Transitioning from Maven 1 to Maven 2 Hands on Tutorial

Transitioning from Maven 1 to Maven 2 project management

Igor Pavlin
Version 0.1
Date: December 1, 2007

This is a "hands on tutorial" for transitioning from Maven 1 to Maven 2 project management. It could be easily a large book, but I will try to keep things short and simple. If you do not understand something please refer to the mentioned literature or feel free to ask me.

For the purposes of this document I will be strict about names Maven 1.x and Maven 2.x, but later we should assume that unless otherwise specified, when we say maven it will mean Maven 2.x.

The reader is assumed to be completely familiar with Maven 1, and not so much with Maven 2. I recommend reading at least the User documentation http://maven.apache.org/.
I found book: **Better Builds with Maven** by Vincent Massol et al. (there is an electronic version available), very valuable. Published by DevZuz Library Press, April 2007.

In this hands on approach, I will convert one of the GLAST projects: **org-glast-groupmanager,** that was built with Maven 1, into a Maven 2 project. I will also assume that NetBeans 6.0 is used for building the project using Netbeans Maven 2 plugin. Initially I would test transitions using command line program mvn.

Install Maven 2 for your OS and set MVN_HOME, as well as create .m2 directory in your home(working) directory. Link is: http://maven.apache.org/

In Appendix One, I list typical GLAST Maven 1 project files and give a log of a successful 'maven' (Maven 1.x) run.

I will concentrate bellow only on the significant changes when transitioning from Maven 1 to Maven 2. The changes will be presented in tabular forms bellow.

1. Default Directory Structure:

The default directory structure is strongly encouraged, as this leads to better project management. Most of the time you do not need to change the Maven 1 directory
structure, as it usually follows the recommended default structure for Maven 2.

The default directory structure is actually defined in the Maven 2 "Super POM" file, which all Maven 2 POM objects inherit from. Here is the GLAST directory structure for Maven 2 projects:

artifactId/	**project directory**
|-- .cvsignore	contains files that are local to build: target, profiles.xml, etc
|-- pom.xml	POM file
|-- profiles.xml	local, user and project dependent profile definitions; this should not be version controlled
|-- LICENSE.txt	license of the project
|-- README.txt	welcome to the reader
|-- src/	original src material; this should be version controlled
|-- main/	the original material for the artifact
|-- java	root of main Java source; will be compiled into target/classe
|-- resources/	main Java resources; will be copied into target/classes
|-- webapp/	web application with standard web application structure
|-- WEB-INF/	
| '-- web.xml	
'-- index.jsp	
|-- assembly/	
'-- dep.xml	assembly descriptor for maven-assembly-plugin
|-- filters/	resource filter properties files for main Java resources
|-- config/	configuration files for the artifact
|-- bash/	sources in other technologies for the artifact
|-- python/	
|-- sql/	

|-- site/	project documentation in different formats; mvn site will produce a project website in target/site based on this material and structure (Doxia)
|-- fml/	documentation in FML format (XML based FAQ format)
'-- faq.fml	
|-- resources/	
|-- css/	
|-- img/	
|-- js/	site resources; will be copied into target/site as-is
|-- site.xml	site descriptor: description of site structure; this will generate menus
|-- xdoc/	
'-- xdoc.xml	documentation in Xdoc format (XML based HTML generation; maven 1 legacy)
|-- test/	original material to test the artifact
|-- java/	root of Java source for testing the artifact; usually JUnit test classes; will be compiled into target/test-classes
|-- resources/	resources for testing the artifact; will be copied into target/test-classes
|-- filters/	resource filter properties files for resources for testing the artifact
|-- python/	sources in other technologies for testing the artifact
|-- target/	generated material; this should not be under version control
|-- artifactId-version.jar	generated artifact
|-- classes/	result of compilation of src/main/java and copy of src/main/resources
|-- exported-pom.xml	consolidated POM
|-- javadoc/	javadoc of src/main/java
|-- site/	project site generated by mvn site
|-- surefire-reports/	test reports
|-- test-classes/	result of compilation of src/test/java and copy of src/test/resources
|-- announcement/	
'-- announcement.vm	org.codehaus.mojo:changelog-maven-plugin generates announcement mail here
|-- denotes subdirectory	'-- denotes file in the directory

(a) Where to begin?

For simpler projects one can simply start with converting project.xml to pom.xml (after saving all the project Maven1 build files into maven1 subdirectory.

For more complex projects the rule of thumb is to produce one artifact (JAR, WAR, etc.) per Maven project file. That means that it is prudent to create a maven2 (or m2) directory
where all the existing code is copied and than rearranged into several subprojects (modules). A parent pom.xml and subproject pom.xml files are than created and each module built separately. I will give an example later in section ... When you are finished with migration you can easily remove the old Maven 1 build.

