This is an update on clasJLib, and a useful companion tool: clasJlibCrawler.

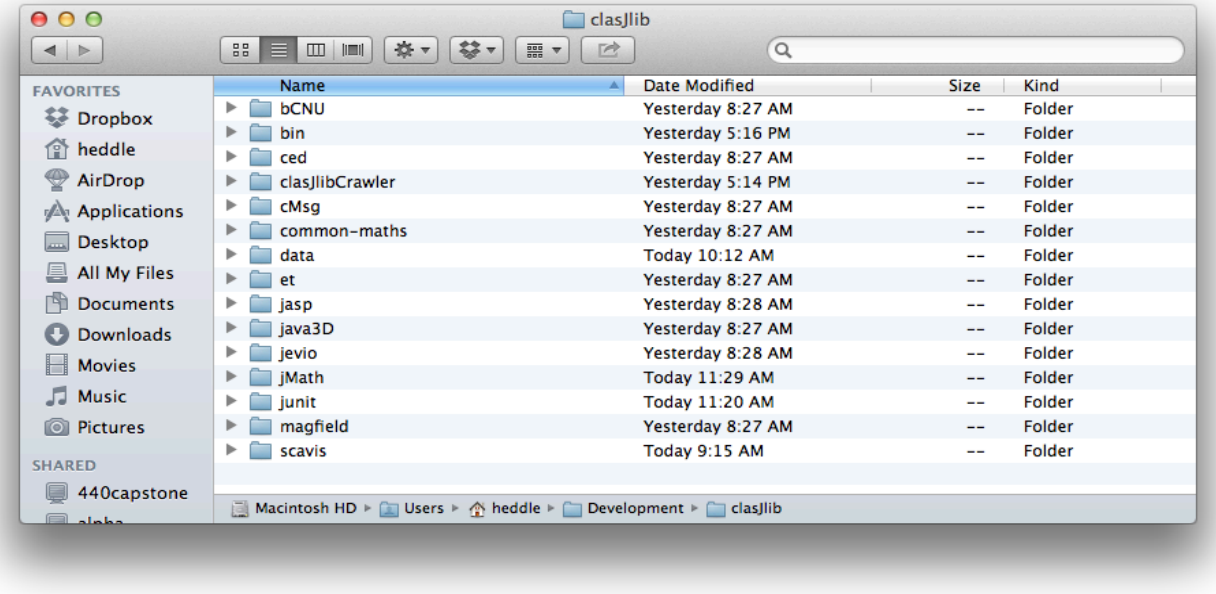
The purpose of the clasJlib project, as described elsewhere, for clasJlib to serve as a lightweight jar file bookkeeper—to bring some order to our proliferation of jars.

Checking out clasJlib from the clas12 svn repository should provide two benefits:

1. All the jars needed to build all clas12 java projects should be in standard locations as described elsewhere, or as figured out by inspection. (At the moment it is not complete. We are adding projects one-by-one. Predictably each project has its own wrinkles.)

2. Scripts should be available to run clas12 java applications, referencing only jar files contained within the clasJlib checkout. As such, it should be a much simpler distribution mechanism for clas12 java applications than clax. (Assuming we can get jToolbox and Clara fit into the clasJlib scheme.)

As of 11/2/13 here are the contents of clasJlib:



