# Maven

Managing software projects
for repeatable results

# What is Maven?

- Organized approach to managing software projects
- Centralization of tasks
- Simplifies build process
- Simplifies documentation process

# What Maven Isn't

- A cure for everything
- Immediately easy – there is a nontrivial learning curve.

The goal of this  presentation is to help lessen that learning curve…

# Important Features

- Repository – centralizes the dependencies, obviates the need for duplicate copies of jars. This is just a standardized directory & file structure.
- Build process – Handles many of the normal tasks for you easily. (Build, documentation, testing, etc)
- Archetypes (Templates) – Create a re-usable template for specific projects. (Web Services – create a new project ready for coding in 30 seconds. No more having every single developer waste a day setting up the build for a project.)
- Project Object Model (POM) – organizes dependencies for a project. (Or template)

# Repository

- Holds copies of all of the relevant jars in a centralized location.
- Has version and artifact information necessary for use in a POM.xml file.
- Can hold multiple versions of the same file without issue.
- Local (for building) and remote (used to get dependencies you don’t currently have in your local repository.)

# Build Process

- mvn package – the command used to create the project artifact.  (Jar, war, ear)
- Uses maven repository and pom.xml to compile the code and deliver the packaged artifact.
- Different archetypes (templates) will deliver different artifacts.

# Archetypes

- Basically a project template
- Should follow best practices as defined by Maven documentation – project layout, etc.
- Can include project dependencies
- Will deliver the final artifact when packaged
- Can be created from an example project and then re-used later.  (Build your own reusable archetype!!!)

# Project Object Model (POM)

- Used to organize the Maven project
- Holds the project dependencies
- Used to designate the compiler version
- Defines the packaging type for project
- Defines the output artifact
- Defines required plugins

# Standard Project Structure

| directory | description |
|---|---|
| /new-app/pom.xml | maven2 project file |
| /new-app/src/ | Sources |
| /new-app/src/main/java/ | Java source tree |
| /new-app/src/test/java/ | Java unit tests |
| /new-app/src/main/resources/ | Java classpath resources |
| /new-app/src/test/resources/ | Resources for unit-tests |
| /new-app/target/classes/ | compiles classes |
| /new-app/target/test-classes/ | compiles test classes |
| /new-app/target/dots | other plugins' output |
| /newwebapp/src/main/webapp | root of webapp |

# Installing Maven

- Install Maven on workstation (Laptop, desktop, etc)
- Place Maven home in the path.
- Configure your repository location in the maven settings.xml file.
- Install plugin for your specific IDE.  (m2Eclipse is fine for Eclipse.
- Run basic commands from command line initially.  (I found it less confusing this way, and I learned the actual commands.)  Not really necessary with the m2eclipse IDE plugin.
- Import the maven project into your IDE.  It should recognize the POM and accept the configurations.  Or you can Maven-ize an existing one via the IDE.  (Add the pom.xml, manually alter the configuration to the standard.)
- Run the command to generate the ANT file for you – makes life simpler to use the basics.  Plus you can add to the build.xml for anything you don't see covered.  The command is "mvn ant:ant".

# Create a New Web Project with Maven

```
mvn archetype:create -DgroupId=com.mycompany.app -DartifactId=my-webapp
    -DarchetypeArtifactId=maven-archetype-webapp
```

- Mvn archetype:create - The command to create a new project via maven
- -DgroupId=com.company.app - This is the package used for the new java code.
- -DArtifactId=my-webapp - The artifact created
- -DarchetypeArtifactId=maven-archtype-webapp - The standard maven web application project template to be used.

# Installing a Jar into the Repository

- I needed JSR181-api.jar but it wasn't available anywhere on the net in one of the normal larger repositories. (Maven will try and download a dependency for you!)
- mvn install:install-file -DgroupId=javax.jws -DartifactId=jsr181 -Dpackaging=jar -Dfile=jsr181-api.jar -Dversion=1.0 -DgeneratePom=true
- Most of this information was readily available and simply a matter of hunting it up.
- A bit slow.  But a one-time process for any jar.
- After the first time, you can simply check the pom.xml dependency information in the repository and copy it into your local pom.  Voila!  Now you can use the jar file.

# Setting the Compiler Version

- Open the pom.xml – edit the plugin information as shown for compiling to java 1.5
- <build>
- <plugins>
- <plugin>
- <artifactId>maven-compiler-plugin</artifactId>
- <configuration>
- <source>**1.5**</source>
- <target>**1.5**</target>
- </configuration>
- </plugin>
- </plugins>

Issue found: If you are running Eclipse 3.2 with java 1.5 – the forked JVM used for compilation is java 1.5. And it will not accept the java 1.6 without throwing an error during compile.

Solution:

<fork>true</fork>
<executable>/path/to/javac_1.6/executable</executable>
<target>1.6</target>

These settings will work. I generally put them in properties at the bottom of the pom.xml file for station specific configurations. (like which version of a compiler you are using, or the location of that compiler.)

