



# Building Applications with Apache Maven

---

dIon Gillard

Multitask Consulting Pty. Ltd.



# Agenda

---

- What is Maven?
- The POM
- Source + POM = Build
- Customizing Maven
- Maven & IDEs
- Migrating from Ant
- Multi-project builds
- Your own plugins



# What Is Maven?

---

- A build tool
- A project comprehension tool
- Site management tool
- Documentation tool
- Quality control tool
- Repository
- Ant replacement
- Project best practices guide



# Maven – Build Tool

---

- Command line or IDE integrated
- Heavily biased towards Java
- Data-driven from the project object model
- Functionality supplied by built-in and third party plug ins, leveraging the POM
- Built in functionality for most common J2SE and J2EE needs
  - Jar, war, ear, compile, test, javadoc, *etc.*

# Maven – Project Comprehension

---

- Tried working with a new project and getting up to speed quickly?
- Where's the source?
- What do each of the jars do?
- What's the API look like?
- How well do unit tests cover the code
- Is there documentation?

# Maven – Project Comprehension *(Continued)*

---

- Each 'project' produces a single jar
- Each project has a single source directory
- Generated sites provide basic information that should always be present.
- Links to external systems for bug tracking and SCM help newbies
- Encourage best practices, such as testing, code quality, *etc.*



# Maven – Site

---

- Project-based site
- Consistent documentation format (xdoc)
- Can integrate other formats (Docbook, Word, latex, HTML)
- Can produce PDF, WAR file
- Easy to deploy to file system or remote machine (*via* ssh/scp)



# Maven – Documentation

---

- Produces project documentation with just source and POM
  - Javadocs, Unit Test reports, Code metrics
  - Test coverage, Checkstyle
  - Change log, File and Developer Activity
  - Coloured Source Cross reference, Task list
- Other reports rely on you
  - FAQ, Changes report



# Maven – Quality

---

- Maven can't enforce quality
- Several plug ins provide project statistics to help measure quality
- Test Coverage: jcoverage and clover
- Code style: Checkstyle
- Code 'smells': PMD, Simian, Checkstyle, Findbugs
- SCM: Developer activity, statcvs



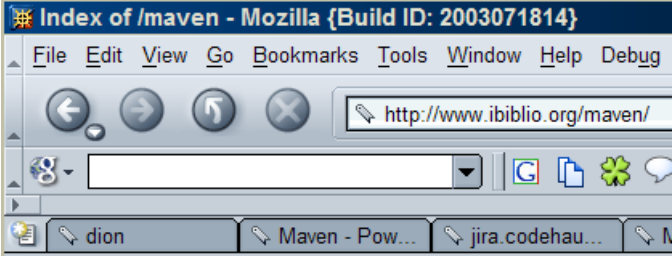
# Maven – Ant Replacement













---

- Maven builds on top of Ant
- Effectively provides a set of reusable Ant targets, with well documented requirements
- Ant needs you to write the logic and provide the data
- Maven needs you to provide the data and knows how to do most things
- Scripting in Maven provided by Jelly, which can easily use Ant tasks

# Maven – Repository

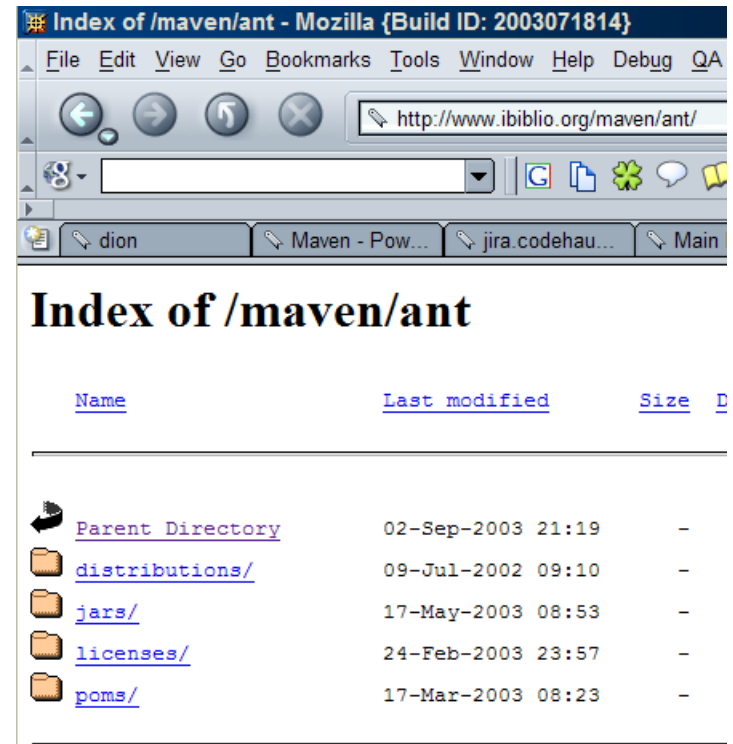
- Hosted @ ibiblio
- Central repo of well named jar files, tld's,....
- Easily accessible
- Simple format








