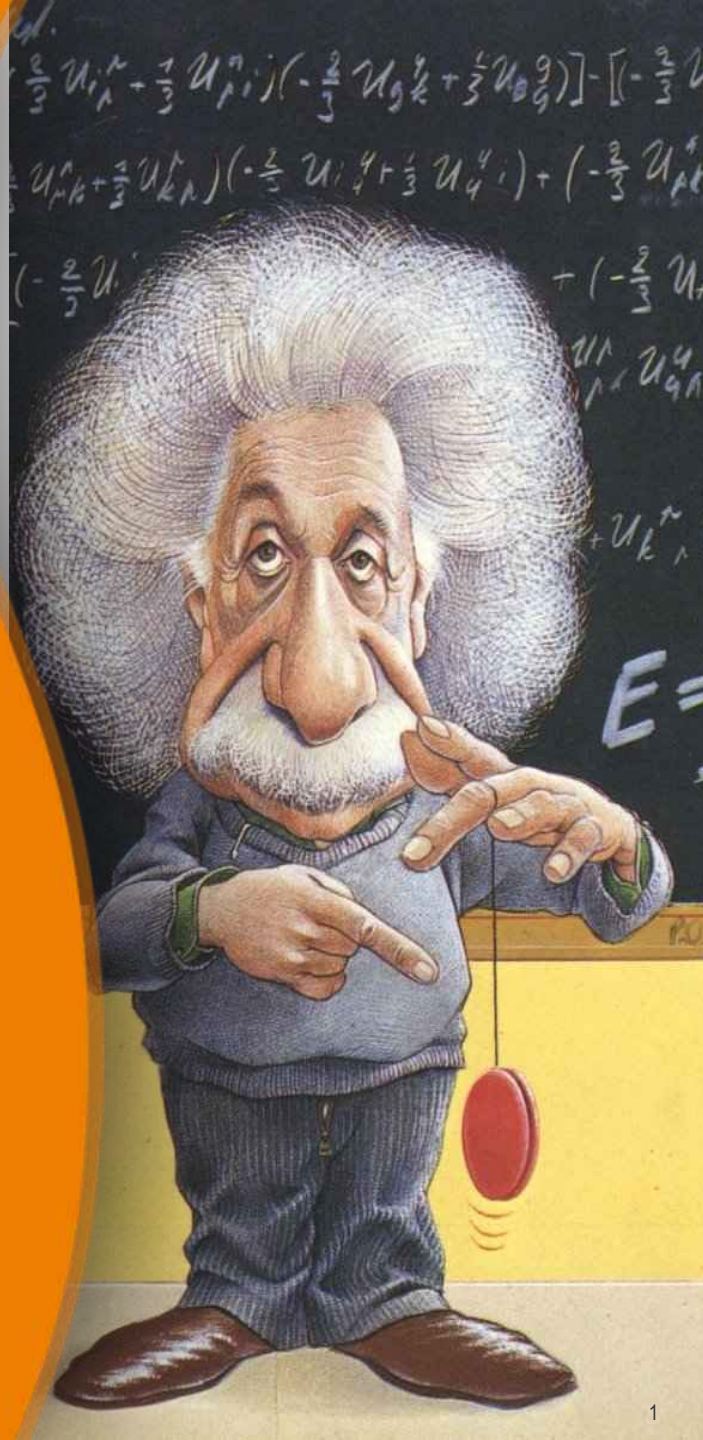
GlassFish V3
Intro to Maven 2.0
v1.0

Paul Sterk
GlassFish V3 Planning Team
November 1, 2007



Goals

- To explain the features and usage of the Maven2 project management tool.
- The focus is on approach this subject from the point-of-view of the Maven *user* and opposed to the Maven *plugin developer*.
- Learn by doing examples.

Agenda

- Why Maven?
- What is Maven?
- Migrating Maven 1.x to 2.0
- Installation and configuration
- Getting started
- Directory layout
- POM
- Plugins
- Lifecycles and phases
- How do I?
- Netbeans plugin
- Resources

Advanced Topics Not Covered

- Maven2 Architecture
- Maven2 Plugins
- Netbeans Maven IDE plugin
- Reports generation
- In-depth lifecycle phase discussion
- Profiles
- New packaging types

Why Maven?

