Getting Started with the Oracle Maven Repository

Hands-on Lab Guide

© Copyright 2015 Oracle Corporation. All rights reserved.
Mark Nelson, Architect, Fusion Middleware Platform Engineering

Table of Contents

Copyright 2015 Oracle Corporation. All Rights Reserved.

1 Introduction

In this Hands-on lab, we will learn how to use the Oracle Maven Repository to create and build applications for Fusion Middleware.

The Oracle Maven Repository contains artifacts provided by Oracle that you may require to compile, test, package, perform integration testing, or deploy reapplications. It is similar to other Maven repositories.

2 Prerequisites

The following software is required to complete this lab:

- WebLogic Server 12.1.3 Developer's (ZIP) Distribution
- JDK 1.7 or 1.8 (latest release)
- Apache Maven 3.2.5

3 Getting Access and Configuring Maven

Access to the Oracle Maven Repository is provided after accepting the appropriate license agreement. In this section we will obtain access and configure Maven to use the Oracle Maven Repository.

3.1 Register for Access

Before using the Oracle Maven Repository, you must register for access and accept the license agreement. You must have an Oracle Technology Network account to complete the registration process.

To start the registration process, open your browser and go to the following page:

```
https://www.oracle.com/webapps/maven/register/license.html
```

Read the license agreement and then click on the button to accept the agreement.

You will be redirected to the documentation site which contains information about how to configure Maven to access the Oracle Maven Repository.

3.2 Configure Maven

Now we need to configure Maven to use the Oracle Maven Repository. This involves telling Maven about the repository and providing the necessary information to authenticate to the repository.

Copyright 2015 Oracle Corporation. All Rights Reserved.

3.2.1 Authentication

Every time you want to access the Oracle Maven Repository, you must provide the user name and password you used to log in to Oracle Technology Network (OTN). Access is only provided over HTTPS. You can store your credentials in the Maven settings file so that you do not have to specify them manually every time. Oracle recommends that you encrypt your password, using the utilities provided with Maven, as shown in the next section.

3.2.2 Create a Master Password

Maven uses a 'master password' to encrypt passwords for repositories. Use the following command to create a master password:

```
mvn --encrypt-master-password secret
```

Replace 'secret' with the master password you want to use.

The output will be a string similar to this:

```
{wJ8chNDD1ksHLtKSjX2o2gepDO4zcEyr2XHjnwZ9CH4=}
```

Save this value in your Maven settings-security.xml in your $HOME/.m2 directory. Here is an example of the contents of this file – if you do not have this file already, please create it:

```
<settingsSecurity>
  <master>{wJ8chNDD1ksHLtKSjX2o2gepDO4zcEyr2XHjnwZ9CH4=}</master>
</settingsSecurity>
```

3.2.3 Encrypt your OTN Password

Now that you have a master password defined, you can encrypt your OTN password using the following command:

```
mvn --encrypt-password mypassword
```

Change 'mypassword' to your OTN password.

The output will be a similar string, keep this value for later – you will need it when configuring Maven.

3.2.4 Adding the Oracle Maven Repository to Your settings.xml

Add a repository definition to your Maven settings.xml file. The repository definition should look like the following:

```
<repositories>
  <repository>
    <id>maven.oracle.com</id>
    <releases>
      <enabled>true</enabled>
```

Copyright 2015 Oracle Corporation. All Rights Reserved.

```
    </releases>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <url>https://maven.oracle.com</url>
    <layout>default</layout>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <id>maven.oracle.com</id>
    <url>https://maven.oracle.com</url>
  </pluginRepository>
</pluginRepositories>
```

3.2.5 Configuring the HTTP Wagon

The Maven settings.xml requires additional settings to support the Oracle Maven Repository. Add the following <server> element to the <servers> section of the Maven settings.xml:

```
 <server>
    <id>maven.oracle.com</id>
    <username>username</username>
    <password>password</password>
    <configuration>
      <basicAuthScope>
        <host>ANY</host>
        <port>ANY</port>
        <realm>OAM 11g</realm>
      </basicAuthScope>
      <httpConfiguration>
        <all>
          <params>
            <property>
              <name>http.protocol.allow-circular-redirects</name>
              <value>%b,true</value>
            </property>
          </params>
        </all>
      </httpConfiguration>
    </configuration>
  </server>
```

Replace the <username> and <password> entries with your OTN user name and the encrypted password you created earlier.

