



Using Maven2 to Get Control over Your Development Process

Hermod Opstvedt

Chief Architect

DnB NOR ITU





Using Maven2 to Get Control over Your Development Process

- What is Maven?
 - It's a kind of Project Management Tool.
 - Focus on standards and best practices.
 - Convention over configuration.
 - Shared language for build management.
 - Repeatable and robust builds.
 - Central point is the Project Object Model (POM).



Using Maven2 to Get Control over Your Development Process

- History

- Maven is a result of trying to make several Apache Software Foundation (ASF) projects work in the same predictable way.
- Started out in 2002.
- Came as an offspring of the ASF Turbine project.
- Initiated by Jason Van Zyl.



Using Maven2 to Get Control over Your Development Process

- Objectives

- Create a common and understandable build infrastructure

- If you know Maven, moving from one project to the next means you can focus on the problem at hand and not spend time trying to understand how it is built.
- Being able to build the same artifact version over and over again, regardless of changes to it over time.



Using Maven2 to Get Control over Your Development Process

- Common problems in a project
 - Traditionally, the larger the project, the more complex and less understandable the build process or build infrastructure gets.
 - Bringing in a new team member often meant spending hours, even days of introduction to and trying to understand how various parts of the project is build.



Using Maven2 to Get Control over Your Development Process

- Very often the build system is copied from a previous project that the senior developers participated in.
- Setting up an initial project through copy and paste means that hard to find errors often are introduced – “It worked the last time, why not now?”



Using Maven2 to Get Control over Your Development Process

- Versioning of the project often becomes a nightmare if not a strict regime of code management is introduced.
- Dependencies on other libraries are hard to track, and often leads to problems when trying to rebuild a given version.
 - Ex: myexternallibrary.jar
 - ✓ No clue as to which version it is
 - ✓ No mention of version in manifest file



Using Maven2 to Get Control over Your Development Process

- Documentation process is often "put off till later".
- Testing infrastructure is at best mediocre.



Using Maven2 to Get Control over Your Development Process

- So how can Maven help us?
 - Common build logic
 - Separation of concerns
 - Standard naming convention
 - Directories
 - Project output
 - All build output is automatically identified with version number.
 - ✓ Ex: myjar-1.0.jar



Using Maven2 to Get Control over Your Development Process

- Single output from a single Maven project
 - *i.e.* the resulting output from a build is a single artifact
 - A Maven project may consist of several subprojects each producing a single output.



Using Maven2 to Get Control over Your Development Process

- Maven standard Build Lifecycle
 - Validate: validate the project is correct and all necessary information is available.
 - Generate-sources: generate any source code for inclusion in compilation.
 - Process-sources: process the source code, for example to filter any values.
 - Generate-resources: generate resources for inclusion in the package.



Using Maven2 to Get Control over Your Development Process

- **Process-resources:** copy and process the resources into the destination directory, ready for packaging.
- **Compile:** compile the source code of the project.
- **Process-classes:** post-process the generated files from compilation, for example to do bytecode enhancement on Java classes.
- **Generate-test-sources:** generate any test source code for inclusion in compilation.



Using Maven2 to Get Control over Your Development Process

- **Process-test-sources:** process the test source code, for example to filter any values.
- **Generate-test-resources:** create resources for testing.
- **Process-test-resources:** copy and process the resources into the test destination directory.
- **Test-compile:** compile the test source code into the test destination directory



Using Maven2 to Get Control over Your Development Process

- **Test:** run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
- **Package:** take the compiled code and package it in its distributable format, such as a JAR.
- **Pre-integration-test:** perform actions required before integration tests are executed. This may involve things such as setting up the required environment.



Using Maven2 to Get Control over Your Development Process

- Integration-test: process and deploy the package if necessary into an environment where integration tests can be run.
- Post-integration-test: perform actions required after integration tests have been executed. This may including cleaning up the environment.
- Verify: run any checks to verify the package is valid and meets quality criteria.