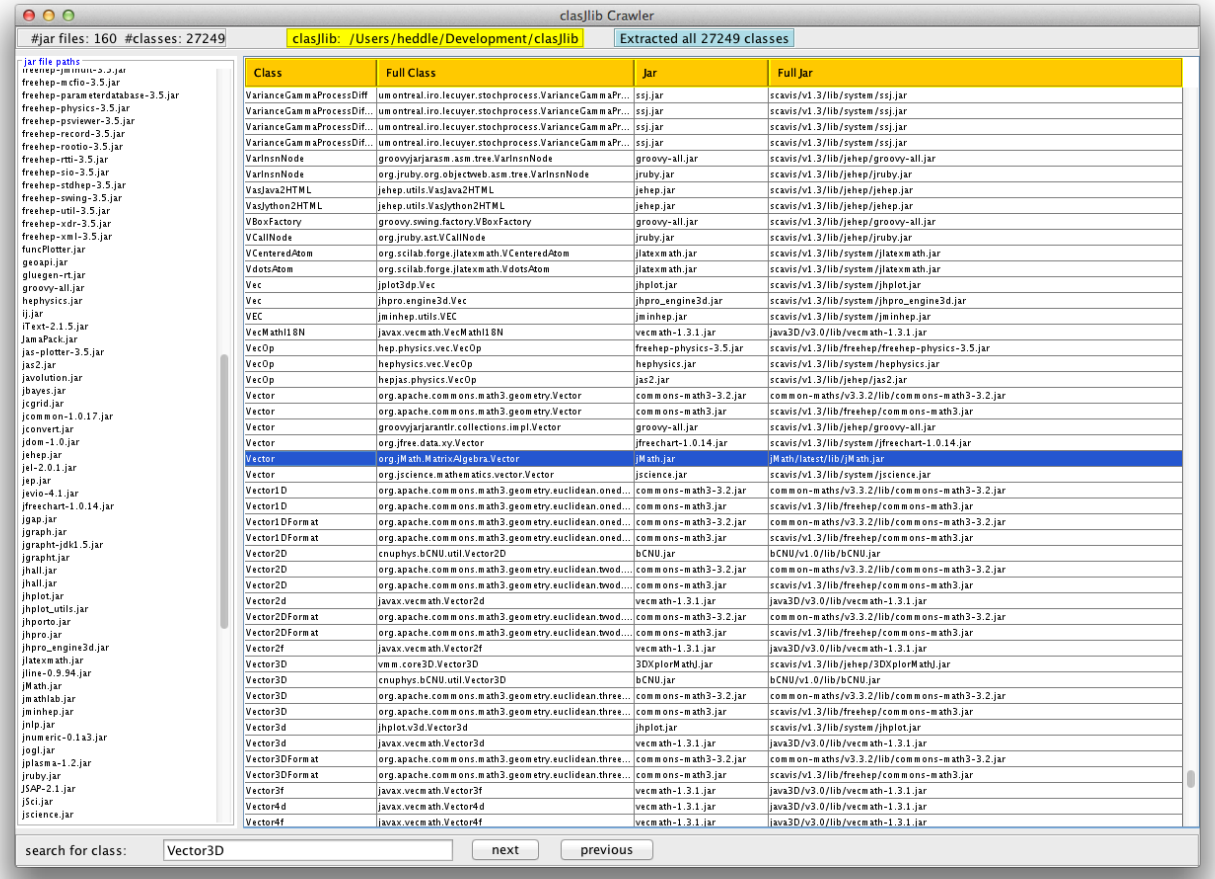
Note that it contains 3rd party libraries, and clas12 projects such as ced and jMath.

All clas12 projects in clasJlib were built against other jars contained therein. This is important—the whole point is that clasjLib is self-contained. This is greatly facilitated by one of the tools we wrote: **clasJlibCrawler**. You can launch clasJlibCrawler using a script in the `clasJlib/bin/latest` folder.

clasJlibCrawler will quickly crawl all the jars in clasJlib and give you a list of all jar files and, more importantly, a table of *all* classes from *all* jars. The picture below is a screenshot. It is hard to see because it is reduced, but you get the idea. The table