3.2.6 Set up Maven to use the Oracle Maven Repository without a POM

When you invoke Maven from the command-line to run a plugin goal directly, Maven uses a standalone POM that does not use the repositories listed in the settings.xml file unless they are part of a profile that is set to be "active by default". For example, the typical way to generate a new project from an

Copyright 2015 Oracle Corporation. All Rights Reserved.

archetype is to invoke the Maven's `archetype:generate` goal from the command line directly without a project POM in place. In order for this to work, the Oracle Maven Repository must be declared within a profile that is active by default, as shown in the following example:

```
<profiles>
  <profile>
    <id>main</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <repositories>
      <repository>
        <id>maven.oracle.com</id>
        <url>https://maven.oracle.com</url>
        <layout>default</layout>
        <releases>
          <enabled>true</enabled>
        </releases>
      </repository>
    </repositories>
  </profile>
</profiles>
```

3.3 About the Oracle Maven Synchronization plugin

If you have used Oracle's Maven support previously, you will probably be familiar with the Oracle Maven Synchronization plugin which is used to populate your local Maven repository from and Oracle Home. When you use the Oracle Maven Repository, you do not need to execute the Oracle Maven Synchronization plugin's goals to populate your local repository – Maven will download anything it needs from the Oracle Maven Repository for you.

However, the Oracle Maven Repository does not contain artifacts with one-off patches applied, so if you require any one-off patches, you will still need to obtain them from Oracle Support, install them on your local Oracle Home and then use the Oracle Maven Synchronization plugin to update your local Maven repository with the patched artifacts.

4 Creating a New Project

Now that you have configured Maven, you are ready to create a new project using an archetype in the Oracle Maven Repository. We will use the 'Basic WebApp' archetype for WebLogic Server. Create a project using the following command (enter it all on one line):

```
mvn archetype:generate
    -DarchetypeGroupId=com.oracle.weblogic.archetype
    -DarchetypeArtifactId=basicWebapp
    -DarchetypeVersion=12.1.3-0-0
    -DarchetypeRepository=https://maven.oracle.com
```

Copyright 2015 Oracle Corporation. All Rights Reserved.

Maven will download the archetype from the Oracle Maven Repository, along with any other dependencies it needs. Then it will ask you to specify the coordinates for your new project. Enter the following values when you are prompted:

```
groupId    = com.test
artifactId = webapp1
version    = 1.0-SNAPSHOT
```

Maven will create your new project in the 'webapp1' directory. Go ahead and take a look at it now.

Other Archetypes in the Oracle Maven Repository

There are several archetypes provided in the Oracle Maven Repository, for creating different kinds of projects.

- Oracle Fusion Middleware 12.1.2
  - com.oracle.coherence:maven-gar-archetype:12.1.2-0-0
  - com.oracle.weblogic.archetype:basic-mdb:12.1.2-0-0
  - com.oracle.weblogic.archetype:basic-webapp-ejb:12.1.2-0-0
  - com.oracle.weblogic.archetype:basic-webapp:12.1.2-0-0
  - com.oracle.weblogic.archetype:basic-webservice:12.1.2-0-0
- Oracle Fusion Middleware 12.1.3
  - com.oracle.adf.archetype:oracle-adffaces-ejb:12.1.3-0-0
  - com.oracle.coherence:maven-gar-archetype:12.1.3-0-0
  - com.oracle.servicebus.archetype:oracle-servicebus-application:12.1.3-0-0
  - com.oracle.servicebus.archetype:oracle-servicebus-project:12.1.3-0-0
  - com.oracle.servicebus.archetype:oracle-servicebus-system:12.1.3-0-0
  - com.oracle.soa.archetype:oracle-soa-application:12.1.3-0-0
  - com.oracle.soa.archetype:oracle-soa-project:12.1.3-0-0
  - com.oracle.weblogic.archetype:basic-mdb:12.1.3-0-0
  - com.oracle.weblogic.archetype:basic-webapp:12.1.3-0-0
  - com.oracle.weblogic.archetype:basic-webapp-ejb:12.1.3-0-0
  - com.oracle.weblogic.archetype:basic-webservice:12.1.3-0-0

5 Building the Project

Let's go ahead and build the project now. From the webapp1 directory, issue the following command:

```
mvn package
```

This tells Maven to compile the source files and build the deployable WAR. The first time you do this, Maven will not have the necessary dependencies locally (on your machine) so it will download them from the Oracle Maven Repository. Depending on your Internet speed, this could take several minutes.

