***Maven***

* *Maven is a full-fledged build tool*
* *A lot of implicit functionality*
* *Consistency across projects*
* *Also capable to achieve inheritance in projects*
* *Transitive dependencies (can be achieved using Ivy with Ant though)*
* *Built around versioning*

***Pros and Cons***

|  |  |
| --- | --- |
| * *Maven can be a black box* * *Steeper learning curve* * *Convention over configuration* * *Better IDE integration* * *Let overheard through use of repos* * *Different mindset, steepest learning curve is not making Maven act like Ant* | * *You can trace through Ant files fairly easily* * *Quicker to learn, but very copy-paste intensive* * *Larger project size in SCM, artifacts stored with project* |

***Notes***

* *Ant is very declarative*
* *Maven follows a convention over configuration model*
* *Ant is maybe easier to learn, but it really is only beneficial as a scripting tool*
* *Maven is really centered around managing your entire project’s lifecycle*
* *Source code goes in src/main/java*
* *Everything is compiled to our target directory*
* *The POM has 4 major parts*
* *Introduction of goals* 
  + *You can chain goals*
* *Basic example of a dependency*
* *Where things are stored in your local repo*
* *How we can override the default behavior*

***Maven Structure***

* *src/main/java*
* *target*
* *pom.xml*



* *Where we store our Java code* 
  + *the beginning of our package declaration* 
    - *com.yourcompanyname.division*
* *What about other languages src/main/groovy* 
  + *Src/main/groovy*
* *What about testing src/test/java* 
  + *Src/test/java*

***Target***

* *Where everything gets compiled to*
* *Also where tests get ran from*
* *Contents in this directory get packaged into a jar, war, ear, etc…*

***POM.xml***

* 1. 

***Can be divided into 4 basic parts #***

* ***Project Information*** 
  + *groupId*
  + *artifactId*
  + *version*
  + *packaging*
* ***Dependencies*** 
  + *Direct dependencies used in our application*
* ***Build*** 
  + *Plugins*
  + *Directory Structure*
* ***Repositories*** 
  + *Where we download the artifacts from*

***Dependencies***

* *Versions*
* *Types*
* *Transitive Dependencies*
* *Scopes*
* *What we want to use in our application*
* *Dependencies are imported by their naming convention* 
  + *Often considered the most confusing part of Maven*
* *We have to know the groupId, artifactId, and the version of what we are looking for*
* *Added to a dependencies section to our pom file*
* *Just list the dependency that we want* 
  + *Transitive dependencies will be pulled in by Maven*
* *Need at a minimum 3 things:* 
  + *groupId*
  + *artifactId*
  + *version*



***Versions***

* *Development starts off as a SNAPSHOT*
  + *myapp-1.0-SNAPSHOT.jar*
  + *Changes always downloaded*
  + *Saves you from rereleasing versions for development*
  + *Never deploy to production with a SNAPSHOT*
* *A release doesn’t have a specific naming convention* 
  + *myapp-1.0.jar*
  + *myapp-1.0.1.jar*
* *Industry common terms, but don’t affect maven* 
  + *myapp-1.0-M1.jar (milestone release)*
  + *myapp-1.0-RC1.jar (release candidate)*
  + *myapp-1.0-RELEASE.jar (release)*
  + *myapp-1.0-Final.jar (release)*

***Types***

* *Current core packaging types are:* 
  + *pom, jar, maven-plugin, ejb, war, ear, rar, par*
  + *The default packaging type is jar*
* *The type of pom is referred to as a dependency pom* 
  + *Downloads dependencies from that pom*



***Transitive Dependencies***

* *The main reason people begin using maven*
* *If we add a dependency:*



* *If we add a dependency it downloads it’s transitive dependencies:*



***Scopes***

* *There are 6 scopes available for dependencies:* 
  + ***compile*** *# default scope, artifacts available everywhere*
  + ***provided*** *# like compile, means that the artifact is going to be provided where it is deployed* 
    - *servlet-api.jar*
    - *xml-apis*
    - *Available in all phases, but not included in final artifact*
  + ***runtime*** *# not needed for compilation, but needed for execution* 
    - *Not available for compilation, but included in all other phases*
    - *Not included in final artifact*
  + ***test*** *# only available for the test compilation and execution phase*
  + ***system*** *# similar to provided, you specify a path to the jar on your file system* 
    - *Very brittle and defeats the purpose of maven, don’t use!*
  + ***import*** *# advanced topic, deals with dependencyManagement sections*