In the m2 directory, you will need to create a parent POM. You will use the parent POM to store the common configuration settings that apply to all of the child modules. For example, each module will inherit the following values (settings) from the parent POM.
? groupId: this setting indicates your area of influence, company, department, project, etc., and it should mimic standard package naming conventions (for GLAST this is just 'glast' , but in general it will be com.yourcompany.projectname). Maven 2 will than install the files under /.m2/repository/glast or (/.m2 /repository/com/yourcompany/projectname)
? artifactId: the setting specifies the name of this module (for example, org-glast-groupmanager). The convention here at GLAST is that all the top projects should be prefixed with org-glast-
? version: this setting should always represent the next release version number appended with - SNAPSHOT - that is, the version you are developing in order to release. During the release process, Maven will convert to the definitive, non-snapshot version for a short period of time, in order to tag the release in your SCM.
? packaging: the jar, war, and ear values should be obvious to you (a pom value means
that this project is used for metadata only - when it is a parent project of several subprojects)

In this parent POM you can also add dependencies such as JUnit, which will be used for testing in every module, thereby eliminating the requirement to specify the dependency repeatedly across multiple modules.

The other possible addition to the parent POM files is specifying build properties that are common to all the modules, for example JDK settings, etc. This will be described later.

You do not have to memorize this directory structure. Most of the time you can use Maven archetypes to create prototypes for particular type of Maven 2 project. See bellow section on using archetypes.

2. Maven 2 project file pom.xml

The most reliable way to convert from project.xml to pom.xml is to follow the respective XML schema:
1. http://maven.apache.org/maven-v3_0_0.xsd for Maven 1.1.
2. http://maven.apache.org/maven-v4_0_0.xsd for Maven 2.0.

Note: The simplest thing to do in transition from Maven 1 to Maven 2 project building is to rename project.xml file into pom.xml file first. However, there are slight differences that need to be addressed and they are presented in the table bellow. I recommend creating subdirectories in the project directory called maven1 and maven2, where project files from both versions are kept (under the version control) until you are comfortable that your project is fully converted to a Maven 2 project.

You can start by moving project.xml to pom.xml, change pomVersion to modelVersion in the table bellow and than follow some of the mappings (bold letters denote elements that need to be changed).

The alternative to the table is to use NetBeans and do XML validation, which will give you which elements need correction according to the new XML schema.

The following table resulted from conversion of project.xml in pom.xml of the **org-glast-groupmanager project**. Mappings between project.xml and pom.xml elements:

Maven 1.x	Maven 2.x	Description
<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/4.0.0http://maven.apache.org/maven-v4_0_0.xsd">	Keep more extended version of project, so you can visit the schema and validate document
<pomVersion>3.0.0< /pomVersion>	<modelVersion>4.0.0</modelVersion>	Differentiates between different versions of POM objects
<groupId>	<groupId>	Stays the same
<artifactId>	<artifactId>	Stays the same
<currentVersion>	<version>	Same meaning
<shortDescription>	<description>	Same meaning
<package>	Not used, JavaDoc is done differently in Maven 2	The Java package name of the project. This value is used when generating JavaDoc.
<siteAddress>	Not used, hostname is part of the corresponding <url> element	The hostname of the web server that hosts the project's web site. This is used when the web site is deployed.
<siteDirectory>	Not used, directory part of the corresponding <url> element	The directory on the web server where the public web site for this project resides
<repository>	Should be nested inside <repositories> element	Specifies various Maven and SCM repositories
<developerConnection>	Should be moved to <scm> element	Defines connection to SCM (Source Control Management) for developers.
<reports>	<reporting> (and <report>*)	This element includes the specification of reports to be included in a Maven-generated site.
<properties>	If used inside <dependency> delete the element and use <scope>, <version>,<type> as described in http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html	Dependency properties
<url>	If used inside <dependency>, delete it.	Different URLs nested inside other elements
<sourceDirectory>	<sourceDirectory>	Used inside <build>, no need to specify, if the default src directory is used.
<issueManagementUrl>	<issueManagement>	url of Jira server

3. File after substitutions:

(Note: This file will still not work, as we need to specify some other Maven 2 properties, like local, GLAST and global repositories, etc.)