<u>Name</u>	<u>Last modified</u>	<u>Size</u>
 <a href="#">Parent Directory</a>	05-Sep-2003 14:40	-
 <a href="#">Global/</a>	03-Feb-2003 14:32	-
 <a href="#">HTTPClient/</a>	04-May-2003 21:29	-
 <a href="#">aelfred/</a>	17-Mar-2003 08:17	-
 <a href="#">ant-contrib/</a>	17-Mar-2003 08:23	-
 <a href="#">ant-doxygen/</a>	02-Sep-2003 12:31	-
 <a href="#">ant/</a>	17-Mar-2003 08:23	-
 <a href="#">antlr/</a>	17-Mar-2003 08:23	-
 <a href="#">anttex/</a>	17-Mar-2003 08:24	-
 <a href="#">aptconvert/</a>	17-Mar-2003 08:24	-
 <a href="#">ashkelon/</a>	12-Mar-2003 15:42	-
 <a href="#">asm/</a>	19-May-2003 09:46	-

# Maven – Repository

- Licenses are available
- Distributions in binary and source
- Jar files
- Maven POMs as well

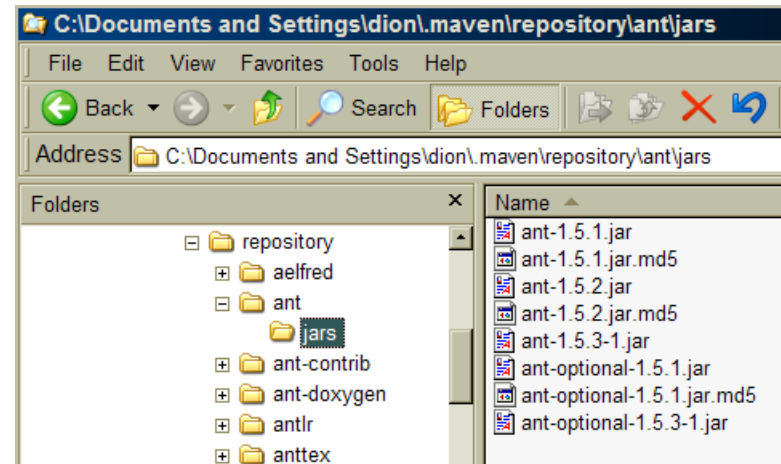


Index of /maven/ant

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>D</u>
 <a href="#">Parent Directory</a>	02-Sep-2003 21:19	-	
 <a href="#">distributions/</a>	09-Jul-2002 09:10	-	
 <a href="#">jars/</a>	17-May-2003 08:53	-	
 <a href="#">licenses/</a>	24-Feb-2003 23:57	-	
 <a href="#">poms/</a>	17-Mar-2003 08:23	-	

# Maven – Local Repository

- Single location for downloads
- Md5 files for verification
- Version numbers as part of the file
- Created automatically





# Maven – Best Practices

---

- Help to document what they are
- Development process URL as part of documentation
- Test failing stop jar creation
- Single jar single project
- Contents of manifest, project distribution, Javadocs
- Dependency visibility and repository management



# Agenda

---

- What is Maven?
- **The POM**
- Source + POM = Build
- Customizing Maven
- Maven & IDEs
- Migrating from Ant
- Multi-project builds
- Your own plugins



# POM

---

- Project Object Model
- Captures important meta data about your development effort
- XML format means it can be generated or transformed for IDEs, *e.g.* Eclipse, Idea, JBuilder, Jdeveloper, *etc.*
- Maven can run without one, but functionality is severely reduced
- Defined by schema



# POM – Elements

---

- `<project>`
  - root element
- `<extend>`
  - a project may optionally extend another and inherit its values, overriding as needed
  - This means many related projects can specify common data in a single place
  - Examples: plug ins, sub-projects, tag libraries, *etc.*



# POM – Elements *(Continued)*

---

- `<pomVersion>`
  - Version of the POM maven is using
- `<id>`
  - Unique name for the project
  - Format like a Java variable
- `<name>`
  - A short name for the project
- `<groupId>`
  - The identifier for the organization/team



# POM – Elements *(Continued)*

---

- `<currentVersion>`
  - The version number of the project, *e.g.* 1.0, 2.1-alpha-5, 1.5-SNAPSHOT
- SNAPSHOT is a 'special' name recognized by maven when downloading
- `<organization>`
  - Details about the development team or organization
  - `<name>`, `<url>` and `<logo>` sub-elements