Copyright 2015 Oracle Corporation. All Rights Reserved.

Maven will keep the files in your local repository, so you only need to download them once. Maven will check once a day (by default) to see if any newer versions of the files have become available.

6 Preparing a WebLogic Server environment for deployment and testing

We want to be able to deploy and test our project as part of the build process. First, we will need a WebLogic Server environment to deploy the project into. We will use the WebLogic Maven plugin to install WebLogic Server from the WebLogic ZIP Distribution, create a domain and start a server. The WebLogic Maven plugin is also available in the Oracle Maven Repository, so we can just add a dependency and Maven will download it for us.

Take a look in the Maven Project Object Model **pom.xml** file in your webapp1 directory. In the **build/plugins/plugin** section there is an entry for the WebLogic Maven plugin. It starts like this:

```
<build>
  <plugins>
    ...
    <plugin>
      <groupId>com.oracle.weblogic</groupId>
      <artifactId>weblogic-maven-plugin</artifactId>
      <version>12.1.3-0-0</version>
      ...
      <configuration>
        <middlewareHome>c:/vts/test/wlshome/wls12130</middlewareHome>
      </configuration>
      <executions>
        ...
```

6.1 Install WebLogic Server

Add the following **execution** block, make it the first one in the **executions** section:

```
  <!-- install weblogic server -->
  <execution>
    <id>install-wls</id>
    <phase>validate</phase>
    <goals>
      <goal>install</goal>
    </goals>
    <configuration>
      <artifactLocation>c:/vts/wls1213_dev.zip</artifactLocation>
      <installDir>c:/vts/test/wlshome</installDir>
    </configuration>
  </execution>
```

Change the **artifactLocation** to point to the WebLogic ZIP Distribution you downloaded, and **installDir** to a suitable location where WebLogic Server will be installed.

Copyright 2015 Oracle Corporation. All Rights Reserved.

This tells Maven to execute the WebLogic Maven plugin's **install** goal during the **validate** phase of the build, i.e. right at the beginning of the build.

To test this, issue the following command:

```
mvn validate
```

Note that on subsequent Maven runs, this goal will notice you already have WebLogic Server installed and will not install it again. If you want to have it perform the installation again, just remove the old installation by deleting the **wls12130** directory.

6.2 Create a Domain

Next, we want to create a domain. We can do this using the WebLogic Maven plugin's **wlst** goal to execute a short WLST script. Add the following **execution** block immediately after the previous one:

```
  <!-- create a domain -->
  <execution>
    <id>create-domain</id>
    <phase>validate</phase>
    <goals>
      <goal>wlst</goal>
    </goals>
    <configuration>
      <fileName>${basedir}/misc/create-domain.py</fileName>
    </configuration>
  </execution>
```

Now create a new directory called **misc** in your **webapp1** directory, and inside that directory create a new file called **create-domain.py** and place the following WLST commands in that file:

```
# create domain
readTemplate('c:/vts/test/wlshome/wls12130/wlserver/common/templates/wls/wls.
jar')
cd('/Security/base_domain/User/weblogic')
cmo.setPassword('welcome1')
writeDomain('c:/vts/test/wlshome/wls12130/user_projects/domains/base_domain')
closeTemplate()
```

Change the locations to match your environment. You can test this is working by issuing the following command:

```
mvn validate
```

Maven will install WebLogic Server (if it is not already installed) and then run this WLST script to create a domain.

Note that this step will fail if you run Maven again because the domain will already exist. We will fix this later on, but for now, if you want to re-run Maven, just delete the domain directory first.

Copyright 2015 Oracle Corporation. All Rights Reserved.

6.3 Start a Server

Next, we will want to have a server running so we can deploy our application. We can do this using the WebLogic Maven plugin's **start-server** goal. Add the following **execution** section immediately after the one we just added to create the domain:

```
  <!-- start server -->
  <execution>
    <id>start-server</id>
    <phase>validate</phase>
    <goals>
      <goal>start-server</goal>
    </goals>
    <configuration>

<domainHome>c:/vts/test/wlshome/wls12130/user_projects/domains/base_domain</d
omainHome>
    </configuration>
  </execution>
```

Adjust the **domainHome** to match your environment. If you want to test this, issue the following command:

```
mvn validate
```

Note that the server will remain running after this command finishes, so you will need to manually stop the server if you want to run Maven again. We will tidy this up next.