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!--
Maven 2 is shortly identified as maven. If we want to give a reference to Maven 1.0.2 we will call Maven 1.
This POM extends SuperPOM, which means that the following elements are already defined:
maven central repository (http://repo1.maven.org/maven2)
maven plugin repository  (http://repo1.maven.org/maven2)

default build directories:

  target
  target/classes
  target/test-classes
  target/site
  src/main/java
  src/test/java
  src/main/scripts
  src/main/resources
  src/test/resources


as well as default profiles: release-profile

and default plugins:
        maven-source-plugin
        maven-javadoc-plugin
        maven-deploy-plugin

-->
<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>
        <artifactId>org-glast-groupmanager</artifactId>
        <groupId>glast</groupId>
        <version>1.6-SNAPSHOT</version>
        <description>Glast Group Manager</description>
        <url>http://glast-ground.slac.stanford.edu/GroupManager/</url>
        <name>Glast Group Manager</name>
        <inceptionYear>2005</inceptionYear>
         <repositories>
                <!-- Not yet defined -->
        </repositories>
          <scm>
                <developerConnection>
                        scm:cvs:ext:${maven.username}@glast   java.slac.stanford.edu:/cvs/java:${pom.
artifactId}  <!-- to be verified -->
                </developerConnection>
         </scm>
         <dependencies>
                <!-- dependencies here -->
                <dependency>
                                <groupId>tomcat</groupId>
                                <artifactId>servlet-api</artifactId>
                                <version>5.0.18</version>
                    </dependency>
                <!-- ... -->
                <!-- Needed by cruisecontrol -->
                <dependency>
                                <groupId>maven</groupId>
                                <artifactId>maven-scm-plugin</artifactId>
                                <version>1.5</version>
                                <type>plugin</type>
                    </dependency>
        </dependencies>
        <build>
                    <resources>
                                <resource>
                        <directory>src/main/resource</directory> <!-- should have been resources -->
                        <includes>
                                  <include>META-INF/*.tld</include>
                        </includes>
                      </resource>
```

```
    </resources>
  </build>
</project>
```

scope tag

The new element that has been introduced in dependencies is the scope tag. It describes what the dependency is used for, how to find the dependency, and when it should be included in the classpath.
There are five options for scope:

- *compile*: This is the default scope (for example, the struts2-core artifact uses this scope because no other was provided), and these dependencies are available on all classpaths. Any dependency with compile scope will be packaged with the final artifact.
- *provided*: The dependency will be provided by the JDK or the application server at runtime. It is required for compilation but will not be packaged in the application.
- *runtime*: The dependency is not required for compilation but is required to run the application. It will be available only on the runtime classpath and the test classpath.
- *test*: The dependency is only required for testing and will be available on the test classpath and the runtime classpath.
- *system*: The dependency is always available (you need to provide the JAR file) and is not retrieved via a repository lookup.

4. profiles.xml file

If we run mvn install target with this POM file, it will still not build completely as we need to locate dependencies that are not found in the parent POM central repositories (modules and plugins): http://repo1.maven.org/maven2

Maven 2 profile files

Profiles are Maven 2 way of letting you create environmental variations in the build life cycle to accommodate things like building on different platforms, building with different JVMs, testing with different databases, or referencing the local file system. Typically, you try to encapsulate as much as possible in the POM to ensure that builds are portable, but sometimes you simply have to take into consideration variation across systems and this is why profiles were introduced in Maven 2.

Profiles modify the POM at build time, and are meant to be used in complementary sets to give equivalent-but-different parameters for a set of target environments (providing, for example, the path of the application server root in the development, testing, and production environments).

As such, profiles can easily lead to differing build results from different members of your team. However, used properly, you can still preserve build portability with profiles. You can define profiles in one of the following three places:

1. The Maven 2 settings file (typically <your -home-directory>/.m2/settings.xml)
2. A file in the the same directory as the POM, called profiles.xml
3. The POM itself

In terms of which profile takes precedence, the local-most profile wins. So, POM-specified profiles override those in profiles.xml, and profiles.xml overrides those in settings.xml. This is a pattern that is repeated throughout Maven 2, that local always wins, because it is assumed to be a modification of a more general case. settings.xml profiles have the potential to affect all builds, so they're sort of a "global" location for profiles. profiles.xml allows you to augment a single project's build without altering the POM.
And the POM-based profiles are preferred, since these profiles are portable (they will be distributed to the repository on deploy, and are available for subsequent builds originating from the repository or as transitive dependencies).

Because of the portability implications, any files which are not distributed to the repository are NOT allowed to change the fundamental build in any way. Therefore, the profiles specified in profiles.xml and settings.xml are only allowed to define:

- repositories
- pluginRepositories
- properties

Everything else must be specified in a POM profile, or in the POM itself, or not at all. For example, if you had a profile in settings.xml that was able to inject a new dependency, and the project you were working on actually did depend on that settings-injected dependency in order to run, then once that project is deployed to the repository it will never fully resolve its dependencies transitively when asked to do so. That's because it left one of its dependencies sitting in a profile inside your settings.xml file.

For example, GLAST specific m2 repository should be specified in profiles.xml file (thus allowing developers outside GLAST to refer to some other repositories), and developer specific m2 repository (if the developer keeps its own repository) can be specified in settings.xml file.

(c) Profile file structure

You can define the following elements in the POM profile:

- repositories
- pluginRepositories
- dependencies
- plugins
- properties (not actually available in the main POM, but used behind the scenes)

- modules
- reporting
- dependencyManagement
- distributionManagement

A subset of the build element, which consists of:

- defaultGoal
- resources
- testResources
- finalName

(d) Activating profiles

Since you build profiles for specific tasks (like debug build or test build), you need to have mechanism to identify and activate the profiles in different situations.

There are several ways that you can activate profiles:

1. Profiles can be specified explicitly using the mvn - P command line option. This option takes an argument that contains a comma-delimited list of profile-ids. When this option is specified, no profiles other than those specified in the option argument will be activated.

For example:

