Maven 2.0

Topics

- Introduction to Maven
- Running Maven
- Archetype & Creating Archetype
- Maven Plugging
- POM Files
- Repositories
- Eclipse Integration

Maven

- Maven, - *accumulator of knowledge,*

- A standard way to build the projects, a clear definition of what the project consisted of, an easy way to publish project information and a way to share JARs across several projects.

- Maven is build tool
  - to build deployable artifacts from source code.
  - preprocessing, compilation, packaging, testing, and distribution

- Project Management Tool
  - To Help In Project Management
  - run reports, generate a web site, and facilitate communication among members of a working team.

Maven- Convention over Configuration

- Maven is based on conventions.
- The Location of source code is known is because of the convention used by It.
- Without customization, source code is assumed to be in ${basedir}/src/main/java
- Having a source in the correct directory, is based requirement if that is done Maven will take care of the rest.

Running Maven

- *Maven is a Java tool, Java should be Installed*
- Download Maven :http://maven.apache.org/download.html
- Unzip the installation archive
- Set the M2_HOME and Path environment variables in the following way:
  - **M2_HOME=C:\apache-maven-2.2.1**
  - **Path=%M2_HOME%\bin**
- **mvn –version**
  - Will print out installed version of Maven, indicating successful installation .

Archetype

- Archetype is a Maven project templating toolkit.
- Its a standard plug-in which comes with Maven 2.
- The Archetype plug-in runs outside of a Maven project build life cycle and is used for creating Maven projects.
- Archetype will help authors create Maven project templates
- Once these archetypes are created and deployed in repository they are available for use by all developers within organization.

Using an Archetype

- To create a new project based on an Archetype,
- **mvn archetype:create goal**
- **maven-archetype-simple**
  - An archetype which contains a simple Maven project.
- **maven-archetype-webapp**
  - An archetype which contains a sample Maven Webapp project.
- mvn archetype:create -DgroupId=com.mycompany.app -DartifactId=my-webapp -DarchetypeArtifactId=maven-archetype-webapp

Maven Plugin

- Most of the action in Maven happens in plugin goals
  - compiling source,
  - packaging bytecode,
  - publishing sites

- Maven Lifecycle and other Goals are done through these Plugins

- Plugins are retrieved from the Maven Repository maintained centrally and shared universally.

- **mvn install**

  - With New Maven installation will retrieve most of the core Maven plugins from the Central Maven Repository.

Maven Repositories

- Maven repositories store a set of artifacts which are used by Maven during dependency resolution for a project.
- Local repositories can be accessed on the local hard disk.
- Remote repositories can be accessed through the network.
- An artifact is bundled as a JAR file which contains the binary library or executable.
- An artifact can also be a war or an ear.
- After downloading repository, Maven will always look for the artifact in the local repository before looking elsewhere.