6.4 Stop the Server

After we have deployed the application and run our tests, we will want to stop the server. We can do this using the WebLogic Maven plugin's **stop-server** goal. Add the following **execution** section to the POM – placing it after the 'deploy' execution that was created from the archetype, i.e. make this the last **execution** in the WebLogic Maven plugin **executions** section:

```
  <!-- stop the server -->
  <execution>
    <id>stop-server</id>
    <phase>post-integration-test</phase>
    <goals>
      <goal>stop-server</goal>
    </goals>
    <configuration>
      <adminurl>t3://localhost:7001</adminurl>
      <user>weblogic</user>
      <password>welcome1</password>

<domainHome>c:/vts/test/wlshome/wls12130/user_projects/domains/base_domain</d
omainHome>
    </configuration>
```

Copyright 2015 Oracle Corporation. All Rights Reserved.

```
  </execution>
```

This tells Maven to stop the server in the **post-integration-test** phase of the build. This phase executes after all of the tests have been completed.

The Maven Default Lifecycle

For your reference, here is a refresher on the phases in the Maven default lifecycle:

`validate`	validate the project is correct and all necessary information is available.
`initialize`	initialize build state, e.g. set properties or create directories.
`generate-sources`	generate any source code for inclusion in compilation.
`process-sources`	process the source code, for example to filter any values.
`generate-resources`	generate resources for inclusion in the package.
`process-resources`	copy and process the resources into the destination directory, ready for packaging.
`compile`	compile the source code of the project.
`process-classes`	post-process the generated files from compilation, for example to do bytecode enhancement on Java classes.
`generate-test-sources`	generate any test source code for inclusion in compilation.
`process-test-sources`	process the test source code, for example to filter any values.
`generate-test-resources`	create resources for testing.
`process-test-resources`	copy and process the resources into the test destination directory.
`test-compile`	compile the test source code into the test destination directory
`process-test-classes`	post-process the generated files from test compilation, for example to do bytecode enhancement on Java classes. For Maven 2.0.5 and above.
`test`	run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
`prepare-package`	perform any operations necessary to prepare a package before the actual packaging. This often results in an unpacked, processed version of the package. (Maven 2.1 and above)
`package`	take the compiled code and package it in its distributable format, such as a JAR.
`pre-integration-test`	perform actions required before integration tests are executed. This may involve things such as setting up the required environment.

Copyright 2015 Oracle Corporation. All Rights Reserved.

`integration-test`	process and deploy the package if necessary into an environment where integration tests can be run.
`post-integration-test`	perform actions required after integration tests have been executed. This may including cleaning up the environment.
`verify`	run any checks to verify the package is valid and meets quality criteria.
`install`	install the package into the local repository, for use as a dependency in other projects locally.
`deploy`	done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

6.5 Remove the Domain

Finally, we want to remove the domain, so that any subsequent Maven build will create its own new clean domain and we don't pollute the test domain with any artifacts left over from previous builds. We do this using the WebLogic Maven plugin's **remove-domain** goal. Add the following **execution** after the one we just created to stop the server:

```
  <!-- remove domain -->
  <execution>
    <id>remove-domain</id>
    <phase>post-integration-test</phase>
    <goals>
      <goal>remove-domain</goal>
    </goals>
    <configuration>

<domainHome>c:/vts/test/wlshome/wls12130/user_projects/domains/base_domain</d
omainHome>
    </configuration>
  </execution>
```

Adjust the **domainHome** to match your environment.

7 Testing the Application

Now that we have configured our build to create a new WebLogic Server environment and remove it when we are finished, we want to add some automated tests to our build process. We will do this using the Robot Framework to execute a Selenium test. The Maven plugins for this are not provided in the Oracle Maven Repository, they are in the Maven Central repository, but we can mix and match plugins from various repositories in our project.

Copyright 2015 Oracle Corporation. All Rights Reserved.