```
mvn -Pprofile1,profile2 install
```

2. Profiles can be activated in the Maven settings, via the activeProfiles section. This section takes a list of activeProfile elements, each containing a profile-id. Note that you must have defined the profiles in your settings.xml file as well. For example:

```
<settings>
[...]
<profiles>
<profile>
<id>profile1</id>
[...]
</profile>
</profiles>
<activeProfiles>
<activeProfile>profile1</activeProfile>
</activeProfiles>
[...]
</settings>
```

3. Profiles can be triggered automatically based on the detected state of the build environment. These activators are specified via an activation section in the profile itself. Currently, this detection is limited to prefix-matching of the JDK version, the presence of a system property, or the value of a system property. Here are some examples:

3.1.

```
<profile>
<id>profile1</id>
[...]
<activation>
<jdk>1.4</jdk>
</activation>
 </profile>
```

This activator will trigger the profile when the JDK's version starts with "1.4" (e.g., "1.4.0_08", "1.4.2_07", "1.4").

3.2

```
<profile>
<id>profile1</id>
[...]
<activation>
<property>
<name>debug</name>
</property>
</activation>
</profile>

This will activate the profile when the system property "debug" is specified with any value.
         3.3
```

<profile>
<id>profile1</id>
...
<activation>
<property>
<name>environment</name>
<value>test</value>
</property>
</activation>
</profile>

```
This last example will activate the profile when the system property "environment" is specified with the value
"test".


h3. (e)        profiles.xml  file for org-glast-groupmanager

Based on the general discussion above of profiles, I  have chosen initially the following profiles.xml file.

Note: settings.xml file in my Maven2 repository directory (~igor/.m2) is just:

{code:title=settings.xml}
<?xml version="1.0" encoding="UTF-8"?>
<settings>
</settings>
```

The profiles file will initially point to project specific repositories that are needed to complete the dependency resolution. In the transition period, when some of the files are in Maven 1 repositories (both GLAST and developer) and only some of the files are in GLAST Maven 2 respository:

http://glast-ground.slac.stanford.edu/maven2/ or
/nfs/slac/g/glast/ground/maven2

we need to establish the set of repositories to follow the following path in the order of precedence:

1.http://repo1.maven.org/maven2/
2. any outside repositories not in maven2 repository above (like apache.org, sun.com, freehep.org)
3.http://glast-ground.slac.stanford.edu/maven2/
4.http://glast-ground-slac.stanford.edu/maven1/
5. and only than developer specific repositories:
6. ~developer/.m2/repository
7. ~developer/.maven/repository

Note: as we download specific dependencies, extra dependencies might be downloaded, based on the dependencies specified in downloaded POM files.

The best practice is to define GLAST wide parent POM file, which defines GLAST specific repositories, settings, etc, which all the GLAST Java projects will use.

Maven 2 repositories

GLAST Maven 2 repository location

Glast Maven 2 repositories are located on the NFS file system

/nfs/slac/g/glast/ground/maven2 (releases)
/nfs/slac/g/glast/ground/maven2/SNAPSHOTS (snapshots)

They can be viewed through the web interface
http://glast-ground.slac.stanford.edu/maven2/

GLAST parent POM

Currently Maven 2 supports several methods of deployment, including simple file-based deployment, SSH2 deployment, SFTP deployment, FTP deployment, and external SSH deployment. In order to deploy, you need to correctly configure your distributionManagement element in your POM, which would typically be your top-level POM, so that all child POMs can inherit this information. In our case, we have decided to create a GLAST wide POM, which all Maven 2 Java modules will inherit from.

GLAST parent POM defines distributionManagement for all children POMs, so developers do not need to bother with distribution management, and also allows centeral management of global GLAST development resources.

In order to deploy to the GLAST Maven 2 repository, we use ssh2 protocol, and the following distributionManagement elements

GLAST parent POM

