FXF-DevOps – Migration from Ant To Maven

Table of Contents

1. Scope

The objective of the document is to enable developers the basic steps required for migration from Apache Ant to Apache Maven. Project teams/individuals who want to migrate the existing build tool Ant to Maven.

Keywords
Ant to Maven, Ant Migration, Maven Migration, Ant, Maven

H/W Platform: Windows 7
Tools and S/W Required: Java, Maven, Ant, Any latest version of Eclipse
Application Type: Web Based, SOA, EJB, and Batch Jobs
Project Type: Development/Maintenance

2. Introduction

Most of the legacy Java/J2EE applications will be using Apache ANT as the build tool.
When the applications are updated or restored to latest technology there might be a need to migrate new build tool from ANT to Maven.
Currently there is no automated tool available which will automatically migrate ANT scripts to Maven. This is because each project is created and organised differently.
This handbook will outline the process to migrate from ANT to Maven.

3. Ant and Maven Installation steps

3.1 Installation of Apache Ant

Apache Ant is a general purpose Java based build tool from Apache Software Foundation. Ant is an abbreviation for Another Neat Tool.Apache Ant's build files are written in XML and they take advantage of being open standard, portable and easy to understand. Ant is primarily used for building and deploying Java projects but can be used for every possible repetitive tasks, e.g. generating documentation.

Following are the steps for installing ANT

1. Make sure you have already downloaded and installed Java Development Kit (JDK) on your computer.
2. Ensure that the JAVA_HOME environment variable is set to the folder where your JDK is installed.
3. Download the Apache Ant binaries from **http://ant.apache.org**
4. Uncompress the downloaded file into a directory eg. c:\folder. using Winzip, winRAR, 7-zip or similar tools.
5. Create a new environment variable called **ANT_HOME** that points to the Ant installation folder, in this case c:\apache-ant-1.8.2-bin folder.
6. Append the path to the Apache Ant batch file to the **PATH** environment variable. In our case this would be the c:\apache-ant-1.8.2-bin\bin folder.

To verify the successful installation of Apache Ant on your computer, type ant -version on your command prompt as shown below..

```
C:\>ant -version
Apache Ant(TM) version 1.8.2 compiled on December 20 2010
```

This will indicate Ant is successfully installed.

3.2 Installation Of Apache Maven

Apache Maven is a software project management and comprehension tool to support the developer at the whole process of a software project. Typical tasks of a build tool are the compilation of source code, running the tests and packaging the result into JAR_ files.

Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

Maven allows the developer to automate the process of the creation of the initial folder structure for the Java application, performing the compilation, testing, packaging and deployment of the final product. It is implemented in Java which makes it platform-independent. Java is also the best work environment for Maven.

Following are the steps for installing Maven.

1. Make sure you have already downloaded and installed Java Development Kit (JDK) on your computer.
2. Ensure that the JAVA_HOME environment variable is set to the folder where your JDK is installed.
3. Download Maven 3.3.3 from http://maven.apache.org/download.cgi
4. Extract the downloaded file into a directory using Winzip, winRAR, 7-zip or similar tools.
5. Right click on MyComputer -> properties -> Advanced System Settings -> Environment variables -> click new button.
6. Now add MAVEN_HOME in variable name and path of maven in variable value. It must be the home directory of maven i.e. outer directory of bin. For example: c:\apache-maven-3.3.3
7. Click on new tab if path is not set, then set the path of maven. If it is set, edit the path and append the path of maven.
8. The path of maven should be %maven home%/bin. For example, c:\apache-maven-3.3.3\bin

To verify the successful installation of Apache Maven on your computer, type mvn -version on your command prompt as shown below..

```
C:\Users\AD5036558>mvn -version
Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T17:27:3
7+05:30)
Maven home: D:\apache-maven-3.3.3
Java version: 1.8.0_73, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_73\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "amd64", family: "dos"
```

This will indicate Maven is successfully installed.

