Unsupported/Desirable Work for HDF-Java

Peter Cao
The HDF Group

This document describes the unsupported features and desirable work in HDF5 Java products and different levels of support for HDF-Java products.

Contents

1 Introduction

Some of the features are not supported in HDF5 Java products due to a variety of technical reasons. This document attempts to describe these features and give a list of what is supported currently and what is desirable to have in the future.

There are three distinct HDF5 Java Products:

- Java HDF5 Interface (JHI5): the Java Native Interface to the standard HDF5 library.
- Java HDF Object Package: a Java package that implements HDF data objects in an object-oriented form.
- HDFView: a visual tool for browsing and editing HDF4 and HDF5 files.

HDFView is built on the Java HDF Object Package, which in turn is built on the Java HDF5 Interface. Because of the dependence among the HDF-Java products, features that are not supported in the JNI level will not be supported in the object level and HDFView.

2 JHI5

2.1 Untested JNI functions

Among the 462 HDF5 C API functions, 369 functions have been implemented in the HDF5 Java wrapper. All the new functions added to support HDF5 1.8 features and some of the HDF5 1.6 functions in HDF5 JNI are tested. About 140 functions are not tested in the JNI unit test (see the complete list at Appendix A). It is very important to add unit tests for all the JNI functions. The work has been scheduled. Based our experience, the work will take about 70 hours to complete (0.5 hour/per test). Our goal is to complete all the HDF5 JNI tests before next release (V2.8).

2.2 Unsupported functions

There are around 460 functions in the HDF5 library (version 1.8). Ninety three of the functions are not supported in JHI5. Most of the unsupported functions have C function pointers, which is not currently implemented in JHI5. Appendix B is a complete list of unsupported functions.

3 Java HDF5 Object Package

While the Java HDF5 Interface (JHI5) gives the user the full power of the underlying HDF5 functions, the object package is designed for simplicity. The tradeoff is that the object package loses some capabilities that the JHI5 offers.

3.1 Unsupported features

The following are some examples of what is not supported in the object layer:

- Any feature related to unsupported JHI5 functions is not supported in the object layer
- Very limited properties can be passed in the object layer. While the HDF5 library (version 1.8) provides more than 60 functions for handling properties, the HDF5 object layer only uses the

default properties in most cases when creating or accessing a file or objects. Below is a list of examples that users can pass different properties in the object layer:

- H5File.open(int plist), allows users to pass access properties when opening a file
- H5File. getAttribute(int objID,int idx_type, int order), allows users to retrieve attributes in different indexing type and order.
- H5Group. getMetadata(int ...attrPropList), allows users to pass different indexing type and order when retrieving attributes. Same for H5ScalrDS and H5Compound.

- No advanced hyperslab selections. Only simple rectangle subsetting is allowed.

3.2 Examples

We have very limited examples for using the hdf-java objects. Users have asked examples and we need to add more examples.

4 HDFView

The capabilities of HDFView are limited by both the Java HDF5 Interface and by the object package.

Some examples of limitations in HDFView are listed here.

- Any feature related to unsupported JHI5 functions or unsupported operations in the Java Object Package is not supported in HDFView.
- Writing compound data is limited to simple cases, e.g. the base compound fields have primitive types such as integers, floats, characters, etc. HDFView does not write complex compound data, e.g. the type of a compound field is a datadet region reference.
- Writing variable length data is not supported except for datasets where each data point is a single variable length string.

5 Other Java-related issues

The Java language offers many features that benefit the HDF Java Products, including:

- Object-oriented language
- Easy to program, with a rich set of ready-to-use packages for GUI, I/O, Database, etc.
- Automatic memory management eliminates the problems of corrupted pointers and code-level memory leaks
- Platform independence

However, there are some issues related to Java that inhibit the ability of the HDF Java products to support all of HDF5's features with good performance. These issues are listed in the following sections.

5.1 Unsigned Integers

There is no unsigned integer type in the Java primitive types. This causes problems between the Java layer and the C library for unsigned integers. For example, the value of 200, a valid number for unsigned 8-bit integer in C, cannot be presented correctly in the Java “byte” primitive type.

Our current solution is to convert the unsigned integer type to a larger size integer. For example, an unsigned 8-bit integer in C will be converted to the 16-bit short integer in Java. This approach has some shortcomings:

- There is no primitive type in Java to use with an unsigned 64-bit integer. The conversion affects the I/O performance since reading data from and writing data to a file requires the data conversion between unsigned C integers and signed Java integers.

