`java4unix`: a solution for integrating Java into the UNIX environment

Luc Hogie
INRIA/CNRS/Université de Nice-Sophia Antipolis

June 27, 2012

Abstract

Beneath its inherent strengths, for the sake of its cross-platform character, Java was never offered a proper integration into operating systems. In particular in UNIX, the sole way to run a Java program is to explicitely invoke the JVM with the main class of the application as an argument. Worse, for this to work, the path to classes must have priorly been set by hand as pointing to the correct JAR files. In addition to this problem of poor integration, Java's API, though very rich, has no shortcuts to common operations. Then writing simple programs in Java often requires long coding. As a consequence programmers are reluctant to use Java as a programming language for UNIX. Most often they rather opt for bash, Python, C/C++, Perl, TCL, etc.

Java4unix comes as a solution which consists in:

- on the developer side: a releasing tool that exports your application and all its dependancies to a website;
- on the user side: an installer which downloads and installs released applications on the local computer as standard UNIX applications.

Additionally, an application that uses the facilities of java4unix benefits from:

- automatic management of the command line;
- the support of configuration files;
- an API for POSIX tools from Java;
- an general-purpose API allowing the fast-coding of common operations.

Contents

1 Description

Beneath all of its strengths and for the sake of its cross-platform character, Java was never offered a proper integration into operating systems. In particular in UNIX, the sole way to run a Java program is to explicitely invoke the JVM with the main class of the application as an argument. Worse, for this to work, the *classpath* must have priorly been set by hand as pointing to the correct JAR files. In addition to this problem of poor integration, Java's API, though very rich, has no shortcuts to common operations. Then writing simple programs in Java often requires long coding. As a consequence programmers are reluctant to use Java as a programming language for UNIX. Most often they rather opt for bash, Python, C/C++, Perl, TCL, etc.

`java4unix` proposes a new alternative described in this document.

2 Features in brief

- network installation of applications;
- access to UNIX specificities through a Java API;
- management of the command-line;
- etc....

3 Installing `java4unix` applications

A `java4unix` application comes in the form a ZIP archive file. To install it, you need to use one of the two `java4unix` installers:

j4uai , the normal installer (you must have downloaded the archive before);

j4uni , the network installer (will use the archive of the last version, which is on the web).

`java4unix` installers are shell scripts. You can download them at the following URLs:

`http://www-sop.inria.fr/members/Luc.Hogie/java4unix/j4uai`

`http://www-sop.inria.fr/members/Luc.Hogie/java4unix/j4nai`.

Because these installers are subject to modifications/improvement, it is not recommended to download them. Instead, it is adviseable to use the online version throught the following command line:

3.1 Normal installation

```
curl -s http://www-sop.inria.fr/members/Luc.Hogie/java4unix/j4uai | sh -s zip_file
```

3.2 Network instalation

```
curl -s http://www-sop.inria.fr/members/Luc.Hogie/java4unix/j4nai | sh -s application_name
```

You can get a list of registered applications at the following URL: `http://www-sop.inria.fr/members/Luc.Hogie/java4unix/db/websites/`

An installer for any registered application can be produced by issuing the following command:

```
1 echo 'curl -s http://www-sop.inria.fr/members/Luc.Hogie/java4unix/j4nai | sh -s application_na
```

3.3 Directories

If you run the installation as a normal UNIX user, the software will then be installed in the `$HOME/.`*name* directory. The following sub-directories will be created.

`$HOME/.`*application*

`$HOME/.`*application*`/lib` is where all the required jar files are;

`$HOME/.`*application*`/bin` is where all the user-invokable scripts are. This directory shall be added to the `$PATH` environment variable, so as the user will not have to type the full path the application script every time he execute one.

Classes in *name*`-`*version*`.jar` that extends `java4unix.AbstractShellScript` will entail the creation of executable files in `$HOME/.`*name*`/bin/`

3.4 Installation on Cygwin

If you plan to use Java on Cygwin, you need to be aware that the JDK is not a Cygwin application but a native Windows one: it is meant in NO way to operate with Cygwin. This leads to a number of issues (location of the home dir, file/path separators, etc).

This said, a `java4unix` application installed onto a Cygwin system will be installed in the user home directory in the sense of Cygwin (e.g. `/home/`*username*). The user's home directory in the sense of Windows (typically `C:\Documents and Settings\`*username*) is not considered.

3.5 Updating a `java4unix` application

To update an already installed application, run the installer again.