4. Migration Steps

4.1: Understand the Source Code Structure:

In application source code, it would be divided across multiple projects i.e. web project, Java project, Generic projects etc. It is essential to understand the purpose of these projects, the dependencies of one project over the other and what third party Jars are required to build a particular project etc.
Once you have understood the source code organization and its dependencies it's time to document them.

Let's assume we have the below mentioned projects associated with an application.

1. "scheduling" is a standalone module and it is the first project which gets built.
2. "Source" is dependent on scheduling.
3. "Portal" is dependent on scheduling and source.

After you have analysed the projects and understood the dependencies create the below table:

Sr No.	Module	Module Dependencies	JAR Dependencies
1.	Scheduling	Independent	Weblogic.jar
2.	Source	Scheduling	castor.jar commons-httpclient-2.0-rc1.jar dom4j.jar
3.	Portal	Scheduling, source	castor.jar commons-httpclient-2.0-rc1.jar dom4j.jar

4.2: Understand the Existing ANT script:

This is one of the complex tasks involved when migrating from ANT to Maven. A thorough understanding of the existing ANT build process is necessary if you want to migrate from
ANT to Maven.

In some of the applications where migrated from ANT to Maven, we had to analyze many
Ant build scripts. Some of these ANT scripts were not even been used in the build process,
some had dead code in them.

Steps to understand the existing ANT scripts:

1. Document the environment variables used by ANT. Variables can be found in OS environment Variables as well as in build.cmd file.

   e.g.

Environment Variable	Value
APP_ROOT_DIR	D:/workspace
WEBLOGIC_HOME	D:/oracle/middleware

2. List down all the ANT scripts available in different Eclipse projects.
3. Manually try to understand each ANT script.

   a. List down the properties been used.
   e.g.

Property	Value	Actual Value
build.property.file.name	${project.dir}/build/build.properties	D:/workspace/build/build.properties
current.src.dir	${project.dir}/src	D:/workspace/src
weblogic.home	${bea.home}/wlserver_10.3	D:/Oracle/Middleware/wlserver_10.3
weblogic.server.lib.dir	${weblogic.home}/server/lib	D:/Oracle/Middleware/wlserver_10.3/server/lib

   b. List down the directories been created
   e.g.

Property	Value	Actual Value
build.dir	${project.dir}/build	D:/workspace/build
staging.dir	${project.dir}/staging	D:/workspace/staging
webapp.dir	${staging.dir}/webapp	D:/workspace/webapp
application.dir	${staging.dir}/application	D:/workspace/application
build.classes.dir	${build.dir}/classes	D:/workspace/classes
build.dist.parent.dir	${build.dir}/dist	D:/workspace/dist
build.dist.dir	${build.dist.parent.dir}/app2	D:/workspace/app2

c. List down all the ANT targets and tasks, what each target/task do.
e.g.

Target	Dependency	Logical Path	Physical Path	Comment
Prepare	init	${build.dir}	D:/workspace/build	Creates directories Prints "preparing target directories"
copy_admin_properties	init	${base.dir}/properties	D:/workspace/src/properties	Copies admin properties file. Prints "Copying portal property files".
compileSrc	Prepare	${batch.src.dir} (Source) ${batch.classes.dir} (Destination)	D:/workspace/source/src	Compiles java source files. Prints "Compiling Java source files for module Source".

d. List down all the artifacts created.

4. Determine the parent ANT script which calls sub ANT scripts.
5. Determine the sequence in which the sub ANT scripts are called.
E.g.

```
<target name="all" depends="prepare, clean, wlw.sched.build,
app.common.classes.jar, package"/>
```

Targets are called in following order:

all
prepare
clean
wlw.sched.build
app.common.classes.jar
package
compileall
package.portal.ear
package.commonaccess.war
compileall

prepare:
init

compileall:
compileSource
prepare
compile
compilePORTAL
prepare
compile
copy_database_scripts
prepare

package.commonaccess.war:
prepare

package.portal.ear:
package.gui.ear
package.mdb
package.entity.ejb….

