CLAS 12 Jar Files

This report outlines a scheme for managing the many (and sometimes conflicting) jar files used by CLAS12 software.

It is a simple, free approach. It is not as good as more expensive enterprise solutions. It will accomplish something less than 100% of what we'd like, but it aims to be a vast improvement over the current unmanaged anarchy.

It is really little more than a directory structure into which the jars will be placed and versioned.

It is not automated; developers must buy-in. They are responsible for putting their jars into correct location. We do not have a librarian dedicated to building all the projects and placing the jars.

The whole directory structure: *all jars, all versions*, will be available as an svn project named clasJlib in the CLAS12 repository. It will start small and grow quickly.

David Heddle
17 October, 2013

CHRISTOPHER NEWPORT UNIVERSITY

Some more introduction

The clasJlib project contains: documentation, jar files, and the occasional native library (e.g., for the native 3D libraries). The project, under no circumstances, contains source code or build scripts.

Developers have the following requirements:

• They must follow the directory structure.

• They must keep source code and build scripts out of the project.

• Once a jar is placed in a versioned folder (as described below) the developer cannot, under any circumstances, break the API. Bugs can be fixed in old versions, but the API cannot change. This is spelled out more fully on the next page.

Versioning

This versioning applies to the jar files in the clasJlib project, independent of any versioning scheme you may have for your source code. Of course it will helpful if they are lockstep. The scheme is very simple and recognizable:

`X.Y.Z`

`X` is the major release number. Change **`X`** when there is a major change in functionality or the API. All effort should be made to the preserve API through overloading, with the old API signatures marked, if appropriate, with a `deprecated` tag.

`Y` is the minor release number. It represents major bug fixes but essentially the same or only slightly augmented functionality. It can add (modestly) to the API but the previous API must be fully supported.

`Z` is a revision number. It represents minor bug fixes, with absolutely no change to the API.

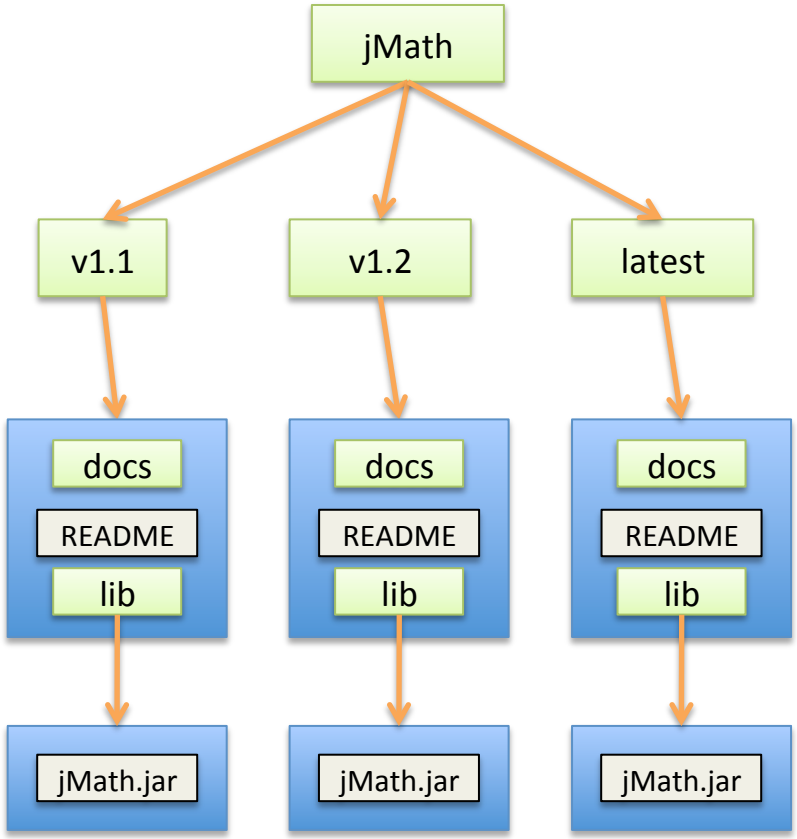
Only **`X`** is required. **`X`**, **`Y`** and **`Z`** are not limited to one digit. 4.32.41 is a later minor release than 4.5.1 (since 32 > 5).

