



Maven

Next Generation Build Tool



Andreas Schaefer, Senior Software Engineer

andreas.schaefer@madplanet.com

OnJava.com: Maven Tips & Tricks

Member of JSR-77 expert group

Co-author of JMX book

Maven: Table of Content



- Maven Presentation:
 - ✱ **Design**
 - ✱ Project Object Model
 - ✱ Plugins & Goals
 - ✱ Properties
 - ✱ Scripting with Jelly
 - ✱ Your own Plugins
 - ✱ Convert your Ant Projects
 - ✱ Troubleshooting
 - ✱ Q&A



- Collection of Tasks
- Tasks are independent
- Build Script executes them one after the other in order they appear in the target and the target's dependencies
- Tasks are compiled
- Scripting possible but difficult

Ant's Problems



- Long Scripts because every Task needs to be fully qualified
- All Properties must be set
- No common Build Logic
- Tasks are compiled Java classes
 - ✱ Hard to Change
 - ✱ Hard to Test
 - ✱ Hard to See what is going on
- Scripting optional and difficult to use



- Project Model
- Collection of Plugins
- Plugins expose as set of Goals
- Plugins can call other Goals
- Maven provides a Build Process through the Project Model and the plugins
- Scripting is integrated part of the build script
- Ant is natively supported

Maven: Build Components



- Project Object Model (POM)
- Base POM if POM is extended
- Properties files
- Maven Scripting File: **maven.xml**
- Project components like source and test code, resource files etc



• To generate a Maven Project:

• maven genapp

- Enter a template or select the **default**
 - Enter the application id
 - Enter a human readable name
 - Enter a project package path like org.test.myapp
- Maven will generate a project with a POM, properties file and source and test code

Maven: Table of Content



- Maven Presentation:
 - ✱ Design
 - ✱ **Project Object Model**
 - ✱ Plugins & Goals
 - ✱ Properties
 - ✱ Scripting with Jelly
 - ✱ Your own Plugins
 - ✱ Convert your Ant Projects
 - ✱ Troubleshooting
 - ✱ Q&A

Project Object Model: POM



- Collection of common Data of a Project
- Defines
 - ✱ Project Name and other Information
 - ✱ Version Control System
 - ✱ Dependencies
 - ✱ Source / Test Code and their Resources
- Usually stored in a file called 'project.xml'



POM: Inheritance and Interpolation

● Inheritance

- ✱ Extend Tag specifies a Base POM
- ✱ Base POM provides Default Values
- ✱ The Properties and Scripting File of the Base POM are **disregarded** (at least it does now)

● Interpolation: **could change in the future**

- ✱ POM is a Jelly Script
- ✱ Properties and Functions can be used



POM: Dependencies

- Artifacts used to compile/run a Project
- Loaded from the Local Repository
- Downloaded from Remote Repository if necessary

```
<dependencies>
  <dependency>
    <groupId>xdoclet</groupId>
    <artifactId>xdoclet-ejb-module</artifactId>
    <version>1.2b1</version>
    <type>jar</type> <!-- 'jar' is the default so not necess.-->
  </dependency>
</dependencies>
```

{repository}/<groupId>/<type>s/<artifactId>-<version>.<type>



• Different Artifact Name

```
<dependencies>
  <dependency>
    <groupId>my-group</groupId>
    <jar>my-archive</jar>
    <type>my-type</type>
  </dependency>
</dependencies>
```

{repository}/<groupId>/<type>s/<jar>.<type>



POM: Dependencies III

- **maven jar:install** places a generated Artifact into the Local Repository
- Local Repository can share binaries between Projects
- Repository to share binaries within a Network

```
# Add the companies repository after the Maven repos.  
maven.repo.remote =  
  http://www.ibiblio.org/maven/  
  http://www.mycompany.com/maven/
```

These Repositories enables a Project to finish a build even when another Project build fails because the binaries maybe already there



- Maven provides many plugins providing Project Reports
 - Junit
 - JavaDoc
 - Checkstyle
 - XRef
 - ...
- Within the POM the list of reports can be reduced
- Project Reports are generated with:
 - **maven site**