6. Determine the dependency of one ANT script over the other.
7. Run the ANT scripts with verbose and debug ON so that you can understand how the build happens.
8. Try to correlate your manual understanding (Step 2) with ANT log output (Step 6).
9. Determine the source code present in Eclipse project is used for creating different artifacts. In one of the eclipse projects we need to create 3 artifacts – 1 jar, 1 Entity Bean Jar and 1 Session Bean Jar.

E.g.

Project	Artifacts
scheduling	JAR
source	JAR, Entity JAR, Session JAR

This is needed because based on the number of artifacts created using 1 project you may end up creating that many POM files.

4.3: Install Maven Third Party JARs:

1. Using Step 1 identify the list of third party JARs which are needed for the build.
2. Determine whether the jars are available in Maven Central repository or Client Maven central repository.
3. If jars are available in Maven repository then we need not install the jars manually into central repository/local repository.
4. If jars are not available then we have to install the JARs manually.
   e.g.

```
mvn install:install-file -Dfile=D:\workspace\JARs\lib\castor.jar
-DgroupId=org.exolab -DartifactId=castor -Dversion=1.0
-Dpackaging=jar
```

```
mvn install:install-file -Dfile=D:\workspace\JARs\lib\dom4j.jar
-DgroupId=org.dom4j -DartifactId=dom4j -Dversion=1.0                    -
Dpackaging=jar
```

5. While installing the JARs we need to determine the value to be used for '-Dversion'.
   The version can be present either in the JAR file name itself or as a part of manifest file within the JAR. If it is not present in both these places then we need to use a default value. Make sure you use a unique value so that you don't overwrite anything in Maven repository.

   Note:
   Create an automated script to install the JARs. Since the JARs might have to be installed in different local maven repositories first and then in the central repository, by automating we can save time and eliminate errors.

4.4: Application Server JARs:

If you are using an Enterprise Appserver i.e. Websphere, Weblogic and if the JARs provided by the Appserver are needed for building the application then you need to decide whether these JARs can be accessed from the Appserver installation directory or you want to install these JARs as well in the Maven repository.

Would strongly recommend to use the JARs from the installation directory since if you start installing the Appserver JARs it can turn out to be a never ending process. For our migration we started installing Weblogic JARs and even after installing 60+ Weblogic JARs the build was still failing. Each time the build failed we had to determine which JAR contained the missing class, then had to install that JAR. Also, sometimes even if you install all the JARs some of the ANT tasks provided in the Appserver JARs, might not work since it couldn't resolve the dependent JARs

4.5: Document different types of artifacts which need to be created:

Each Maven POM file is used to create just one artifact i.e. JAR, WAR, EAR, TAR etc. So we need to determine different types of artifacts which needs to be created and create sample POMs for that.
The **pom.xml** file is the core of a project's configuration in Maven. It is a single configuration file that contains the majority of information required to build a project in just the way you want. The POM is huge and can be daunting in its complexity, but it is not necessary to understand all of the intricacies just yet to use it effectively.

In the application which we migrated we had to create the following artifacts:

1. JAAS authentication JAR
2. JAR
3. EJB – Entity
4. EJB – Session
5. EJB – Message
6. WAR
7. EAR
8. TAR

4.6: Determine if source code directory structure can be refactored:

Maven requires specific project structure to be in place for the POM to work out of box.

src/main/java	Application/Library sources
src/main/resources	Application/Library resources
src/main/filters	Resource filter files
src/main/webapp	Web application sources
src/test/java	Test sources
src/test/resources	Test resources
src/test/filters	Test resource filter files
src/it	Integration Tests (primarily for plugins)
src/assembly	Assembly descriptors
src/site	Site
LICENSE.txt	Project's license
NOTICE.txt	Notices and attributions required by libraries that the project depends on
README.txt	Project's readme

Decide whether you want to rearrange the existing source code or you want to maintain the existing directory structure as it is. In most cases it will be tedious to re-arrange the code. Also, you may not to want disturb the existing structure.

4.7: Start writing Maven POM files:

Once you have understood the source code, ANT scripts and what artifacts you have to build its time to write maven POM files.

