MAVEN STS/ECLIPSE INTEGRATION

INTRODUCTION :

* In this module , we will cover Spring STS , eclipse integration with Maven.
* Most modern IDE’s have integration with Maven and can be configured using Maven
* Installing Spring STS which is a flavor of Eclipse
* Converting existing projects to Maven
* Importing projects using Maven.
* POM viewer , dependency overview tool , dependency hierarchy , Effective POM inside application

INSTALLTION :

* Spring STS suite and Eclipse , they don’t use the registry so direct download and extract to use.
* Some IDE’s use a bundled version of Maven.
* To download Spring STS – <http://www.springsource.org/downloads/sts-ggts>

DEMO INSTALLATION :

* Once we have Java and Maven already installed, we need to download a IDE.
* Go to the above link mentioned in the previous section.
* Go and download either one , a ZIP extension is more easier than an EXE extension cause its just a bundled version and will provide the ZIP.
* Extract the version to a specific path on your local disk.
* Open STS.exe that launches the Spring tool suite.
* It doesn’t use the registry , so I can transfer it to an external HDD or give it to a friend and it would work properly.
* Launch Spring STS , it will ask for the workspace , C:/dev/workspace , your choice.

IMPORTING PROJECTS :

* Importing existing Maven projects is really quite easy.
* Most IDE’s have Maven installed within them and allows us to already run some particular goals that come with it.
* IDE configuration and classpath will be set from Maven.
* Locate the project on the disk
* RC on package explorer -> Maven -> Import existing Maven project.
* Imports all of our sourcecode and sets our classpath.
* Open Spring STS and do the above , It should recognize the POM.xml file within the existing Maven project that we have.
* Click on Finish.
* It will build the workspace.
* The icon on SRC/MAIN/JAVA -> Says that it contains java files.
* It shows our test and resources directory
* It also shows our Maven Dependencies and the JRE System Library.
* It also shows the POM.xml file and it has the dependencies within it.

CONVERTING EXISTING PROJECTS :

* Converting existing projects , is also similar to importing existing projects.
* If we have a POM file we can convert it to Maven Project.
* RC on the project which contains the POM file -> Configure -> Convert to Maven project.
* Once the project is converted the project will set the classpath and automatically build the project and it will show the errors.

DEMO :

* Create a sample project called Foo.
* It’s a skeleton project.
* The HelloWorld project has an M icon above it where as Foo doesn’t.
* Foo has a pom.xml within it , which is empty and has only the groupId , artifactId and version.
* RC on the Foo project -> Configure -> Convert to Maven Project.
* Once we convert it to Maven , the M symbol has come over the project name.
* Maven defaults the Java version to 1.5 , its telling eclipse to use 1.5 for this application.

POM VIEWER & DEMO:

* The POM viewer is the default view when you open the POM file.
* It gives the POM overview which are the high level elements of the project.
* Not super useful other than the name.
* Any changes made here are directly changing the source.
* Things that we can change are the groupId , artifactId , version and the packaging , most people generally go to the source code.
* Open the newly created application Foo.
* Double-click on the POM file to see the POM viewer.
* It’s the Meta-Info and change the groupId and see the changes in the source.
* Change it in the source pom.xml and again see that it reflects in the POM overview.

DEPENDENCY VIEWER AND DEMO :

* Dependencies are more easily added in the IDE.
* Dependencies tab is very useful , dependency management tab is there only when we are using parent pom.
* The add screen has a searching capability , which makes it a nice feature and gives us a dropdown when we type a keyword and adds the dependency to our project.
* Open up the POM.xml
* It will open the POM viewer and we have the dependency tab.
* Click on add.
* Lets add Junit to our application.
* The center box allows me to search , we can assign the correct scope also , it searches a local repository which we have.
* It will add it to our POM.xml
* We can see that it has pulled that dependency in.
* If I want to add one of the dependency by hand then we can see that it has also added it in the dependency viewer once we save the file which has the default scope present for it.

DEPENDENCY HEIRARCHY AND DEMO :

* The dependency hierarchy view , it’s the most beneficial view inside our IDE.
* It shows the transitive dependencies as well as the conflict of dependency if any.
* Scope is also displayed.
* There is nothing to show if we have just junit or commons-lang
* It can help us troubleshoot the application.
* Right now it appears as though we don’t have any transitive dependencies.
* Junit has a hamcrest core as the transitive dependency.
* Add the hibernate-core dependency
* Now look at the dependency hierarchy to see all the transitive dependency that it has taken once the file is saved.
* We can also see the conflict of a dependency jboss-logging with two versions CR2 and GA when the IDE automatically took the GA version which is the final release over the CR2 version.

EFFECTIVE POM AND DEMO :

* The effective POM can be a very wordy document , it is the POM underneath our hood , everything inherited from the parent POM and the super POM.
* More of a debugging tool to see what the POM is doing.
* We will see an example of an effective POM , its very lengthy.
* It has all of our settings combined from all of our locations in it.
* Click on POM.xml for the Foo project.
* We have three dependencies and our group information.
* We know that Maven when it installed has the super POM.
* That defines the default location.
* It builds this Effective POM , look on the tabs to the bottom to find it.
* It has all of the information like defaults , plugins , repositories that we are using inside our application.
* It has all our three dependencies.
* We can find the central repository link within the repository configure URL tag.
* Snapshots are false.
* We can also see the pluginRepository.
* The source and target location for our code in our workspace.
* The default goals , run , clean and compile.
* Pulls everything into one , what our Maven installation is running.

SUMMARY :

* In this module , we saw the integration with IDEs.
* Unzipping a bundled application.
* Adding dependencies inside IDE is easier cause of searching tools.
* Existing projects and converting projects.
* Solving dependency resolution conflict is easier in IDE.
* Configuring IDE is lot more convenient with Maven.