- First, understand the problem. Each project has its own:
  - directory layout
  - process of building binaries
  - method of resolving dependencies
  - set of build technologies (SCM, Ant)
  - build lifecycle
  - way of defining a project
  - process for publishing artifacts (jars, docs)
- No best practices or collaboration

Why not just use Ant?

- Does not solve problems on previous slide. Also...
- Ant build scripts are much longer than Maven scripts because every task needs to be fully qualified
- Ant tasks difficult to debug
- Scripting is more difficult to integrate

What is Maven?

- A tool for building and managing any Java project
- A software project management and comprehension tool.
- 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.
- Glassfish v3 uses version 2.0

A maven is a trusted expert in a particular field, who seeks to pass his or her knowledge on to others. Comes from the Yiddish meyvn and Hebrew mevin, which in turn derives from the Hebrew binah, meaning understanding.

Maven Objectives

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Providing guidelines for best practices development
- Allowing transparent migration to new features

http://maven.apache.org/what-is-maven.html

Comparing Maven 1.x to 2.0

- Complete rewrite of Maven 1.x
- Maven 2.0 has the following:
  - Uses command 'mvn' instead of 'maven'
  - Supports Java-based plugins
  - Offers a managed lifecycle
  - Supports multi-project builds
  - Updated Project Object Model (POM)
  - Uses pom.xml instead of project.xml
  - Resolves transitive dependencies
- GF v3 uses Maven 2.0
- Possible to mix 1.x and 2.0 versions

Converting From 1.x to 2.0

- Move content in project.xml to pom.xml
- Move build.properties and project.properties to settings.xml
- Discard maven.xml
- Move files to Maven2 directory structure
- Migrate plugins

http://maven.apache.org/guides/mini/guide-m1-m2.html

Maven: Installation

- Maven depends on a JRE in your env
- Download and install:
  http://maven.apache.org/download.html
- Add the system variable M2_HOME
- Add $M2_HOME/bin directory to your system path.
- Type the following:
  mvn –version
- You should see:
  Maven version: 2.0.7

Maven: Configuration

- Configuration occurs at three levels:
  - *Project*: most static configuration occurs in pom.xml
  - *Installation*: this is configuration added once for a Maven installation (covered in previous slide)
  - *User*: this is configuration specific to a particular user

http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

Maven: Configuration

- User configurations are specified in: ${user.home}/.m2/settings.xml
- Able to configure:
  - a local repository
  - a proxy:
  - security and deployment settings
  - repository mirrors
  - profiles

- http://maven.apache.org/guides/mini/guide-configuring-maven.html

Maven: Configuration

- Able to pass command line flags:
  - mvn <plugin>:<goal> [-Doption1 -Doption2]
    Basic maven invocation of <goal> (multiple goals possible)
  - mvn <phase> [-Doption1 -Doption2 ...] Execute maven until <phase> (multiple phases possible)
  - To make maven install without launching the tests:
    mvn install -Dmaven.test.skip=true
  - To continue the maven build even if a test fails:
    mvn -DtestFailureIgnore=true <goal>

- http://maven.apache.org/guides/mini/guide-configuring-maven.html

Maven: Getting Started

- On your command line, execute the following maven goal:
  - mvn archetype:create -DgroupId=com.mycompany.app -DartifactId=my-app
- Maven will then download required artifacts (plugin jars) from a remote repository and place in your local repository (by default):
  - ${user.home}/.m2/repository
- The create goal created a directory called my-app

New project: what did I just do?

- You executed the Maven goal archetype:create and passed parameters to it
- The prefix archetype is the plugin that contains the goal. This is similar to an Ant task.
- The create goal created a simple project based upon an archetype.
- A plugin is a collection of goals with a general common purpose

New Project: Directory Layout

- cd to my-app directory
- Notice the directory layout:

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

New Project: Directory Layout (cont)

- The src/main/java directory contains the project source code
- The src/test/java directory contains the test source
- The pom.xml is the project's Project Object Model, or POM.

Project Object Model (POM)

- The pom.xml file is a single configuration file that contains the majority of information required to build a project
- Declarative project configuration:
  - Name and description
  - Source control, issue tracking, etc.
  - Company and developers
  - Source layout
  - Dependencies on external projects
  - Build requirements and configuration

http://maven.apache.org/pom.html

Project Object Model (POM)

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
    <modelVersion>4.0.0</modelVersion>
    <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: Key Elements

- **project** this is the top-level element in all Maven pom.xml files
- **modelVersion** version of the object model this POM is using (4.0.0)
- **groupId** the unique identifier of the organization or group that created the project. Typically the fully qualified domain name of your organization.
- **artifactId** the unique base name of the primary artifact being generated by this project. A typical artifact will have the form <artifactId>-<version>.<extension> (e.g., myapp-1.0.jar).