# POM – Elements *(Continued)*

---

- `<inceptionYear>`
  - The year the project started – *e.g.* used for copyright in Javadocs
- `<package>`
  - Java package base for the source code
  - JUST ONE - why?
- `<logo>`
  - Project logo



# POM – Elements *(Continued)*

---

- `<gumpRepositoryId>`
  - If your project is being built by gump, it may have a different id
- `<description>`
  - Description of the project, suitable for an about page
- `<shortDescription>`
  - Suitable for an index listing. Typically a one-liner



# POM – Elements *(Continued)*

---

- `<url>`
  - Web site for the project
- `<issueTrackingUrl>`
  - Web site for bug tracking, *e.g.* Bugzilla, Jira, *etc.*
- `<siteAddress>`
  - Host name for the Web site used for publishing
- `<siteDirectory>`
  - Directory on that host where the Web site is to be placed



# POM – Elements *(Continued)*

---

- `<distributionSite>`
  - Host name where source and binary distributions are published
- `<distributionDirectory>`
  - Directory on that host where the distributions are to be placed



# POM – Elements *(Continued)*

---

- `<repository>`
  - Details about the SCM system being used
    - `<connection>` – public connection string
    - `scm:cv:s:pserver:anoncvs@cv.s.apache.org:/home/cvspublic:maven`
    - Format is `scm:type:type-specific-details`
    - `<developerConnection>` – read write connection
    - `<url>` – Web based access to source, *e.g.* `viewcvs`



# POM – Elements *(Continued)*

---

- `<versions>`
  - Details about the versions present in the scm system
  - `<version>`
    - Details about a single version, `<id>`, `<name>`, `<tag>`
- `<branches>`
  - Like versions, except development is happening rather than being fixed



# POM – Elements *(Continued)*

---

- `<mailingLists>`
  - Nested `<mailingList>` elements
  - `<name>`, `<subscribe>` url, `<unsubscribe>` url
- `<developers>`
  - Nested `<developer>` elements
  - `<name>`, `<id>` from scm, `<email>`,  
`<organization>`, `<roles>/<role>`, `<url>`,  
`<timezone>`



# POM – Elements *(Continued)*

---

- `<contributors>`
  - Like `<developers>` but no `scm <id>`
- `<licenses>`
  - Nested `<license>` elements
    - `<name>`
    - `<url>`
    - `<distribution>` manual or *via* central repository



# POM – Elements *(Continued)*

---

- `<dependencies>`
  - Jars and other artifacts your project requires to build or test or run or...
  - Nested `<dependency>` elements
  - `<groupId>` – as before
  - `<artifactId>` – id of artifact produced by group
  - `<version>` – specific requirement by your project
  - `<jar>` – file name if not `<artifactId>-<version>-<type>`



# POM – Elements *(Continued)*

---

- `<dependencies>` continued...
  - `<type>` – jar, war, ear, plugin, tld, *etc.*
  - `<url>` – typically where to download it or the dependent project's home page
  - `<properties>`
    - Arbitrary string properties used to mark the dependency for processing later, *e.g.*  
`<war.bundle>true</war.bundle>`



# POM – Elements *(Continued)*

---

- `<build>`
  - Build time information
  - `<nagEmailAddress>` - email address to send to on build failure (not used by maven)
  - `<sourceDirectory>` - only ONE
  - `<sourceModifications>` - exclude/include code based on classes being available
    - `<className>`, `<excludes>/<exclude>`, `<includes>/<include>`

# POM – Elements *(Continued)*

- `<build>` cont...
  - `<unitTestSourceDirectory>`
  - `<integrationUnitTestSourceDirectory>`
  - `<aspectSourceDirectory>`
  - `<unitTest>`
    - `<includes>/<include>`, `<excludes>/<exclude>`
    - `<resources>`
      - ✓ `<directory>` – where to find them
      - ✓ `<targetPath>` – where they go
      - ✓ `<excludes>/<exclude>`, `<includes>/<include>`, `<filtering>`



# POM – Elements *(Continued)*

---

- `<build>` cont...
  - `<resources>` for inclusion in the jar, *etc.*
    - `<directory>` – where to find them
    - `<targetPath>` – where they go
    - `<excludes>/<exclude>`, `<includes>/<include>`, `<filtering>`
- `<reports>`
  - Nested `<report>` element
  - Named for the plug in producing the report



# POM – Elements *(Continued)*

---

- `<properties>`
  - Arbitrary string content about the project for use in later processing
  - Not used by Maven yet
- Mandatory top level
  - `pomVersion`, `id`, `name`, `currentVersion`, `organization`, `inceptionYear`, `shortDescription`, `developers`, `build`



