jahia

HOW TO PACKAGE YOUR OSGI BUNDLE WITH DEPENDENCIES

DEVELOPMENT DOCUMENTATION

TABLE OF CONTENT

1 CASE 1: INTRODUCE A NEW FRAMEWORK THAT HAS NO LINK WITH THE ONES PROVIDED BY JAHIA

That means the framework(s) you are using are not extending one provided by Jahia as is, they are using some of them most probably but they are not adding or overriding any of those frameworks.

In this case the integration should be straightforward, add your dependencies in your pom.xml file as usual.

Then you need to configure the maven-bundle-plugin that will package your bundle into a jar, in this configuration you need to specify what package you want to export, and which dependencies you need to embed inside your bundle.

```
<plugin>
 <groupId>org.apache.felix</groupId>
 <artifactId>maven-bundle-plugin</artifactId>
 <extensions>true</extensions>
 <configuration>
   <instructions>
     <Jahia-Module-Type>system</Jahia-Module-Type>
     <Jahia-Deploy-On-Site>all</Jahia-Deploy-On-Site>
     <Jahia-Depends>default,siteSettings,serverSettings</Jahia-Depends>
     <Import-Package>
       ${jahia.plugin.projectPackageImport},
       *
     </Import-Package>
     <!-- Export all local packages-->
     <Export-Package>
       {local-packages}
     </Export-Package>
     <Embed-
    Dependency>*;groupId=dep_groupid|other_dep_groupid;scope=compile;
    type=!pom; inline=false</Embed-Dependency>
   </instructions>
 </configuration>
</plugin>
```

The Import-Package instruction helps you define which package your module is using, The Jahia Maven plugin will try to optimize and parse all your files to find all the needed imports, they will be accessible through the variable **jahia.plugin.projectPackageImport** this plugin will parse your

jsps, tlds, spring, rules files etc. Then you need to add the * entries so that maven plugin can add all needed package from your dependencies and transient dependencies that will not have been found by the Jahia plugin before.

The Export-Package instruction allows you to define the package you want to export to other bundle, those ones will be the only accessible class from another bundle. Here you can use the Maven macro 'local-packages' to define that you want to export all your packages and/or list them manually this is needed if you want to expose some classes of the embedded framework.

The Embed-Dependency instruction is here to help you define which of the dependencies you want to see embedded inside your bundle, those dependencies can be embedded as is or inline along your classes.

To have more information about the directives you can use in those instructions refer to this page http://felix.apache.org/site/apache-felix-maven-bundle-plugin-bnd.html

2 CASE 2: ADD CUSTOM SPRING HANDLERS AND NAMESPACES TO THE ONES PROVIDED BY JAHIA

Let's say that you want/need to use one of the several sub projects developed by Spring Framework like Spring Social. Those projects provide some spring namespace handlers to extends the spring xml files directives those files named META-INF/spring.schemas and META-INF/spring.handlers will be found by the OSGI classloader inside your embedded jars and will be the only one found, which will break the resolution of other namespaces and then make your bundle impossible to start even if correctly installed and resolved by the OSGI platform.