3.6 Uninstalling an `java4unix` application

To uninstall an installed application, delete its root directory `$HOME/.`*application*.

4 The command line

4.1 Definitions

A *positional parameter* is any text that comes after the script name on the command-line. There exist two kinds of positional parameters:

- An *option* is a positional parameter starting by `-`;

- An *argument* is any positional parameter that is not an option.

Arguments indicates the application *what* data to process, while options indicates the appliation *how* to process it.

4.2 Options

4.2.1 Long name and short name

An option is identified by it *long name*. It optionally has a shortcut, referred to as its *long name*. The `--verbose` option and its `-v` shortcut is a common example of this.

Long options are of the form `--`$l+$, e.g. they must start with `--`. Short options are of the form `-`l, where $l \in [a - zA - Z]$. They must start with `--`.

4.2.2 Valued options

An option might take a value.

The way the value is given to an option depends on wether its long name or short name is used. For example, the value for a short option `-o` is given by `-o` *`value`*; while the value for the long option `--fubar` is given be `--fubar=`*`value`*.

4.2.3 Declaring an option

From the developer point of view, options are defined via `java4unix.OptionSpecification` objects:

```
1 new OptionSpecification(long_name, short_name,
2         value_regex, default_value, description)}
```

For example, the definition:

```
1 new OptionSpecification("--repetitions", "-r", "[0-9]+",
2         "1", "Specifies the number of repetitions")}
```

declares a new option that can be invoked via either `--repetition` or `-r`, and which accept a positive integer as its value. If the value is not specify at runtime, the default value 1 will be considered.

4.2.4 The special case of the `--` option

The `--` option indicates than the following item on the command-line is not an option but an argument. This enables items starting by the character `-` to be treated as arguments.

For example, consider you want to process a file whose the name starts by `-`. If you do not use the `--` option, the name of the file will be considered as an option.

4.3 Arguments

4.4 $0

In bash scripts, the argument `$0` gives the filename containing the script. In `java4unix` this file is not mixed to the argument list. instead it is given as the first parameter to the user-implemented `run()` method.

5 Configuration file

`java4unix` commands optionally use a configuration file located at:

```
1 $HOME/.application/command_name.conf}
```

The syntax for the configuration file is the one of Java properties, that is is it a line-by-line sequence of $key = value$ pairs.

This configuration file allows the same keys as the one defined by the command line. At startup, the values defined in the command-line override those defined in the configuration file.

6 Data files

Applications may need to store data in files. Instead of spreading files anywhere on the filesystem, the application should use they own file space, located in `$HOME/.application/data`.

For example if you need your application to store data on the local `fubar.dat` file, you are advised to write such a code to get a reference on the file:

```
1 public RegularFile getFubarFile()
2 {
3         return getDataFile("fubar.dat");
4 }
```

7 POSIX utility methods

- `Collection<AbstractFile> find(Directory file)`
- `Relation<Integer, File> findFileInodes(File searchRoot)`
- `Directory getUserHome()`
- `String cat(RegularFile... file)`
- `boolean isLinux()`
- `boolean isMacOSX()`
- `double[] getLoadAverage()`
- `void chmod(AbstractFile file, String mode)`
- `boolean isSymbolicLink(AbstractFile file)`
- `String getFileUser(AbstractFile file)`
- `String getFileGroup(AbstractFile file)`
- `String getFilePermissions(AbstractFile file)`
- `void makeSymbolicLink(AbstractFile source, AbstractFile link)`
- `AbstractFile getSymbolicLinkTarget(AbstractFile symlink)`
- `List<Directory> retrieveSystemPath()`

- `RegularFile locateCommand(String cmd)`
- `boolean commandIsAvailable(String name)`
- `Map<String, RegularFile> listAvailableCommands()`

8 Writing a basic script

At installation time, every class that extends `AbstractShellScript` will entail the generation of an executable file into `$HOME/.application/bin`.

8.1 Licensing

8.2 User input/output

8.2.1 Printing to stdout

```
1 printMessage(message)}
```

8.2.2 Printing to stderr

```
1 printWarning(message)
```

```
1 printNonFatalError(message)
```

```
1 printFatalError(message)
```

A call to the last method prints a message to the standard error and subsequently forces the application to stop.

```
1 printDebug(message)}
```

```
#Debug:
```

8.2.3 Reading from stdin

```
1 readUserInput(prompt, regexp)
```

```
1 select()
```

8.3 Error handling

The model for error management in `java4unix` is non-permissive, meaning that the application should not try to recover from any error. Instead it should throw an exception fed with an appropriate error message. `java4unix` will catch the exception at the lowest layer and prints the error message to *stderr*. If a message is missing, or if the `--Xdebug` option is set, the full exception trace gets printed.