POM: Key Elements

- **packaging** the package type to be used by this artifact. Default is 'jar'.
- **version** the version of the artifact generated by the project. The version will often contain 'SNAPSHOT' to indicate that it is in a state of development.
- **name** This element indicates the display name used for the project.
- **url** This element indicates where the project's site can be found.
- **description** This element provides a basic description of your project.

POM: Additional Elements

- **dependency** An artifact on which this project depends. Each one has a scope. Default is compile.
- **build** Contains info on how to build the current artifact.
  - **sourceDirectory** Contains java source files. Default is src/main/java
  - **scriptSourceDirectory** Contains script files. Default is src/main/script
  - **testSourceDirectory** Contains test files. Default is src/test/java
  - **outputDirectory** Where to place compiled files, scripts and resources. Default is target/classes
  - **testOutputDirectory** Default is target/test-classes

POM: Additional Elements

- **build** How to build the current artifact.
  - **resources** Points to the resource directories. This contains files that are not compiled. Content copied to outputDirectory. Default is src/main/resources
  - **testResources** Points to test resource directories. Content copied to outputTestDirectory. Default is src/test/resources
  - **directory** Top-level directory where built files are placed. Default is target
  - **finalName** The name to use for build objects like jar. Default is ${artifactId}-{$version}
  - **filters** Points to properties files used for filtering.
  - **plugins** The plugins required to build the artifact

POM: Additional Elements

- **profiles** A profile setting that controls which build elements are used. Sets up a standard environment (development, test, QA, production)
- **modules** Child artifacts of the current artifact
- **repositories** Locations from where artifacts can be downloaded
- **pluginRepositories** Locations from where plugins can be downloaded
- **reporting** Special plugins used for site generation
- **properties** Name-value pairs used to simplify configuration

Maven: Getting Started

- Execute: 'mvn package'
- The command line will print out various actions, display the message 'BUILD SUCCESSFUL' and place the created artifact in the target directory
- You may test the newly compiled and packaged JAR with the following command:
  java -cp target/my-app-1.0-SNAPSHOT.jar com.mycompany.app.App

Maven Phases

- Unlike the first command (archetype:create) the second is 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.

Maven Plugins

- When a user executes a goal or a phase, a configured plugin is executed.
- Each plugin is made up of one or more Maven Java Objects (Mojos).
- Each Mojo is mapped to a goal or can belong to a phase.
  - mvn org.apache.maven.plugins:maven-**jar**-plugin:jar
  - mvn **jar**:jar
  - mvn package

http://maven.apache.org/guides/introduction/introduction-to-plugin-prefix-mapping.html

Maven Plugins

- **Plugin Description**
- antrun Run a set of ant tasks from a phase of the build.
- assembly Build an assembly (distribution) of sources and binaries.
- checkstyle Generate a checkstyle report.
- clean Clean up after the build.
- compiler Compiles Java sources.
- deploy Deploy the built artifact to the remote repository.
- ear Generate an EAR from the current project.
- eclipse Generate an Eclipse project file for the current project.
- ejb Build an EJB (and optional client) from the current project.
- help Get information about the working environment for the project.

Maven Plugins

- **Plugin** **Description**
- install Install the built artifact into the local repository.
- jar Build a JAR from the current project.
- javadoc Generate Javadoc for the project.
- jxr Generate a source cross reference (analog to javadoc).
- netbeans Netbeans IDE plugin: http://maven.apache.org/netbeans-module.html
- resources Copy the resources to the output directory for including in the JAR.
- site Generate a site for the current project.
- source Build a JAR of sources for use in IDEs and distribution to the repository.
- surefire Run the Junit tests in an isolated classloader.
- war Build a WAR from the current project.

http://maven.apache.org/plugins/

How do I use plugins?

- Whenever you want to customize the build for a Maven project, this is done by adding or reconfiguring plugins.
- plugins in Maven 2.0 look much like a dependency
- plugin will be automatically downloaded and used - including a specific version if you request it (the default is to use the latest available).
- The configuration element applies the given parameters to every goal from the compiler plugin

How do I use plugins?

