2008: Version 1 circulated for comment within The HDF Group.

August 1, 2008: Version 2 incorporates feedback; circulated for internal comment.

August 15, 2008: Version 3 was a temporary version which was made obsolete by the arrival of the RFC template.

September 24, 2008: Version 4 circulated for comment within The HDF Group, final call for comments before public release.

October 1, 2008: *Version 5* is published and posted for public comment. Comments should be sent to nfortne2@hdfgroup.org or help@hdfgroup.org