Boxer

A Java Packaging Tool for Eclipse Projects

by

Udo Schuermann

This document was created with LibreOffice 4.1 on Kubuntu 13.04

The header font is Fontin, the body font is Gentium, sample text uses DejaVu Sans.

Boxer uses Mark James awesome Silk icon set: http://www.famfamfam.com/labs/icons/silk

Table of Contents

1 Introduction

Boxer generates JAR files from an Eclipse Java[1] project.

This may not sound exciting until you learn that it can merge dependent project JAR as well as library file contents into your primary JAR, or reference them as external resources; it can generate archive files (.zip, .tar, .tar.gz) complete with all libraries and dependent projects JAR files to distribute your work; optionally it can include the source code in your archive, add arbitrary files to the JAR and/or the distribution archive, and build JNLP files to launch your Java software straight from a browser. In short, Boxer is intended to be a one-click solution for creating a distributable package of your project.

Configuring a project tends to require a graphical desktop to run the GUI, but the packaging process is easily executed from a script on the console by giving it the name of one or more projects (project names are matched partially, with exact matches taking precedence).

And yes, naturally we're using Boxer to package Boxer itself, as well as other projects. We eat our own dog food, as the saying goes.

It should be noted that Boxer does not trigger Eclipse compilations, and has no idea about the state of your project, whether it is properly built, or the code is full of errors. Boxer simply packages what the Eclipse project's configuration lays out. It remains your responsibility to ensure that your project is functional and distributable and, if the project's structure has changed, ensure that Boxer is aware of the changes by adjusting Boxer's view of the project again.

Although Boxer goes to some length to be easy to use, and uses tool tips extensively to educate you as to what it expects, this document should serve you as a reference in case you require more detailed explanations.

Note: Boxer is aiming to support Java 8's JavaFX GUI but at the time of this writing, it is not yet prepared to do more than present a small dialog stating that it is not ready for JavaFX 8. Boxer can be forced, in such situations, to launch the Swing GUI by giving it the `-7` switch on the command line.

1 Java, but not JEE projects: Boxer is not prepared to work with, and not intended for JEE project at this time.

2 Boxer's Windows

Boxer begins with a project selection window. A workspace selector lets you decide whether to see projects in all workspaces, or only projects in a specific one. You can add/register workspaces with the 'Add...' button, and forget/unregister them again with the 'Forget' button.

Boxer will remember registered workspaces, and also the choice you make (show projects in all workspaces, or only show projects in a specific workspace).

To open the a project's Configuration Window, either double click the project name or click the 'Configure' button (near the bottom of the window). You can open multiple project configuration windows, but in order to avoid opening potentially dozens or even hundreds of windows, Boxer requires you to open each project's configuration window one at a time.

To build a project, click the 'Package' button (near the bottom of the window). You can also build a project from its Configuration Window; you can also select and build multiple projects all at once from the project selection.

2.1 The Main Window

Boxer remembers your 10 most recently chosen workspaces and lists within the currently selected workspace all projects for which Boxer can find a .classpath file. The .classpath file is built and maintained by Eclipse. Not all Eclipse project types have a .classpath file. If Boxer does not show you a particular project, then the lack of a .classpath file is the reason, and Boxer cannot work with that project.

You may press the 'Pick' button to select a different workspace, type one into the drop down, pick another from the drop-down, or remove elements from the drop down: Removing an element requires that you select it, then delete all characters *and press the ENTER key* to register the removal.

To configure how Boxer builds a project, either double click the project name, or select the project and then click the "Configure" button.

To package a project that has been configured previously, select the project and click the "Package" button. If you choose to package a project without configuring it first, you will produce a JAR

file in your account home directory with the name of the project and some reasonable defaults which may or may not work for you:

1. A likely execution entry point ("Main-Class") is chosen for you.
2. Source code will *not be included.*
3. Library file resources will be included in the JAR under the path that they are defined in the project. This may not match how your code expects to access the files, however.
4. Library JAR files are referenced as external resources on the JAR's classpath.
5. JAR files of dependent projects are referenced as external resources on the JAR's classpath.
6. Binaries are packaged into the JAR.

The JAR is not signed, and a distribution archive will not be built from the result. Ideally this is a minimally functional JAR, but you are strongly urged to configure your project first. In particular, choose a distinct directory for the output of various files, to simplify cleanup later.

We will discuss project configuration first, packaging after that.

2.2 The Configuration Window

Typically, four or more tabs will be available in the Configuration Window, depending on the complexity of your project. Each of the tabs is discussed in this section. You can either save the project configuration with the "Save" button, or discard changes (with one exception) using the "Cancel" button. The exception involves changes made editing the Signing Properties file.

2.2.1 The Main Tab

2.2.1.1 Package Build Directory

The package build directory represents the workspace directory where Boxer assembles library and JAR files according to the various distribution-related paths that you have specified. For example, you may have specified (we'll explain below exactly how) that several required third-party libraries are to be referenced in a "third-party" sub-directory, while a utility library that you wrote yourself is to be referenced without a sub-directory reference. Boxer assembles all these files, including your primary JAR in this package build directory.