- **/xdocs/navigation.xml** allows to customize the generated Project Site (XDoc)

```
<project name="JDoppio">
  <title>JDoppio: The Next Generation Application Server</title>
  <body>
    <links>
      <item name="Maven" href="http://maven.apache.org/" />
    </links>
    <menu name="Definition">
      <item name="Download" href="download.html" />
      ...
    </menu>
    <menu name="Projects">
      <item name="Utility Project" href="./multiproject/jdoppio.util/index.html" />
      ...
    </body>
  </project>
```

Maven: Table of Content



- Maven Presentation:
 - ✱ Design
 - ✱ Project Object Model
 - ✱ **Plugins & Goals**
 - ✱ Properties
 - ✱ Scripting with Jelly
 - ✱ Your own Plugins
 - ✱ Convert your Ant Projects
 - ✱ Troubleshooting
 - ✱ Q&A



- A Plugin provides a certain Building Process
- Plugins can be viewed as **Reusable Build Scripts**
- It may **depend** on other Plugins and can call them
- If not needed it will not be executed
- **Properties** can change their behavior
- Maven provides some plugins, some are provided by other sources like Source Forge
- Plugins can be created and installed **locally**



- Every Plugin exposed a Set of Goals
- Goal Convention: **<Plugin Shortcut>:<Goal>**
- A Plugin can specify a default goal which is executed when called with only the **Plugin Shortcut**
- Examples:
 - maven java:compile
 - maven test:test
 - maven ssh-deploy

Goals II



- List all plugins and their goals
 - **maven -g**
- The Goals of a Plugin: Java
 - **maven -P java**

Goals in java

=====

```
[java]                ( NO DEFAULT GOAL )
  compile .....Compile the project
  jar .....Create the deliverable jar file.
  jar-resources .....Copy any resources that must be present in the deployed JAR file
  prepare-filesystem ....Create the directory structure needed to compile
```



- Plugins can be found:
 - ✱ 1.0 or higher: `~/.maven/cache`
 - ✱ Before 1.0: `~/.maven/plugins`
- Plugin Scripts are in file: `plugin.jelly`
- Plugin Default Properties are in file: `plugin.properties`
- Newer versions of a Plugin can be installed with:
 - ✱ `maven plugin:download -DgroupId=maven`
`-DartifactId=maven-dashboard-plugin -Dversion=1.6`
- Plugin Scripts are great Source of Ideas how to Script in Maven



Third Party Plugins

- To install them from a Remote Repository:
 - ✱ Execute: `maven plugin:download`
 - ✱ It will ask you for the plugin name, the group name and the plugin version
 - ✱ Then it will download and install this plugin **if found**
- To install them otherwise:
 - ✱ Download the Plugin's archive
 - ✱ Place it into the **<maven-home-directory>/plugins**
 - ✱ The next time Maven is executed the plugin cache will be updated and the plugin is installed

Maven: Table of Content



- Maven Presentation:
 - ✱ Design
 - ✱ Project Object Model
 - ✱ Plugins & Goals
 - ✱ **Properties**
 - ✱ Scripting with Jelly
 - ✱ Your own Plugins
 - ✱ Convert your Ant Projects
 - ✱ Troubleshooting
 - ✱ Q&A



- Configures the Build Process
- Overwrites Plugin Default Values
- Can be **Read** and **Set** Dynamically at Runtime
- Processing Order
 - Plugin Default Properties (if a Property of a Plugin)
 - `${project.home}/project.properties`
 - `${project.home}/build.properties`
 - `${user.home}/build.properties`
 - Properties set by the Command Line with **-D**
- The **Last Definition** wins (in contrast to Ant)

Plugin Properties



- Most Plugin provide Defaults for their Properties
- If **not** the user **must provide one**, otherwise build fails
- Plugin Properties can be read at runtime:
 - `<maven:get plugin="maven-java-plugin" property="maven.compile.source" var="source.version" />`
- Plugin Properties can be set at runtime:
 - `<maven:set plugin="maven-test-plugin" property="maven.junit.sysproperties" var="sys.properties" />`
-

