MAVEN - PLUG-INS

What are Maven Plugins?

Maven is actually a plugin execution framework where every task is actually done by plugins. Maven Plugins are generally used to :

- create jar file
- create war file
- compile code files
- unit testing of code
- create project documentation
- create project reports

A plugin generally provides a set of goals and which can be executed using following syntax:

```
mvn [plugin-name]:[goal-name]
```

For example, a Java project can be compiled with the maven-compiler-plugin's compile-goal by running following command

```
mvn compiler:compile
```

Plugin Types

Maven provided following two types of Plugins:

Type	Description
Build plugins	They execute during the build and should be configured in the <build/> element of pom.xml
Reporting plugins	They execute during the site generation and they should be configured in the <reporting/> element of the pom.xml

Following is the list of few common plugins:

Plugin	Description
clean	Clean up target after the build. Deletes the target directory.
compiler	Compiles Java source files.
surefile	Run the JUnit unit tests. Creates test reports.
jar	Builds a JAR file from the current project.
war	Builds a WAR file from the current project.
javadoc	Generates Javadoc for the project.
antrun	Runs a set of ant tasks from any phase mentioned of the build.

Example

We've used **maven-antrun-plugin** extensively in our examples to print data on console. See Maven Build Profiles chapter. Let to understand it in a better way let's create a pom.xml in C:\MVN\project folder.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.companyname.projectgroup</groupId>
<artifactId>project</artifactId>
<version>1.0</version>
<build>
<plugins>
   <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-antrun-plugin</artifactId>
   <version>1.1</version>
   <executions>
      <execution>
         <id>id.clean</id>
         <phase>clean</phase>
         <goals>
            <goal>run</goal>
         </goals>
         <configuration>
            <tasks>
               <echo>clean phase</echo>
            </tasks>
         </configuration>
      </execution>
   </executions>
   </plugin>
</plugins>
</build>
</project>
```

Next, open command console and go to the folder containing pom.xml and execute the following **mvn** command.

```
C:\MVN\project>mvn clean
```

Maven will start processing and display clean phase of clean life cycle

```
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------
[INFO] Building Unnamed - com.companyname.projectgroup:project:jar:1.0
[INFO]    task-segment: [post-clean]
[INFO] ------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] [antrun:run {execution: id.clean}]
[INFO] Executing tasks
     [echo] clean phase
[INFO] Executed tasks
[INFO] ------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------
[INFO] Total time: < 1 second
[INFO] Finished at: Sat Jul 07 13:38:59 IST 2012
[INFO] Final Memory: 4M/44M
[INFO] ------------------------------------------------------------------
```

The above example illustrates the following key concepts:

- Plugins are specified in pom.xml using plugins element.

- Each plugin can have multiple goals.
- You can define phase from where plugin should starts its processing using its phase element. We've used **clean** phase.
- You can configure tasks to be executed by binding them to goals of plugin. We've bound **echo** task with **run** goal of *maven-antrun-plugin*.
- That's it, Maven will handle the rest. It will download the plugin if not available in local repository and starts its processing.