# Agenda

---

- What is Maven?
- The POM
- **Source + POM = Build**
- Customizing Maven
- Maven & IDEs
- Migrating from Ant
- Multi-project builds
- Your own plugins



# Source + POM = Build

---

- Adding a project.xml to your project
- Configure it to reflect your code
  - project.xml
  - src/java/...
  - src/test/...
- We'll use the simple-java example that comes with Maven as a start



# Source + POM = Build

---

- maven --goals tells you which plug-ins are installed and available
- Produce a jar file
  - maven jar
- Run tests
  - maven test
- Delete generated files
  - maven clean



# Source + POM = Build

---

- Generate Javadocs
  - maven javadoc
- Produce project Web site
  - maven site
- Code coverage
  - maven clover (commercial)
  - Maven jcoverage (open source)
- Generate unit tests
  - Maven junitdoclet

# Source + POM = Build

## Directory Structure

---

- Maven specifies directory structure
- Configurable *via* properties, except
  - project.xml
  - maven.xml
  - Must be in the root directory, or use
    - maven --find
    - maven --dir

# Source + POM = Build

## Directory Structure

---

- basedir – directory of project.xml
- maven.src.dir – \${basedir}/src
- maven.build.dir – \${basedir}/target
- maven.build.dest – \${maven.build.dir}/target
- maven.docs.dest – \${maven.build.dir}/docs
- Most configuration happens through properties a plugin provides

# Source + POM = Build

## Property Resolution

---

- `${user.dir}/build.properties`, *e.g.*  
`~/build.properties` on Unix, or  
`%USERPROFILE%\build.properties` on Windows
- `${basedir}/build.properties`
- `${basedir}/project.properties`
- `${plugin.dir}/plugin.properties`
- Maven's `defaults.properties`



# Agenda

---

- What is Maven?
- The POM
- Source + POM = Build
- **Customizing Maven**
- Maven & IDEs
- Migrating from Ant
- Multi-project builds
- Your own plugins



# Customizing Maven

---

- Customizing *via* the POM
- Reporting choices
- Customizing *via* properties
- Providing goals in maven.xml
- Writing plug-ins



# Customizing Maven - POM

---

- This is the first way you will customize Maven
- Tailoring the sourceDirectory and other elements tells Maven and its plug-ins about your directory structure and project meta data.
- We've also covered the POM heavily in the previous section, so it's mentioned here as reference only



# Customizing Maven – Reports

---

- This is usually customized *via* the POM
- Can also use preGoal and postGoal in your maven.xml file
- Reports all come with similar goals:
  - `<report>:register`, `<report>:deregister`, `<report>:report`
- You can call these goals yourself from a custom maven.xml file



# Customizing Maven – Reports

---

```
<reports>
  <report>maven-jdepend-plugin</report>
  <report>maven-checkstyle-plugin</report>
  <report>maven-changes-plugin</report>
  <report>maven-changelog-plugin</report>
  <report>maven-findbugs-plugin</report>
  <report>maven-file-activity-plugin</report>
  <report>maven-developer-activity-plugin</report>
  <report>maven-javadoc-plugin</report>
  <report>maven-jxr-plugin</report>
  <report>maven-junit-report-plugin</report>
  <report>maven-tasklist-plugin</report>
  <report>maven-pmd-plugin</report>
  <report>maven-simian-plugin</report>
</reports>
```



# Customizing Maven – Reports

---

- Remove or add a report

```
<postGoal name="xdoc:register-  
reports">  
  <attainGoal name="maven-  
    linkcheck-plugin:  
    deregister"/>  
</postGoal>
```

# Customizing Maven – Properties

- `${user.dir}/build.properties`
  - Across all projects
- `${basedir}/build.properties`
  - Your changes for a build
- `${basedir}/project.properties`
  - Properties provided by the project
- `${plugin.dir}/plugin.properties`
  - Plugin defaults
- Maven's `defaults.properties`

# Customizing Maven – Installation Properties

- `$MAVEN_HOME`, `maven.home.local`
  - Where maven install files and user working files are kept
- `maven.repo.local`
  - `${maven.home.local}/repository`
  - Where the local repository of artifacts is kept
- `maven.plugin.dir`
  - `${maven.home}/plugins`
  - Where plug-in jars are kept

# Customizing Maven – Installation Properties

- `maven.plugin.unpackeddir`
  - `${maven.home.local}/plugins`
  - Where plug-ins are expanded and processed
- `maven.repo.remote`
  - <http://www.ibiblio.org/maven/>
  - Where dependent jars can be downloaded
- `maven.scp.executable`, `maven.ssh.executable`
  - For site deployment

