Advanced Customisation

It is anticipated that the customisation features described in the JSPUI and XMLUI customisation sections will be sufficient to satisfy the needs of the majority of users, however, some users may want to customise DSpace further, or just have a greater understanding of how to do so.

Additions module

This module was added in DSpace 3.0 and should be used to store dspace-api changes, custom plugins, ... Classes placed in `[dspace-source]/dspace/modules/additions` will override those located in the dspace-api. This module will be used for all the webapps located in the `[dspace-source]/dspace/modules` directory and in the command line interface. **It is recommended to place all dspace-api changes in this module** so the changes made are contained in a single module, making it easier to get an overview of changes made.

If you are using **Tomcat 8 or later**, the jar-files on a webapps classpath are not loaded alphabetically anymore. This means you need to explicitly instruct Tomcat to first load the additions.jar file, and after that all other jar-files (a technique called class shadowing). You can do this by adding the following to your webapp context XML file:

```
<Resources className="org.apache.catalina.webresources.StandardRoot">
    <PreResources className="org.apache.catalina.webresources.JarResourceSet"
      base="/path/to/dspace/lib/additions-5.x.jar"
      internalPath="/"
      webAppMount="/WEB-INF/classes" />
</Resources>
```

This dynamically (and virtually) puts the additions.jar (or any other jar you want) in the `WEB-INF/classes` folder of the webapp. Tomcat will always start with loading files from the `WEB-INF/classes` directory thus giving them a higher priority.

Note that you need to do this for every DSpace webapp you deploy in Tomcat 8+.

Maven WAR Overlays

Much of the customisation described in the JSPUI and XMLUI customisation sections is based on Maven WAR Overlays. In short, any classes or files placed in `[dspace-source]/dspace/modules/*` will be overlayed onto the selected WAR. This includes both new and amended files.

For more details on Maven WAR Overlays and how they relate to DSpace, see this presentation from Fall 2009: Making DSpace XMLUI Your Own (Please note that this presentation was made for DSpace 1.5.x and 1.6.x, but much of it still applies to current versions of DSpace.)

DSpace Source Release

If you have downloaded the 'dspace-src-release' (or checked out the latest DSpace Code via GitHub), there are two main build options that are available to you:

1. **Full Build:** Running `mvn package` from the root `[dspace-source]`directory
    - This option will rebuild **all** DSpace modules from their Java Source code, then apply any Maven WAR Overlays. In other words, all subdirectories of `[dspace-source]` are recompiled/rebuilt.
2. **Quick Build:** Running `mvn package` from the `[dspace-source]/dspace/`directory
    - This option performs a "quick build". It does **not** recompile/rebuild all DSpace modules. All it does is rebuild and re-apply any Maven WAR Overlays to the previously compiled source code. In other words, the ONLY code that will be recompiled/rebuilt is code that exists in `[dspace-source]/dspace/modules/*` (the Maven WAR Overlay directories)

Which build option you need to use will depend on your local development practices. If you have been careful to utilize Maven WAR Overlays for your local code/changes (putting everything under `[dspace-source]/dspace/modules/*`), then the **Quick Build** option may be the best way for you to recompile & reapply your local modifications. However, if you have made direct changes to code within a subdirectory of `[dspace-source]` (e.g. `/dspace-api`, `/dspace-xmlui`, `/dspace-jspui`, etc.) then you will need to utilize the **Full Build** option in order to ensure those modifications are included in the final WAR files.