Maven: Table of Content



- Maven Presentation:
 - ✱ Design
 - ✱ Project Object Model
 - ✱ Plugins & Goals
 - ✱ Properties
 - ✱ **Scripting with Jelly**
 - ✱ Your own Plugins
 - ✱ Convert your Ant Projects
 - ✱ Troubleshooting
 - ✱ Q&A



Introduction into Scripting

- **Maven = Jelly** Scripting (!)
- Project Script is in file **maven.xml**
- Simplest Scripting File:

```
<project default="jar:jar"  
  xmlns:j="jelly:core"  
  xmlns:maven="jelly:maven"  
  xmlns:ant="jelly:ant"  
>  
</project>
```

j, **maven**, **ant** are Jelly name spaces
jar:jar is the default goal to be executed



• Usages of Goals

```
<!-- New Goal -->
<goal name="my-very-own-goal">
  <!-- Call another Goal -->
  <attainGoal name="test:compile"/>
  ...
<!-- Add this script in front of a Goal -->
<preGoal name="test:compile">
  ...
<!-- Add this script at the end of a Goal -->
<postGoal name="test:compile">
  ...
```

- ✱ Pre/Post Goals are extending this goals
- ✱ Pre/Post Goals are executed if the goal is called somewhere else



• Jelly Scripting to build a Manifest's Class-Path:

```
<j:set var="ear.archives" value="" />
<j:forEach var="dep" items="${pom.dependencies}">
  <j:if test="${dep.getProperty('ear.library')==true}">
    <j:set var="ear.archives" value="${ear.archives} ${dep.artifact}" />
  </j:if>
</j:forEach>
<ant:jar jarfile="${component.build.root.dir}/ear/ejbs.jar">
  ...
```

- List of all Entries in the POM Dependencies
- Check if a certain property is set
- If set it will add it to a property separated by a space

Create Own Jelly Tag Library



- Create Tag and Tab Library Class:

```
package jelly.patch;
import org.apache.commons.jelly.TagLibrary;
public class PatchTagLibrary extends TagLibrary {
    public PatchTagLibrary() {
        registerTag("patch", jelly.patch.PatchTag.class);
        registerTag("getStatic", jelly.patch.GetStaticTag.class);
    }
}
```

For more Info on Tag Classes check out Jelly:
<http://jakarta.apache.org/commons/jelly/>

- Create archive and put it into **<maven>/lib** directory



Use Own Jelly Tag Library

- Declare Tag Library and Name Space
- Use the Tags with its Name Space as Prefix

```
<project default="jar:jar"  
  xmlns:j="jelly:core"  
  xmlns:myjelly="jelly:jelly.patch.PatchTagLibrary"  
>  
  <goal name="my-very-own-goal">  
    <myjelly:getStatic .../>  
  </goal>  
</project>
```

Maven: Table of Content



- Maven Presentation:
 - ✱ Design
 - ✱ Project Object Model
 - ✱ Plugins & Goals
 - ✱ Properties
 - ✱ Scripting with Jelly
 - ✱ **Your own Plugins**
 - ✱ Convert your Ant Projects
 - ✱ Troubleshooting
 - ✱ Q&A

Create Own Plugin



- Create a Maven Project (POM) to build the Plugin
- Create a plugin.jelly file containing the Plugin's script that looks like a maven.xml file
- Create a plugin.properties containing the default values
- Install the plugin: **maven plugin:install**



● Plugin Structure defined in the POM

✿ Basic Components

```
...
<build>
  <resources>
    <resource>
      <directory>${basedir}</directory>
      <includes>
        <include>plugin.jelly</include>
        <include>plugin.properties</include>
        <include>project.properties</include>
        <include>project.xml</include>
      </includes>
    </resource>
  ...
```



- Plugin Structure defined in the POM

- Plugin Resources

```
...
  <resource>
    <directory>${basedir}/src/resource/lib</directory>
    <targetPath>plugin-resources</targetPath>
    <includes>
      <include>*.jar</include>
    </includes>
  </resource>
</resources>
</build>
...
```

