RFC: New Datatypes

Jerome Soumagne
Quincey Koziol / Elena Pourmal

The C99 standard introduced support for boolean and complex types. Boolean and complex types are also supported in C++, Fortran, Java, Python. This RFC describes how support for these types can be added within the HDF5 library.

1 Introduction

The HDF5 library defines already datatypes for all common types, i.e., char, short, int, long, etc, as well as their specific encoded variation, for instance all standard types, as well as little/big endian encoding, etc. However, new datatypes were requested by HDF5 users (ESDIS, LLNL, etc) who work with scientific applications to support both *boolean* and *complex* types. Boolean types should be seen as a distinct conceptual type, not as another integer type. An important aspect of having a boolean datatype, despite the convenient mapping that it can provide between C/C++ codes that use boolean concepts, is to no longer require the use of integers for storing boolean data and therefore reduce storage needs. Regarding complex types, they are already natively defined by the C standard, therefore not only for convenience but also for performance reasons would it makes sense for the HDF5 library to provide a native datatype so that users are no longer forced to either use separate datasets or create a compound type (which would in this case be composed of real/imaginary fields) in order to store and retrieve data. As these types were introduced by the C99 standard, this set of datatypes can be defined as new HDF5 native types.

2 New Datatypes

The following types are considered: `_Bool`, `float _Complex`, `double _Complex`, `long double _Complex`. Next sections are a reminder of the rules that the C standard [1] defines for these types.

2.1 `_Bool`

An object declared as type `_Bool` is large enough to store the values 0 and 1. While the number of bits in a `_Bool` object is at least `CHAR_BIT`, the width (number of sign and value bits) of a `_Bool` may be just 1 bit.

Conversion rules: When any scalar value is converted to `_Bool`, the result is 0 if the value compares equal to 0; otherwise, the result is 1.

2.2 _Complex

An object declared as type complex is composed of two parts: real and imaginary, the real part is always before the imaginary part. Values of complex types are equal if and only if both their real parts are equal and also their imaginary parts are equal. The integer constant 1, `__STDC_IEC_559_COMPLEX__`, intended to indicate adherence to the IEC 60559 compatible complex arithmetic specifications. The integer constant 1, `__STDC_NO_COMPLEX__`, intended to indicate that the implementation does not support complex types or the `<complex.h>` header.

```
#include<stdio.h>
#include<complex.h>

union Example
{
  float f[2];
  float complex c;
};

int main()
{
  union Example example;
  example.c = 1.0f + 0.0f*I;
  printf("First element of float: %.4f\n", example.f[0]);
  printf("Second element of float: %.4f\n", example.f[1]);
  printf("Real part of complex: %.4f\n", __real__(example.c));
  printf("Imaginary part of complex: %.4fi\n", __imag__(example.c));
  return 0;
}
```

As mentioned in [2], on both big endian and little endian, the first element of f is 1.0000 and the second element of f is 0.0000. The real part of the complex number c is 1.0000 and the imaginary part is 0.0000i.

```
First element of float: 1.0000
Second element of float: 0.0000
Real part of complex: 1.0000
Imaginary part of complex: 0.0000i
```

The byte order of the elements of a floating-point and a complex number is different between big endian and little endian, but the element order is the same.

Conversion rules: As summarized in Table 1, when a value of complex type is converted to another complex type, both the real and imaginary parts follow the conversion rules for the corresponding real types. When a value of real type is converted to a complex type, the real part of the complex result value is determined by the rules of conversion to the corresponding real type and the imaginary part of the complex result value is a positive zero or an unsigned zero. When a value of complex type is converted to a real type, the imaginary part of the complex value is discarded and the value

Table 1 Conversion to/from `_Complex` (`a+b*i`).

Types	Complex	Real	Imaginary
Complex	`conv(a)+conv(b)*i`	`conv(a)`	`conv(b)*i`
Real	`conv(a)+0*i`	-	`0*i`
Imaginary	`0+conv(b)*i`	`0`	-

of the real part is converted according to the conversion rules for the corresponding real type. When a value of imaginary type is converted to a real type other than `_Bool`, the result is a positive zero. When a value of real type is converted to an imaginary type, the result is a positive imaginary zero. When a value of imaginary type is converted to a complex type, the real part of the complex result value is a positive zero and the imaginary part of the complex result value is determined by the conversion rules for the corresponding real types. When a value of complex type is converted to an imaginary type, the real part of the complex value is discarded and the value of the imaginary part is converted according to the conversion rules for the corresponding real types.

