Basics

`mvn <plugin>:<goal> [-Doption1 -Doption2 ... ]`	Basic maven invocation of <goal> (multiple goals possible)
`mvn <phase> [-Doption1 -Doption2 ... ]`	Execute maven until <phase> (multiple phases possible)
`mvn -h`	Getting help on command line parameters
`mvn -o`	Working offline
`mvn -N`	Skip visiting of child artifacts (the given goal or phase will be executed only for the current artifact - no recursive descent)
`mvn -DtestFailureIgnore=true <goal>`	To continue the maven build even if a test fails
`mvn install -Dmaven.test.skip=true`	To make maven install without launching the tests
`mvn test`	To run unit tests
`mvn clean`	To clean up
`mvn package`	To build the package file (jar, war, etc)
`mvn -P<profile>`	Activate a profile

Important phases

- Build lifecycle: validate, compile, test, package, integration-test, verify, install, deploy
- Clean lifecycle: clean
- Site lifecycle: site, site-deploy

Configuration

Environment variable	Significance
M2_HOME	Root of where the `mvn` executable is located
MAVEN_OPTS	Command line options of the JVM launching maven
M2_REPO	Location of the local maven repository

Configuration file for maven: `~/.m2/settings.xml`.

We recommend to set the following parameter on the command line of maven: `-fae` "fail at end" Rationale: (1) you see explicitly what tasks have been run and (2) maven continues until the end (without breaking on intermediate errors) BTW: We added this parameter to the default maven launch script.

Eclipse Plugin

`mvn eclipse:eclipse`	To generate Eclipse project descriptor after configuring the dependencies in pom.xml (see also next point)
`mvn eclipse:clean eclipse:eclipse -DdownloadSources=true`	To setup the source versions of all included libraries (for debugging convenience)

Site generation

`mvn site`	To create a complete documentation website containing Javadoc, Checkstyle, JUnit and other report pages for the current artifact. JUnit tests will be directly executed for the current artifact. The generated site can be found at "target/site"
`mvn -Dmaven.test.skip=true clean site`	To generate site documentation without running the tests (handy while updating the APTs)
`mvn javadoc:javadoc`	To generates Javadoc into directory "target/site/apidocs"
`mvn checkstyle:checkstyle`	To check the style of the code by applying the EL4J's Checkstyle rules and generate reports at "target/site/checkstyle.html"

Cargo Plugin (to deploy artefacts in a web or EJB container)

`mvn cargo:start`	To start the configured container. By default this is Tomcat 5.5
`mvn cargo:deploy`	To deploys the configured deployable. If the current artifact is of type war this war will be deployed
`mvn cargo:undeploy`	To undeploy the configured deployable. If the current artifact is of type war this war will be undeployed
`mvn cargo:stop`	To stop the configured container. By default this is Tomcat 5.5

Varia

`mvn exec:java -Dexec.args="\"A single argument\""` [-Dexec.executable="maven"] [-Dexec.workingdir="/tmp"]	To execute the default java class of this module
`mvn depgraph:fullgraph -Ddepgraph.groupFilter="ch.elca"`	Get a dependency graph
`mvn depgraph:fullgraph -Ddepgraph.groupFilter="(ch.elca.el4j.modules)|(ch.elca.el4j.demos)|(ch.elca.el4j.apps)" -Ddepgraph.filterEmptyArtifacts=true -Ddepgraph.dotFile=el4j.dot`	Another dependency graph
`mvn assembly:assembly -DdescriptorId=src`	To distribute the source code
`mvn install:install-file -Dfile=foo.jar -DgroupId=bar -Dversion=x.y -Dpackaging=jar -DartifactId=blah`	To install a jar file on local repo
`mvn install -DperformRelease=true`	Installs all artefacts of the project in the current directory in local repo (includes src, javadoc, tests)
`mvn assembly:assembly -DdescriptorId=jar-with-dependencies`	To generate a single jar files with all its dependencies
`mvn archetype:create -DarchetypeGroupId=ch.elca.el4j -DarchetypeArtifactId=EL4JArchetypeCore -DarchetypeVersion=1.3 -DgroupId=ch.elca.el4j -DartifactId=myFirstProject -DremoteRepositories=http://el4.elca-services.ch/el4j/maven2repository`	To create a new EL4J project
`mvn install:install-file -Dfile=foo.jar -DgroupId=org.foosoft -DartifactId=foo -Dversion=1.2.3 -Dpackaging=jar`	Installing 3rdParty jar in local repository.

