PROPOSED Debian policy for Java

Ola Lundqvist

Stephane Bortzmeyer

PROPOSED Debian policy for Java

by Ola Lundqvist, Stephane Bortzmeyer, and

$Revision:$ $Date:$ Edition

Abstract

This is the proposed java policy for Debian. It begins with a background description, continues with the real policy, some issues to discuss and ends with some advices to java packagers.

The policy covers java virtual machines, java compilers, java porgrams and java libraries.

Table of Contents

Chapter 1. Background

An important warning: this text is a *proposal*. I put it here, publically, so it can be read, discussed, implemented, ignored, etc. It has no sort of endorsement from any authority in Debian or elsewhere.

Feel free to report me (Ola Lundqvist <`opal@debian.org`>) comments and disagrements. I'll put them on this text and forward them to <`debian-java@lists.debian.org`>, if you don't object.

There are several "subpolicies" in Debian. They all want to make the Debian Policy (http://www.debian.org/doc/debian-policy/) more precise when it comes to a specific subject. See the Emacs subpolicy in package emacsen-common for instance. As far as I know, the only subpolicy for a programming language, is that of Perl (http://non-us.debian.org/~hertzog/perl-policy.html/).

This policy is intended to be in a package java-common, whose maintainer will be Java Debian <`debian-java@lists.debian.org`>, when the policy have been officially accepted.

Chapter 2. Policy

A package java-common is created, containing this policy and some basic tools.

Virtual packages are created: *java-compiler*, *java2-compiler*, *java-virtual-machine*, *java1-runtime* and *java2-runtime*.

Packages written in Java are separated in two categories: programs and libraries. Programs are intended to be run by end-users. Libraries are intended to help programs to run and to be used by developers. Both *must* depend on *java-virtual-machine*.

Both are shipped as Java bytecode (`*.class` files, packaged in a `*.jar` archive) and with an "Architecture: all" since Java bytecode is supposed to be portable.

This policy does not yet address the issue of documentation (for instance HTML pages made with javadoc).

2.1. Virtual machines

Java virtual machines *must* provide *java-virtual-machine* and depend on java-common. They can also provide the runtime environment that the package contains (*java1-runtime* and/or *java2-runtime*). If it does not provide the files itself it *must* depend on the needed runtime environment.

I *should* use `/etc/alternatives` for the name 'java' if they are command-line compatible with the Sun's java program.

They *should* have a CLASSPATH predefined which include the needed runtime environment.

If a given source (like the JDK does) brings both a compiler and a virtual machine, you *may* name the compiler package xxxx-dev.

2.2. Java compilers

Java compilers *must* provide *java-compiler* and/or *java2-compiler* and depend on java-common. They *must* also depend on the needed runtime environemnt (*java1-runtime* and/or *java2-runtime*).

They *should* use `/etc/alternatives` for the name 'javac' if they are command-line compatible with Sun's JDK javac. They *should* have a CLASSPATH predefined to include the java core classes need for the compiler.

2.3. Java programs

Programs *must* have executable(s) in `/usr/bin` and be executable. They can be Java classes (using binfmt_misc) or wrappers. In any case, they *must* run without specific environment variables (see Policy 3.8 (http://www.debian.org/doc/debian-policy/ch3.html#s3.8)), for instance CLASSPATH. They *must* respect the Policy rules for executables (for instance a manual page per executable, see Policy 6.1 (http://www.debian.org/doc/debian-policy/ch6.html#s6.1)).

If they have their own auxiliary classes, they *must* be in a jar file in `/usr/share/java`. The name of the jar *should* folow the same naming conventions as for libraries.

Programs *must* depend on *java-virtual-machine* and the needed runtime environment (*java1-runtime* and/or *java2-runtime*).

There is no naming rules for programs, they are ordinary programs, from the user point of view.

2.4. Java libraries

Libraries are not separated between developers (-dev) and users versions, since it is meaningless in Java.

Java libraries packages *must* be named libXXX[version]-java (without the brackets), where the version part is optional and *should* only contain the necessary part. The version part *should* only be used to avoid naming colisions. The XXX part is the actual package name used in the text below.

Their classes *must* be in `jar` archive(s) in the directory `/usr/share/java`, with the name `packagename[-extraname]-fullversion.jar`. The extraname is optional and used internaly within the package to separate the different jars provided by the package. The fullversion is the version of that jar file. In some cases that is not the same as the package version.

Some package *must* also provide a symbolic link from `packagename-extraname.jar` to the most compatible version of the available `packagename-extraname-version.jar` files.

All jar files *must* have a well-documented CLASSPATH, so that developers should know what to add to their wrappers.

This applies only to libraries, *not* to the core classes provied by a the runtime environment.

2.5. Main, contrib or non-free

About politics: packaging Java stuff changes nothing to the rules Debian uses to find if a program is free or not. Since there are not many free Java tools, keep in mind the following:

- If your source package can compile (correctly) only with non-free tools (the only free Java compilers seem to be guavac, gcj and jikes, it cannot go to main. If your package itself is free, it *must* go to contrib.
- If your binary package can run only with non-free virtual machines (the only free Java virtual machine seems to be kaffe - and the one included in libgcj), it cannot go to main. If your package itself is free, it *must* go to contrib.

Chapter 3. Issues to discuss

The following points are discussions about the policy, either because they have to be studied more, or are controversial.

- Name and existance of the repository. It was removed in the latest version.
- The symbolic links in /usr/share/java be made by a script instead, similar to the c-libraries.
- Core classes (`java.*`). More study needed.
- Sun's Community Source Licence. Can we use it? How? Where can we find the text (http://www.sun.com/software/communitysource/faq.html)?
- All jars must have a good CLASSPATH documentation, but how should it be documented. The best solution is probably in some computer parsable format to make it even easier for the developer.

  It should exist some tool to parse this. How should it work?

  Should the tool also be used to create the necessary symbilic links needed by servlets under tomcat?

- Should there be a default classpath, similar to a repository? Which jars should be included in that? A standard and one optional part? If there are a default classpath (in the wrapper) how should it be overridden?
- How to check for a good enough jvm, and to select a proper one to use. Are /etc/alternatives not good enough?
- Should the jvm internal classes be possible to override entirely and how?

Chapter 4. Advices to Java packagers

Warning: they are just advices, they are not part of the policy.

- Be sure to manage all dependencies by hand in `debian/control`. Debian development tools cannot find them automatically like they do with C programs and libraries (or like dh_perl does it for Perl, a volunteer to write dh_java would be welcome).
- You can suppress many calls in `debian/rules` which are meaningless for Java, like dh_strip and dh_shlibdeps.
- Source package handling is painful, since most Java upstream programs come with `.class` files. I suggest to make a new `.orig` tarball after cleaning them, otherwise, dpkg-source will complain.
- Java properties files are probably better under `/etc` and flagged as configuration files (this will be integrated in the policy, one day).