Some more on versioning

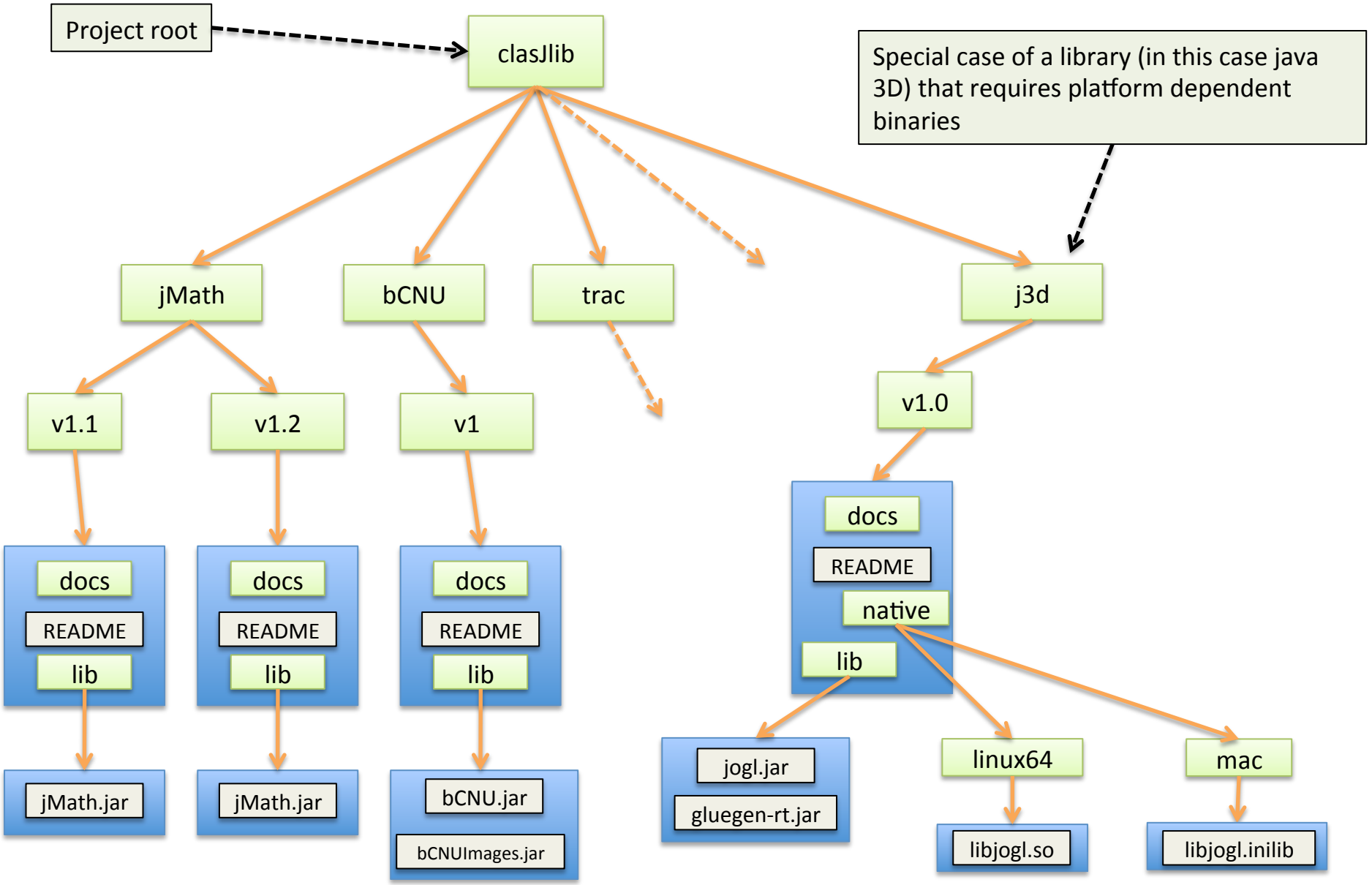
It is recommended that a jar file does not contain a version number in its name, but maintains a generic name. For example, there may be many folders under the `jMath` path, each with unique version numbers, but within each of those folders (in a `lib` folder) you will find `jMath.jar`. (Differing, of course.)