`cd $EL4J_ROOT/external` ; `mvn -N deploy:deploy-file -DgroupId=org.springframework -DartifactId=spring -Dversion=2.0.5 -Dpackaging=jar -Dfile=D:/tools/spring-framework-2.0.5/dist/spring.jar -DrepositoryId=ftpEl4ElcaServices -Durl=ftp://el4.elca-services.ch/htdocs/el4j/maven2repository`	Install 3rdParty jar to remote repository (requires pom.xml in . and requires login/password for repo)
`cd $EL4J_ROOT/external` ; `mvn -N deploy:deploy-file -DgroupId=org.springframework -DartifactId=spring -Dversion=2.0.5 -Dpackaging=source -Dfile=D:/tools/spring-framework-2.0.5/dist/spring-src.zip -DrepositoryId=ftpEl4ElcaServices -Durl=ftp://el4.elca-services.ch/htdocs/el4j/maven2repository`	Install 3rdParty source zip to remote repository

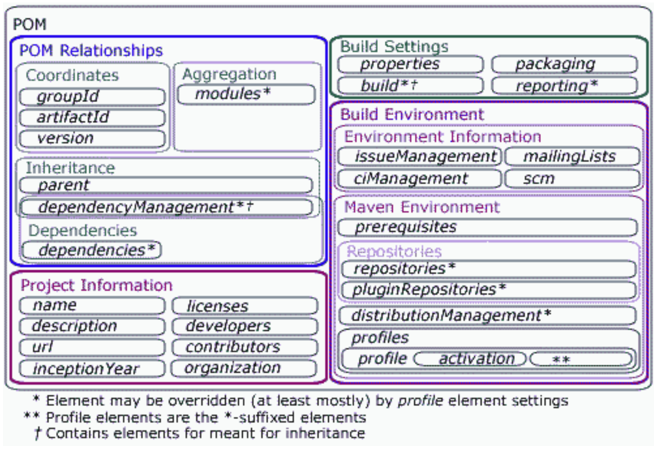
Launching of the database via the database plugin

`mvn db:prepareDB cargo:undeploy cargo:deploy cargo:start`	Initialize the database, install the deployable and start the container
`mvn db:start db:silentDrop db:create`	Does basically the same as `db:prepareDB`
`mvn -Ddb.connectionPropertiesSource=scenarios/db/raw/keyword-core-tests-override-db2.properties -Ddb.sqlSourceDir=etc/sql/db2/ db:prepareDB db:block`	Initialize and launch the db for testing keyword data access tests under `applications/templates/common/keyword/tests` CAVEATS: (1) connectionPropertiesSource is mandatory (2) sqlSourceDir must be relative to classpath (3) db:prepareDB does not block (even with db.wait flag)

Debugging

`mvn -X`	Enable debug output
`mvn –N help:effective-settings`	Prints the effective settings on the console
`mvn –N help:effective-pom`	Prints the effective pom on console
`mvn –N help:active-profiles (–P.. –D..)`	To test what profiles of the current artifact are currently active. In addition you can set profiles (-P) or system properties (-D) on the command line to see what profiles would be active in that case.
`mvn project-info-reports:dependencies`	Makes a report on dependencies. See in target/site/dependencies.html
`mvn –N help:describe -DgroupId… –DartifactId… -Dfull=true`	Describes all goals of the given plugin (groupId & artifactId).
set `MAVEN_OPTS="-Xmx768M -XX:MaxPermSize=512M -Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000"`	To be able to debug a running maven with e.g. eclipse

pom overview



References

- Extensive maven presentation: http://el4j.sourceforge.net/docs/pdf/Maven2Course_v1_2.pdf
- Documentation of standard Maven 2 plugins: http://maven.apache.org/plugins/
- Documentation of Maven 2 plugins at codehaus (a bit out of date): http://docs.codehaus.org/display/MAVEN/Maven+Plugin+Matrix
- Maven book: http://www.mergere.com/m2book_download.jsp
- EL4J: http://el4j.sourceforge.net/
- PluginDatabase
- PluginDepGraph

Revision: r1.12 - 07 Aug 2007 - 15:37 - PhilippHOser

EL4J > ComponentsEL4J > MavenBuildSystem > MavenCheatSheet