```
<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>glast</groupId>
  <artifactId>maven-parent</artifactId>
  <version>1.0-SNAPSHOT</version>
  <organization>
    <name>Glast</name>
    <url>http://glast-ground.slac.stanford.edu</url>
  </organization>
  <repositories>
    <repository>
      <id>glast-maven</id>
      <name>GLAST Maven 2 Repository</name>
      <url>
        http://glast-ground.slac.stanford.edu/maven2/
      </url>
      <releases>
        <enabled>false</enabled>
      </releases>
    </repository>
    <repository>
      <id>glast-maven-snapshots</id>
      <name>GLAST Snapshot Repository</name>
      <url>
        http://glast-ground.slac.stanford.edu/maven2/SNAPSHOTS
      </url>
      <releases>
        <enabled>false</enabled>
      </releases>
    </repository>
  </repositories>
</project>
```

The Maven GLAST POM declares the elements that are common to all of its sub-projects.

The GLAST snapshot repository and the release repository deployment locations should be defined in children of the parent POM.
It is the best practice to follow open source releases which separate SNAPSHOTS and PLUGINS directory from the release directory.

In children, you should point to the following GLAST distribution repositories:

GLAST repositories

```
<!-- use the following in  if you're not using a snapshot version. xxxx-->
  <repository>
     <id>glast-maven2</id>
     <name>GLAST Maven 2 central repository</name>
     <url>scp://glast-java.slac.stanford.edu:/nfs/slac/g/glast/ground/maven2/</url>
  </repository>
  <!-- use the following if you ARE using a snapshot version. -->
  <snapshotRepository>
     <id>glast-maven2-snapshots</id>
     <name>GLAST Maven2 centreal SNAPSHOTS repository</name>
     <url>scp://glast-java.slac.stanford.edu:/nfs/slac/g/glast/ground/maven2/SNAPSHOTS</url>
     <!-- this is set to false, to prevent multiple versions of SNAPSHOT files -->
     <uniqueVersion>false</uniqueVersion>
   </snapshotRepository>
```

Or if you want to maintain a set of plugins use:

GLAST plugin repository

```
<pluginRepositories>
        <pluginRepository>
            <id>glast-maven2-plugins</id>
            <name>GLAST Maven2 central PLUGINS repository</name>
            <url>scp://glast-java.slac.stanford.edu:/nfs/slac/g/glast/ground/maven2/PLUGINS</url>
            <uniqueVersion>false</uniqueVersion>
        </pluginRepository>
    </pluginRepositories>
```

An issue that can arise, when working with this type of hierarchy, is regarding the storage location of the source POM files. Source control management systems like CVS and SVN (with the traditional intervening trunk directory at the individual project level) do not make it easy to store and check out such a structure. These parent POM files are likely to be updated on a different, and less frequent schedule than the projects themselves. For this reason, it is best to store the parent POM files in a separate area of the source control tree, where they can be checked out, modified, and deployed with their new version as appropriate. In fact, there is no best practice requirement to even store these files in your source control management system; you can retain the historical versions in the repository if it is backed up (in the future, the Maven Repository Manager will allow POM updates from a web interface).

Using GLAST parent POM.

All what is necessary is to include parent in each of the project POM files. This is done as in the following example:

Using GLAST parent 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">
  <!-- every project POM must include GLAST parent POM -->
  <parent>
      <groupId>glast</groupId>
      <artifactId>maven-parent</artifactId>
      <version>1.0</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <groupId>glast</groupId>
  <artifactId>sample-maven-webapp</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Sample Maven Webapp</name>
  <url>http://glast-ground.slac.stanford.edu</url>
  <dependencies>
      <!-- enter project specific POM -->
  </dependencies>
  <build>
    <finalName>sampleMavenWebapp</finalName>
  </build>
</project>
```


project inheritance

In order for you to see what happens during the inheritance process, you will need to use the handy **mvn help:effective-pom** command. This command will show you the final result for a target pom.

You will notice that the POM that you see when using the mvn help:effective-pom is bigger than you expected. Remember that the Super POM sits at the top of the inheritance hierarchy. Looking at the effective POM includes everything and is useful to view when trying to figure out what is going on when you are having problems.

Deploying your project in GLAST Maven 2 repository:

If you can deploy to cvs server and deploy to Maven 1, you are already set for deploying to Maven 2.

If not, make sure that you can ssh and scp to glast-java.slac.stanford.edu

Using ssh at SLAC is nicely described in:
http://glast-ground.slac.stanford.edu/workbook/ (Under Getting Conntected, Secure Shell).

