

Maven 2

The Maven Team @ Apache



Agenda

- Maven time line
- The Maven Philosophy
- Maven 2
- Continuum



Maven Time Line

- August 2001: Started under Jakarta Alexandria project
- February 2002: Moved to Turbine
- Summer 2002: Jelly was added
- March 2003: Moved as a top-level Apache project
- Early 2003: Component refactoring begun – No Jelly!
- July 2004: Version 1.0
- Winter 2005: Version 2.0



The Maven Philosophy

- Build unification
 - `m2 install`
- Project meta data
 - Repositories
 - Non-Java
- Site
 - Documentation
 - Reports



Maven 2

- Is a complete rewrite from Maven 1.0/1.1
- Beta testing from TODAY!
- Built around *reusable libraries*:
 - Artifact code, used in Ant tasks
 - Wagon, SCM and project loading being used in Maven 1.1



Why Rewrite Maven for 2.0?

- Started parallel development in early 2003, well before Maven 1.0 final!
- More consistent definition of all parts of the system
- Architecture supports features that the original couldn't
- *Faster, lighter, smaller* - embeddable
- Making it simpler to use required, reworking many core concepts



Maven 2.0 Architecture - Plexus

- *Based on Plexus* – an inversion of control (IoC) container supporting component oriented programming (COP) encouraging a clear separation of concerns (SoC)
- Plug-ins are handled as Plexus *components* though plug-ins have no direct dependency on Plexus



Maven With Non-Java Projects

- Builds NUnit
 - Assemblies deployed to the repository
 - `m2 install`
- Builds make and autotools-based projects
 - `m2 install`
- Common points, POM is used to contain project meta data
 - CI integration
 - Unified way to build everything

Maven 2.0 Reusable Libraries

- Maven SCM
 - CVS, Subversion, ClearCase, Perforce, StarTeam
- Maven Wagon
 - File/Directory transport
 - File, HTTP(S), FTP, SSH/External SSH, SCM
- Doxia – Documentation system



Maven 2.0 New Features

- Enhanced dependency support
- Build lifecycle
- Unified project file
- Enhanced plug-in support
- Multi-module project support
- Site and documentation enhancements
- Release management
- Build Profiles



Transitive Dependencies

- Always enabled in Maven 2.0
- Don't need to declare dependencies of dependencies yourself
- Frequently requested, but has more consequences than often realised...
 - Version conflicts
 - Unwanted dependencies
 - Bad published meta data – report at <http://jira.codehaus.org/browse/MEV>
 - Not a hard problem with good data



Dependency Scope

- compile (default), runtime, test, provided, system
- Control class path and distribution bundling
- Helpful for transitive dependencies (don't get test transitively)
- *Only need to specify one* – others may be implied



Dependency Mediation

- Allows specification of a range of versions for a dependency
- Maven will help resolve the best version available
- Techniques for handling conflicts
 - Fail, Newest, Nearest
- Release tool fills in versions later

```
<dependency>  
  <groupId>jaxb</groupId>  
  <artifactId>jaxb-api</artifactId>  
  <version>[1.0.1,)</version>  
</dependency>
```

Snapshot Handling

- Deploying to a shared repository gives a version with a *time stamp and build #*
- Don't need to update dependency version to get updated builds
- Updates daily, on-demand, or at a particular interval
- Developers can get *access to co-workers changes earlier* without the need to update and build



Build Lifecycle

- Builds using Maven follows a *pattern*
- Ensures developers moving between projects do not learn a new process



Build Lifecycle – The Phases

- validate
- initialize
- generate-sources
- process-sources
- generate-resources
- process-resources
- compile
- process-classes
- generate-test-sources
- process-test-sources
- generate-test-resources
- process-test-resources
- test-compile
- test
- package
- integration-test
- verify
- install
- deploy



Build Lifecycle - Plug-ins

- Plug-ins can augment the build lifecycle
- For example:
 - JAXB could register an XSD to Java goal in `generate-sources`, then add it to the compilation list
 - XDoclet could register a goal to create a Hibernate mapping in `process-sources`.
- The developer still only needs to know to use `m2 install`



