Ways to improve the quality of HDF-Java products

Peter Cao

More and more users depend on the HDF-Java products. Currently HDFView is the main visualization tools for viewing HDF4 and HDF5 files. With the increasing demanding of the HDF-Java products, there is also a great need for higher quality. This document presents a few ways for improving the quality of HDF-Java products.

There are three distinct HDF5 Java Products; each requires different approaches for quality improvement:

* 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.

# Java HDF5 Interface (JHI5)

* Check library changes -- sometimes, even if there is no change in the signature of library API functions but the change of API behavior may cause unwanted results in the JNI layer, such as memory leak or lower performance.
* Apply test-driven development (TDD) – there is no unit test for the JNI functions that were implemented before. We have experienced problems in the JNI layer because the JNI calls are not tested. As we move forward, especially, adding new JNI functions to support HDF5 1.8 APIs, we will develop a full set of unit tests and apply TDD.
* Ensure memory leak free – Memory leak in the JNI code is hard to track as no Java tool can track the memory usage in the C code and no C-purposed tool, e.g. valgrind or purify, can track the memory at the Java virtual machine. Here are some ways that we can use:
	+ Write high quality code.
	+ Ensure all allocated memory (malloc or calloc) is freed at the end.
	+ Close all open IDs of the library, e.g. if “did=H5Dopen()” is used, H5Dclose(did) must be called.
	+ Run memory leak test before checking in any changes to the source repository.
	+ Exploit ways of debug tools for memory leak at JNI level, such as Sun’s dbx .
	+ Write C program that create JVM (using JNI\_CreateJavaVM) from C and use C debug tools.
* Add a stack of open IDs for tracking memory left by applications (implemented for hdf-java 2.6 release)

# Java HDF Object Package

* Run unit tests for all changes
* Add more test cases for the object package
* Create better example file for the unit tests, e.g. an example file that covers all types of datasets and attributes.
* Ensure memory leak free – run memory leak test for all changes

# HDFView

* Test the full list of HDFView features and update the list when new changes are made.
* Separate GUI components and I/O layer to make HDFView more testable for automatic tools.
* Implement automatic tools for testing HDFView GUI components.
	+ Figure out test cases which end up same result screen output. GUI testing tool compares the screen output with expected output.
	+ Figure out test cases which a user can perform same actions
		- Can make sequence of action and compare the result screen
	+ Generate test cases with a sequence of actions which visit menu items and icons. GUI testing tool records all the user actions and replay.
		- This may not be able to check if the output value is correct but it can check if each actions are in working condition or not.
		- For example: 1. HDF5 file creation with groups and datasets and close the file. The result can be compared since the input will stay same.
		- This can be also used as regression test as evolves
	+ Test case which requires to check correct value and the expected value can be acquired
		- For example: Check if HDF5 Library Version is correct
		- Generate a pair test cases. (Relate to separation between GUI and I/O layer)
			1. Check GUI action: ‘Help -> HDF5 Library Version’. Show related popup window?
			2. Check value: Use the same function to get the value and compare with expected value.
	+ Test case which requires to check correct value and expected value doesn’t exist
		- Do the same as above. Generate a pair test cases.
		- Display the output value and let user choose if the value is correct or not (Y/N).
		- This test will be considered as Half-automated/interactive test. However it will still reduce time and effort from a tester.
	+ Test Monkeys – random non-intelligent testing. It’s more of unexpected action test.
		- This may not fit our case, but it may find some broken feature.
	+ Open GUI testing tools to review
		- Abbot, GUITAR, Pounder, qftestJUI, Eclipse TPTP's Automated GUI Recorder
		- Also refer to <http://www.opensourcetesting.org/>
* Collect ideas and opinions from testers. Especially testing first time for the first impression and second time when not totally used to the current features.

# General rules

* Test all changes.
* Apply TDD.
* Use better example to cover as many cases as possible.
* Follow release procedure.
* Document the cause of all bugs and ways to avoid them.

# Release procedures

* Fixing bugs for the release
* Revise User's guide
	+ First round
* Second roundHDFVIew GUI test
	+ Tested by developer
	+ "Beta Release for ERDC, HUGS, Energistics, NASA, NPOESS"
	+ Add test files/folder
		- Memory leak
		- "examples with 1.8 new objects, 1.6 compatible objects, basic files created from HDFView"
		- "Examples related NASA, NPOES, Engistics, HUGS, and bug fixes"
	+ Tested by staff/users
* Build binaries
	+ for linux and solaris
	+ for Winodws and Mac
* Make HDFView installation program
	+ pack the installation
	+ test the installation
* Revise hdf-java webpage
* Pack and release
	+ Pack the source and binaries
	+ upload to ftp
	+ download from ftp and final test
* Send newsletter for the release

# Revision History

September 12, 2009: Version 1 circulated for comment within The HDF Group.