# Impact

- Will not be trivial initially…
- The first project to move over will require the most time –setting up repository, loading jars, creating basic project configuration, moving code over, testing, etc.
- The largest gain is in repeatability & management of artifacts.
- The next gain is in time – after the learning curve you will be able to create project very fast without worry. Especially good for similar projects. No more cutting and pasting.
- Additional gains indocumentation - javadoc, web site about project, testing, and complying with standards.
- After the first project it should go significantly faster. All new projects using a standard configuration (Or created template) will benefit greatly in setup and panaging the project.
- Maven 2 has the ability to manage projects in a hierarchy. This should significantly lessen the maintenance of the ANT scripts. No more hierarchical ant build scripts. Dependencies can be managed via the POM. Large projects can use a relatively flat structure more easily.

# Steps to creating a new web project (war) via Maven/Eclipse

- 1. Determine the parameters for the archetype to use
- 2. Create the project in the workspace directory
- 3. Import into IDE (Eclipse for me.)
- 4. run "mvn ant:ant" form the project top directory (Creates the build.xml file)
  - This will allow you to add your own specific tasks and run them via ant. Useful for web service stub generation for example. Yes Maven can do it, but sometimes it's just easier to write that small part. (Especially since you don't have to worry about dependencies – they are maintained by maven for the any script.)
- 5. Edit POM for compiler level (Also set this for the IDE!!!!)
- 6. Edit POM for dependancies. You should see the IDE trying to download stuff for the dependencies once you save the POM.
- 7. Set up the src/main/java source directories
- 8. Test a build of the artifact. Run the "mvn package" command on the command line from the head of the project. (The WAR ant task has an issue so far, command line seems to work fine.)

# Creating archetypes

- Create new project. (You can use another archetype to start & modify by hand.)
-
  - Create pom.xml at top of archetype project. This is the archetype project, not the example project.
  - Create a src/main/resource/archetype-resources directory
  - Place the minimal pom for the archetype you are creating in this directory.
  - Create a src/main/META-INF/maven directory.
  - Create an archetype.xml file in this directory. It will detail the files that will be seeded into the projects created from this new archetype plugin.
- Good resource for doing it manually:

  http://maven.apache.org/guides/mini/guide-creating-archetypes.html
- Unfortunately the archtypes/project layouts have changed from M1 to M2. So the creation of an archetype is a bit tedious right now. I ended up spending 20-30 minutes altering the project layout due to incorrect layout of the archetype resources. Not a major problem, just annoying.

# Installing Archetype into Repository

- To install the archetype into the repository, you run the following command from command line of the project.
- mvn install  (You can accomplish this from the m3eclipse floating menu as well.  That seems to work fine.)
- Adding an archetype to the archetype-catalog file
  - Create a valid file:  mvn archetype:crawl
  - Edit as necessary to show the archetypes as you create them.

Currently I haven't figured out how to get the eclipse IDE to recognize the local archetype-catalog.xml file.

# Creating a Project From a New Archetype

- Run the following:

```
mvn -e archetype:create -DarchetypeGroupId=com.mantech -
    DarchetypeArtifactId=maven_archetype-clearspace-plugin -DarchetypeVersion=0.1 -
    DgroupId=com.mantech -DartifactId=testPlugin -DarchetypeRepository=/Users/
    stevekeener/Documents/repository
```

# Explanation of the “archetype:create” Command

- -e
  - Turn on debug information.  (an absolute must to debug.)
- archetype:create
  - Use archetype plugin, create action.
- -DarchetypeGroupId=com.mantech
  - Use the com.mantech pathing to find the archtype plugin in the repository.
- -DarchetypeArtifactId=maven_archetype-clearspace-plugin
  - The archetype name
- -DarchetypeVersion=0.1
  - The archetype version
- -DgroupId=com.mantech
  - The group (pathing) for the project being created.  (See pom.xml)
- -DartifactId=testPlugin
  - The name of the project (resultant artifact)
- -DarchetypeRepository=/Users/stevekeener/Documents/repository
  - Path to local repository where the archtype resides

# Issues:

- 1. Archetypes are not well documented. This makes creating new ones a challenge. Workaround is to use an example project as a baseline and reverse engineer into an archetype. (Haven't done this yet.)
- Archetytpe-catalog.xml – Currently haven't gotten this to work with the Eclipse IDE. So I will document the command necessary to use from the command line. Since this is only done once per project it's not high on my list of thigns to figure out.
- The order of the fields in the archetype.xml file is strict. If you get them out of order the results are a bit unpredictable.
- Repository is just a file system. Can I use SVN instead. No. But you can create your repository normally and then create a SVN/CVS project and import it. That way you have a record of the system and other developers should be able to check it out and be up and running quickly.

# Resources

- Maven home - http://maven.apache.org/
- Maven remote repository - http://repo1.maven.org/maven2/
- Maven plugins (Commands) - http://maven.apache.org/plugins/
- Maven – getting started - http://maven.apache.org/guides/getting-started/index.html
- Creating archettypes guide

  http://maven.apache.org/guides/mini/guide-creating-archetypes.html
- GREAT maven guide

  http://www.scribd.com/doc/238927/BetterBuildsWithMaven?query2=maven+daytrader