Possible solutions: *there isn’t any other better solution other than convert a unsigned integer to a larger size integer.*

5.2 Function Pointers

C-type function pointers are not supported in Java.

Currently, the C functions (or the library APIs) with function pointers are not implemented in the HDF Java wrapper.

Possible solutions: we either have to hard-code the function name in the JNI code, or tell the JNI code somehow what the name of the function is. More research is needed to find out how this can be done and the effort of the work.

5.3 JNI Memory Debugger

There is no tool or any easy way to debug memory leaks in the JNI C code. A C debug tool will not be able to trace memory leaks for programs starting in JVM. Java debug tools cannot trace memory leaks in C function calls.

Our goal is to prevent this type of memory leak from the implementation stage of the JNI C:

- Making sure that all memory allocated in the JNI C level is freed after use
- Checking the physical memory use at the OS level to ensure there is no memory increase by running the testing program in loops

5.4 Library memory leak

There is no direct way to check resources held by open objects from library calls. For example, leaving datatype and dataspace open in memory causes memory leak. HDF5 Java provides two functions, H5.getOpenIDCount() and H5. getOpenID(), for applications to check this type of memory leak.

5.5 JVM memory limit

Opening a large dataset, e.g. 2GB, may cause Java “OutOfMemoryError” even though there is enough physical memory in your machine since the memory of an Java application is limited by the JVM.

5.6 Poor performance on multiple dimensional arrays

HDF-Java has poor performance on directly handling multiple dimensional arrays. A 1D array allocated in Java is directly used in the JNI C. There is very little overhead between Java and C. For example, when you allocate a float array in C, float a = new float[1000], the same memory of array "a" will be used in C. For multi-dimensional array, there is no directly memory match between Java and C. For example, there is no match between Java int[][] and C int[][]. In Java, int[][] means multiple arrays, i.e. each int[i] is an object. In C int[][] is just an array (1D array in disk) and each int[i] is part of the contiguous memory block. HDF Java allows you to pass an multiple array to C, e.g int[][] ; however, there are a lot of data conversions from C int[][] to Java int[][], which also requires a lot of memory for the conversion too. Not a good idea for both memory use and CPU time. For this reason, we always use 1D Java array to deal with data (in file storage, it is 1D anyway) in the HDF Java object layer. We leave it to the applications to map the 1D array to multiple dimensions. We can improve our currently implementation in the HDF Java wrapper by using the 1D array between Java and C and converting the data to multi-dimension array in Java.

6 Other desirable future work

6.1 Public Widget

Some GUI components in HDFView can be taken out and used to create general-purpose widgets. For example, the code that creating image from raw data can be separated from its main class and used for other applications. Some advantages of the public widgets include:

- Reusable code – the general code can be used by other applications.
- Maintainable code – the code will be easier to maintain.
- Testable code – the code will be much easier to test.

The purpose of work is to investigate the classes in the HDFView package, ncsa.hdf.view, and remove common codes from the main classes to create public widgets.

6.2 Better Movie Play

The goal is able to spatially and temporally subsample images to speed up movie play and add a zoom button for animation. Integrating the movie play into the current image view will only work for small images or low resolution of large images. It will not work for large images. Since movie play requires that all the images should be loaded into memory, loading many frames of large images into memory will cause two problems: a) loading/creating images takes a long time; b) The Java virtual machine will not be able to handle the large memory size.

6.3 Reduce Response Time when Showing Images

HDFView takes a long time to load large images, e.g. images of 8k x 8k. The request is to show the images quickly. We suggest three solutions:

- Incrementally show image while reading data from file instead of waiting a long time to load and display a large image

- Initially show image in low resolution while loading the image with full resolution. For example, we can set the stride to 10 or larger so that HDFView only reads one pixel for every 10 or more pixels.
- Store a low resolution image along with the full resolution image in the file. The low resolution image can be an attribute or another dataset

6.4 Writing Variable Length Data

The Java interface does not work on vlen except of vlen of strings. In order to support vlen (other than strings) we need to map the vlen_t data structure between C and Java, which is not implemented. Many users have requested the support of general variable length.

6.5 Writing Compound Data

Writing compound data is limited to simple cases, e.g. the base compound fields have primitive types such as integers, floats, characters, etc. HDFView does not write complex compound data, e.g. the type of a compound field is a datadet region reference.

6.6 Using Buffered Array to Handle Large Dataset

Opening a large dataset, e.g. 2GB, may cause Java “OutOfMemoryError” even though there is enough physical memory in your machine since the memory of a Java application is limited by the JVM. One solution is to use a memory buffer that dynamically loads data in the view area.