To learn about ssh and different authentication strategies http://www.linux.com/articles/34958?tid=78&tid=82

Basic requirement is that you need to be able to do scp without password authentication.

Using archetypes to create specific type of projects.

Often, you would like to use a default Maven 2 sceleton project for particular type of project, that are not already part of NetBeans or your IDE. In that case you need to locate your favorable 'archetype' on the web, and use it to build a sceleton project. For example, if there were many projects that are going to use Struts2 framework, one would download Struts2 starter project archetype, called struts2-archetype-starter.

To generate the starter project, select the working directory, and issue the following command:

Creating a prototype project

```
mvn archetype:create
-DgroupId=org.glast.struts2
-DartifactId=app
-DarchetypeGroupId=org.apache.struts
-DarchetypeArtifactId=struts2-archetype-starter
-DarchetypeVersion=2.0.9-SNAPSHOT
-DremoteRepositories=http://people.apache.org/maven-snapshot-repository
```

This command will create a Struts2 application prototype, in the 'app' directdory. From there you can use the files in directory 'app' as a starting point for your new Struts2 application project.

Configuring the Dependencies

Most of the time there is no configuration for repositories in the pom.xml configuration file because there is a built-in default repository (pointing to http://repo1.maven.org/maven2). Most of the time, open source projects will publish artifacts to the central repository, but they can also host their own repositories and not made them available.

Hibernate, which is now under the JBoss umbrella of projects, is such an example.

To integrate Hibernate into the web application, two different types of dependencies are required: for the Hibernate code itself, and for the JPA and transactional APIs that are implemented. The JBoss repository contains both of these types, so configuring the pom.xml configuration file is easy. A new repositories tag is added under the top-level project tag, which in turn contains a repository tag with an id tag (providing a unique identifier) and a url tag (that supplies the URL of the repository).
<project>
...
<repositories>
<repository>
<id>jboss</id>
<url>http://repository.jboss.com/maven2</url>
</repository>
</repositories>
<project>

As well as the common repositories outside our organization, we have created repositories inside GLAST. This is a great strategy because developers know that if an artifact is installed in the GLAST repository, it can be used. Along with external artifacts, this repository hosts internal projects and libraries to provide a central access point here at SLAC.

Installing new artifacts in the GLAST Maven 2 repository

When dealing with a build tool that automates dependency management (especially downloading the libraries for you) such as Maven2, the artifact or library that you require may not be available.
The following are two scenarios when JAR files are not automatically retrieved and installed:
? The distributing organization does not publish the JAR files to the standard remote repositories (such as ibiblio.com).
? A legal restriction such as a user acknowledgment or license agreement needs to be accepted before using the files.
In either of these cases, additional steps are required before the JAR file can be configured in the pom.xml configure file and utilized in your application:

1. Locate the download location for the libraries that are needed
2. Accept the license agreement (if applicable), and download the file.
3. If the files are downloaded as archives (other than JAR files), expand the archive into a working directory.
4. Install the required libraries into your local Maven2 repository (usually in ~/.m2)

Once downloaded, Maven2 provides a command to install the library into the correct local repository location and to create the necessary metadata files within the repository. The command has the following form:

```
mvn install:install-file -DgroupId=<groupId>
-DartifactId=<artifactId>
-Dversion=<version>
-Dpackaging=jar
-Dfile=</path/to/downloaded/file>
```

For an example, let's take the Hibernate annotations file (remember this is an example and this step doesn't need to be performed). Following is the dependency configuration that will be added into the web application's pom.xml configuration file:

```
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-annotations</artifactId>
<version>3.2.1.ga</version>
</dependency>
```

This provides all the information needed to issue the install command from earlier. By substituting the groupId, artifactId, and version information, and issuing the command from the directory that the hibernate-annotations.jar (the name of the file that was downloaded) is located, the Maven2 command to install the library becomes the following:

```
mvn install:install-file -DgroupId=org.hibernate
-DartifactId=hibernate-annotations
-Dversion=3.2.1.ga -Dpackaging=jar -Dfile=hibernate-annotations.jar
```

The preceding pom.xml configuration is now able to access the dependency from your local development environment.

The next step is to configure Maven2 so that the application can access the Hibernate dependencies. We saw above that this is achieved by adding the dependency file information to the dependencies node of the pom.xml configure file. When configured in this manner, Maven2 goes out and retrieves the JAR files from either a local repository or the newly configured remote repository.

```
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate</artifactId>
<version>3.2.1.ga</version>
</dependency>
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-annotations</artifactId>
<version>3.2.1.ga</version>
</dependency>
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-entitymanager</artifactId>
<version>3.2.1.ga</version>
</dependency>
```

Note Each Hibernate dependency also has its own dependencies, known as transitive dependencies. When the project you are using is built for Maven2, the transitive dependencies are configured and downloaded automatically with the dependency that you have configured. But occasionally, you will encounter a project that has not been built for Maven2. In this case, you need to add each of the transitive dependencies individually to the pom.xml configuration file manually.

