

Maven 101: A Maven primer

[Revision History 1](#_Toc345488240)

[What is Maven? 2](#_Toc345488241)

[The grand-father: make 2](#_Toc345488242)

[The father: ant 2](#_Toc345488243)

[Here comes Maven 3](#_Toc345488244)

[Maven design goals 3](#_Toc345488245)

[Making the build process easy 3](#_Toc345488246)

[Providing a uniform build system 3](#_Toc345488247)

[Providing quality project information 3](#_Toc345488248)

[Providing guidelines for best practices development 4](#_Toc345488249)

[Allowing transparent migration to new features 4](#_Toc345488250)

[Benefits of Maven 4](#_Toc345488251)

[FAQ 5](#_Toc345488252)

[Code samples and first use 7](#_Toc345488253)

[Examples of Maven outputs 7](#_Toc345488254)

[Some useful shortcuts to create in $MAVEN/bin or useful commands 8](#_Toc345488255)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Reason For Changes | Revision |
| Olivier DUPUY | December 25, 2012 | Initial version | 1.0 |
| Olivier DUPUY | January 4 | Added logo, formatting, fixed typos | 1.1 |

# What is Maven?

Maven is an open source project management tool hosted by the Apache group but let’s have a look at the history behind it.

## The grand-father: make

Developers know about the [make system](http://en.wikipedia.org/wiki/Make_%28software%29) where a make file is made of targets (e.g. compile) and rules expressing dependencies. You can realize a target from the command line. Make is nearly as old as Unix and provides a common syntax. Make does not come with much more then this base logic of rules and dependencies. The build master has to write nearly everything from scratch, has to know about the command lines, options, syntax of the underlying commands (e.g. the cc C compiler). The debugging can be tedious from a platform to another one. Make knows nothing about Java by default.

## The father: ant

To overcome these limitations, the [Tomcat](http://tomcat.apache.org/) project used internally a Java based command line to build, assemble and test the different parts of the project. This is [Ant](http://ant.apache.org/). Ant was quickly promoted as a top level Apache project of its own because it could be used outside of Tomcat. Ant “knows” about Java and J2EE and provides a simple interface to every standard Java binary command (javac, java, jar…) with some base options. We have the same notion of targets with dependencies to other targets. To realize a target, different [Ant task](http://ant.apache.org/manual/index.html)s are used, each Java command line tool (e.g. jar) being the subject of one or more tasks.

Compared to make, Ant makes a bridge with the Java world and is highly extensible. XML is used to drive the dependencies and tasks. You can easily write your own Java based ant tasks.

However the build master is still left with the job of writing the base logic of writing the target dependencies again and again. E.g. to compile the tests, compile the code first then add the compiled classes to the classpath used to compile the tests or compile the code first to create a JAR file. Some logic is quite repetitive is and applies from one project to the other. Some options (command options, folder locations, names of the files, path…) differ from a project to another. You can try to use some template pattern, put all the logic in one XML file and all the driving properties aka configuration (paths, options…) in another included file. This provides a way to reuse some logic. However this does not scale so well and the logic has to be reinvented by each team.

In any case, Ant does not address modern development with many recursive dependencies to external libraries (Apache, Spring, Google…), SCCS branches and the management of the deliverables, jar files, source, documentation, licenses…

## Here comes Maven

[Maven](http://maven.apache.org/) has been designed to simplify the building of Apache Turbine which used different Ant based components. The [design goals of Maven](http://maven.apache.org/what-is-maven.html) were to have a standardized build environment with the possibility to publish and share jar files across projects and a quick learning process.

Maven is built in Java and is especially good at Java development but can be used for any kind of project, C# or other.

# Maven design goals

## Making the build process easy

Maven knows that your tests depend on your code and all these kinds of things. The details are cached to you but this is because some default conventions are implemented.

As long as you do not know the conventions, everything looks like magic in Maven while in Ant you can refer to some code. Once the conventions known, all the projects look simpler and similar.

## Providing a uniform build system

The POM structure, the file organization ([maven standard directory layout](http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html)), the names of the targets, their dependencies are about conventions versus configuration. If you know Maven, it is very easy to discover a new one. Also it is very easy to compose bigger projects from simple one.

On a project following the Maven conventions, Maven dramatically reduces the number of lines for the configuration. The ratio Maven versus Ant lines of code is from experience around 1 to 10 with anyone easily able to run the different operations from the command line AND to make some changes often in a single spot. A build guru is much less needed.

## Providing quality project information

You can describe the developers, the dependencies direct or indirect, the license, the SCCS and bug systems links… You get this, the unit test results and more. As well, this allows as a good example to create the whole site information in one command, mvn site (below) where all the information is published.

Look at the [log4j site](http://logging.apache.org/log4j/1.2/dependencies.html) to see what Maven can document for you. No doubt that so many projects have adopted Maven.

## Providing guidelines for best practices development

To cite a few, sound naming conventions, separating source code from test, generated code from compiled one, resources in dedicated folders, directory structure, etc.

