**Creating a bespoke JRE**

Maybe this can be done within Eclipse using Maven, but if so I haven’t been able to figure it out. So the following steps are all done in the command line interface. They can be done in any folder, as long as your environmental variables are pointing to the correct Java folder so that it can find the executables.

Whenever you are going to add a new dependency, you should perform Step 1 on the new library jar file. Compare the output to the modules already included in the bespoke JRE (see step 2) – with a little luck, they are already in there and you don’t even need to make a new JRE.

If you are updating the entire version of Java, you should run this on the Pamguard jar file to make sure you have everything you need.

**Step 1 – Figure out which JDK library files are actually needed**

You do this with the jdeps command, which you run on a jar file and save the output to a text file (that’s what I’ve found to be easiest). The command is:

D:\Work>jdeps --ignore-missing-deps -R -verbose:package JarFileToCheck.jar > OutputFile.txt

Obviously replace *JarFileToCheck.jar* and *OutputFile.txt* with your own filenames. The -R (for recursive search) and -verbose:package switches make the output a lot longer, but I found after lots of experimentation that adding those two flags (and then sifting through the output) was the only way I could get all of the necessary JDK libraries.

Note that this can take quite a while - if you’re running this on the full Pamguard jar, it will take 10-15 minutes.

This is a typical line in the output file:

Acquisition.filedate -> javax.swing java.desktop

It’s the last column that you’re interested in – that is the module you need to include in the jre. I usually load the file into excel and then try and break the rows into separate columns using [space] as the delimiter. Note that the first few rows are shorter, so they will not line up properly with the rest. Move the last column of the first few rows in line with the last column of everything else, highlight the entire column and then hit the ‘Remove Duplicates’ button. What you have left is a list of Java modules that will be needed in your bespoke JRE.

Here is the list, when run on the PAMGuardBeta-2.01.06.jar file:

|  |
| --- |
| java.base |
| java.compiler |
| java.datatransfer |
| java.desktop |
| java.instrument |
| java.logging |
| java.management |
| java.naming |
| java.net.http |
| java.prefs |
| java.rmi |
| java.scripting |
| java.security.jgss |
| java.security.sasl |
| java.sql |
| java.transaction.xa |
| java.xml |
| java.xml.crypto |
| jdk.jfr |
| jdk.jsobject |
| jdk.unsupported |
| jdk.unsupported.desktop |
| jdk.xml.dom |

**Step 2 – Create the JRE**

Now that you have the list, you can create the JRE. You do this using the jlink command in the DOS window.

First check to make sure that we are using Java16 as the default (or whichever version you are hoping for):

D:\Work>java -version

openjdk version "16" 2021-03-16

OpenJDK Runtime Environment (build 16+36-2231)

OpenJDK 64-Bit Server VM (build 16+36-2231, mixed mode, sharing)

Now create a new JRE:

D:\Work\>jlink --no-header-files --no-man-pages --module-path "C:\Program Files\JavaFX\javafx-sdk-16\lib" --add-modules java.base, java.compiler, java.datatransfer, java.desktop, java.instrument, java.logging, java.management, java.naming, java.net.http, java.prefs, java.rmi, java.scripting, java.security.jgss, java.security.sasl, java.sql, java.transaction.xa, java.xml, java.xml.crypto, jdk.crypto.ec, jdk.jfr, jdk.jsobject, jdk.unsupported, jdk.unsupported.desktop, jdk.xml.dom --output Java16MavenJRE

The path in quotes after –module-path should point to wherever you have the JavaFX files installed on your computer. The modules that you are bundling together are all listed after –add-modules. Finally, the folder name in which the custom JRE will be saved is given after –output (in this example, the folder is Java16MavenJRE).

**\*\*\* One important thing to note**: both the Alarm module and the Deep Learning module need one extra Java module in order to function: **jdk.crypto.ec**. They only try to access this module during runtime, which is why the dependency doesn’t show up when using the jdeps command. Without it, you will receive an SSL Handshake error when the Alarm module tries to access an output email, or when the Deep Learning module tries to download the necessary AI files (which occurs when you open the Deep Learning params window).

So now you have a bespoke JRE. Copy all the files in the output folder into the PamguardCompiler’s JRE12Plus folder (can be found in the Tools folder).

If you ever need to know which modules were included or which version of Java this is, go into the bespoke JRE folder and view the ‘release’ file.