/plugin-resources is the default directory for a plugin to look for its resources



Use your own Plugin

- Can be used like any other Plugin
 - Call the plugin directly with:
maven myplugin:mygoal
 - Call the plugin goal inside your **maven.xml** file with
<attainGoal>
 - Set the **default goal** of you project to a goal of your plugin

Maven: Table of Content



- Maven Presentation:
 - ✱ Design
 - ✱ Project Object Model
 - ✱ Plugins & Goals
 - ✱ Properties
 - ✱ Scripting with Jelly
 - ✱ Your own Plugins
 - ✱ **Convert your Ant Projects**
 - ✱ Troubleshooting
 - ✱ Q&A

Simple Man's Conversion



- A straight approach to convert an Ant project is:
 - ✱ Rename **build.xml** into **maven.xml**
 - ✱ Drop the **name** and **basedir** from the **<project>** tag
 - ✱ Rename all **<target>** tags into **<goal>**
 - ✱ Rename all '**depends**' attribute to '**prereqs**' in the goal tag
 - ✱ Execute the build with **Maven**
- Beware of the **different properties setting** order



Advanced Conversion Tips

- Convert **<antcall>** into **<attainGoal>** tags
- Convert **<ant>** by the following:
 - Copy the **build.xml** into another directory (if not already done)
 - Rename the **build.xml** file to **maven.xml**
 - Create a POM there ('**maven genapp**' for example)
 - Define the Maven Jelly Taglibrary in the **<project>** tag:
xmlns:maven="jelly:maven"
 - Replace **<ant>** by **<maven:maven>**:
<maven:maven descriptor="sub/project.xml" goals="main"/>



- Consider a Rewrite instead of a Conversion:
 - ✱ Taking advantage of the Building Process within Maven
 - ✱ Using the POM to declare the Project
 - ✱ Shortening the Build Scripts
 - ✱ Taking advantage of the full power of Maven Scripting
- Declare an Ant Name Space and use it as prefix for Ant Tasks
 - ✱ Add into the `<project>` tag: `xmlns:ant="jelly:ant"`
 - ✱ Add a prefix to ant tasks
 - ◆ `<ant:echo>` instead of just `<echo>`

Maven: Table of Content



- Maven Presentation:
 - ✱ Design
 - ✱ Project Object Model
 - ✱ Plugins & Goals
 - ✱ Properties
 - ✱ Scripting with Jelly
 - ✱ Your own Plugins
 - ✱ Convert your Ant Projects
 - ✱ **Troubleshooting**
 - ✱ Q&A



- Maven Web Site: maven.apache.org
 - ✿ POM: maven.apache.org/reference/project-descriptor.html
 - ✿ List of Plugins: maven.apache.org/reference/plugins/index.html
 - ✿ User Guide: maven.apache.org/reference/user-guide.html
 - ✿ Wiki: wiki.codehaus.org/maven/
- My Article with Tips about Maven
 - ✿ www.onjava.com/pub/a/onjava/2004/08/04/maventips.html

Need Help?



● Jelly Scripting:

- ✿ Check out the Plugins' Script to see how they do it
- ✿ Check out the Jelly Web Site: jakarta.apache.org/commons/jelly/

● Maven:

- ✿ Check out open-source projects using Maven
- ✿ Maven's **Powered-By** List:
maven.apache.org/misc/powered.html
- ✿ Join the **Maven Users** Mailing List:
maven.apache.org/mail-lists.html

Maven: Table of Content



- Maven Presentation:
 - ✱ Design
 - ✱ Project Object Model
 - ✱ Plugins & Goals
 - ✱ Properties
 - ✱ Scripting with Jelly
 - ✱ Your own Plugins
 - ✱ Convert your Ant Projects
 - ✱ Troubleshooting
 - ✱ **Q&A**

Q & A



Andreas Schaefer

andreas.schaefer@madplanet.com

Web Log

www.madplanet.com/weblog/blog/schaefa

Presentation can be found on

www.madplanet.com/presentations.tea