

Maven

Brian Schlining

What is Maven?

- A Build *Framework*
 - Cleanly separates your code from
 - Configuration files
 - Documentation
 - Dependencies

Benefits of Maven

- Standardized project layout
- Standardized dependency management
- Multiple project/module support
- Instant downloads of Maven plugins and features as needed.
- Website generation
- Integration with source control

Getting Started

```
mvn archetype:create /  
  -DgroupId=org.mbari.foo /  
  -DartifactId=bar-app
```

Getting Started

```
mvn archetype:create /  
    -DgroupId=org.mbari.foo /  
    -DartifactId=bar-app
```

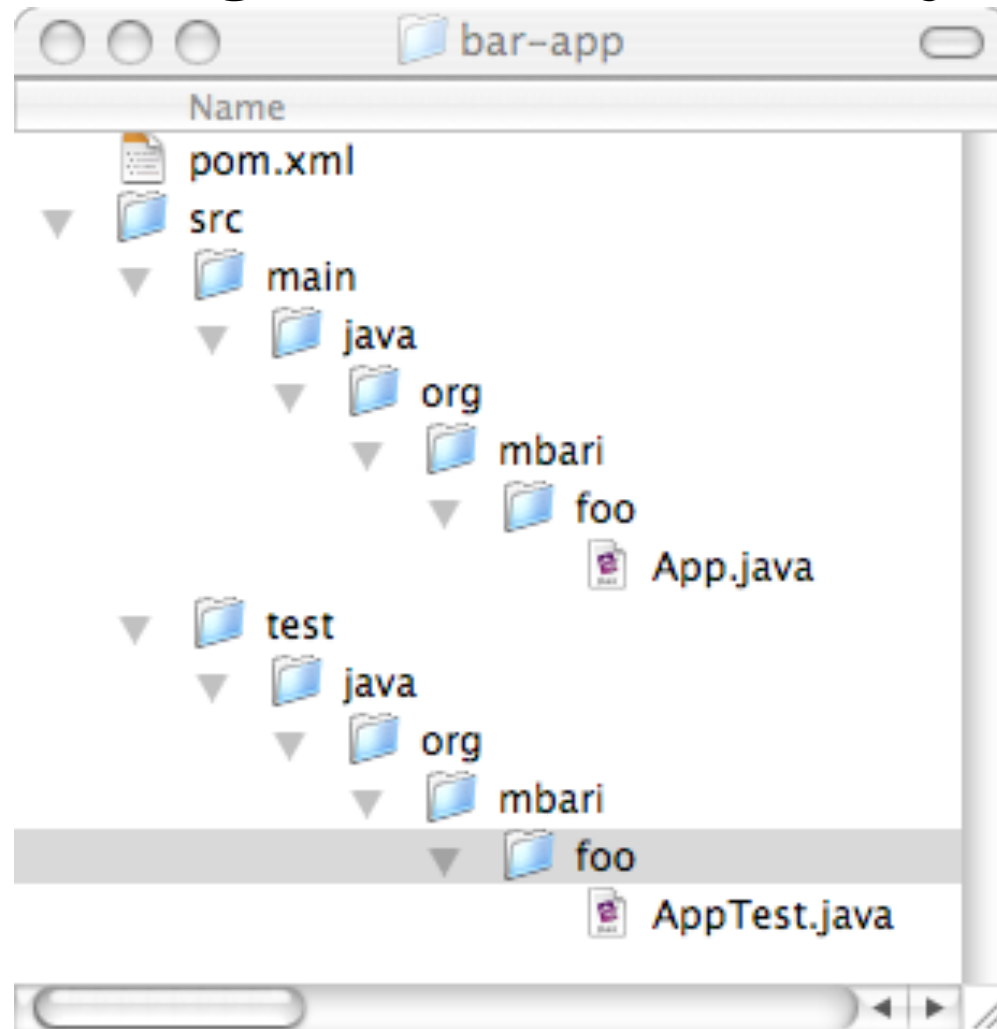
Generally corresponds to the *package* of your project. Some projects use the same value as artifactId

Getting Started

```
mvn archetype:create /  
    -DgroupId=org.mbari.foo /  
    -DartifactId=bar-app
```

The name of your application, jar, war, or ear. Called an *artifact*. Each Maven project creates exactly **one** artifact. The Above command creates a *jar* artifact project