2.2.1.2 Main Class

The Main Class is your program's entry point. Boxer scans all your binaries and lists in this drop down field all the classes that offer a method with the following signature:

```
public static void main( String[] )
```

You may pick one of these, or type the name of another class. If you do not want any "Main-Class" specified in the JAR file's manifest, thereby preventing the JAR from becoming directly executable, you should blank out the contents of this field: Simply delete all characters in it.

2.2.1.3 Digital Signature

Digital signatures are required nowadays if you plan to distribute your project as an Applet or have your customers launch it via Web Start. Boxer does not store the credentials for your keystore and signing key directly, but in a separate file of your choosing. The reason is that Boxer's configuration is stored in the project directory, and may well be checked into your source code repository along with all other files, and is therefore shared between different developers. When referencing an external file for the credentials, each developer can have his/her own keystore, so long as the credentials file is kept in the same place (because its name is referenced in Boxer's `.boxer-conf` file.

The “Secrets” entry specifies the file that stores the credentials. It is a standard Java Properties file, but Boxer can encrypt the sensitive passwords in this file, the password for your keystore as well as the password for the key you use to sign your JAR files. The credentials file should not be checked into your source code repository or shared with anyone, because the certificates and keys identify you, and if someone else got a hold of the keystore and passwords, they could sign anything they wanted, including malware, and the software would then be certified to have come from you. Keep it secret, keep it safe, as Gandalf said in the movie.

Once you have selected a file, whether it exists or not, you can click the “Edit” button. If the file exists, and contains encrypted passwords, you will have to provide the password that decrypts them, and you cannot edit the file (with Boxer) until you provide the correct password.

> NOTE: If you forget the password for this file, but still remember the passwords for your keystore, you can use an external text editor to remove the “encryption-*” lines from the credentials file, and change the two Base64-encoded password lines to plain text passwords.

If the file's passwords are not encrypted (perhaps you prefer that, so that other tools can use the same file), then pressing the “Edit” button brings up the editor without requiring you to enter any password first.

The editor consists of two sections, one for the information relating to encrypting the keystore and signing key passwords, and the other for the four “actual” values that will be fed to the JarSigner tool for signing your project JARs.

The signing credentials are encrypted so long as the checkbox is selected that indicates to encrypt the fields, and you enter a password into the two fields (they must match to keep you from making embarrassing typographical mistakes with your password). When you unselect the checkbox and save the file, the encryption will be dropped and passwords will be stored in plain text again.

2.2.1.4 Create Digest Signatures

Digest signature files a standard way of publishing a cryptographic digest (hash) that recipients of a file (including software) can use to verify that the received file matches what the author intended, and it was not altered (perhaps to nefarious effect) by someone else.

Using digests, when available, is a smart way to gain confidence the file is an unadulterated copy.

You can select up to three (or none) files with different algorithms; distributing multiple hashes is a good idea, because a determined attacker might be able to figure out how to trick one digest to

give the same result even though the file was altered in extremely clever ways (this is called a hash collision), but succeeding against two or even three different algorithms becomes exponentially more difficult:

1. SHA-256 — This generates a 256-bit digital hash, a very good choice, the best one if you pick only one digest to distribute. Note that there is also SHA-384 and SHA-512 which Boxer will support once they become standard features of the Java platform.
2. SHA-1 — A decent choice, but no longer the best. It is still a good secondary one, to ship along with SHA-256.
3. MD5 — A choice that should be a secondary, but not your primary one, as MD5 is becoming increasingly vulnerable with today's distributed networks and the staggering computing power that they can bring to bear for a determined attacker.

2.2.1.5 Distribution Archive

A distribution archive is a one-file package to distribute your software. Boxer recognizes four distinct file name extensions:

1. .zip — The ZIP file format. Good compression, but the archive directory is at the end of the file, and if that part is missing, no part of the file is accessible. This tends to be a pointless concern for distributing software, however.
2. .tar — The TAR file format, a streaming format that does not by itself support compression. Its primary value lies in being a stream to which additional data can be easily appended, and if the file is chopped off at one point, everything up to that point is still accessible.
3. .tar.gz — The GNU Zip (gzip) compressed version of TAR, a .tar file that has been gzip compressed. All the advantages of .tar but with good compression (slightly better compression than Zip).
4. .tgz — An MS-DOS/MS-Windows compatible file name extension for .tar..gz (otherwise the same thing, gzip compressed .tar).

2.2.1.6 Splash Screen Image

The name of the picture that you want Java to use for the splash screen when the application starts. This must be the name of the file as it appears in the archive when it is built. This requires a little knowledge from you, not just to pick the correct file, but the correct path for the file, too.

If you place all pictures from your project into an “img” subdirectory, then your splash image name must begin with “/img/...” (yes, with a leading '/' symbol).

If you get the name wrong, no image will show up. You can leave this field blank to prevent Java from showing an automatic splash image.