6.7 Automated Testing for HDFView

HDFView has only used a manual checklist to perform an acceptance test, usually before a release. Manual checking is very time consuming and is error prone since the checklist is very long and some tests are very complex. There are tools which can automate not only the acceptance test, but also provide unit testing of HDFView's visual components and interactions.

6.8 HDFView Graphical Design

The current GUI components in HDFView were designed by developers. It would be very helpful if those components can be reviewed by professionals in graphical design so that HDFView will have professional looking.

7 Levels of HDF Java Product Support

The HDF Java support is divided into two levels: what we currently have now and what we desire for. The activities of the two levels of support are summarized the in the following tables.

Table 1 Activities of currently supported

Activity	Description
User support	Providing help for forum discussions
	Assisting help desk

	Identifying and analyzing bugs/problems/features reported from users
General Maintenance	Tracking the daily test failure and identifying what test failed
	Prioritizing tasks and fixing critical bugs for releases
	Releasing products (major, minor, beta, and patch)
	Maintaining web pages
	Making the current products work with the latest version of the library
Support platforms	Major platforms (32-bit and 64-bit Windows, Linux, Solaris, and Mac Intel)
Features from the libraries	Implementing all 1.6 functions in JNI
	Implementing 1.8 functions of high priorities in JNI
	Implementing all 1.8 functions in JNI

Table 2 Activities of desirable future work

Activity	Description
Support platforms	All platforms that are supported by HDF4 and HDF5 libraries and JVM
Features from the libraries	Adding unit tests for all implemented JNI functions
	Implementing all library functions in JNI
	Adding basic features in object layer and HDFView to support HDF5 1.8 features, such as handling external links, using H5Ocopy() for better performance
	Adding other features from HDF5 1.8 and above in the object layer and HDFView
Other advanced features	Improving performance on multiple dimensional arrays
	Writing dataset of complex types, e.g. variable length arrays and nested compound datatypes
	Using buffered array to handle large dataset
	Converting HDF4 objects to HDF5 objects in HDFView
	Importing/exporting XML for HDF5
	Importing/exporting GeoTIFF for HDF5
	Using drag/drop in HDFView
	Providing public GUI widgets
	Using automatic GUI testing

8 HDF4 related work

Most of the work mentioned so far is about HDF5 Java. There are a lot of more things needed to be done for HDF4 Java. The most urgent one is to add unit test for HDF4 JNI and HDF4 objects.

Revision History

January 05, 2011: Version 1 Moved from HDF-Java products support document.

March 22, 2011: Version 2 Added section for untested JNI.

Appendix A: a list of untested functions (143)

H5Aget_num_attrs	H5Pget_filter1	H5Punregister
H5Aopen_idx	H5Pget_filter2	H5Sget_select_bounds
H5Aopen_name	H5Pget_gc_references	H5Sget_select_elem_pointlist
H5Arename	H5Pget_hyper_vector_size	H5Sget_select_hyper_blocklist
H5check_version	H5Pget_istore_k	H5Sget_select_hyper_nblocks
H5close	H5Pget_layout	H5Sget_select_npoints
H5Dextend	H5Pget_nfilters	H5Sget_simple_extent_npoints
H5Dfill	H5Pget_nprops	H5Soffset_simple
H5Dget_storage_size	H5Pget_preserve	H5Sselect_all
H5Fget_obj_ids	H5Pget_size	H5Sselect_elements
H5Funmount	H5Pget_sizes	H5Sselect_hyperslab
H5Gget_comment	H5Pget_small_data_block_size	H5Sselect_none
H5Gget_linkval	H5Pget_sym_k	H5Sselect_valid
H5Gget_num_objs	H5Pget_userblock	H5Sset_extent_none
H5Gget_objname_by_idx	H5Pget_version	H5Sset_extent_simple
H5Gget_objtype_by_idx	H5Pisa_class	H5Tcommit_anon
H5Glink	H5Pmodify_filter	H5Tcommit1
H5Glink2	H5Premove	H5Tcommit2
H5Gmove	H5Premove_filter	H5Tcommitted
H5Gmove2	H5Pset	H5Tcompiler_conv
H5Gset_comment	H5Pset_alignment	H5Tdecode
H5Pall_filters_avail	H5Pset_alloc_time	H5Tdetect_class
H5Pclose_class	H5Pset_btree_ratios	H5Tencode
H5Pcopy	H5Pset_buffer	H5Tget_array_dims1
H5Pcopy_prop	H5Pset_cache	H5Tget_array_dims2
H5Pexist	H5Pset_deflate	H5Tget_create_plist
H5Pfill_value_defined	H5Pset_edc_check	H5Tinsert
H5Pget	H5Pset_external	H5Tis_variable_str
H5Pget_alignment	H5Pset_family_offset	H5Tlock
H5Pget_alloc_time	H5Pset_fapl_core	H5Topen1
H5Pget_btree_ratios	H5Pset_fapl_family	H5Topen2
H5Pget_buffer	H5Pset_fapl_log	H5Tpack
H5Pget_cache	H5Pset_fclose_degree	H5Tset_cset
H5Pget_chunk	H5Pset_fill_time	H5Tset_ebias
H5Pget_class	H5Pset_fill_value	H5Tset_fields
H5Pget_class_name	H5Pset_filter	H5Tset_inpad
H5Pget_class_parent	H5Pset_fletcher32	H5Tset_norm
H5Pget_edc_check	H5Pset_gc_references	H5Tset_offset
H5Pget_external	H5Pset_hyper_vector_size	H5Tset_order
H5Pget_external_count	H5Pset_istore_k	H5Tset_pad
H5Pget_family_offset	H5Pset_layout	H5Tset_precision
H5Pget_fapl_core	H5Pset_preserve	H5Tset_sign
H5Pget_fapl_direct	H5Pset_shuffle	H5Tset_strpad
H5Pget_fapl_family	H5Pset_sizes	H5Tvlen_create
H5Pget_fclose_degree	H5Pset_small_data_block_size	H5Zfilter_avail
H5Pget_fill_time	H5Pset_sym_k	H5Zget_filter_info
H5Pget_fill_value	H5Pset_szip	H5Zunregister
H5Pget_filter_by_id1	H5Pset_userblock	