Remember the following while writing the POM files:

1. An eclipse project can contain more than 1 POM.
2. Write one POM to create each one of the artifacts identified. Once you have done this, it will mostly copy paste to create similar artifacts.
3. Using ANT you can copy Java classes from different projects and create a single JAR out of it. But through Maven, you cannot create a single JAR – the single JAR would be broken into multiple sub jars. This won't affect the functionality.
4. Try to use Skinny WAR plugin if the JARs are used by multiple WARs/EJBs.
5. If an eclipse project contains more than 1 POM, then you have excluded the MAVEN target directories explicitly.
6. If you are using ANT tasks provided by Enterprise Appservers in the ANT scripts then you can use them as in Maven.
7. Create an environment variable for the Maven build number and use them in the POMs. If you hard code it then you have modify in multiple places in future.
8. Create a Parent POM and call the sub POMs from parent POMs.
9. If you haven't rearranged the directory structure then you have to provide the source directory explicitly.

```
<build>
   <resources>
               <resource>
               <!—Adds resource directory for compiling java classes -->
<directory>src</directory>
```

The Sample POM

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2
   001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/x
   sd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.
5.   <groupId>com.mycompany.app</groupId>
6.   <artifactId>my-app</artifactId>
7.   <version>1.0-SNAPSHOT</version>
8.   <packaging>jar</packaging>
9.
10.  <name>Maven Quick Start Archetype</name>
11.  <url>http://maven.apache.org</url>
```

```
12.
13.   <dependencies>
14.     <dependency>
15.       <groupId>junit</groupId>
16.       <artifactId>junit</artifactId>
17.       <version>4.8.2</version>
18.       <scope>test</scope>
19.     </dependency>
20.   </dependencies>
21. </project>
```

You executed the Maven goal archetype:generate, and passed in various parameters to that goal. The prefix archetype is the plugin that contains the goal. If you are familiar with Ant, you may conceive of this as similar to a task. This goal created a simple project based upon an archetype. Suffice it to say for now that a plugin is a collection of goals with a general common purpose.

Build the Project

```
1. mvn package
```

The command line will print out various actions, and end with the following:

```
1.   ...
2. [INFO] ------------------------------------------------------------------------
3. [INFO] BUILD SUCCESSFUL
4. [INFO] ------------------------------------------------------------------------
5. [INFO] Total time: 2 seconds
6. [INFO] Finished at: Mon May 22 13:34:52 CEST 2017
7. [INFO] Final Memory: 3M/6M
8. [INFO] ------------------------------------------------------------------------
```

Unlike the first command executed (archetype:generate) you may notice the second is simply a single word - package. Rather than a goal, this is a phase. A phase is a step in the build lifecycle, which is an ordered sequence of phases. When a phase is given, Maven will execute every phase in the sequence up to and including the one defined. For example, if we execute the compile phase, the phases that actually get executed are:

1. validate
2. generate-sources
3. process-sources
4. generate-resources
5. process-resources
6. compile

4.8: Running Maven Tools:

Maven Phases

Although hardly a comprehensive list, these are the most common default lifecycle phases executed.

- **validate**: validate the project is correct and all necessary information is available
- **compile**: compile the source code of the project
- **test**: test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- **package**: take the compiled code and package it in its distributable format, such as a JAR.
- **integration-test**: process and deploy the package if necessary into an environment where integration tests can be run
- **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.

There are two other Maven lifecycles of note beyond the default list above. They are

- **clean**: cleans up artifacts created by prior builds
- **site**: generates site documentation for this project

Note:

```
1. mvn clean dependency:copy-dependencies package
```

This command will clean the project, copy dependencies, and package the project (executing all phases up to package, of course).

Generating the Site

```
1. mvn site
```

This phase generates a site based upon information on the project's pom. You can look at the documentation generated under target/site.

4.9: Compare the artifacts:

Once you have generated the artifacts using the POMs it's time to compare them with the artifacts generated using ANT. We used BeyondCompare for this activity. This will help to statically compare the artifacts and make sure they are same.