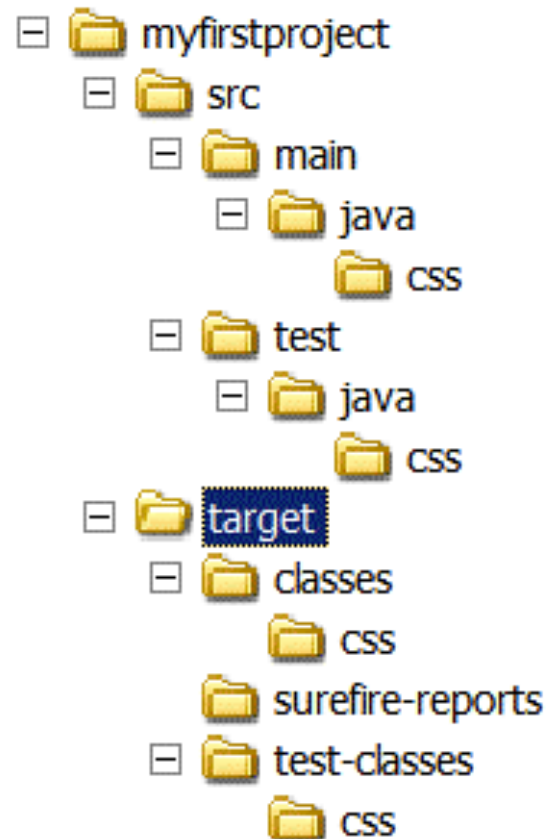


Using Maven2 to Get Control over Your Development Process

- **Install:** install the package into the local repository, for use as a dependency in other projects locally.
- **Deploy:** done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

Using Maven2 to Get Control over Your Development Process

- Standard directory structure:





Using Maven2 to Get Control over Your Development Process

- Installing Maven
 - Download from <http://maven.apache.org>
 - Unzip maven-xxx-bin.zip to the directory you wish to install Maven.
 - Add the bin directory to your path.
 - Run *mvn -version* to verify installation:
 - Should reply with: Maven version: x.x.x

Using Maven2 to Get Control over Your Development Process

- Create a file: settings.xml in your home directory.

➤ **Windows:** <drive>:\Documents and Settings\<<user>\.m2

```
<settings>
<profiles>
<profile>
  <id>Snapshots</id>
  <repositories>
  <repository>
    <id>Maven Snapshots</id>
    <url>http://snapshots.maven.codehaus.org/maven2/</url>
    <snapshots>
      <enabled>>true</enabled>
    </snapshots>
    <releases>
      <enabled>>false</enabled>
    </releases>
  </repository>
</repositories>
</profile>
</profiles>
</settings>
```



Using Maven2 to Get Control over Your Development Process

```
<pluginRepositories>
  <pluginRepository>
    <id>Maven Snapshots</id>
    <url>http://snapshots.maven.codehaus.org/maven2/</url>
    <snapshots>
      <enabled>>true</enabled>
    </snapshots>
    <releases>
      <enabled>>false</enabled>
    </releases>
  </pluginRepository>
</pluginRepositories>
</profile>
</profiles>
<activeProfiles>
  <activeProfile>Snapshots</activeProfile>
</activeProfiles>
</settings>
```



Using Maven2 to Get Control over Your Development Process

- Settings.xml controls your build environment
 - Proxies
 - Repository locations
 - Profiles (more on that later)



Using Maven2 to Get Control over Your Development Process

- Archetypes

- What's an archetype?

- Template of a project which is combined with some user input to produce a working Maven project that has been tailored to your requirements.
- Used to initialize a new project of a particular kind.



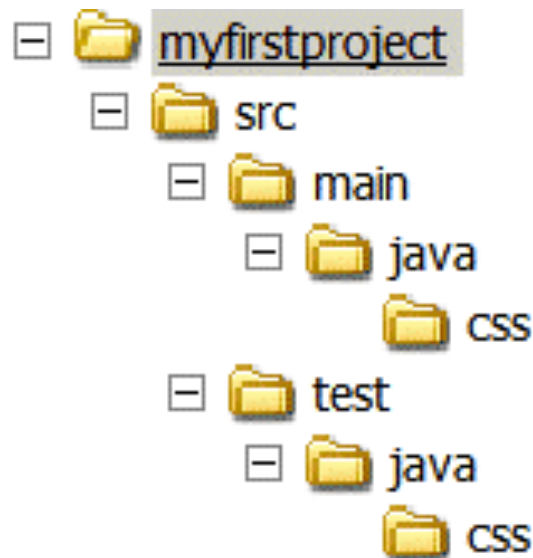
Using Maven2 to Get Control over Your Development Process

➤ Several archetypes available:

- j2ee-simple
- marmalade-mojo
- mojo
- portlet
- profiles
- quickstart
- simple
- site-simple
- site
- Webapp
- *etc.*

Using Maven2 to Get Control over Your Development Process

- Running an archetype:
 - `mvn archetype:create -DgroupId=css -DartifactId=myfirstproject`
 - Creates a simple standard Maven project:





Using Maven2 to Get Control over Your Development Process

- One directory containing source and one directory containing test source.
 - src/main/java/<package>
 - src/test/java/<package>



Using Maven2 to Get Control over Your Development Process

➤ POM file for the project:

```
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/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>css</groupId>
  <artifactId>myfirstproject</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```



Using Maven2 to Get Control over Your Development Process

- Contains a sample source java file and a sample test java file
 - App.java
 - AppTest.java

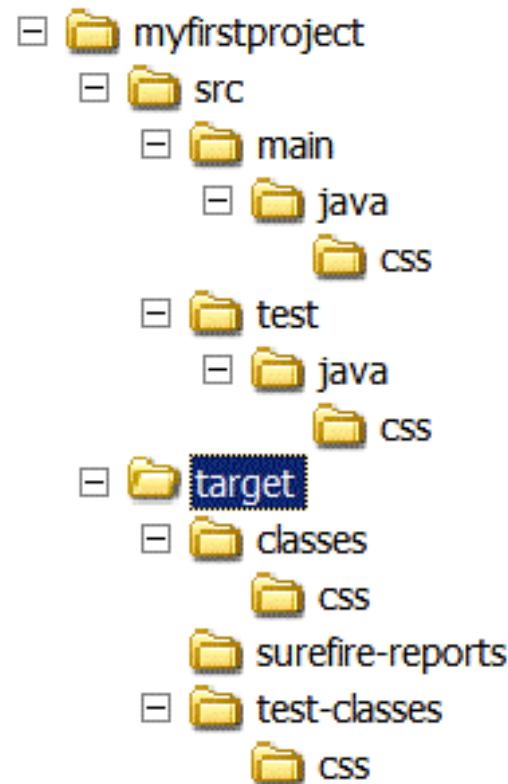


Using Maven2 to Get Control over Your Development Process

- Running Maven to build the simple sample:
 - mvn clean install
 - Clean ensures that any previously build artifacts are removed prior to the execution
 - Install tells Maven to go through the lifecycles and build the single project artifact.

Using Maven2 to Get Control over Your Development Process

- Produces a standard single artifact, and creates a standard output directory structure.





Using Maven2 to Get Control over Your Development Process

- After execution the result is:
 - myfirstproject-1.0-SNAPSHOT.jar



Using Maven2 to Get Control over Your Development Process

- Creating your own archetype:
 - Create a directory to contain your archetype
 - Then create a pom for the archetype in this directory:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>css2006.maven</groupId>
  <artifactId>my-archetype</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>maven-plugin</packaging>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```



Using Maven2 to Get Control over Your Development Process

- Then create a new directory:
 - *src/main/resources/META-INF/*
- Create an archetype descriptor in it.
 - Archetype.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<archetype>
  <id>css2006-archetype</id>
  <resources>
    <resource>src/main/java/css2006/maven/archetype1/StdBean.java</resource>
    <resource>src/main/test/css2006/maven/test/archetype1/TestStdBean.java</resource>
  </resources>
</archetype>
```



Using Maven2 to Get Control over Your Development Process

- Create a directory a new directory that will contain the resources that should be standard
 - `src/main/resources/archetype-resources`
- Then create a standard Maven project structure under that, which will contain all the resources that should be part of you standard setup:
 - `src/main/resources/archetype-resources/src/main/java`
 - `src/main/resources/archetype-resources/src/main/test`
 - `src/main/resources/archetype-resources/src/main/resources`
 - *etc.*



Using Maven2 to Get Control over Your Development Process

- Under the java directory we create a package structure for our standard classes and the classes themselves:
 - `src/main/resources/archetype-resources/src/main/java/css2006/maven/archetype1`
 - ✓ `StdBean.java`

- Under the test directory we create a package structure for our standard classes and the test classes themselves:
 - `src/main/resources/archetype-resources/src/main/test/css2006/maven/test/archetype1`
 - ✓ `TestStdBean.java`

Using Maven2 to Get Control over Your Development Process

➤ So now we have a directory structure as such:





Using Maven2 to Get Control over Your Development Process

➤ StdBean.java

```
package css2006.maven.archetype1;
public class StdBean {

    public StdBean() {
        super();
    }

    public String sayHello(String name) {
        if (name != null) {
            return "Hello" + name;
        }
        return "Who are you?";
    }
}
```