# Customizing Maven – Directory Structure

- basedir – directory of project.xml
- maven.src.dir – \${basedir}/src
- maven.conf.dir – \${basedir}/conf
- maven.build.dir – \${basedir}/target
- maven.build.dest – \${maven.build.dir}/target
- maven.docs.src – \${basedir}/xdocs
- maven.docs.dest – \${maven.build.dir}/docs
- maven.gen.docs – \${maven.build.dir}/generated-xdocs

# Customizing Maven – Java Plugin Properties

---

- `maven.compile.debug = on`
- `maven.compile.optimize = off`
- `maven.compile.deprecation = off`
- `maven.compile.target = 1.1`
- `maven.compile.src.set` – code to be compiled as an ant path
- `maven.compile.compilerargs`
- `maven.compile.fork` *etc*

# Customizing Maven – Test Plugin Properties

- Uses all compile options from Java plugin
- `maven.test.dest = ${maven.build.dir}/test-classes`
- `maven.test.reportsDirectory = ${maven.build.dir}/test-reports`
- `maven.test.skip = false`
- `maven.junit.jvmargs=maven.junit.fork = no`
- `maven.junit.usefile = true`
- `maven.test.failure = true`

# Customizing Maven – xdoc Plugin Properties

---

- `maven.xdoc` prefix
- `date`, `date.format`, `date.locale`
- `developmentProcessUrl` = Maven URL
- `includeProjectDocumentation` = `yes`
- `poweredBy.image`, `.title`, `.url`
- `maven.ui.*` - 'stylesheet' values

# Customizing Maven – xdoc Sample



Last published: 30 July 2003 | Doc for 2.0-rc2

## ScWebconf

- Overview
- Features
- Demo
- Download
- News and Status
- Getting Started
- Technical

## Misc

- FAQ
- Tasks

## Project Documentation

- About ScWebconf
- Project Info
- Project Reports
- JavaDocs
- Source XReference
- Development Process



## Overview

ScWebconf is a Configuration Management Application for complex products and services. It uses a pluggable rule engine that allows for a more natural expression of business rules with regards to business objects.

*Many enterprise software systems today already include the concept of rules. Most times, these rules are directly implemented in code and are difficult to adapt to a changing business landscape. The domain model often includes business logic which may change several times. When these business 'rules' are coded using normal systems programming techniques, modification and maintenance of the logic can become difficult.*

Bob McWhirter in [Theory, Usage and Reference of the Drools Rule Engine](#)

## How does it work

This section gives a broad overview of a normal workflow scenario.

Below you see an image of an example configuration.



# Customizing Maven – War plug-in Properties

- `maven.war.src = src/webapp`
- `maven.war.webxml = ${maven.war.src}/WEB-INF/web.xml`
- `maven.war.webapp.dir =`  
`${maven.war.build.dir}/${pom.artifactId}`  
where the Web app is created from source
- `<properties><war.bundle>true</war.bundle></properties>`  
on a dependency

# Customizing Maven – Java Runtime Properties

---

- `maven.dependency.classpath`
  - All dependent jars as an Ant path
- `sourcesPresent` – if the `sourceDirectory` is specified and exists
- `unitTestSourcesPresent` – same for unit tests
- `maven.compile.src.set`
- `maven.test.compile.src.set`

# Customizing Maven – maven.xml

---

- maven.xml is to maven as build.xml is to Ant
- Where you can write your own scripts
- Uses Jelly, not just Ant tasks
- Have complete access to Maven plugins and core, as well as any Java object you can instantiate

# Customizing Maven – maven.xml goals

---

- Like Ant targets, a named action to be executed
- May consist of many 'tasks'
- Maven adds 'preGoal', and 'postGoal' to allow easy hooking of existing processes
- Syntax is Jelly – a scripting language in XML format
- Provided by Werkz

# Customizing Maven – maven.xml goals

- Sample goal
- Sample preGoal
- Sample postGoal

```
<project default="jar:jar">  
  
  <goal name="helloWorld">  
    <echo>Hello World!</echo>  
  </goal>  
  
  <preGoal name="java:compile">  
    <echo>Compile code</echo>  
  </preGoal>  
  
  <postGoal name="jar:jar">  
    <copy  
      file="{maven.final.name}"  
      toDir="c:/install"/>  
    </postGoal>  
  </project>
```

# Customizing Maven – maven.xml scripting

---

- Jelly has a raft of tag libraries that are available for use in Maven
- The core, ant, define and util tag libraries are available simply by using an xmlns declaration in maven.xml
- `<project xmlns:ant="jelly:ant">`