- To find out what configuration is available for a plugin, you can see the Plugins List: http://maven.apache.org/plugins/

```
...
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>1.5</source>
        <target>1.5</target>
      </configuration>
    </plugin>
  </plugins>
</build>
...
```

Maven Lifecycles

- Maven knows by default the following three lifecycles
  - **default** Is used for most activities on artifacts like performing a traditional build.
  - **clean** Is mostly used to delete generated parts.
  - **site** Is used to generate a website for the current artifact.
- A lifecycle has one or more phases, and a plugin can join a phase.

Maven Lifecycle Phases

- Phases are actually mapped to underlying *goals*.
- The specific goals executed per phase is dependent upon the packaging type of the project. (e.g., `mvn package` executes `jar:jar` if the project type is a JAR)
- Typically, when phases of the lifecycles above are activated, some predefined plugin-goals are automatically executed.

Default Lifecycle: Common Phases

- **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.

http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

Default Lifecycle: Common Phases

- **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.

How do I compile source files?

- In the my-app directory execute:
  mvn compile

```
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Quick Start Archetype
[INFO]    task-segment: [compile]
[INFO] ------------------------------------------------------------------------
[INFO] artifact org.apache.maven.plugins:maven-resources-plugin: \
  checking for updates from central
...
[INFO] artifact org.apache.maven.plugins:maven-compiler-plugin: \
  checking for updates from central
...
[INFO] [resources:resources]
...
[INFO] [compiler:compile]
Compiling 1 source file to <dir>/my-app/target/classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3 minutes 54 seconds
[INFO] Finished at: Fri Sep 23 15:48:34 GMT-05:00 2005
[INFO] Final Memory: 2M/6M
[INFO] ------------------------------------------------------------------------
```

How do I compile source files?

- The *first time* you execute this (or any other) command, Maven will need to download all the plugins and related dependencies it needs to fulfill the command.
- The compiled classes were placed in ${basedir}/target/classes
- If you follow the standard directory layout, you do not have to specify source files or the target directory.

How do I compile and run tests?

- Execute 'maven test'
- Maven downloads dependencies and plugins necessary for executing the tests.
- Before compiling and executing the tests Maven compiles the main code.
- If you simply want to compile your test sources (but not execute the tests), execute 'maven test-compile'

How do I compile and run tests?

- Note that the surefire plugin (which executes the test) looks for tests contained in files with a particular naming convention. By default the tests included are:
  - **/*Test.java
  - **/Test*.java
  - **/*TestCase.java
- And the default excludes are:
  - **/Abstract*Test.java
  - **/Abstract*TestCase.java

How do I create a Jar of my compiled source file?

- Execute 'mvn package'
- In the default POM, the packaging element is set to jar. This is how Maven knows to produce a JAR file from the above command
- The jar file is placed in the ${basedir}/target directory

How do I add resources to my jar file?

- Add files to src/main/resources

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   |-- java
    |   |   `-- com
    |   |       `-- mycompany
    |   |           `-- app
    |   |               `-- App.java
    |   `-- resources
    |       `-- META-INF
    |           `-- application.properties
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

What does Maven add to my jar file?

- Maven adds files from the resources directory. It also adds pom.properties and pom.xml, and generates a MANIFEST.MF file

```
|-- META-INF
|   |-- MANIFEST.MF
|   |-- application.properties
|   `-- maven
|       `-- com.mycompany.app
|           `-- my-app
|               |-- pom.properties
|               `-- pom.xml
`-- com
    `-- mycompany
        `-- app
            `-- App.class
```

http://maven.apache.org/guides/getting-started/index.html#How_do_I_add_resources_to_my_JAR

How do I filter resource files?

- Sometimes a resource file will need to contain a value that can only be supplied at build time.
- Put a reference to the property that will contain the value into your resource file using the syntax ${<property name>}
- Add the property value to src/main/filters/filter.properties
- In pom.xml, set <filtering> to true and add reference to filter.properties

http://maven.apache.org/guides/getting-started/index.html#How_do_I_filter_resource_files

How do I use external dependencies?