Using Maven2 to Get Control over Your Development Process

➤ TestStdBean.java

```
package css2006.maven.test.archetype1;
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;
import css2006.maven.archetype1.StdBean;
public class TestStdBean extends TestCase {
    public static Test suite()
    {
        return new TestSuite( TestStdBean.class );
    }
    public void testSayHello()
    {
        StdBean sb=new StdBean();
        String retval=sb.sayHello("CSS2006");
        assertEquals(retval,"Hello:CSS2006");
    }
}
```



Using Maven2 to Get Control over Your Development Process

- Augmenting the archetype
 - Resources
 - Property files
 - *etc.*
 - Site resources for the generated site
 - Stylesheets
 - Images
 - *etc.*



Using Maven2 to Get Control over Your Development Process

- A little more complex archetype: multiproject
- Now we will create a archetype that contains a main project and two subprojects
 - A web project
 - A utility jar project
- We repeat the process from the previous archetype, but make some differences to the master pom for the project to be created.



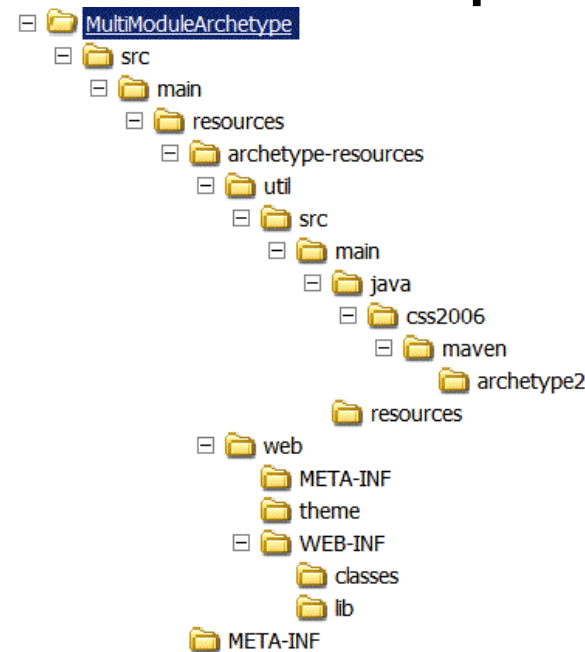
Using Maven2 to Get Control over Your Development Process

- The revised project master pom:

```
<?xml version="1.0" encoding="UTF-8"?>
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>css2006.maven</groupId>
  <version>1.0</version>
  <artifactId>projectroot</artifactId>
  <packaging>pom</packaging>
  <name>CSS2006 Multimodule project</name>
  <modules>
    <module>util</module>
    <module>web</module>
  </modules>
</project>
```

Using Maven2 to Get Control over Your Development Process

- We added the modules definition section for the two modules that we want to be created.
- Remember: What we put under archetype-resources is a mirror of the Maven project we want create.





Using Maven2 to Get Control over Your Development Process

- Complex archetypes – *i.e.* execute something
 - Can not be done using the regular way
 - Archetypes are simple in nature – they do not execute code.
 - Lets say that you have a standard project structure in your organization. If you want to be able to introduce a new team member into a project as painless as possible, you can do the following:



Using Maven2 to Get Control over Your Development Process

- Start out by creating an archetype for your standard structure.
- Then in the pom for the project, you add a scm section that will point to where your code is stored.
- Then first run the `mvn archetype:create`
- Change to the root of your newly created project and then do: *mvn install*



Using Maven2 to Get Control over Your Development Process

- An alternative method is to create an Ant based mojo.
 - A mojo is a Maven plugin that executes some code at a given lifecycle.
 - It can run Java code, Ant scripts and more
- The reason for using an Ant based mojo is that to be able to run more than one task in a mojo we need to use what is known as the Embedder.



Using Maven2 to Get Control over Your Development Process

- The Embedder can however not be run as a mojo – hence we introduce Ant to help us. With the Ant script we are able to fork a new Java process in a new environment, thereby circumventing the mojo/Embedder limitation.