In addition to numbered folders there can be a folder named `latest` that has the latest and potentially unstable release. It may or may not be identical (it probably starts out life that way) with the highest version in the numbered folders.

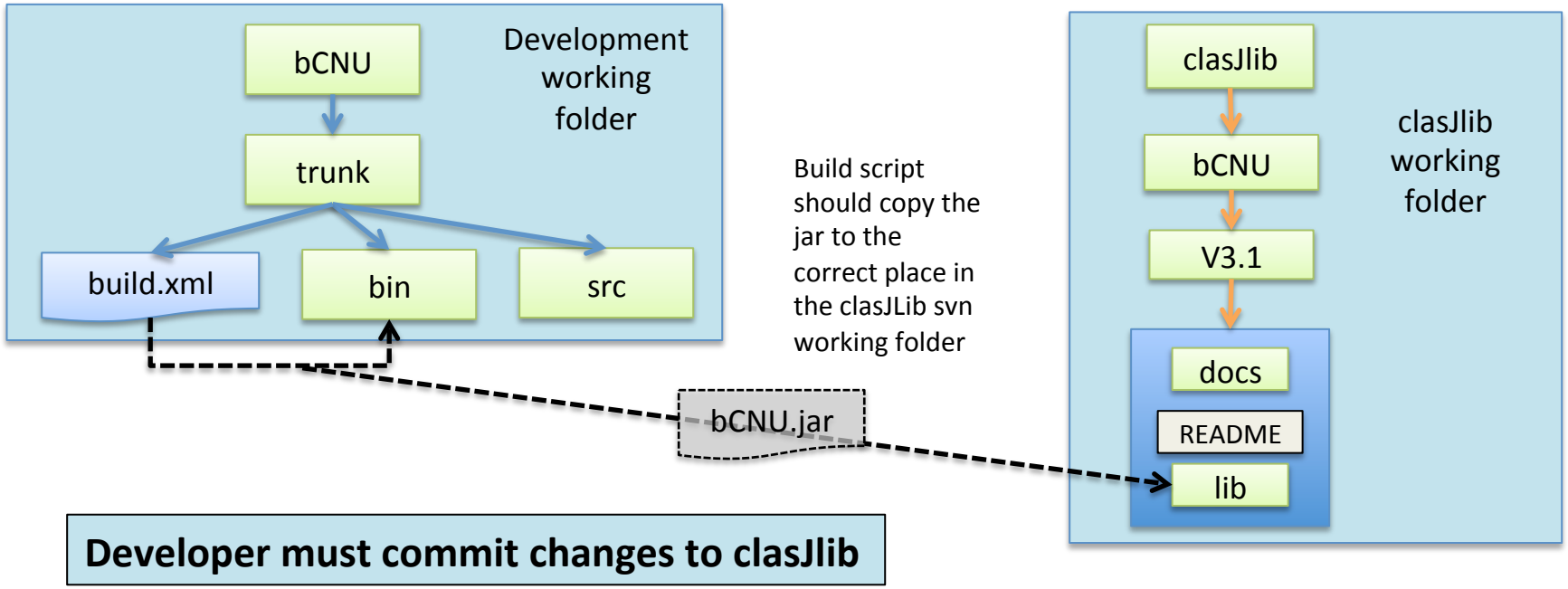
All jars (there may be more than one for some projects) are in the lib folder.



clasJlib structure



Responsibilities? Requires developer buy-in (no code czar)



Developer must commit changes to clasJlib

A target (note no environment variables) in bCNU's *build.xml*

```
<property name="version" value="v3.1b"/>
<property name=”vdir” value="../../clasJLib/bCNU/${version}/lib"/>
<target name="bcnujar">
  <mkdir dir=“${vdir}" />
  <jar
    destfile="${vdir}/bCNU.jar"
    basedir="../bin">
  </jar>
</target>
```