Here is the build lifecycle of 8 phases with multiple goals associated to each phase and a plug-in involved for each goal and sometime for multiple goals.

|  |
| --- |
| process-resources compile process-test-resources test-compile test package install deploy |

## Allowing transparent migration to new features

Maven 2 is “maven 2” while technically the version is 3. In any case, maven is very stable. A new version can come with some updated plug-ins but the compatibility is preserved. Plug-ins are automatically downloaded and used. The first times you do a build or use a plug-in, you may see a long list of downloads but once done and cached they will not occur frequently.

# Benefits of Maven

Maven makes your product more industrial versus an hand crafted system were you need to be a “maven” to do a change.

Maven enables agility and easy change control, especially for the dependencies. Without maven, things are hard to figure out, we don’t know what depends on what. Because of this, we are shy about doing a change and when we do it this is difficult to test it.

With maven this will be much easier to experiment. We can list all the existing dependencies. We can add a new library; change the version of another one et voila.

# FAQ

**Q: Can I call Maven from Ant?**  
A: Yes but there is no Maven task shipped by default (with Ant). The exec task does the job.

|  |
| --- |
| <target name="buildProject" description="Builds the individual project">  <exec dir="${source.dir}\${projectName}" executable="cmd">  <arg value="${env.MAVEN\_HOME}\bin\mvn.bat"/>  <arg line="clean install" />  </exec>  </target> |

The other alternative is to use the Ant tasks provided by the Maven project itself. [A subset of the Maven functions](http://maven.apache.org/ant-tasks/index.html) is available but you can leverage and integrate the interesting ones such as taking care of the dependencies, adding them to your Ant paths. So you can have a project mixing the two.  
**Q: Can I call Ant from Maven?**  
A: It may make sense to do this in some cases such as when you have an existing Ant base that you move progressively or very complex task. If highly configured and not using maven profiles and the [maven standard directory layout](http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html) (as you should) Maven can become quite verbose. The [Maven AntRun Plug-in](http://maven.apache.org/plugins/maven-antrun-plugin/) is what you look for.  
**Q: Which one to use, Ant or Maven?**  
A: Well it depends but on a brand new project, the winning choice is normally Maven. You can see very clear market trends [here](http://www.indeed.com/jobanalytics/jobtrends?q=ant%2C+maven&l_=), ant glory days are over. Maven is much more powerful for dependency management, small projects and integration with other products such as continuous integration. The ramp up time for is also smaller for Maven because the uniformity of the configuration and commands.  
**Q: Maven cannot find or download some plug-ins**A: You type some Maven command and you receive an error message. Check first or modify your <proxies> settings by uncommenting the relevant section in "C:\Documents and Settings\TXXXXX\.m2\settings.xml". The full proxy rules are picked up from the [PAC script](http://pac.tsl.telus.com/standard.pac). If you open the file in FF, the **current** default proxy is at the end and is 198.161.14.25:8080*.* After that, some libraries may give some warnings as the source or the javadoc is not available. You also have the case of non published libraries as this is the case for some Sun/Oracle ones. Alternatives such as OpenJDK, Tomcat, OpenEJB… can provide an alternative implementation of some specifications allowing you to compile and test.  
**Q: Do we have Oacis using Maven?**A: The RTFEditorKit project (in CVS as JDK/RTFEditorKit) has been Mavenized on the late as the Eclipse files were missing and to introduce unit testing. Sadly it does not have the [maven standard directory layout](http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html) which would simplify the POM file. Jar signing, using local libraries instead of downloaded ones… makes this more complex than usual. The industrial solution is to move some of the configuration points (e.g. signing) to shared profiles to write them once for all and to follow the standard layout. The README file provides the steps to use it. The architecture team upstairs uses extensively maven, a maven repository…  
**Q: What does a complex project looks like?**A: Different Maven modules depending on each other under the same project, e.g. one for a common jar file, one for the WAR file … They depend on each other, they inherit their dependency. Building the upper one triggers building the lower ones…  
**Q: What can I use Maven for in my daily work?**A: Certainly, some simple answers are to:  
- jumpstart a small project or library where you don’t need any Oacis code. For any new “library”, Maven should be your choice. Start your project using the archetype plug-in (below).  
- test the impact of a library upgrade, e.g. for a Spring upgrade, create a simple project using the archetype then add a spring dependency. A test dependency used only to compile and run the test would also have <scope>test</scope>. The default scope is “compile” (will be added to WEB-INF/lib or other), “provided” is for the libraries used to compile but coming with the environment (e.g. mail.jar on a J2EE server)

<dependency>  
 <groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>3.1.1.RELEASE</version>

</dependency>  
look at the dependencies (*mdt* (below) with the output redirected to a file) then update the version and compare the two trees (use [Winmerge](http://winmerge.org/) to compare the files) to see how the dependencies have evolved   
- nearly all the OSS libraries are now mavenized, use the [Maven repository](http://mvnrepository.com/artifact/org.springframework/spring-core/3.1.1.RELEASE) to discover their dependencies and which library of your project is using them. You also simply find the type of license, the packages used and more…  
**Q: Where are the plug-ins and libraries downloaded from?**A: They come from the Maven 2 Central Repository where the standard ones are known and available in a cache but you can add other public repositories in your configuration. Typically an enterprise, will create its own repository where are stored the different version of its products (Jar, War, Ear files…) and which can act as a cache to limit the internet downloads.  
Apache Archiva and Sonatype Nexus are two common repository managers (aka build artifact repositories) frequently used with Maven. You can publish to it your project snapshots and versions under development.