***pom.xml with our new dependency***



***Goals***

***Clean*** *# Delete the target directory and any generated resources*

***Compile*** *# compiles all sources code, generate any files, copies resources to our class’s directory*

***Package*** *# runs the compile command first, run any tests, package the app based off of its packaging type*

***Install*** *# runs the package command and then installs it in your local repo*

***Deploy*** *# runs the install command and then deploys it to a corporate repo*

***Repositories***

***Local Repo***

* *Where Maven stores everything it downloads* 
  + *Installs in your home directory\.m2* 
    - *C:\Users\<yourusername>\.m2\repository*
* *Stores artifacts using the information that you provided for artifactId, groupId, and version* 
  + *C:\Users\<yourusername>\.m2\repository\commons-lang\commons-lang\2.1\commons-lang-2.1.jar*
* *Avoids duplication by copying it in every project and storing it in your SCM*



* *Simply just a http accessible location that you download files from*
* *Super pom.xml* 
  + *Default with the Maven installation*
* *Default location* 
  + *http://repo.maven.apache.org/maven2*
* *Multiple repositories allowed*
* *Corporate Repository* 
  + *Nexus (this is what the default repo is built on)*
  + *Artifactory*

***Dependency Repository***

* *Where we download all of our dependencies from* 
  + *Can contain just releases and/or snapshots*
  + *Not uncommon to have them in separate repositories*
* *How do we specify our own repository*



***Plugin Repository***

* *Identical to Dependency Repositories, just deals with Plugins*
* *Will only look for Plugins, by design usually a separate repository*



***Releases / Snapshots***

* *Snapshots and releases can come from the same repo*
* *Why would projects not upload everything to the central repo* 
  + *Snapshots*
  + *Milestones*
  + *Release Candidate*
  + *Release policies*



***Notes***

* *Dependency repositories and Plugin repositories can be separate or the same repository*
* *Projects will often not upload their SNAPSHOT code up to the central repo even though their release project is hosted there*
* *Plugins are usually in the same repo as dependencies*
* *Companies should use a corporate repository internally to help lighten the load on the central repo*

***Plugins***

***Outline***

* *Goals*
* *Phases*
* *Compiler plugin*
* *Jar plugin*
* *Sources plugin*
* *Javadoc plugin*

***Goals***

* *The default goals are plugins configured in the maven install*
* *Clean, compile, test, package, install, deploy*
* *Super pom has these goals defined in it, which are added to your effective pom:*



* *Goals are tied to a phase*

***Phases***

* ***Validate #*** *Validate the project is correct and all necessary information is available*
* ***Compile #****Compile the source code of the project*
* ***Test #*** *Test the compiled source code*
* ***Package #*** *Packages the code in its defined package, such as a JAR*
* ***Integration-test #****Deploy and run integration tests*
* ***Verify #*** *Run checks against package to verify integrity*
* ***Install #*** *Install the package in our local repo*
* ***Deploy #*** *Copy final package to a remote repository*

***Compiler Plugin***

* *Used to compile code and test code*
* *http://maven.apache.org/plugins/maven-compiler-plugin/index.html*
* *Invokes Javac, but with the classpath set from the dependencies*
* *Defaults to Java 1.5 regardless of what JDK is installed*
* *Configuration section allows customization Includes/Excludes* 
  + *Fork*
  + *Memory*
  + *Source/target*



***Jar Plugin***

* *Used to package code into a jar*
* *http://maven.apache.org/plugins/maven-jar-plugin/index.html*
* *Tied to the package phase*
* *Configuration section allows customization* 
  + *Includes/Excludes*
  + *Manifest*



***Source Plugin***

* *Used to attach source code to a jar*
* *http://maven.apache.org/plugins/maven-source-plugin/index.html*
* *Tied to the package phase* 
  + *Often overridden to a later phase*



***Javadoc Plugin***

* *Used to attach Javadocs to a jar*
* *http://maven.apache.org/plugins/maven-javadoc-plugin/index.html*
* *Tied to the package phase* 
  + *Often overridden to a later phase*
* *Usually just use the defaults, but many customization options for Javadoc format*



***Summary***

* *Goals are really just configured plugins in your application*
* *Plugins are tied to one of the defined phases, but can usually be overridden.*
* *The compile plugin is already defined for you, but is often changed to use a specific version of Java.*
* *The jar plugin is one of the default plugins and can be configured to produce artifacts to specific needs.*
* *Source and Javadocs can easily be generated to be installed in your corporate repository for use by other developers.*

***Eclipse Integration***

***Outline***

* *Installing Eclipse/Spring STS*
* *Importing Maven Projects*
* *Pom viewer*
* *Dependency Overview*
* *Adding a dependency*
* *Dependency Heirarchy*
* *Effective Pom*