- The dependencies section of the pom.xml lists all of the external dependencies that our project needs in order to build
- For each external dependency, you'll need to define groupId, artifactId, version, and scope.
- The scope element indicates how your project uses that dependency, and can be values like compile , test , and runtime .

http://maven.apache.org/guides/getting-started/index.html#How_do_I_use_external_dependencies

How do I use external dependencies?

- With this information, Maven will be able to reference the dependency when it builds the project.
- Maven looks in your local repository (~/.m2/repository is the default) to find all dependencies.
- If not found locally, Maven will download the dependency from a remote repository. Default repo is: http://repo1.maven.org/maven2/

http://maven.apache.org/guides/introduction/introduction-to-repositories.html

How do I use external dependencies?

- It is possible to disable transitive dependency resolution using the `excludeTransitive` parameter of the `dependency:resolve` goal. See: http://maven.apache.org/plugins/maven-dependency-plugin/resolve-mojo.html

How do I use external dependencies?

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <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>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.12</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
</project>
```

How do I install the Jar file in my local repository?

- Execute 'mvn install'
- In the default POM, the packaging element is set to jar. This is how Maven knows to produce a JAR file from the above command
- The jar file is placed in the ${basedir}/target directory
- You will see the following output:

http://maven.apache.org/guides/introduction/introduction-to-repositories.html

Maven install output

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
[surefire] Running com.mycompany.app.AppTest
[surefire] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.001 sec

Results :
[surefire] Tests run: 1, Failures: 0, Errors: 0

[INFO] [jar:jar]
[INFO] Building jar: <dir>/my-app/target/my-app-1.0-SNAPSHOT.jar
[INFO] [install:install]
[INFO] Installing <dir>/my-app/target/my-app-1.0-SNAPSHOT.jar to \
   <local-repository>/com/mycompany/app/my-app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5 seconds
[INFO] Finished at: Tue Oct 04 13:20:32 GMT-05:00 2005
[INFO] Final Memory: 3M/8M
[INFO] ------------------------------------------------------------------------
```

How do I create site docs?

- Execute: mvn archetype:create -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-site -DgroupId=com.mycompany.app -DartifactId=my-app-site

```
my-app-site
|-- pom.xml
`-- src
    `-- site
        |-- apt
        |   |-- format.apt
        |   `-- index.apt
        |-- fml
        |   `-- faq.fml
        |-- fr
        |   |-- apt
        |   |   |-- format.apt
        |   |   `-- index.apt
        |   |-- fml
        |   |   `-- faq.fml
        |   `-- xdoc
        |       `-- xdoc.xml
        |-- xdoc
        |   `-- xdoc.xml
        |-- site.xml
        `-- site_fr.xml
```

How do I create site docs?

- There is a $basedir/src/site directory which contains a site descriptor along with various directories corresponding to the supported document types.
  - Xdoc format
  - APT format, "Almost Plain Text", is a wiki-like format
  - FML format is the FAQ format
- Execute 'mvn site'
- site.xml describes the site layout

http://maven.apache.org/guides/getting-started/index.html#How_do_I_create_documentation

What kind of reports can Maven create?

- Reports can be generated to show the current state of the project
- Maven can generate code coverage reports (e.g., Clover), test results, code style, and others.

How do I...?

- Deploy my site? 'mvn site-deploy'
- Create javadocs? 'mvn javadoc:javadoc'. See:
  http://maven.apache.org/plugins/maven-javadoc-plugin/javadoc-mojo.html
- Add arbitrary resources to my site, support internationalization and do report configuration?
  http://maven.apache.org/guides/getting-started/index.html#How_do_I_deploy_my_site
- Create different projects such as web apps?
  http://maven.apache.org/guides/getting-started/index.html#How_do_I_build_other_types_of_projects

Is there a Maven2 Netbeans plugin?