2.2.1.7 Define ${VERSION} tag

You may want to include in the name of your distribution archive a variable derived from your source code (or another file). For example, if your source code contains some type of version label, you can extract it using a regular expression and include the resulting value in the Distribution Archive file name so: `/tmp/myarchive-${VERSION}.tar.gz`

You need three elements for this to work:

1. The source file name. Usually this would be a source file in your project, in which case the name must be a relative path within your project directory; it is not recommended to use absolute file names, but Boxer will not keep you from it, if you insist.
2. A regular expression to define one or more groups to extract value(s) from the source file. The pattern is surrounded with `.*` symbols, so you do not need to (and should not) specify them. An example pattern might be the following:

   ```
   \bVERSION\s*=\s*"(.*?)"
   ```

3. A combination of text and/or group references that defines the `${VERSION}` tag's value. The first group is named `${1}`, the second one, `${2}`, etc. The example above has only one group.

2.2.2 The Sources Tab

If you are working with Free/Open Source Software ("FOSS"), you may want to include source code inside the executable JAR that you distribute, rather than as a separate file. Source code compresses well, so the size of the executable JAR may not expand by much if you include sources this way.

By default, source code is not included, of course. You will see this by the grayed out default setting, which indicates that '*' (all files) are ignored.

Pressing the "(+)" button adds another entry above the default line. In the Extension(s) column, enter one or more filename extensions (including the period), such as ".java" (no quotes, though); for C/C++ projects, you might enter something like ".c .c++ .cxx .cpp .h"

The Action column indicates whether to include the files with the indicated file name extension(s) in the JAR, or ignore them. There are only two choices.

You can store the sources in a sub-directory: You really want to do that, otherwise your sources will be mixed up with the binaries, which is not necessarily a disaster, but would get messy for end users when they want to extract the sources. There is no need to give a leading '/' here.

The Flatten option removes all path names from the source directory tree before storing the files in the JAR. There might be reasons you want that, but most of the time you should not check that option.

The "(-)" button on the right removes the row again. Be careful, as there is no way to restore a deleted row once it's gone.

2.2.3 The Libraries Tab (Dynamic)

The Libraries tab shows up only if your project includes library elements.

Libraries come in different forms. Library JAR files, ones included in the project, or external to it, can be handled in three different ways:

1. Reference the library as an external resource, in a directory of your choosing relative to the main JAR, with the recommended option to strip the JAR's existing directory path away.

   This option reflects the way many or even most Java projects tend to be delivered. It allows any one single component to be updated, which is more efficient than updating all at once.

   In most cases you will want to flatten the directory, meaning that the library path as Eclipse stores it, is ignored, and only the “Store in” path is used in the JAR's classpath. The same path is used when you choose to create a distribution archive package.

2. Merge the library contents into your JAR, which eliminates an external dependency; This is a good choice if you want to simplify delivery, especially if *all* dependent files are merged in this manner, but it is probably not a good idea if a given JAR is distributed under a different license.

   The downside of this simplicity is that you cannot update merely one component of your project, but must always update the combined portions.

   At this time, Boxer does not yet merge the manifests from multiple JARs, a matter that may complicated the matter of carrying forward digital signatures from multiple JARs.

3. Ignoring the library dependency may be an option if your project is a library itself (no need for a classpath in the JAR manifest) and is itself part of a larger project, which will satisfy any dependencies on its own.

2.2.4 The Projects Tab (Dynamic)

The Projects tab shows up only if your project references other projects.

Projects upon which your project depends can be handled in three different ways:

1. Reference the project's contents as a separate JAR, just like a library JAR, in a directory of your choosing relative to the main JAR. There is no path associated with another project's JAR, so unlike with libraries, there is no option to flatten such a path.

   This option reflects the way many or even most Java projects tend to be delivered. It allows any one single component to be updated, which is more efficient than updating all at once.

2. Merge the other project's contents into your JAR, which eliminates an external dependency: This is a good choice if you want to simplify delivery, especially if *all* dependent files are merged in this manner, but it is probably not a good idea if a given JAR is distributed under a different license.

Boxer has a dependency on another small library project, so Boxer merges that project's files into Boxer's own JAR for simplicity.

3. Ignoring the project dependency may be an option if your project is itself a dependency of another, and that one is already taking care of managing the files and resolving dependencies on its own.

2.2.5 The Binaries Tab

Binaries tend to be critical to any project, without them you have nothing to run. There may be times when you wish to exclude certain files from the output directory. Perhaps your build process or post-build analysis tools generated additional debugging files, symbol table files, or extraneous information that is not necessary for execution, and even undesirable for inclusion in the JAR.

In that case, you may choose to exclude such files by giving their file name extensions.

2.2.6 The Additions Tab

Documentation, license files, and other additions to the project, perhaps not integral to running the software, but it may certain be desirable or important for other reason to include such files.

The File path chosen is relative to the project directory. For files within the project directory a relative file path is best; for files elsewhere in the file system, an absolute path tends to be best.

You can choose to include the file in the JAR, in the distribution archive package, or both. For each of these options, you can pick a directory for the JAR or distribution archive.

2.3 License

Boxer is distributed under the GNU General Public License Version 3 (or later at your option).

A copy of the license, under the file name COPYING, is included inside the .jar — You may extract the file, or view it with any unzip tool by treating the .jar as a zip file. Alternately, you may download a copy of the GNU General Public License from http://www.gnu.org/licenses/