***Installation***

* *Eclipse/Spring STS doesn’t use the registry*
* *Java and Maven installed the same regardless of using an IDE*
* *Some IDEs do include a bundled version of Maven*
* *http://www.springsource.org/downloads/sts-ggts*

***Importing Maven Projects***

* *Modern IDEs have Maven integration built into them*
* *Maven integration will allow us to execute default maven goals within our IDE*
* *IDE configuration and Classpath will be set from Maven*
* *Right Click in the Package Explorer > Import > Maven > Existing Maven Projects*



***Converting Existing Projects***

* *If you have a pom.xml file, you can convert the project to a Maven project*
* *Right click on the project containing the pom.xml file > Configure > Convert to Maven Project*



* *Once converted the project will set the classpath and automatically build the project*

***Pom Viewer***

* *Default view when you open the pom file*
* *Pom overview shows the high level elements of your project*
* *Changes made here are directly changing the source*



***Dependencies***

* *Shows which dependencies we have installed and allows us to manipulate dependencies too*
* *Dependency Management (advanced topic) is also displayed*
* *The add screen has searching capability*



***Dependency Heirarchy***

* *Displays the complete dependency tree, including transitive dependencies as well overridden dependencies*
* *Scope of the resource is also displayed*



***Effective Pom***

* *The complete pom with everything inherited from the project pom, if we have a parent pom, and the default super pom*
* *More of a debugging tool to see what the pom is doing*



***Summary***

* *Eclipse/STS installation is really unzipping*
* *Existing projects can be imported easily and converting projects can be easily as well*
* *Adding dependencies inside the IDE can be easier using the searching tools*
* *Solving dependency resolution errors is more convenient in the IDE*
* *Configuring your IDE is also more convenient with Maven*

***Notes*** *Backlog Completed*

<scm>

<connection>scm:git:https://github.allstate.com/ConnectedCarProductEngineering/DriveWise-BatchMonitor</connection>

</scm>

<!-- Utilities -->

<dependency>

<groupId>cglib</groupId>

<artifactId>cglib</artifactId>

</dependency>

<dependency>

<groupId>dom4j</groupId>

<artifactId>dom4j</artifactId>

</dependency>

<dependency>

<groupId>org.codehaus.castor</groupId>

<artifactId>castor-core</artifactId>

</dependency>

<dependency>

<groupId>org.codehaus.castor</groupId>

<artifactId>castor-xml</artifactId>

</dependency>

<dependency>

<groupId>org.codehaus.castor</groupId>

<artifactId>castor-xml-schema</artifactId>

</dependency>

<dependency>

<groupId>net.sf.ehcache</groupId>

<artifactId>ehcache-core</artifactId>

</dependency>

<dependency>

<groupId>com.sun.mail</groupId>

<artifactId>javax.mail</artifactId>

</dependency>

<dependency>

<groupId>net.sourceforge.jexcelapi</groupId>

<artifactId>jxl</artifactId>

</dependency>

<dependency>

<groupId>net.sf.opencsv</groupId>

<artifactId>opencsv</artifactId>

</dependency>

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi</artifactId>

</dependency>

<dependency>

<groupId>org.antlr</groupId>

<artifactId>stringtemplate</artifactId>

</dependency>

<dependency>

<groupId>org.javassist</groupId>

<artifactId>javassist</artifactId>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<dependency>

<groupId>taglibs</groupId>

<artifactId>standard</artifactId>

</dependency>

**Points**

Maven will pick one version using the *"nearest in the dependency tree"* strategy.

**Commands**

To create new project# mvn archetype:generate

To import project to eclipse # eclipseWorkspace\yourproject > mvn eclipse:eclipse

***Assessment***

*which goal would you use with the dependency plugin to determine which include used as well as those you are using that you haven’t defined?*

* *dependency:evaluate*
* *dependency:analyze*
* *dependency:properties*
* *dependency:tree*

*A compile time dependency of a dependency for your project, is often called a\_\_\_\_\_dependency*

* *provided*
* *runtime*
* *test*
* *transitive*

*what is the name of the artifact that apache maven utilizes to perform its build operation?*

* *Ant files*
* *plug-ins*
* *properties*
* *configuration*

*what maven construct is used to build a template project structure that you can then leverage when building new maven-based java project*

* *build*
* *Template*
* *Starter*
* *Archetype*

*when two dependencies of your maven project introduce the same transitive dependency versions, which one(s) will be included in the dependency list for your projects*

* *the dependency with higher versions*
* *the transitive dependency version of the artifact that is closest to your project*
* *the transitive dependency that comes from the dependency listed first in the POM.xml file*
* *the transitive dependency that comes from the dependency listed last in the POM.xml*

*what is the default scope for maven dependencies?*

* *Test*
* *build*
* *compile*
* *runtime*