3 HDF5 Implementation

Implementing these new types requires modifications of multiple HDF5 components: the core library, the high-level library, the Fortran library, the C++ library, the tools. This implies modification of the corresponding tests that support these components, as well as of the file format.

3.1 Core Library Support

As already described in [3], implementation within the HDF5 library of a new datatype requires the following changes:

1. Detect the size of the new type in configure/CMake along with other native datatypes. After the configuration, a new macro called `H5_SIZEOF_XXX` is defined as in `H5pubconf.h` in the src/ directory under the build directory. If the compiler does not support this datatype, the value of `H5_SIZEOF_XXX` would be 0.
2. Add the new type for detection in `src/H5detect.c`. After compiling `H5detect.c` and running `H5detect`, the properties of the type is generated in `H5Tinit.c`.
3. Declare the variables related to the new type in the top of `H5T.c`.
4. Release the variable in `H5T_term_interface` of `H5T.c`.
5. Make the datatype ID public by defining a macro in `H5Tpublic.h`.
6. Declare the variables for alignments in `H5Tpkg.h`.
7. Add a printout for the new type in `H5_trace` of `H5trace.c` along with other types.
8. Add the prototypes of hard conversion functions for this new datatype in `H5Tpkg.h`. There should be functions between the new datatype and all other native integer types and all native floating-point number types.

9. Add the definitions of the hard conversion functions for this new datatype in `H5Tconv.c`.
10. Add test cases for the new type in the test suite, especially the data conversion test `dt_arith.c`.

3.1.1 Boolean Type

`H5T_NATIVE_BOOL` will be detected and its size[1] defined accordingly. Adding support for the type within the core library should be straightforward. Conversion routines will follow rules already defined by the C standard.

HDF5 boolean type: It is also worth noting that HDF5 used to define a boolean type, referred to as `hbool_t`. This boolean type is currently defined as an integer, therefore for consistency it will now be defined as `_Bool` when possible.

3.1.2 Complex Types

`H5T_NATIVE_FLOAT_COMPLEX`, `H5T_NATIVE_DOUBLE_COMPLEX`, and `H5T_NATIVE_LDOUBLE_COMPLEX` will be detected and their size defined accordingly. Adding support for this type may require some more work as one may need to access real and imaginary parts within the functions that are already defined. Conversion routines will follow rules already defined by the C standard.

3.2 High-level Support

The `H5LT` API makes minimal use of datatypes and support for these additional datatypes will be added (straightforward). High-level read or write make use of datatypes, they will need to be tested using these new types.

3.3 Fortran Support

ISO C bindings are used to provide us with a standard mechanism to pass data between Fortran and C. Fortran `LOGICAL` can be mapped to the C `_Bool` type by using a special integer kind (`C_BOOL`).

Note. We will leave Fortran HL out for now.

3.4 C++ Support

C++ supports `bool` types, as well as complex numbers, which are available through the standard library.

[1] By default, when compiling with GCC, `sizeof(bool)` is 4 when compiling for Darwin/PowerPC and 1 when compiling for Darwin/x86.

3.5 Python Support

Python supports boolean values `True` and `False`, as well as complex types. Therefore these new datatypes make perfect sense for Python users.

3.6 Java Support

Java provides boolean types. Complex numbers are defined through classes.

3.7 Tools Support

The following list of tools must be adapted: `h5dump`, `h5ls`, `h5diff`, `h5import`, `h5repack`, `h5debug`.

4 Additional Questions

Revision History

April 29, 2015:	Version 1 circulated for comment within The HDF Group.
June 10, 2015:	Version 2 circulated for comment within The HDF Group.
June 11, 2015:	Version 3 circulated for comment within The HDF Group.

References

[1] "C Standard Committee Draft," April 2011. N1570 ISO/IEC 9899:201x available at `http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf`.

[2] N. Negherbon, R. Zurob, and N. Ivanovic, "Targeting your applications—what little endian and big endian IBM XL C/C++ compiler differences mean to you," December 2014. Available at `http://www.ibm.com/developerWorks/`.

[3] R. Lu, "A Maintainer's Guide for the Datatype Module in HDF5 Library," January 2012. Available at `http://svn.hdfgroup.uiuc.edu/hdf5doc/trunk/library_maintenance/datatypes/Maintainer_guide.docx`.