To solve this you will have to use the Maven Shade plugin this plugin will allow you to merge same named files such as spring.schemas and spring.handlers into a single file. Shade works only with at least compile scoped dependencies not provided so first you need to declare your dependencies correctly:

```
<properties>
       <spring-social.version>1.1.0.M4</spring-social.version>
       <spring-social-facebook.version>1.1.0.M4</spring-social-facebook.version>
       <spring-social-twitter.version>1.1.0.M4</spring-social-twitter.version>
       <spring-social-linkedin.version>1.0.0.RC3</spring-social-linkedin.version>
       <spring-security-crypto.version>3.1.4.RELEASE</spring-security-
    crypto.version>
   </properties>

   <dependencies>
       <!-- Add spring framework as local dependencies to enforce same version as in
    Jahia -->
       <!-- scope is provided only -->
       <dependency>
           <groupId>org.springframework</groupId>
           <artifactId>spring-beans</artifactId>
           <version>${spring.version}</version>
           <!--<scope>provided</scope>-->
       </dependency>
       <dependency>
           <groupId>org.springframework</groupId>
           <artifactId>spring-core</artifactId>
           <version>${spring.version}</version>
           <!--<scope>provided</scope>-->
       </dependency>
```

```
        <dependency>
            <groupId>org.springframework.webflow</groupId>
            <artifactId>spring-webflow</artifactId>
            <version>${springwebflow-version}</version>
            <!--<scope>provided</scope>-->
        </dependency>
        <dependency>
            <groupId>org.eclipse.gemini.blueprint</groupId>
            <artifactId>gemini-blueprint-core</artifactId>
            <version>${gemini.blueprint.version}</version>
            <!--<scope>provided</scope>-->
        </dependency>
        <dependency>
            <groupId>org.springframework.social</groupId>
            <artifactId>spring-social-core</artifactId>
            <version>${spring-social.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.social</groupId>
            <artifactId>spring-social-web</artifactId>
            <version>${spring-social.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.social</groupId>
            <artifactId>spring-social-security</artifactId>
            <version>${spring-social.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.social</groupId>
            <artifactId>spring-social-facebook</artifactId>
            <version>${spring-social-facebook.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-crypto</artifactId>
            <version>${spring-security-crypto.version}</version>
        </dependency>
    </dependencies>
```

Then you need to configure both the Maven Bundle plugin like previously explained but this time you will have to inline your embedded dependencies:

```
<!-- Export all local packages-->
<Export-Package>
{local-packages},
org.springframework.social.connect,
org.springframework.social.facebook.api,
org.springframework.social.twitter.api
</Export-Package>
<!-- Embed local dependency only by excluding org.springframework (avoid issue at
    runtime when starting the module, module should not provide their own spring :) -
    ->
<Embed-
    Dependency>*;groupId=org.springframework.social|org.springframework.security;scop
    e=compile; type=!pom; inline=!META-INF/**</Embed-Dependency>
```

Here, we are inlining `org.springframework.social` and `org.springframework.security` but we are excluding the contents of all META-INF folders.

Now we configure the Maven Shade plugin to go through all the needed artifact and merge all found handlers and schemas files.

```
<plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>2.1</version>
                <executions>
                    <execution>
                        <phase>install</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <promoteTransitiveDependencies>false</promoteTransitiveDe
    pendencies>
                            <artifactSet>
                                <includes>
                                    <include>org.springframework:*</include>
                                    <include>org.jahia.modules:*</include>
                                    <include>org.springframework.webflow:*</include>
                                    <include>org.springframework.social:*</include>
                                    <include>org.springframework.security:*</include>
                                </includes>
                            </artifactSet>
                            <filters>
                                <filter>
                                    <artifact>org.springframework:*</artifact>
                                    <includes>
                                        <include>META-INF/spring.handlers</include>
                                        <include>META-INF/spring.schemas</include>
                                    </includes>
                                </filter>
                                <filter>
                                    <artifact>org.springframework.webflow:*</artifact
    >
                                    <includes>
                                        <include>META-INF/spring.handlers</include>
                                        <include>META-INF/spring.schemas</include>
                                    </includes>
                                </filter>
                            </filters>
                            <transformers>
                                <transformer
                                        implementation="org.apache.maven.plugins.shad
    e.resource.AppendingTransformer">
                                    <resource>META-INF/spring.handlers</resource>
                                </transformer>
                                <transformer
```

```
                                    implementation="org.apache.maven.plugins.shad
e.resource.AppendingTransformer">
                                <resource>META-INF/spring.schemas</resource>
                            </transformer>
                        </transformers>
                    </configuration>
                </execution>
            </executions>
        </plugin>
```

This configuration will scan all the dependencies jars and lookup only for spring.schemas and spring.handlers files and will then merge them altogether during the install phase after the bundle plugin as shade is working directly on the jar itself.

By playing with all the instructions of Shade it is easy to customize what you will find in the resulting jar file.