Packaging WO Applications as true WAR with Maven

If you have used the woapplication-archetype to create your project, jump to the step 3.

You have to follow some instructions to build a true WAR package:

Step 1: Create a web.xml file

You need to create a web.xml file. You can download a simple web.xml file here. Don't forget to change the displayName and the WOMainBundle properties:

```
<web-app>
   ...
   <display-name>Your Application Name</display-name>
   ...
   <context-param>
      <param-name>WOMainBundle</param-name>
      <param-value>your-app-name</param-value>
   </context-param>
   ...
</web-app>
```

Step 2: Create/generate an Info.plist file

You also need to create or generate a valid Info.plist file into your resources folder. Here is a sample Info.plist. You have to change the ${your-app-name} and ${package} occurrences with the respective application name and Application class package.

Step 3: Package your classes, resources and webserver resources

The application jar must follow the NSJarBundle format. The NSJarBundle is a package organized in Resources and WebServerResources folders. In addition, the Resources folder must contain a valid Info.plist file. Your application classes, resources and webserver resources must be package as a jar. It is easy to configure Maven to do this:

```
<build>
  ...
  <resources>
   ...
   <resource>
      <targetPath>Resources</targetPath>
      <directory>${basedir}/src/main/resources</directory>
   </resource>
   <resource>
      <targetPath>Resources</targetPath>
      <directory>${basedir}/src/main/components</directory>
   </resource
   <resource>
      <targetPath>WebServerResources</targetPath>
      <directory>${basedir}/src/main/webserver-resources</directory>
   </resource>
   ...
  </resources>
  ...
  <plugins>
   ...
   <plugin>
      <artifactId>maven-war-plugin</artifactId>
      <configuration>
         <archiveClasses>true</archiveClasses>
      </configuration>
   </plugin>
   ...
  </plugins>
   ...
</build>
```

Step 4: Add the required dependencies

You must add the following dependency to run the application as a true WAR:

```
<dependency>
   <groupId>com.webobjects</groupId>
   <artifactId>JavaWOJSPServlet</artifactId>
   <version>${woversion}</version>
</dependency>
```

NOTE: If you are using WebObjects 5.2.x or 5.3.x you have to add this additional dependency:

```
<dependency>
   <groupId>com.webobjects</groupId>
   <artifactId>JavaWOJSPServlet_client</artifactId>
   <version>${woversion}</version>
</dependency>
```

NOTE: If your application uses Wonder you must add the ERXServlet dependency:

```
<dependency>
   <groupId>wonder.core</groupId>
   <artifactId>ERServlet</artifactId>
   <version>1.0</version>
</dependency>
```

Read this tutorial to find how to configure the ERXServletAdaptor in your application.

Step 5: Change the packaging type

The default maven-war-plugin can handle the war packaging correctly. You have to change the packaging of your POM to 'war' in order to use this plug-in:

```
<packaging>war</packaging>
```

It's done.

Step 6: Filtering variables with Maven (OPTIONAL)

Maven supports variable substitution during build time. It is called resource filtering. You can find more information about resource filtering here an d here.

If you are using variables in your resource files (i.e. the Info.plist), you have to configure Maven to filter the Resources folder like this:

```
<build>
  ...
  <resources>
   ...
   <resource>
      <targetPath>Resources</targetPath>
      <directory>${basedir}/src/main/resources</directory>
      <filtering>true</filtering>
   </resource>
   ...
  </resources>
  ...
</build>
```

If you are using variables in the web.xml file, you have to configure Maven to filter deployment descriptors like this:

```
<build>
  ...
  <plugins>
   ...
   <plugin>
      <artifactId>maven-war-plugin</artifactId>
      <configuration>
         ...

<filteringDeploymentDescriptors>true</filteringDeploymentDescriptors>
      </configuration>
   </plugin>
   ...
  </plugins>
  ...
</build>
```

Running your application as true WAR

You can use the maven-jetty-plugin to run and test your application.

Step 1: Configure the maven-jetty-plugin

Add the following configuration to your POM:

```
<build>
   ...
   <plugin>
      <groupId>org.mortbay.jetty</groupId>
      <artifactId>maven-jetty-plugin</artifactId>
   </plugin>
   ...
</build>
```

Step 2: Start the Jetty container with Maven

Just execute:

```
mvn clean jetty:run-war
```

Step 3: See the result

Open a browser and type the URL for your application like this: http://locahost:8080/your-app-name/WebObjects/