7.1 Add Robot Framework to our Project

To tell Maven to execute the Robot Framework during our build, we need to add another **plugin** entry to the **build/plugins** section. Add this after the (rather lengthy) **plugin** entry for WebLogic Maven plugin:

```
<plugin>
  <groupId>org.robotframework</groupId>
  <artifactId>robotframework-maven-plugin</artifactId>
  <version>1.4.3</version>
  <executions>
    <execution>
      <id>robot-test</id>
      <phase>integration-test</phase>
      <goals>
        <goal>run</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

This tells Maven to execute the **robotframework-maven-plugin**'s **robot-test** goal in the **integration-test** phase of the build, i.e. after we have installed WebLogic Server, created a domain, started a server and deployed the WAR.

7.2 Add Selenium Library

Since we want to use Selenium to drive our test, we also need to add a dependency to the POM for the Robot Framework Selenium library. Add this new **dependency** to the **dependencies** section of the POM:

```
<dependency>
  <groupId>com.github.markusbernhardt</groupId>
  <artifactId>robotframework-selenium2library-java</artifactId>
  <version>1.4.0.7</version>
  <classifier>jar-with-dependencies</classifier>
</dependency>
```

This will make the Selenium Library available for our tests.

7.3 Creating the Tests

By default, Robot Framework will look for the tests in the following directory (under your **webapp1** directory):

```
src/test/robotframework/acceptance
```

Create this directory now, and create a new file in that directory called **Quick_Test.robot** with the following content:

```
*** Settings ****
Library  Selenium2Library
```

Copyright 2015 Oracle Corporation. All Rights Reserved.

```
*** Test Cases ***
Quick Test
   Open Browser                        http://localhost:7001/basicWebapp
   Page Should Contain                 Please Enter Your Account Name and
Amount
   Page Should Contain Textfield       j_idt10:name
   Page Should Contain Textfield       j_idt10:amount
   Page Should Contain Button          j_idt10:j_idt18
   Input Text                          j_idt10:name             Bob
   Input Text                          j_idt10:amount           25.00
   Click Button                        j_idt10:j_idt18
   Wait Until Page Contains            The money have been deposited to Bob,
the balance of the account is 25.0
   Close Browser
```

Robot Framework tests are reasonably easy to read and understand, even if you have not seen them before. This test is going to open a browser and point it to our newly deployed application, then it will check that the expected page is shown, that it has the right fields on it, etc. Next it will enter some values in the fields, click on the submit button, then wait for the next page to load and check it has the expected content.

8 Running our build

Now we have everything in place, let's run the build. If you have been running the build to test things, you might need to clean up first. Make sure you do not have a WebLogic Server instance running, and remove the old domain directory. You can also delete the WebLogic Server installation if you want to.

Now execute the following command:

```
mvn verify
```

Maven will take the following actions:

- Install WebLogic Server from the WebLogic Server ZIP Distribution (if not already installed)
- Create a domain
- Start a server
- Compile and package our application into a WAR
- Deploy the WAR on the server and start the application
- Run the automated tests against the application
- Stop the server
- Remove the domain

Copyright 2015 Oracle Corporation. All Rights Reserved.

9 Review

In this Hands-on lab we have seen how to use the Oracle Maven Repository. When building applications for Fusion Middleware with Maven, we need a number of artifacts. In the 'normal' Maven world, Maven downloads artifacts it needs from remote repositories. The Oracle Maven Repository makes artifacts from Oracle available to developers in the 'normal' way – so you can work with Maven the way you expect to.

The Oracle Maven Repository contains Maven archetypes to create new projects, Maven plugins that provide the logic to build, test and deploy Fusion Middleware applications, and the libraries (dependencies) that you may need to build applications.

Before accessing the Oracle Maven Repository, you must register for access and accept the license agreement.

You need to update your Maven configuration to tell it about the Oracle Maven Repository and provide the authentication details so Maven can connect to the repository.

Once this is done, the Oracle Maven Repository behaves just like any other remote repository. You can use it to obtain and use artifacts in the 'normal Maven way'.

You can mix and match dependencies from multiple repositories in a project. In this Hands-on lab, we learned how to use artifacts from the Oracle Maven Repository – like the Basic WebApp Archetype and the WebLogic Maven plugin – alongside artifacts from other repositories – like the Robot Framework Maven plugin.

Copyright 2015 Oracle Corporation. All Rights Reserved.