Local Repository

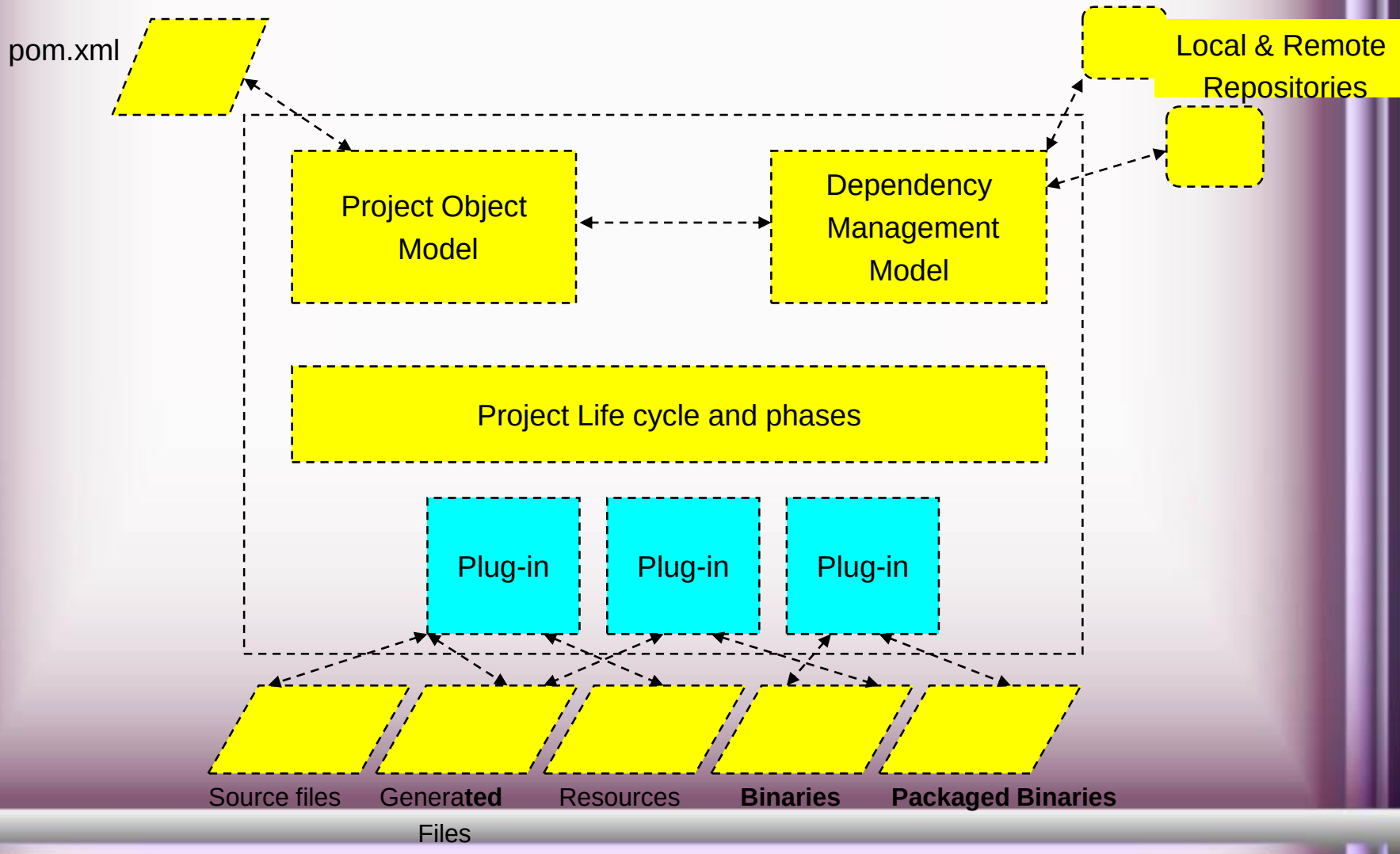
- On Windows XP,
  - **C:\Documents and Settings\USERNAME\.m2\repository**
- **mvn install** command,
- Maven will install project's artifact in local repository. .

Resolving Dependency

- During the build process, the project dependencies are resolved with the help of Maven 2 dependency management engine.
- Dependencies are specified in <dependencies> elements within a pom.xml file.
- Project dependencies are stored on repository servers. The dependencies resolver attempts to resolve the dependencies in the following order:

a. Your local repository is checked for the dependency.
    a. **A list of remote repositories is checked for the dependency.**
    b. **In case, 1 and 2 fail, an error is reported.**
        - **By default, the first remote repository contacted in step 2 is a worldwide-accessible centralized Maven 2 repository containing artifacts for most popular open source projects.**

Physical Overview of Maven 2



POM File

- The src/test/java directory contains he pom.xml is the project's Project Object Model, or POM.
- POM is Maven’s understanding of a project.
- Dependencies are specified as a part of pom.xml file.

- Dependent components are known as artifacts, they are resolved in remote repositories are download to the local repository.

- The plug-ins are handled as artifacts by the dependency management model and are downloaded on demand.

POM.XML

```
- - - - schema defination ---
<groupId>com.mycompany.app</groupId>
<artifactId>my-app</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
<name>Maven Quick Start Archetype</name>
<url>http://maven.apache.org</url>
<dependencies>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>3.8.1</version>
  <scope>test</scope>
</dependency>
</dependencies>
</project>
```

POM.XML

- **project**
  - This is the top-level element in all Maven pom.xml files.
- **groupId**
  - This element indicates the unique identifier of the organization or group that created the project. The groupId is one of the key identifiers of a project and is typically based on the fully qualified domain name of your organization.
- **artifactId**
  - This element indicates the unique base name of the primary artifact being generated by this project. The primary artifact for a project is typically a JAR file.

POM.XML

- **packaging**
  - This element indicates the package type to be used by this artifact (e.g. JAR, WAR, EAR, etc.).
- **Executing Phases of Application**
- **mvn compile**
  - To Compile the application
- **Mvn test**
  - To Test the application

Phases in Maven

- *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.