Guide to using Maven 1.x repositories with Maven 2.x

When you are migrating from Maven 1.x to Maven 2.x you will first be trying to convert your build and to make this easier we have provided a way for you to use your existing Maven 1.x repository so that you don't have to convert your repository before trying to migrate your projects. To use a Maven 1.x repository with your Maven 2.x project you need to specify this in your POM as follows:<project>
...
<repositories>
<repository>
<snapshots>
<enabled>true</enabled>
</snapshots>
<id>my-m1-repository</id>
<name>Maven 1.x Repository</name>
<url>http://repostory.mycompany.com/maven1</url>
<layout>legacy</layout>
</repository>
</repositories>
...

</project>

Enabling the snapshots is important as Maven 2.x makes a distinction between repositories that contain snapshots and those that don't. In Maven 1.x there is no distinction, so setting snapshots to true will give you the Maven 1.x style repository behavior while using Maven 2.x

Web application deployments on tomcat servers

Todo

Appendix A

Here we give Maven 1.x files for reference:

project.xml file:

project.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
   <pomVersion>3</pomVersion>
   <artifactId>org-glast-groupmanager</artifactId>
   <groupId>glast</groupId>
   <currentVersion>1.6-SNAPSHOT</currentVersion>
   <shortDescription>Glast Group Manager</shortDescription>
   <url>http://glast-ground.slac.stanford.edu/GroupManager/</url>
   <name>Glast Group Manager</name>
   <package>org.glast.groupmanager</package>
   <inceptionYear>2005</inceptionYear>
   <siteAddress>glast-java.slac.stanford.edu</siteAddress>
   <siteDirectory>/nfs/slac/g/glast/ground/docs/${pom.artifactId}</siteDirectory>
   <reports>
      <report>maven-jdepend-plugin</report>
      <report>maven-changes-plugin</report>
      <report>maven-changelog-plugin</report>
      <report>maven-developer-activity-plugin</report>
      <report>maven-file-activity-plugin</report>
      <report>maven-javadoc-plugin</report>
      <report>maven-jxr-plugin</report>
      <report>maven-junit-report-plugin</report>
      <report>maven-linkcheck-plugin</report>
      <report>maven-tasklist-plugin</report>
      <report>taglib</report>
  </reports>
   <repository>
      <developerConnection>
                scm:cvs:ext:${maven.username}@glast-java.slac.stanford.edu:/cvs/java:${pom.artifactId}
        </developerConnection>
        <url>http://www-glast.stanford.edu/cgi-bin/viewcvs/${pom.artifactId}/?root=java</url>
   </repository>

   <dependencies>
      <dependency>
         <groupId>tomcat</groupId>
         <artifactId>servlet-api</artifactId>
         <version>5.0.18</version>
      </dependency>

        <!-- more dependencies -->

              <!- Needed by cruisecontrol ->

      <dependency>
         <groupId>maven</groupId>
         <artifactId>maven-scm-plugin</artifactId>
         <version>1.5</version>
         <type>plugin</type>
         <url>http://www.ibiblio.org/maven/maven/plugins/</url
      </dependency>
   </dependencies>

   <build>
      <sourceDirectory>src/main/java</sourceDirectory>
      <resources>
         <resource>
            <directory>src/main/resource</directory>
            <includes>
               <include>META-INF/*.tld</include>
            </includes>
         </resource>
      </resources>
   </build>
</project>
```

project.properties file:

project.properties