Appendix B: a list of unsupported HDF5 functions in HDF5 Java (93)

H5Aget_create_plist **	H5Odecr_refcount **
H5Aiterate_by_name *	H5Oincr_refcount **
H5Aiterate1 *	H5Oopen_by_addr **
H5Aiterate2 *	H5Oopen_by_idx **
H5Ddebug **	H5Pcreate_class *
H5Diterate *	H5Pget_char_encoding *
H5Eget_auto1 *	H5Pget_chunk_cache **
H5Eget_auto2 *	H5Pget_driver *
H5Epush1 *	H5Pget_elink_cb *
H5Epush2 *	H5Pget_filter_by_id2 **
H5Eset_auto1 *	H5Pget_mdc_config **
H5Eset_auto2 *	H5Pget_meta_block_size **
H5Ewalk1 *	H5Pget_multi_type **
H5Ewalk2 *	H5Pget_obj_track_times **
H5FDalloc *	H5Pget_sieve_buf_size **
H5FDclose *	H5Pget_type_conv_cb *
H5FDcmp *	H5Pget_vlen_mem_manager *
H5FDflush *	H5Pinsert1 *
H5FDfree *	H5Pinsert2 *
H5FDget_eoa *	H5Piterate *
H5FDget_eof *	H5Pregister1 *
H5FDget_vfd_handle *	H5Pregister2 *
H5FDquery *	H5Pset_attr_phase_change **
H5FDread *	H5Pset_char_encoding *
H5FDregister *	H5Pset_chunk_cache *
H5FDset_eoa *	H5Pset_driver *
H5FDtruncate *	H5Pset_dxpl_mpio_chunk_opt *
H5FDunregister *	H5Pset_dxpl_mpio_chunk_opt_num *
H5FDwrite *	H5Pset_dxpl_mpio_chunk_opt_ratio *
H5Fget_info *	H5Pset_dxpl_mpio_collective_opt *
H5Fget_mdc_config *	H5Pset_elink_cb *
H5Fget_vfd_handle *	H5Pset_filter_callback *
H5Fset_mdc_config *	H5Pset_mdc_config *
H5Giterate *	H5Pset_meta_block_size *
H5Iclear_type **	H5Pset_multi_type **
H5Idec_type_ref **	H5Pset_obj_track_times **
H5Idestroy_type **	H5Pset_sieve_buf_size **
H5Iinc_type_ref **	H5Pset_type_conv_cb *
H5Iis_valid **	H5Pset_vlen_mem_manager *
H5Iregister *	H5Scombine_hyperslab **
H5Iregister_type *	H5Scombine_select **
H5Itype_exists **	H5Sget_select_type **
H5Lcreate_ud *	H5Tfind *
H5Lis_registered *	H5Tregister *
H5Lregister *	H5Tunregister *
H5Lunpack_elink_val *	H5Zregister *
H5Lunregister *	

* -- C function pointer, ** -- low priorities