- **compile:**
  - compile the source code of the project

- **package:**
  - take the compiled code and package it in its distributable format, such as a JAR.

- **deploy:**
  - done in an integration or release environment, copies the final package to the remote

Creating a Simple Project

- Maven Archetype plugin is used to Start a new Maven project,
- **archetype:generate**
- select archetype #15, and then enter "Y" to confirm and generate the new project:
- To build and package the application, the following command is used from the directory containing pom.xml
- **Mvn compile - fr**om the command line.

Execute a Simple Project

- **mvn exec:java -Dexec.mainClass=com.training.App**

- **java -cp target/simple-1.0-SNAPSHOT.jar org.sonatype.mavenbook.App**

Creating a Goal

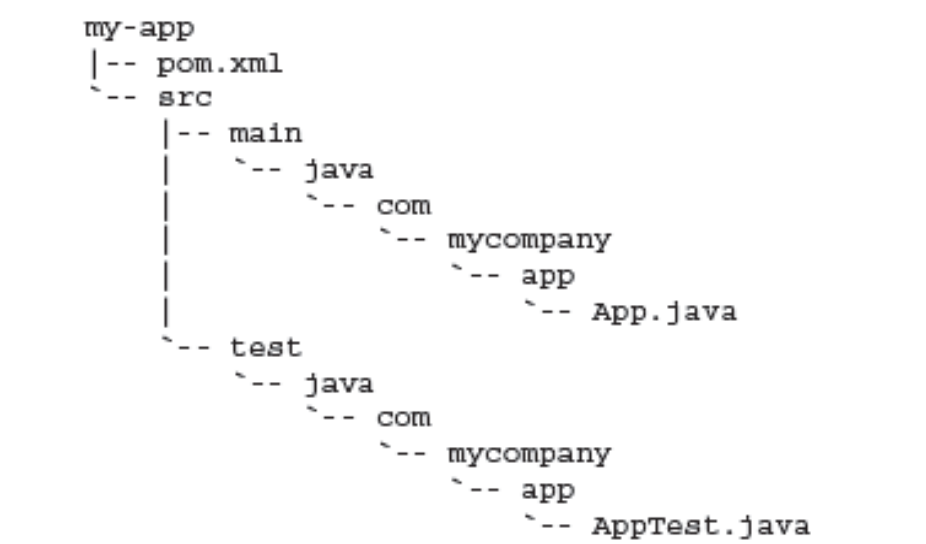
```
<plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
         <version>1.1</version>
    <executions>
    <execution>
     <phase>test</phase>
     <goals>
     <goal>java</goal>
     </goals>
     <configuration>
          <mainClass>com.training.App</mainClass>
  </configuration>
    </execution>
   </executions>
  </plugin>
```

Creating a Project

- **mvn archetype:create -DgroupId=com.mycompany.app -DartifactId=my-app**

- *Ceate goal will create a directory with the same name given as the artifactId.*

- Under this directory standard project structure will be created



Execute the Application

- **mvn package** will execute a series of phases like
  - Validate
  - generate-sources
  - process-sources
  - generate-resources
  - process-resources
  - compile
- **java -cp target/my-app-1.0-SNAPSHOT.jar com.mycompany.app.App**
  - Which Will execute the Application App.java

Adding a Unit test

- It is a development best practice to provide unit tests for all code modules.
- Maven 2 created a placeholder AppTest.java unit test
- Can Remove the file and place customized unit testing code
- Use **mvn test** command.

Site Generation and Reporting

- Maven is its ability to generate documentation and reports.
- **mvn site**
- This will execute the site lifecycle phase.
- This lifecycle is concerned solely with processing site content under the src/site directories and generating reports.
- After this command executes, you should see a project web site in the target/site directory.

Creating the Simple Web Project

- To create web application project,
- **mvn archetype:generate**
  - **choose archetype #18 -> maven-archetypewebapp",**
  - **Enter an artifactId and** a groupId.
- Press Y to confirm and a new web application project is created

WebApplication Directory

- |-- src |
- `-- main |
  - `-- java |
- |-- resources |
- |-- webapp |
- | `-- WEB-INF |
  - | `-- web.xml
  - | `-- index.jsp `
- -- pom.xml

Configuring Maven Plugin – Java 5

- POM for the simple-web project with compiler configuration

```
[……]
<build>
    <finalName>simple-webapp</finalName>
<plugins>
    <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
                <source>1.5</source>
                <target>1.5</target>
      </configuration>
</plugin>
</plugins>
</build>
[…….]
```

Jetty Plugin

- Before launching the Jetty plugin, compile
  - **mvn compile**
- Web Application is launched by
  - **mvn jetty:run**

- The command is executed from
  - **simple-webapp/ project**

- After the Jetty Servlet container, starts,type in web browser

  - **http://localhost:8080/simple-webapp/**

- Maven expects the document root of the web application to be stored in src/main/webapp.