```
maven.repo.remote=http://mirrors.ibiblio.org/pub/mirrors/maven,http://java.freehep.org/maven,http://glast-
ground.slac.stanford.edu/maven
maven.war.src=src/webapp
maven.tomcat.war.context=/GroupManager
maven.war.final.name=GroupManager.war
maven.tomcat.precompile=false
maven.repo.list=glast
maven.repo.glast=scpexe://glast-java.slac.stanford.edu
maven.repo.glast.directory=/nfs/slac/g/glast/ground/maven
maven.site.deploy.method=ssh
maven.compile.target=1.5
maven.compile.source=1.5
taglib.src.dir=src/main/resource/META-INF
```

maven.xml file:

maven.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:j="jelly:core"
           xmlns:u="jelly:util" default="war">
    <goal name="webInstall">
            <attainGoal name="tomcat:install"/>
   </goal>
    <goal name="webReload">
        <attainGoal name="tomcat:remove"/>
        <attainGoal name="tomcat:install"/>
    </goal>
    <goal name="webRemove">
        <attainGoal name="tomcat:remove"/>
    </goal>
    <goal name="tomcat03Redeploy">
        <u:properties file="${user.home}/tomcat03.properties" />
         <attainGoal name="clean"/>
        <attainGoal name="tomcat:remove"/>
        <attainGoal name="tomcat:deploy"/>
    </goal>
    <goal name="tomcat03Deploy">
        <u:properties file="${user.home}/tomcat03.properties" />
        <attainGoal name="clean"/>
        <attainGoal name="tomcat:deploy"/>
    </goal>

    <goal name="tomcat01Redeploy">
        <u:properties file="${user.home}/tomcat01.properties" />
        <attainGoal name="clean"/>
        <attainGoal name="tomcat:remove"/>
        <attainGoal name="tomcat:deploy"/>
    </goal>
    <goal name="tomcat01Deploy">
        <u:properties file="${user.home}/tomcat01.properties" />
        <attainGoal name="clean"/>
        <attainGoal name="tomcat:deploy"/>
    </goal>
</project>
```

build.properties file:

build.properties

```
maven.netbeans.exec.run=webReload
maven.netbeans.exec.build=war:war
tomcat01.properties:
maven.repo.remote =http://mirrors.ibiblio.org/pub/mirrors/maven,http://java.freehep.org/maven,http://glast-
ground.slac.stanford.edu/maven,http://www.codeczar.com/maven
maven.tomcat.host=glast-tomcat01.slac.stanford.edu
maven.tomcat.port=8080
maven.tomcat.username=glast
maven.tomcat.password=...
maven.tomcat.precompile=false
maven.compile.compilerargs=-Xlint:unchecked
maven.tomcat.war.context=/dp2
maven.war.src=src/main/webapp
maven.war.final.name=dp2.war
```

mavenrun.log

```
__  __
|  \/  |__ _Apache__ ___
| |\/| / _` \ V / -_) ' \  ~ intelligent projects ~
|_|  |_\__,_|\_/\___|_||_|  v. 1.0.2

build:start:
war:init:
war:war-resources:
mkdir Created dir: /Users/igor/work/java/glast/org-glast-groupmanager/target/org-glast-groupmanager
mkdir Created dir: /Users/igor/work/java/glast/org-glast-groupmanager/target/org-glast-groupmanager/WEB-INF
copy Copying 15 files to /Users/igor/work/java/glast/org-glast-groupmanager/target/org-glast-groupmanager
copy Copying 1 file to /Users/igor/work/java/glast/org-glast-groupmanager/target/org-glast-groupmanager/WEB-INF
java:prepare-filesystem:
mkdir Created dir: /Users/igor/work/java/glast/org-glast-groupmanager/target/classes
java:compile:
echo Compiling to /Users/igor/work/java/glast/org-glast-groupmanager/target/classes
javac Compiling 6 source files to /Users/igor/work/java/glast/org-glast-groupmanager/target/classes
java:jar-resources:
Copying 1 file to /Users/igor/work/java/glast/org-glast-groupmanager/target/classes
test:prepare-filesystem:
mkdir Created dir: /Users/igor/work/java/glast/org-glast-groupmanager/target/test-classes
mkdir Created dir: /Users/igor/work/java/glast/org-glast-groupmanager/target/test-reports
test:test-resources:
test:compile:
echo No test source files to compile.
test:test:
echo No tests to run.
war:webapp:
echo Assembling webapp org-glast-groupmanager
mkdir Created dir: /Users/igor/work/java/glast/org-glast-groupmanager/target/org-glast-groupmanager/WEB-INF/lib
mkdir Created dir: /Users/igor/work/java/glast/org-glast-groupmanager/target/org-glast-groupmanager/WEB-INF/tld
mkdir Created dir: /Users/igor/work/java/glast/org-glast-groupmanager/target/org-glast-groupmanager/WEB-INF
/classes
copy Copying 1 file to /Users/igor/work/java/glast/org-glast-groupmanager/target/org-glast-groupmanager/WEB-INF
/lib
...
copy Copying 11 files to /Users/igor/work/java/glast/org-glast-groupmanager/target/org-glast-groupmanager/WEB-
INF/classes
war:war:
echo Building WAR org-glast-groupmanager
jar Building jar: /Users/igor/work/java/glast/org-glast-groupmanager/target/GroupManager.war

BUILD SUCCESSFUL

Total time: 4 seconds
Finished at: Thu Nov 29 15:59:57 PST 2007
```