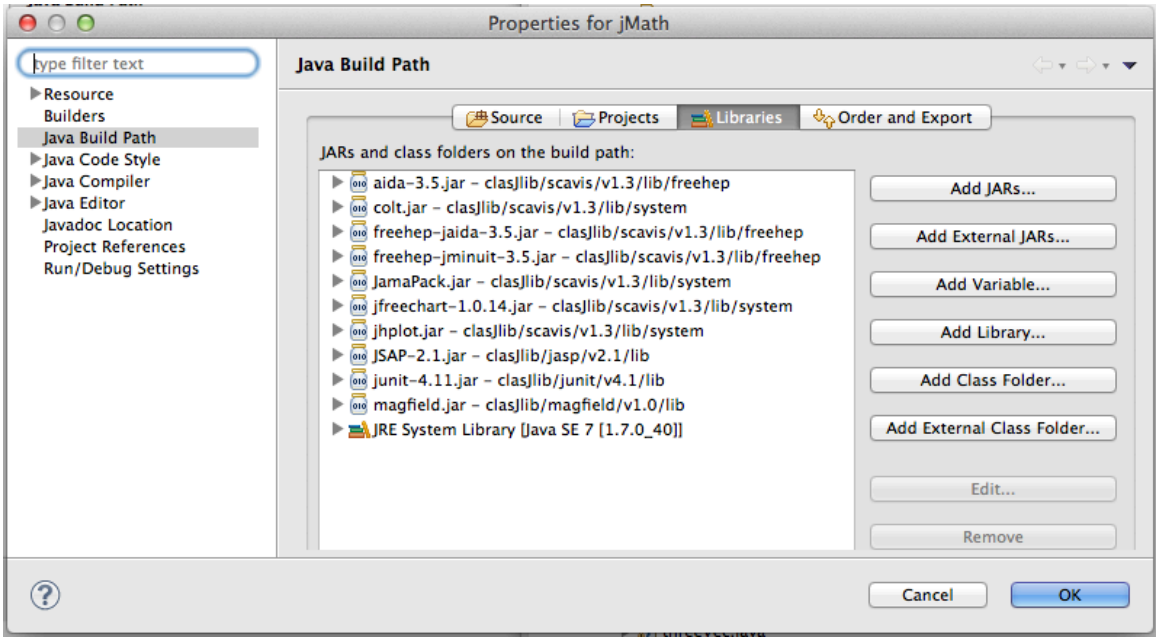
shows every class (27249 of them, at present): its base name (e.g., “Vector”) and its fully qualified name (e.g., org.jMath.MatrixAlgebra.Vector). It also shows the bare jar file name (e.g., jMath.jar) and the full path relative to the clasJlib folder (e.g., jMath/latest/lib/jMath.jar). Clicking on the column headers allows one to sort.

Most useful is the search capability at the bottom of the window. In the picture shown, there was a sort for “Vector”. The result reveals that already in clasJlib there are six classes named “Vector.”



Here is how this tool is helpful. We loaded the jMath project into eclipse. In its build path there were approximately a gazillon scavis jars—many of which we discovered were not needed. We removed everything. Of course the source lit up like a Christmas tree with lots of errors. So, one by one, we found a class that was not resolved, say “HistogramFactory” and searched for it with clasJlibCrawler. The search results told us what jar had to be added to the class path. Typically each jar you add removes many errors—and in this process you find the right jars and also the minimum set of required jars. When completed, the build path for jMath was much cleaner than it was before. as shown below.

Did this take long? Not at all: as you see it needed ten jars, so more or less ten times we searched for an unresolved class and in loading the corresponding jar all the classes that needed the same jar were simultaneously resolved.



Notice that every included jar was loaded via “add Jars…”, not “Add External Jars…” and every loaded jar is contained in clasJlib.

There is some strategy. In searching for “Histogram1D” you will find (at the moment) four jars that have such a class. How to tell which one?

Well, if you are trying to build existing code, such as jMath, the import statement (which will also be flagged as an error) will give you the hint you need. For example, it will be something like: import hep.aida.ref.histogram.Histogram. From the “Full Class” column you match to the fully qualified class name which then indicates the right jar.

The other alternative is to pick less generic unresolved class names (the weirder the better) that are less likely to appear in multiple jars.

Comments are welcomed.