# Customizing Maven – maven.xml scripting

- Core:
  - arg, break, case, catch, choose, default, expr, file, forEach, if, import, include, invoke, jelly, new, otherwise, remove, scope, set, setProperties, switch, thread, useBean, useList, when, while, whitespace
- Define (allows defining new tags inline)
  - attribute, bean, classLoader, dynaBean, extend, invoke, invokeBody, jellyBean, script, super, tag, taglib

# Customizing Maven – maven.xml scripting

- Util:
  - available, file, loadText, properties, replace, sleep, tokenize
- Ant:
  - Any ant task (well, almost)
  - ant, filescanner, setProperty
- Samples



# Agenda

---

- What is Maven?
- The POM
- Source + POM = Build
- Customizing Maven
- **Maven & IDEs**
- Migrating from Ant
- Multi-project builds
- Your own plugins



# Maven & IDEs

---

- Maven is still essentially a command line tool
- However it does provide integration with various IDEs
  - Eclipse
  - JDeveloper
  - JBuilder
  - Idea
  - JDEE



# Maven & IDEs – Eclipse

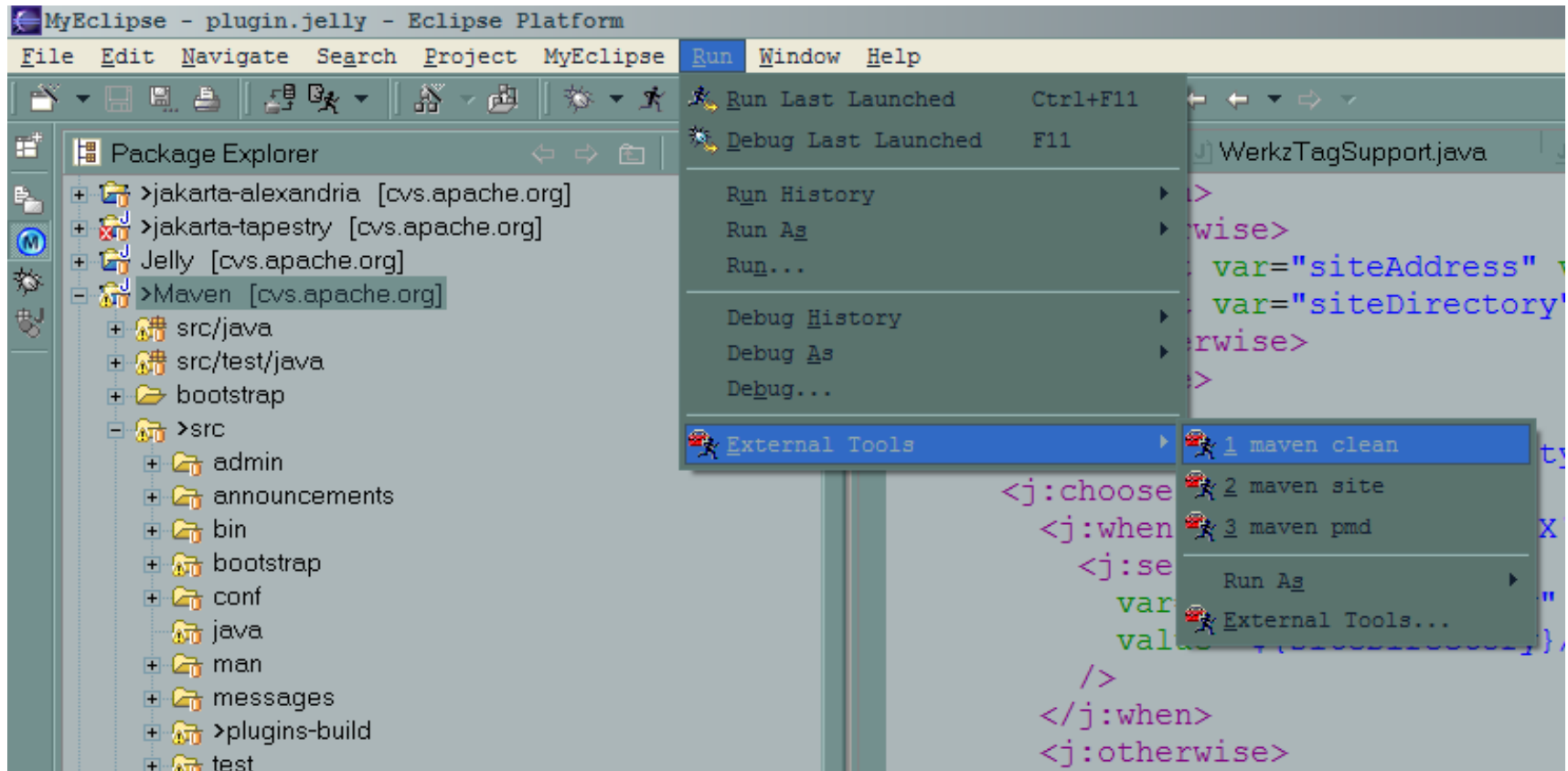
---

- Eclipse plugin
  - add-maven-repo: adds the MAVEN\_REPO variable to Eclipse
  - external-tools: adds maven goals to the Run menu of Eclipse
  - generate-classpath: creates a .classpath file with the dependencies listed in the POM
  - generate-project: creates a .project file with Java nature