Using Maven2 to Get Control over Your Development Process

- We start by creating our mojo, using the mojo archetype available to us
- Then add a src/main/scripts directory
- In this directory we create our Ant script – `css2006build.xml`
- Then we add our execution script to this



Using Maven2 to Get Control over Your Development Process

- Script

```
<project default="MvnRun" xmlns:artifact="antlib:org.apache.maven.artifact.ant">
  <target name="MvnRun">
    <java jvm="${env.JAVA_HOME}/java/bin/java.exe" fork="true"
      classname="css2006.maven.embedder.MvnRun" newenvironment="true" dir=".">
    </java>
  </target>
</project>
```



Using Maven2 to Get Control over Your Development Process

- Then we need to create a `css2006build.mojos.xml` file which is a description of our Ant mojo.

```
<pluginMetadata>
  <mojos>
    <mojo>
      <goal>MvnRun</goal>
      <call>MvnRun</call>
      <requiresDependencyResolution>runtime</requiresDependencyResolution>
      <description>Run maven goals</description>
    </mojo>
  </mojos>
</pluginMetadata>
```



Using Maven2 to Get Control over Your Development Process

- Then we need to create the java file that we want to execute when the Ant script is run.



Using Maven2 to Get Control over Your Development Process

```
public class MvnRun extends AbstractMojo{
    public MvnRun()
    {
        super();
    }

    public void execute() throws MojoExecutionException, MojoFailureException {
        MavenEmbedder embedder = new MavenEmbedder();
        embedder.setClassLoader(Thread.currentThread().getContextClassLoader());
        embedder.setLogger(new MavenEmbedderConsoleLogger());
        try {
            embedder.start();
        } catch (MavenEmbedderException mee) {
            throw new MojoExecutionException("Embedder",mee);
        }
    }
}
```



Using Maven2 to Get Control over Your Development Process

```
Settings settings;  
try {  
    settings = embedder.buildSettings(embedder  
        .getUserSettingsPath(null), embedder.getGlobalSettingsPath(),  
        false, false, false, Boolean.FALSE);  
    } catch (SettingsConfigurationException sce) {  
        throw new MojoExecutionException("Embedder", sce);  
    }  
}
```

...

```
List goals = new ArrayList();  
goals.add("archetype:create");
```



Using Maven2 to Get Control over Your Development Process

```
MavenExecutionRequest request = new DefaultMavenExecutionRequest()
    .setBasedir(new File(".")).setGoals(goals)
    .setLocalRepositoryPath(
embedder.getLocalRepositoryPath(settings)).setSettings(
    settings).setProperties(main_ props).addEventMonitor(
    new DefaultEventMonitor(new ConsoleLogger(
    ConsoleLogger.LEVEL_DISABLED, "logger")));

try {
    embedder.execute(request);
} catch (MavenExecutionException mee) {
    throw new MojoExecutionException("Embedder",mee);
}
```

...



Using Maven2 to Get Control over Your Development Process

```
goals.clear();
goals.add("scm:checkout");
request = new DefaultMavenExecutionRequest().setPomFile(pomFile)
        .setBasedir(new File(pomFile)).setGoals(goals)
        .setLocalRepositoryPath(
embedder.getLocalRepositoryPath(settings)).setSettings(settings).setProperties(main_props).addEventMonitor(
new DefaultEventMonitor(new ConsoleLogger(ConsoleLogger.LEVEL_DISABLED, "logger")));
try {
    embedder.execute(request);
} catch (MavenExecutionException mee) {
    throw new MojoExecutionException("Embedder",mee);
}
...
}
```



Using Maven2 to Get Control over Your Development Process

- We create this Java file in a project of its own, and we declare a dependency on it in our Ant based mojo pom.
- We now install both projects into the repository and we are ready to go.
- Next we alter the project pom for our simple archetype.



Using Maven2 to Get Control over Your Development Process

```
...  
<scm>  
    <connection>scm:svn:http://localhost/css2006/simple/</connection>  
    <developerConnection>  
        scm:svn::http://localhost/css2006/simple/  
    </developerConnection>  
    <url:http://localhost/css2006/simple/</url>  
</scm>  
...
```

- Here we have said that our source for this particular project is stored in Subversion

Using Maven2 to Get Control over Your Development Process

- Then we run the plugin:
 - `mvn css2006:mvnrun`
- We then get created our standard project, and then the project pom is run as a normal Maven run, with the goal `scm:checkout`.
- The result of this is that we now have a ready to go project for a new developer in the project with all the source/artifacts in place.



Using Maven2 to Get Control over Your Development Process

- Time permitting, some more samples.



Using Maven2 to Get Control over Your Development Process

■ References:

- <http://maven.apache.org>
- <http://people.apache.org/~oching/maven-archetype-plugin/>
- http://www.mergere.com/m2book_download.jsp
- <http://cvs.peopleware.be/training/maven/maven2/>



Using Maven2 to Get Control over Your Development Process