9 Exporting to an installable archive

`java4unix` releaser is a tool for the releasing of Eclipse Java project. It basically creates the JAR tarballs and put them at the appropriate location so that the website of the project will be automatically updated.

9.1 Configuration

`java4unix` releaser needs to know where is your workspace. The default location of the workspace is the current directory. You can configure the workspace location by using the `-w` on the command line or by adding the following file into your `java4unix` releaser configuration file, located at `$HOME/.java4unix-eclipse-release/epr.conf`:

```
1 workspace=path_to_your_workspace
```

9.2 Operation

`java4unix` releaser takes as input an Eclipse Java project. It computes 2 files:

- a JAR file, which contains the bytecode of the application and, optionally, its source code;
- a ZIP file, which contains the JAR file of the application as well as all the JAR files the application depends one. The ZIP file is hence so-called *self-contained*.

Three optional operation are then performed:

- these two files are copied to the website of the application;
- the source code is tagged into the subversion repository;
- the manual of the project is optionally copied to the website of the application.

9.3 Project settings: the .java4unix file

The description file for a given project is located at the root of the project, along with Eclipse's `.classpath` and `.project`.

Its syntax uses the common paradigm *key* = *value*. Three keys have been defined, as described in the following.

9.3.1 Updating the website

The website of the project is assumed to accessible throught SSH (SCP). Its root website must contain a directory named `releases`. This is where the release will be copied to.

```
1 scp_destination = location
```

The location is in the form *host:path* where *host* is the name (or IP address) of the SSH server and *path* is the path in the filesystem heading to where the website of the project is located.

You may optionally provide the public URL for the project. This will be used to check that the releasing did its job.

```
1 website = url
```

9.3.2 Adding a download link for the last release

In order to add to the HTML page a link that refers to the last release tarball, you need to add the following PHP code:

```
<a href="releases/epr-<?php include 'releases/last-version.txt'; ?>
        .selfcontained.zip">foobar
</a>
```

where *foodbar* is the clickable text displayed by the web browser.

9.3.3 Tagging the SVN

If the source code of the project is stored into a subversion repository, `java4unix` releaser will tag it, meaning that it will create a time-stamped copy of the current source code into the `/tags` directory of the repository. You then need to specify the following line:

```
svn_repository = url
```

where *url* is the URL of the root of the subversion repository.

9.3.4 Distributing the source code (or not)

f the license of the project allows the distribution of the source code, you need to use the following key:

```
source_is_public = true | false
```

If the source is public, it will be embedded into the JAR file.

9.3.5 Providing documentation

`java4unix` releaser can automatically update the user manual of the project if you specify the path to its file, just like:

```
manual = path
```

The file is copied at the root of the project website (where the HTML file should be) and its name of the file remains unchanged. You need to add the following HTML code to add a download link:

```
<a href="my_file">Manual</a>
```

Where *myfile* is the name of the documentation file. The PDF format is strongly recommended.

If you do not specify the `--nojavadoc` flag, `java4unix` releaser will generate a javadoc for all the classes of the projects and upload it to the `javadoc` subdirectory of your project website. The following HTML code will add a link to the javadoc into your main HTML page:

```
<a href="javadoc/index.html">source documentation</a>
```

9.3.6 Example

As an exemple, take a look at the `.java4unix` file of the Grph project:

```
website=http://www-sop.inria.fr/members/Luc.Hogie/grph/
source_is_public=true
scp_destination=mascotte.inria.fr:web/mascotte/software/grph/
svn_repository=svn+ssh://lhogie@scm.gforge.inria.fr/svn/grph/
manual=/home/lhogie/latex/grph/grph.pdf
```

9.3.7 Obtaining information of a project

In order to get information (number of lines, paths, etc) on a given project, you can use the `java4unix-eclipse-info` command.

9.3.8 Releasing a project

Releasing a project is done by the use of the `java4unix-eclipse-release` command. Just like `java4unix-eclipse-info`, `java4unix-eclipse-release` takes as argument the name of the project you want to release. The releasing process is entirely automatised.

9.4 Export by hand

The tarball called *`name-version`*`.selfcontained.zip` must contain at least an entry named *`name-version`*`.jar`. You may include other jar files in the archive, these jar files will then be treated as dependancies.