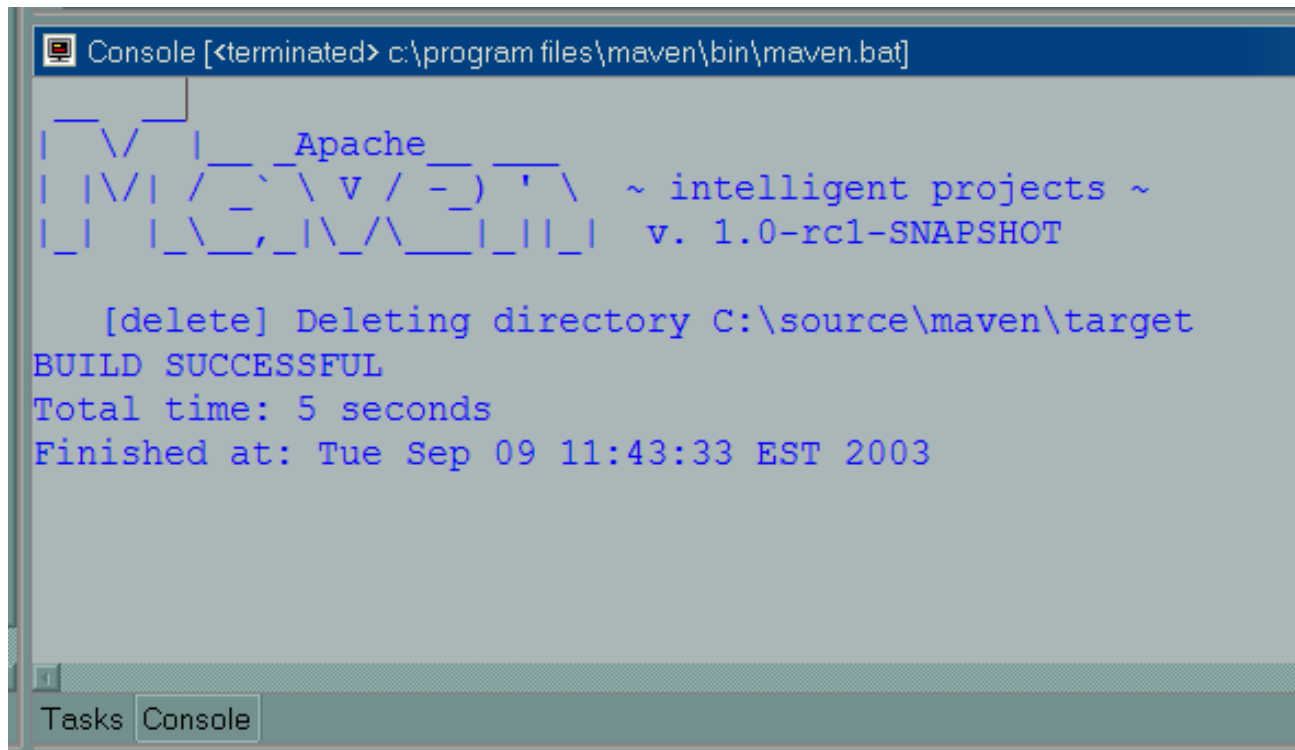
# Maven & IDEs – Idea, Jbuilder, JDeveloper

- `idea:generate-project` – generate `.iml`, `.ipr` and `.iws` files from POM
- `jbuilder:generate-library` – generates `project.library` file from POM
- `jbuilder:generate-project` – generates `project.ipr` file from POM
- `jdeveloper:generate-project` – generates `.jpr` file from POM
- `Idee:generate-project` – generate project file from POM

# Maven & IDEs – Eclipse



# Maven & IDEs – Eclipse



The screenshot shows a console window titled "Console [<terminated> c:\program files\maven\bin\maven.bat]". The output text is as follows:

```

  ____
 |  \  | |   Apache
 |  \| /  \  \ v / -) ' \ ~ intelligent projects ~
 |  | \ ,  | \ / \  | | | | v. 1.0-rc1-SNAPSHOT

      [delete] Deleting directory C:\source\maven\target
BUILD SUCCESSFUL
Total time: 5 seconds
Finished at: Tue Sep 09 11:43:33 EST 2003

```

At the bottom of the console window, there are two tabs: "Tasks" and "Console".