Configuring the Jetty Plugin

- To Deploy a Web Application after compilation in servlet container
- Maven Jetty plugin can be used to run web application within Maven by configuring project's pom.xml.

```
<project>
[...]
<build>
    <finalName>simple-webapp</finalName>
      <plugins>
          <plugin>
                <groupId>org.mortbay.jetty</groupId>
                <artifactId>maven-jetty-plugin</artifactId>
        </plugin>
</plugins>
</build>
[...]
</project>
```

Configuring Maven with eclipse

- Eclipse needs to know the path to the local maven repository.
- Classpath variable **M2_REPO** has to be set.

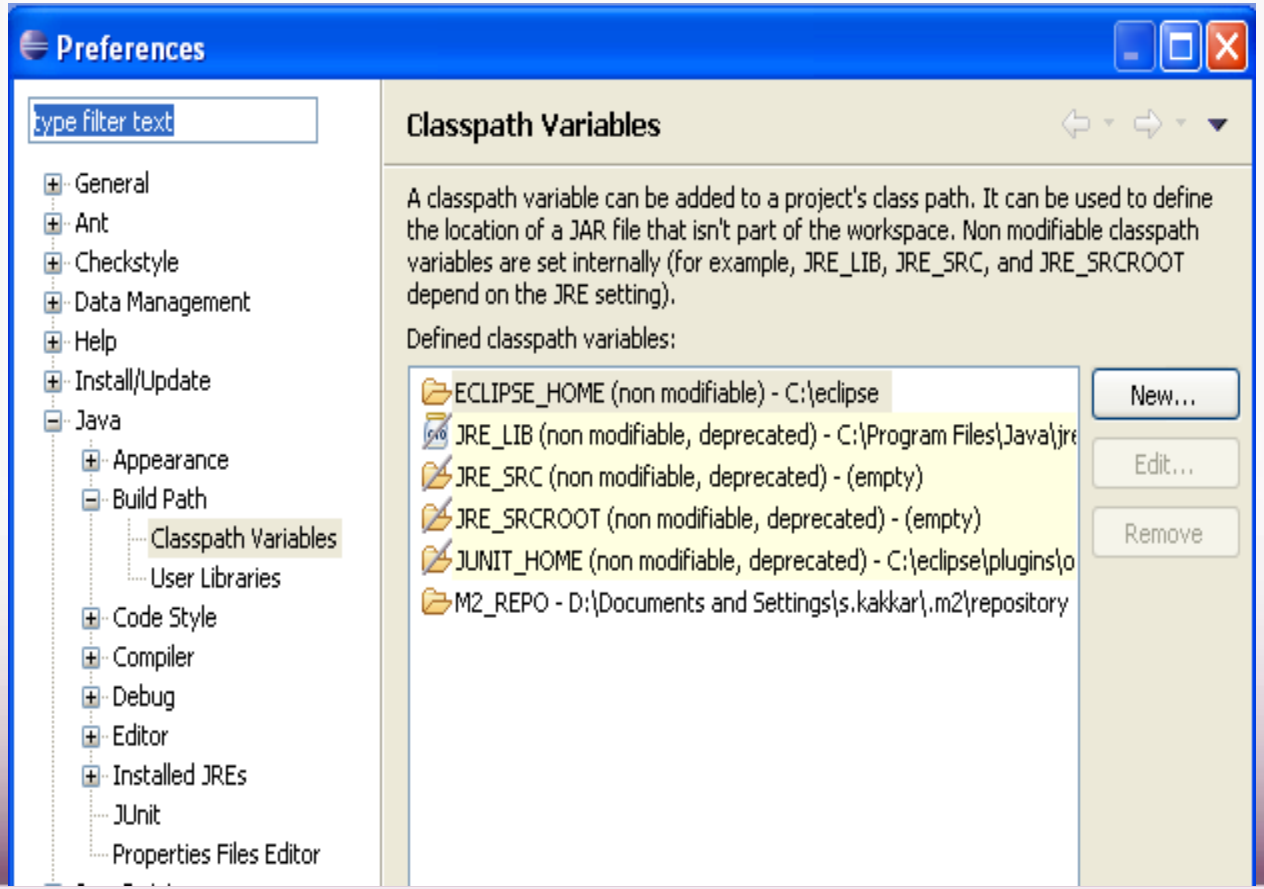
```
C:\MavenExample-1>mvn -Declipse.workspace=C:\Test eclipse:add-maven-repo
```

