



I Got My Mojo Workin'

Gary Murphy

Hilbert Computing, Inc.

<http://www.hilbertinc.com/>
glm@hilbertinc.com





Agenda

- Quick overview on using Maven 2
- Key features and concepts
- A look at the structural components of Maven 2
- Developing Maven plug-ins, comprised of “mojos”.



What is Maven?

- **Framework** for development processes. Can be used for:
 - Build runtime artifacts, such as jars and wars
 - Build developer documentation, such as JavaDoc
 - Create reports for project management
 - Distribute applications to dev, prod, *etc.*
- ... and anything else for which you write a mojo.



Installing Maven?

- Maven is a project managed by the Apache Software Foundation:
<http://maven.apache.org/>
- Download and install a small core of Maven functionality
 - Maven will automatically load and install dependencies for each of the plug-ins you request



Running Maven

- This talk is more about writing mojos for Maven, but we should have a feel for how it works:

Maven Demo Time



Maven Objectives

- Make the build process easy
- Provide a uniform build system
- Provide quality project information
- Provide guidelines for best practices development
- Allow for transparent migration to new features.



Meeting Those Objectives

- Maven is easy to get started, since it bootstraps most of its install.
- Maven build process is easy **when it works**. Difficult to diagnose when it doesn't.
 - Been reliable for me.
 - Some idiosyncrasies on some machines
 - Way better than Maven 1. Don't judge Maven 2 based on Maven 1 experience



Meeting Those Objectives

- High marks for project information.
Plug-ins for:
 - JavaDoc generation
 - Unit test reports
 - Code style
- Pushes documentation closer to developer processes:
 - Better than external documentation
 - Documentation versioned with code



Meeting Those Objectives

- Provides standard directory structure layout:
 - Hard one for me to bite off
 - Defaults generate artifacts in the same directory structure as Eclipse projects
 - Default path structure can be changed, but more configuration work
- In short, Maven 2 is a big improvement over Maven 1 and does an adequate job of meeting its objectives



Key Features

- Project set up *via* a “project object model” (POM).
 - Default “super POM” that is hard-coded and you override what differs
 - POMs have an inheritance hierarchy
 - Parent POM contains information common to all projects
 - Parent POM references all child projects
 - **Demo time:** AEntendre Frameworks POM structure. *Note distribution management.*



Key Features

- Dependency Management
 - The POM contains the dependencies and their scope (*e.g.* compile, runtime, test, ...)
 - Maven will handle the installation of direct and transitive dependencies
- Versioned jars are maintained in a repository
 - In Eclipse, recommend using a classpath variable (*e.g.* M2_REPO) to point to the repository root.



Key Features

- Artifact deployment
 - Can install runtimes to remote systems using SCP or other means.
- Project documentation:
 - Automatically generates a web site with the project information
 - **Demo time:** AEntendre Framework site



Concepts

- Goals
 - Accomplish a single, atomic task.
 - Similar in concept to an Ant goal
 - One goal per mojo
- Plug-ins
 - Packaged jar of mojos
- Lifecycle (more on next slide)
 - A collection of goals bound to a lifecycle
 - Model a high-level process



Concepts - Lifecycle

- Most Common Build Lifecycle Phases
 - **install** – Place the packaged artifact (e.g. a jar) in the local repository. This will compile, test, package, verify and install.
 - **deploy** – Place the packaged artifact in the remote repository
- These are run at the command line:
mvn install



Underlying Structure

- Plexus (<http://plexus.codehaus.org>)
 - Inversion-of-control container
 - Uses field injection of requirements and configuration
 - Perhaps easy for the Maven 2 developers, but hard on the Mojo developers.
 - Not expressive!



Underlying Structure

- **Modello** (<http://modello.codehaus.org/>)
 - Describes an object model in XML. In this case, the POM.
 - Generates Java objects from the XML
 - Generated code contains untyped collections
 - As a result, very difficult to follow
- **Surefire** (<http://maven.apache.org/surefire>)
 - Testing framework for JUnit



Underlying Structure

- Doxia (<http://maven.apache.org/doxia>)
 - Documentation engine
 - APT – Almost plain text
 - DocBook
 - FML – FAQ Markup Language
 - LaTeX
 - RTF
 - XDoc
 - XHTML



Questions





What is a Mojo?

- A mojo is:
 - a play on POJO - **M**aven plain **O**ld **J**ava **O**bject
 - an executable goal in maven
- A plug-in is a distribution of one or more related mojos.



Introduction to Mojos

- Can be written in Java or other scripting languages.
 - This will cover Java mojos only.
- The interface is defined in:
`org.apache.maven.plugin.Mojo`
- Abstract base class in:
`org.apache.maven.plugin.AbstractMojo`



Introduction to Mojos

- Base class includes an implementation of a logging facility.
 - Use instead of `System.out.println`
 - Allows integration with GUI applications
- Concrete implementations must supply an `execute()` method.



plugin.xml

- The jar containing the plug-in classes must contain a descriptor file:
 - META-INF/maven/plugin.xml
 - Contains metadata about each mojo:
 - goal name
 - dependencies
 - parameters in the pom.xml that can be overridden
- Can (and should be) generated from JavaDoc annotations



settings.xml

- Maven needs to know the location (group) for your plugins:

```
<pluginGroups>  
  <pluginGroup>org.aentendre.maven</pluginGroup>  
</pluginGroups>
```



Annotations

- Mojo metadata specified *via* JavaDoc annotations (not Java annotations). Most common follow:
 - @goal *goalname* – used on the command line
 - @phase *phasename* – used if you want your mojo to be part of the standard build



Annotations - Parameter

- Makes objects available to mojo *via* Plexus injection:
 - @required – Parameter must be available for the mojo to work
 - @readonly – Can't be configured by user (*i.e.* access to internal objects)
 - @parameter expression="`#{expr}`" - bind to available objects.
*Expression values are still largely undocumented.
... the explore mojo will help.*



Explore Mojo

- Developed by me to reverse engineer how mojos work.
- Will be included on the Conference CD
- Used other open-source mojos for guidance.

Let's go exploring...



Questions





Thanks

- Thank you for attending.
- Please fill out the evaluations. They are helpful to Wayne and Peggy as well as me.
- Feel free to ask questions during the week.
- Time permitting, we can look at some peripheral issues...



Time Permitting...

- Eclipse and Maven 2
 - There is a plug-in for Maven 2 that I haven't used.
 - Eases setup of classpath in Eclipse
- By default, Maven builds in the same directory structure as Eclipse.
 - I changed the build directory locations
- Eclipse doesn't allow nesting of projects in the filesystem