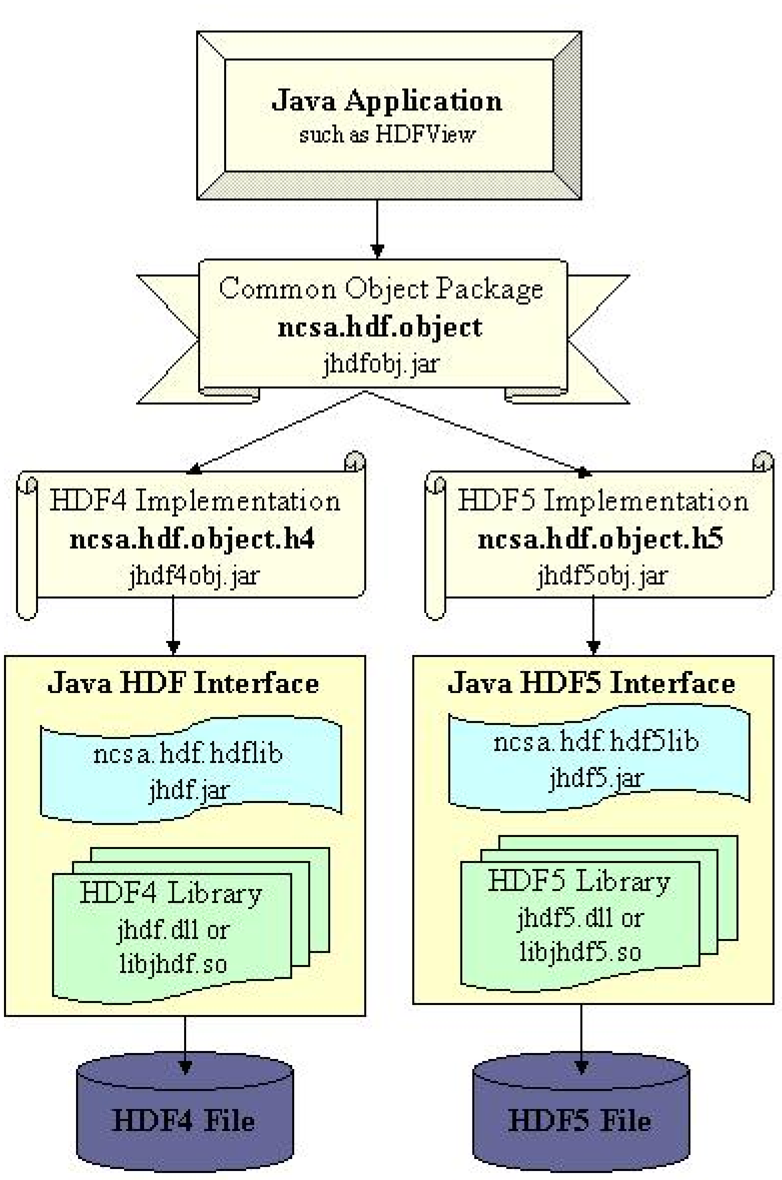
Splitting hdf-java into native project and java project

Current project



The hdf-java project consists of two major layers; the JNI or native code libraries and the Java class libraries.

The native code layer includes the platform specific interface libraries which wrap the HDF library APIs and the Jav jar file which communicates with those interface libraries. This layer uses C for the platform libraries and Java for tl interface jar file. This layer is the same for HDF4 as for HDF5, with 99% of the functions being wrappers around th C Library functions. There are no functions for the HL or C++ libraries. The JNI platform libraries should be built wi the same options as the C Library.

The Java class libraries are the Java object model (ncsa.hdf.object.*) jar files and the HDFView application. This layer uses only Java for all the class jar files and therefore is platform independent.

Problem

The current hdf-java project combines both the platform native interface libraries with the Java object class files and the HDFView application. Because the platform native libraries must be built with the same options as the C library that the JNI functions wrap, the project must be built on every platform and compiler combination supported by the C Library. HDF uses Autotools and CMake to build these libraries. The Java object libraries and the HDFView application only need Java, which is platform independent and only needs to be built once. Java products are best built with the Ant build system, which is used by most Java projects, although CMake does an adequate job of building and packaging Java applications.

Proposal

HDF proposes to split the Java class libraries, which includes the object model jar files and the HDFView application, into a separate project repository. This project will be named hdf-view and will use ant (no CMake) as the build system. Ant allows a property file, entries with a key=value, to be used to locate the platform libraries. The native code project repository will continue to be named hdf-java and use CMake as the build system for the native platform libraries.

The native code will need to build the libraries (including dependent external libraries) and the interface test. Finally, all the libraries will need to be packaged on all the standard platforms that HDF supports, Windows, OS X, and Linux.

The HDFView and object model build and test process will need to include the native package for the platform under test. The release process has three targets; the HDFView and class jar files, or the HDFView and class jar files with platform native libraries, or just the platform native libraries. These jar files will be packaged on each machine designated as the "Distribution" machine with a current version of Java. In addition, each distribution will include a JRE.

The JNI and the HDFView packages will be released for the summer release of hdf-java 2.12.

Because the JNI native libraries wrap the native C Library functions it is logical that these java wrappers should be produced by the HDF4 and HDF5 build process. As with Fortran and C++ libraries the hdf-java JNI libraries should also be an option within the HDF4 and HDF5 build systems. This will reduce hdf-java support significantly by using the current C Library configuration with a moderate increase for the C Library build process. The hdf-java wrappers could be add with a single --with-java option using Autotools or HDF5_BUILD_JAVA option using CMake. The hdf-java project already has Autotools and CMake support.

Adding the JNI native interfaces to the hdf4 and hdf5 library distribution will allow users to use the JNI java jars in their applications without the overhead of the hdfview application. While the HDFView release will still include everything required in a single package.