- Yes. Mevenide2-Netbeans is a Netbeans plugin which integrates all the maven2 project management possibilities into Netbeans.
- See: http://mevenide.codehaus.org/m2-site/

Resources

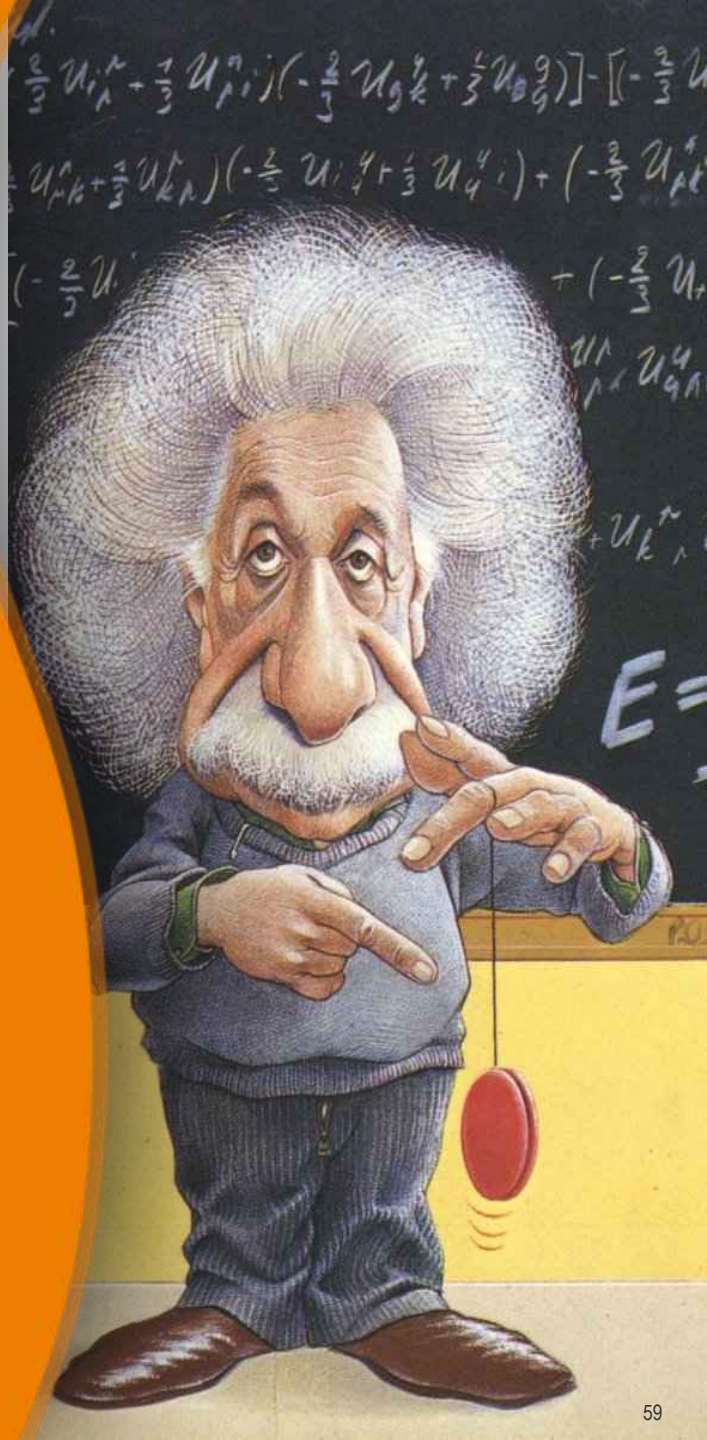
- users@maven.apache.org
- http://maven.apache.org
- maven-users@sun.com
- nb-maven@sun.com
- http://el4j.sourceforge.net/docs/pdf/MavenCheatSheet_EL4J.pdf
- http://maven.apache.org/general.html
- http://maven.apache.org/users/getting-help.html

Q & A

- Questions or comments?
- Suggestions for an advanced Maven session?

GlassFish V3
Intro to Maven 2.0
v1.0

Paul Sterk
GlassFish V3 Planning Team
November 1, 2007



Appendix

- Maven2 Architecture
- Default (Build) Lifecycle Phases

Maven2 Architecture

- User invokes the 'mvn' CLI
- The CLI invokes a plugin inside the Plexus Container.
- The Plexus Container is an Inversion of Control (IoC) container that injects object dependencies into the plugin
- The plugin is managed by a Plugin Manager
- The Plugin Manager calls the Artifact Handler which uses Wagon.
- Wagon a transport abstraction that is used in up/download artifacts. Supports file, http, https, ftp, sftp and scp.

Build Lifecycle Phases

- validate validate the project is correct and all necessary information is available.
- 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.

Build Lifecycle Phases

- 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
- 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.

Build Lifecycle Phases

- pre-integration-test perform actions required before integration tests are executed. This may involve things such as setting up the required environment.
- 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.