# Agenda

---

- What is Maven?
- The POM
- Source + POM = Build
- Customizing Maven
- Maven & IDEs
- **Migrating from Ant**
- Multi-project builds
- Your own plugins



# Migrating from Ant

---

- Maven is a project based tool, Ant is a scripting tool
- Maven requires meta data up front, Ant is easy entry to hack away
- You have to learn Ant's tasks and their attributes to write a simple 'jar' target
- No procedural scripting in the distributed Ant
- Ant is complete freedom



# Migrating from Ant

---

- How many build files do you have?
- Is your Ant build file a single Maven project?
- Are the targets covered by Maven's plugins?
- Does your directory structure fit with Maven's expectations, *e.g.* single source directory, test in same tree, *etc.*
- What do you want to get from the migration?
- Are there reusable Ant targets that you can turn into plugin(s)?



# Migrating from Ant

---

- Set up project.xml
- Set up project.properties
- Copy relevant targets from build.xml to maven.xml as goals
- Use plugins wherever possible
- Test and iterate



# Agenda

---

- What is Maven?
- The POM
- Source + POM = Build
- Customizing Maven
- Maven & IDEs
- Migrating from Ant
- **Multi-project builds**
- Your own plugins



# Multi-project Builds

---

- Maven itself is a typical example
- Core code and lots of 'plugins'
- See Jelly and its taglibs, Log4J and its appenders, James and Mailets, Tomcat and valves, *etc.*, Apache and Modules, Netscape and plugins, *etc.*
- Multi-project plugin is the start of addressing these needs

# Multi-project Builds – Reactor

## Tag

- Tag provided with Maven

```
<maven:reactor
  basedir="."
  banner="Reactored build"
  includes="*/project.xml"
  goals="clean,jar:jar"
  postProcessing="true"
  ignoreFailures="false"/>
```

# Multi-project Builds – The Plugin

- Part of 1.0-beta-10
- Goals:
  - site: generate a combined site for all projects
  - create-overview-page: create a page with descriptions of all projects
  - goal: Run the same goal in all projects
  - artifact: builds an artifact based on type
  - install: install the artifact for all projects
  - deploy: deploy all projects

# Multi-project Builds – The Plugin *(Continued)*

---

- Goals continued...
  - deploy-snapshot: deploy a snapshot of all projects
  - report: register, deregister and run the dependency convergence report
  - clean: clean all projects

# Multi-project Builds – the plugin properties

- Basedir – base directory to search from
- Includes – ant pattern to find projects
- Excludes – ant pattern to ignore files
- IgnoreFailures – whether to stop if a single project dies
- Navigation – aggregate or independent site structure
- Type – jar, war, ear, plugin, *etc.*



# Agenda

---

- What is Maven?
- The POM
- Source + POM = Build
- Customizing Maven
- Maven & IDEs
- Migrating from Ant
- Multi-project builds
- **Your own plugins**



# Writing Plugins

---

- Once you've experimented with `maven.xml`, you may find reusable pieces
- These reusable pieces are easiest for others to use as a plugin
- Plugins are installed into `$MAVEN_HOME/plugins`
- Automatically expanded and unpacked.
- Plugins can be specified in the dependencies

# Writing Plugins – Details

---

- A plugin is like any other project, you need a `project.xml`
- See the ones in Maven's source tree for a start
- Next, create a `plugin.jelly` file to hold the goals and tags you will create
- Put properties you expect a user to supply in `plugin.properties`

# Writing Plugins – Details

## *(Continued)*

---

- Any properties you need to compile and build your plugin, add to `project.properties` as per usual
- If you think you've got it right run:
  - `maven plugin:install`
- Now run one of the goals in your plugin
  - `maven myplugin:mygoal`
- Generate docs: `maven plugin:generate-docs`

# Writing Plugins – Details

## *(Continued)*

---

- Create a site for your plugin
  - maven site
- Distribute to users
- Change, `plugin:install`, test
- Done!
- Another option
  - edit in place `~/.maven/plugins/<plugin-name>/plugin.jelly`



# Summary

---

- Maven is a powerful build tool
- It can create lots of useful documentation
  - What are you going to do with it?
- You can work with it or against it
- It's only open source
- Suggestions and improvements are always welcome



# Resources

---

- <http://maven.apache.org>
- <http://ant.apache.org>
- <http://jakarta.apache.org/commons/jelly/>
- <http://maven-plugins.sourceforge.net>
- <http://mevenide.sourceforge.net>
- [users@maven.apache.org](mailto:users@maven.apache.org)
- <http://wiki.codehaus.org/maven/>
- [dion@multitask.com.au](mailto:dion@multitask.com.au)