Unified Project File

- *Self contained*, and no file system references making it usable from the repository
- `project.properties` **and** `maven.xml` replaced by configuration and custom plug-ins
- `parent` **element replaces** `extend`

```
<parent>
  <groupId>org.apache.maven</groupId>
  <artifactId>maven</artifactId>
  <version>1.0-SNAPSHOT</version>
</parent>
```

Configuring Plug-ins

- Configure how your project is built
- In Maven 1, `project.properties` file
- Example: compile with JDK 5.0 options

```
<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>
```

On-demand Features

- Plug-ins can be *requested on-demand* from the command line
- In Maven 1, this required manual installation
- For example, `idea:idea` will generate an IDEA project file without modifications to your project



Advanced Plug-in Version Resolution

- Plug-in registry
- Can opt not to declare a plug-in version in your project
- Will regularly check for a new release, and download it if desired
- Users can opt to get prompted for new releases of plug-ins
- Release tool will record the active version for reproducible builds



Plug-in Languages

- Java is the most common
- Beanshell is new, useful for *rapid prototyping*
- Can support others with a small amount of work if there is demand
 - For example: Jython, Groovy, JRuby



Plug-in Development

- A few core plug-ins are developed at the Maven project itself
- The rest is hosted at the Mojo Project
 - Increases re-usability
 - Lower bar for acceptance
- Try to move plug-ins to the projects themselves



Multiple Modules

- Maven 2 natively deals with multi-module builds
- A module refers to another project in the build tree
- Goals are performed on all found modules by default, so `m2 install` will perform an install across modules
- Modules can in turn have modules

```
<modules>
  <module>wagon-provider-api</module>
  <module>wagon-providers</module>
</modules>
```

Enhanced Reactor

- Aggregation
- Plug-ins can be reactor aware
 - Faster than reactor aware Maven 1 plug-ins because they don't start their own reactor
- Artifacts can be referenced within the reactor



Release Assistance

- Improvement on the tasks from the SCM plug-in in Maven 1.0
- Resolves information in the project *to make the release reproducible*
- Updates the version information, commits and tags a release
- Does a clean checkout, builds and deploys the release
- Can work across a set of modules from the group level



Build Profiles

- Change the build depending on the *environment*
 - Dependencies, repositories, plug-ins and configuration
- *Trigger* by operating system, JDK, existence of software, and so on, as well as command line parameter
- Per user or per project
- Used to *set up standard environments*:
 - Development, Test, QA and Production



Support for Other Languages

- Being implemented as plug-ins
- Currently have seen work on a C# compiler, and plan to support C/C++ environments on Unix and Windows
- May not be available at Maven 2.0 final release



Why Continuum

- Aims to make managing continuous integration of Maven projects trivial
- Supports Ant, Maven 1, Maven 2 and shell scripts
- Very easy to use
 - Web based configuration
 - Can unzip and start with defaults
- Supports running as services on Windows, Linux, OS X and Solaris
 - Currently only in development version



Continuum Features

- Current functionality
 - A XML-RPC interface for automation and remoting (also SOAP in alpha-3)
 - Notification: Mail, Jabber, MSN, IRC
 - CVS and SVN support
 - Preliminary StarTeam, Clearcase, Perforce support
 - Polled and triggered builds
- Will match features of other OSS and free CI servers by 1.0 release



Adding Projects

- Can read POM from either an URL or from SCM
- Uses module section
 - Will add an entire product with a single click
- Maven 1-based products will be added using a Maven 1 plug-in and the reactor
- Ant and shell projects can be added manually



Getting Involved with M2 & Continuum

- As with any open source project, there are several ways you can get involved
 - Join the mailing list and answer other user's questions
 - Report bugs, feature requests and other issues in the issue tracking application.
 - Submit patches to reported issues (both those you find, or that others have filed)
- Non-core work equally important
 - Plug-ins
 - Documentation



Links & Resources

- [Maven homepage](#)
- [Very first version of Maven](#)
- [Maven Plugin Matrix](#)
- [Maven Blogs](#)
- [How To Help](#)
 - Categorized issues based on complexity



Questions?