**Q: Can I force the versions used?**A: Yes you can like when you have different versions of the same library required by other libraries. Using defined versions and profiles is the best way to do it. You can also force some dependent libraries to be excluded if you wish.

**Q: What is a bill of material?**A: This refer to you moving to a profile a common configuration for the base components e.g. Java version, J2EE version, Spring version…. You can create multiple such profiles and reuse them across your projects. This provides a common platform to your projects.  
**Q: How to mavenise a big project?**A: Good question. Some clues are to introduce some Maven elements. Maybe some missing targets or task could be implemented with Maven. More interestingly using [Maven Ant Tasks](http://maven.apache.org/ant-tasks/index.html), you can use Maven to manage the list of dependencies and inject these ones in your classpath or other or publish your ant jars to the local maven repository to use them in other projects.

# Code samples and first use

Play with the RTFEditorKit project, it is pretty complete even if some parts could (should) be externalized in common Maven profiles to make it simpler and cleaner.

Use *mvn archetype:generate* (below) to create a sample project.  
Then start adding some dependencies to other libraries and start to use them (e.g. spring). You can try multiple top level dependencies having some common dependency such as log4j or commons logging to see different scenarios.  
More content in the next Maven paper.

## Examples of Maven outputs

Table 1 List of the dependencies for a project

|  |
| --- |
| C:\workspace\testfwk>mvn dependency:tree  [INFO] Scanning for projects...  ----------------------------------------------------------------------  [INFO] Building testfwk 1.0-SNAPSHOT  ----------------------------------------------------------------------  [INFO] --- maven-dependency-plugin:2.1:tree (default-cli) @ testfwk --  [INFO] com.oacis:testfwk:jar:1.0-SNAPSHOT  [INFO] +- org.springframework:spring-core:jar:3.0.7.RELEASE:compile  [INFO] | +- org.springframework:spring-asm:jar:3.0.7.RELEASE:compile  [INFO] | \- commons-logging:commons-logging:jar:1.1.1:compile  [INFO] +- commons-collections:commons-collections:jar:3.2.1:compile  [INFO] +- junit:junit:jar:4.10:test  [INFO] | \- org.hamcrest:hamcrest-core:jar:1.1:test  [INFO] \- org.easymock:easymock:jar:3.1:test  [INFO] +- cglib:cglib-nodep:jar:2.2.2:test  [INFO] \- org.objenesis:objenesis:jar:1.2:test  ----------------------------------------------------------------------  [INFO] BUILD SUCCESS  ----------------------------------------------------------------------  [INFO] Total time: 9.224s  [INFO] Finished at: Tue Jul 10 23:13:25 EDT 2012  [INFO] Final Memory: 5M/15M |

## Some useful shortcuts to create in $MAVEN/bin or useful commands

|  |
| --- |
| // general syntax mvn plugin-name:goal-name  // create a project to start from // see <http://maven.apache.org/guides/introduction/introduction-to-archetypes.html> mag.bat = mvn archetype:generate (answer (ENTER)/6(or ENTER)/com.telushealth/projectofmine/1.0-SNAPSHOT(or ENTER)/com.telushealth(or ENTER)/Y(or ENTER) Your project is ready. CD to the new folder. You can complete the POM file and use the commands below.  // compile code then tests, run the tests and create the different jar files mci.bat = mvn clean install %1 %2 %3 %4 %5  // compile code then tests, skip running the tests and create the different jar files mcis.bat = mvn clean install -DskipTests %1 %2 %3 %4 %5  // cleans the eclipse artifacts created by maven mec.bat = mvn eclipse:clean %1 %2 %3 %4 %5  // creates an eclipse project with all the transitive dependencies mee.bat = mvn eclipse:eclipse -Dwtpversion=2.0 %1 %2 %3 %4 %5  // creates an eclipse project with all the transitive dependencies downloading source and javadoc as needed and making them available meed.bat = mvn eclipse:eclipse -DdownloadSources=true -DdownloadJavadocs=true -Dwtpversion=2.0 %1 %2 %3 %4 %5  // show the transitive tree of dependencies // mdt > mytree.txt to save it mdt.bat = mvn dependency:tree %1 %2 %3 %4 %5  // show missing or non required top level dependencies in your code mda.bat = mvn dependency:analyze %1 %2 %3 %4 %5  // create your jar files for the source, code, tests… mvn package  // create your javadoc mvn javadoc  // creates the site documentation in target/site // Complete POM.xml to have more content mvn site |