- The execution of the above command should result in the following:

```
C:\MavenExample-1>mvn -Declipse.workspace=C:\Test eclipse:add-maven-repo
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'eclipse'.
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Default Project
[INFO]    task-segment: [eclipse:add-maven-repo] (aggregator-style)
[INFO] ------------------------------------------------------------------------
[INFO] [eclipse:add-maven-repo {execution: default-cli}]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2 seconds
[INFO] Finished at: Sat Oct 03 17:35:46 GMT+05:30 2009
[INFO] Final Memory: 7M/13M
[INFO] ------------------------------------------------------------------------
```

Configuring Maven with eclipse (Continued..)

- Now open up your eclipse and check if this variable **M2_REPO** has been added.



Creating eclipse specific files

- To generate the eclipse metadata files from your **pom.xml** you execute the following command:

```
C:\MavenExample-1\SampleProject>mvn eclipse:eclipse
```

- The above command generates eclipse related files like .project and .classpath .

```
[INFO] Wrote Eclipse project for "SampleProject" to C:\MavenExample-1\SampleProj
ect.
[INFO]
       Sources for some artifacts are not available.
       Please run the same goal with the -DdownloadSources=true parameter in ord
er to check remote repositories for sources.
       List of artifacts without a source archive:
         o junit:junit:3.8.1

       Javadoc for some artifacts is not available.
       Please run the same goal with the -DdownloadJavadocs=true parameter in or
der to check remote repositories for javadoc.
       List of artifacts without a javadoc archive:
         o junit:junit:3.8.1

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4 seconds
[INFO] Finished at: Sat Oct 03 17:44:40 GMT+05:30 2009
[INFO] Final Memory: 7M/15M
[INFO] ------------------------------------------------------------------------
```

Importing the project in Eclipse

- After generating the Eclipse Classpath and Project files for your project, import the project into Eclipse with the following steps:

1. From the Eclipse workspace menu select File > Import

2. Expand General and select Existing Projects into Workspace

3. Mark Select root directory and hit Browse. Browse for the directory in which your POM file is in. Press OK.

4. You should now see your project checked under the Projects: box. Press Finish.

5. You should now see your project in Eclipse.

Running Maven outside eclipse

- Delete all the files under C:\MavenExample-1\SampleProject\target
- Let's now use Maven to build it as a jar and try to run it outside eclipse. Execute the following command from the command line:

```
C:\MavenExample-1\SampleProject>mvn clean package
```

- You can see SampleProject-1.0-SNAPSHOT.jar under C:\MavenExample-1\SampleProject\target. Based on the pom.xml file, the jar was built.

- You can run the Java application in the command line as follows:

```
C:\MavenExample-1\SampleProject>java -cp .\target\SampleProject-1.0-SNAPSHOT.jar
 com.accenture.works.SimpleExample
```

Running Maven from Eclipse

- Eclipse provides a feature which allows you to run external programs.
- This feature can be utilized to insert Maven commands to a list of external tools which are available to your workspace.
- Go through the following steps for the setup:
  1. From the Eclipse workspace menu select **Run > External Tools > External Tools Configurations...**
  2. Select **Program** from the left.
  3. Right click on **Program** and select **New.**
  4. Provide a name for this configuration. This name is used for identifying your maven command in the list of external tools. E.g. **mvn install**.
  5. Press **Variables…** (on the right hand side).
  6. Press **Edit Variables…**

Running Maven from Eclipse (Continued)

7. Press **New…**

8. Enter the following:

   - **Variable** = maven_exec
   - **Value** = C:\apache-maven-2.2.1\bin\mvn.bat
   - **Description** = Maven Executable

9. Press **Ok**

10. Select **maven_exec** from the **Select Variable** and press **ok**.

11. Press **Browse Workspace...** under **Working Directory** and select the folder in which your project's POM file is located if you want to launch a Maven goal within a single project

Running Maven from Eclipse (Continued)