# Si Vis Bellum Para Bellum

## KC with Maven Overlays: Best Practices, Caveats, and Anti-Patterns

Leo Przybylski

November 8, 2013

# Overview

- Welcome & Colophon.
- About Presenters.
- Maven Overlays.
- Multi-module vs. Regular Webapp Project.
- Appserver Plugins.
- Rice DataDictionary Reloading vs. JRebel.
- Tips and Tricks.
- Proposal: Contributing a Modification as a Maven Dependency.

# Welcome and Colophon

- Hi!
- "Si Vis Bellum Para Bellum" is a play on "Si Vis Pacem Para Bellum" and Web Archive (WAR) files. It means "If you wish for war, prepare for war." WAR being a web archive rather than turning your project into a battlefield.

# About Presenters

# Maven Overlay Basics

- How Overlays work.
- What overlays are not.
- An Overlay RIGHT NOW!

# How Overlays Work

- Simply put, it allows one project war output to overlay/overlay another.
- The project being overlapped is a dependency. Its source is not required.

# Why Overlay a WAR?

- Dependency project source code is not required.
- No mixing changes. No merges. No patches.
- Change isolation.
- Upgrading the dependency is easy.
- Own your POM.

# What Overlays are Not

- Project Inheritence.
- A replacement/alternative for multi-module projects.
- A silver-bullet for all webapp projects.
- Change management pattern.

# Advantages of Overlays

- Offers flexibility in your project by allowing granular modifications at the project level.
- Smaller projects for tracking changes with.
- Easier to modularize your project.
- Better code reuse.

# First Overlay

Using KC as an Example to switch out CAS support
for the login dummy

1. Package KC.
2. Install as a prototype.
3. Setup prototype jar.
4. Create overlay project.
5. Add CAS dependency.
6. Add dependencies for KC.
7. Configure war plugin for overlay.

# Package and Install KC as a prototype

1. **Do install**

   ```
   mvn install
   ```

2. **Create a and Install Prototype Jar**

   ```
   mvn jar:jar
   mvn install:install-file \
       -DgroupId=org.kuali.kra \
       -DarchetypeId=kc_project \
       -Dversion=5.1  \
       -Dpackaging=jar \
       -Dfile=target/kc_project-5.0.1.jar \
       -DpomFile=pom.xml
   ```

   … or …

   ```
   mvn deploy:deploy-file \
       -DgroupId=org.kuali.kra \
       -DarchetypeId=kc_project \
       -Dversion=5.1  \
       -Dpackaging=jar \
       -Dfile=target/kc_project-5.0.1.jar \
       -DpomFile=pom.xml
   ```

# Create an Overlay Project the Old-Fashioned Way

- Configure the maven-war-plugin

```xml
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <version>${maven-war-plugin.version}</version>
</plugin>
```

# Configure the maven-war-plugin

```xml
<dependencies>
<!--
    These are the prototype dependencies. It is used by the overlay to refer back
    to the classes in the prototype.
  -->
  <dependency>
    <groupId>${kcPrototypeGroupId}</groupId>
    <artifactId>${kcPrototypeArtifactId}</artifactId>
    <version>${kcPrototypeVersion}</version>
    <type>war</type>
  </dependency>

  <dependency>
    <groupId>${kcPrototypeGroupId}</groupId>
    <artifactId>${kcPrototypeArtifactId}</artifactId>
    <version>${kcPrototypeVersion}</version>
    <type>jar</type>
  </dependency>
</dependencies>
```

# Overlay Archetypes

- Leo has developed an archetype for creating and overlay KC project.
- https://github.com/r351574nc3/kualigan-maven-plugins/tree/master/kc-maven-plugin
- To create an Overlay Project from Archetype

```
mvn -Dkc:create-overlay \
-DgroupId=com.rsmart.kuali.kra \
-DartifactId=kc -Dversion=5.1
```

# Overlay Patterns

- Overlaying configuration modifications.
- Overlay to activate a module.
- Overlay theming for UX/UI.
- Filtering overlay resources.

# Solutions to Multi-module Issues

- Build classes into `web/target/classes`
  - Requires a profile to only do this for developers
  - Not appropriate for production at all

- JRebel will watch multiple src folders for changes

- Jetty will also watch multiple src folders

# Configuring JRebel

```xml
<plugin>
    <groupId>org.zeroturnaround</groupId>
    <artifactId>jrebel-maven-plugin</artifactId>
    <configuration>
      <alwaysGenerate>true</alwaysGenerate>
      <addResourcesDirToRebelXml>true</addResourcesDirToRebelXml>
      <!--
        root is 2 directories away from jar/war modules
      -->
      <relativePath>../../</relativePath>
      <!--
        use a system property for specifing root directory (note the double $)
        start your application with -Drebel.root=c:/projects/
      -->
      <rootPath>${rebel.root}</rootPath>
    </configuration>
</plugin>
```

# JRebel and the KC Archetype

- The KC Archetype automatically configures your overlay as a multi-module project
- Comes with JRebel configuration already

# Tips and Tricks

- Use JRebel! It's free!
- Take advantage of the default classpath for runtime Data Dictionary overrides.
- Overriding Rice Services.
- Overriding KC Services.

# Proposal: Contributing a Modification as a Maven Dependency

- Develop modifications as a separate maven project or module.

- Include the modifications as dependencies in your Maven Overlay project.

- Distribute modifications with documentation to a Maven Repository.

# The end

Thanks