Getting Started - Layout

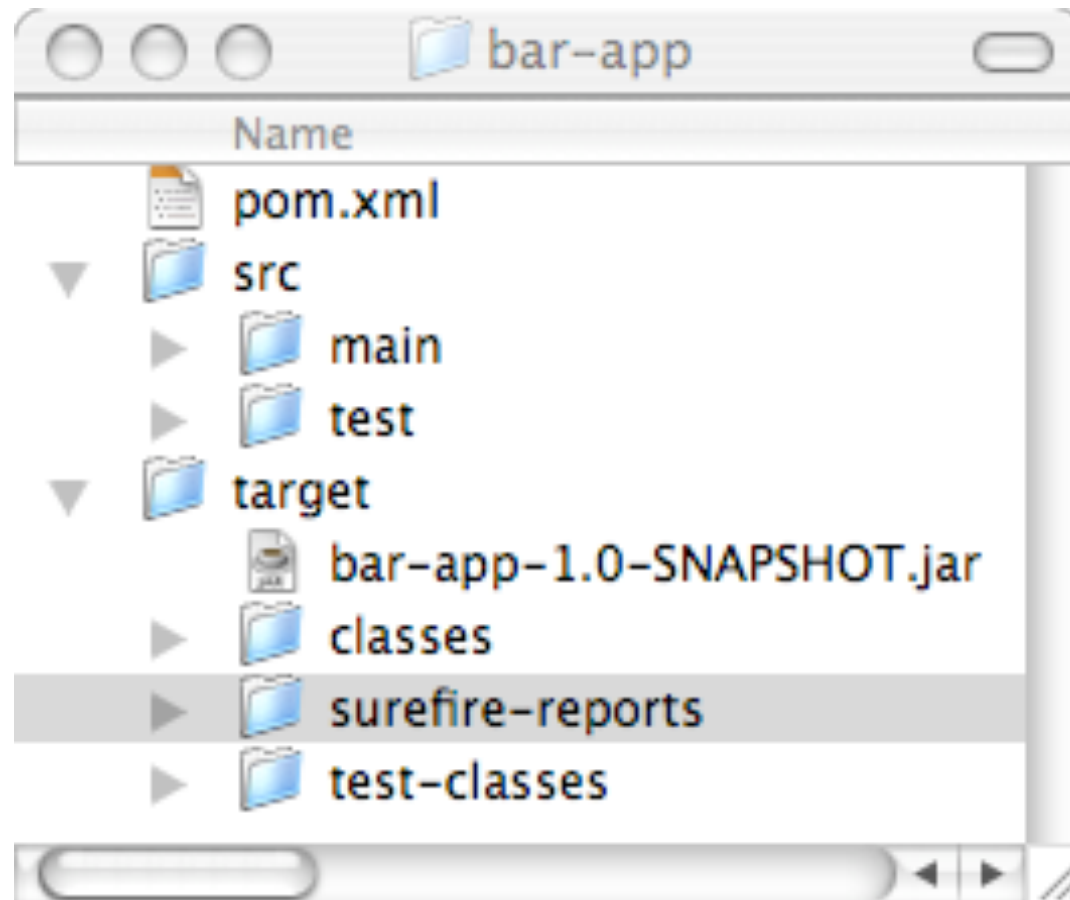


Getting Started - Building

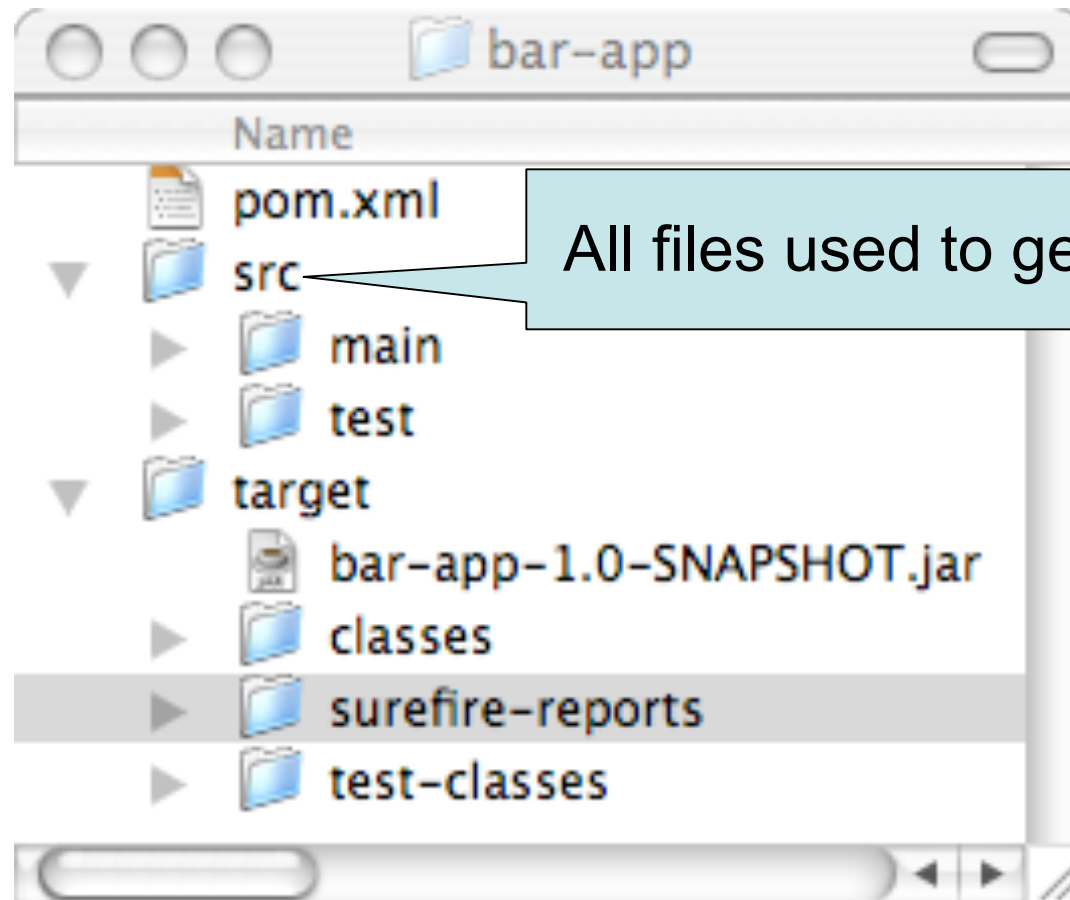
`mvn package`

This compiles, compiles tests, executes tests and creates the artifact (e.g. a jar file)

Getting Started - Building

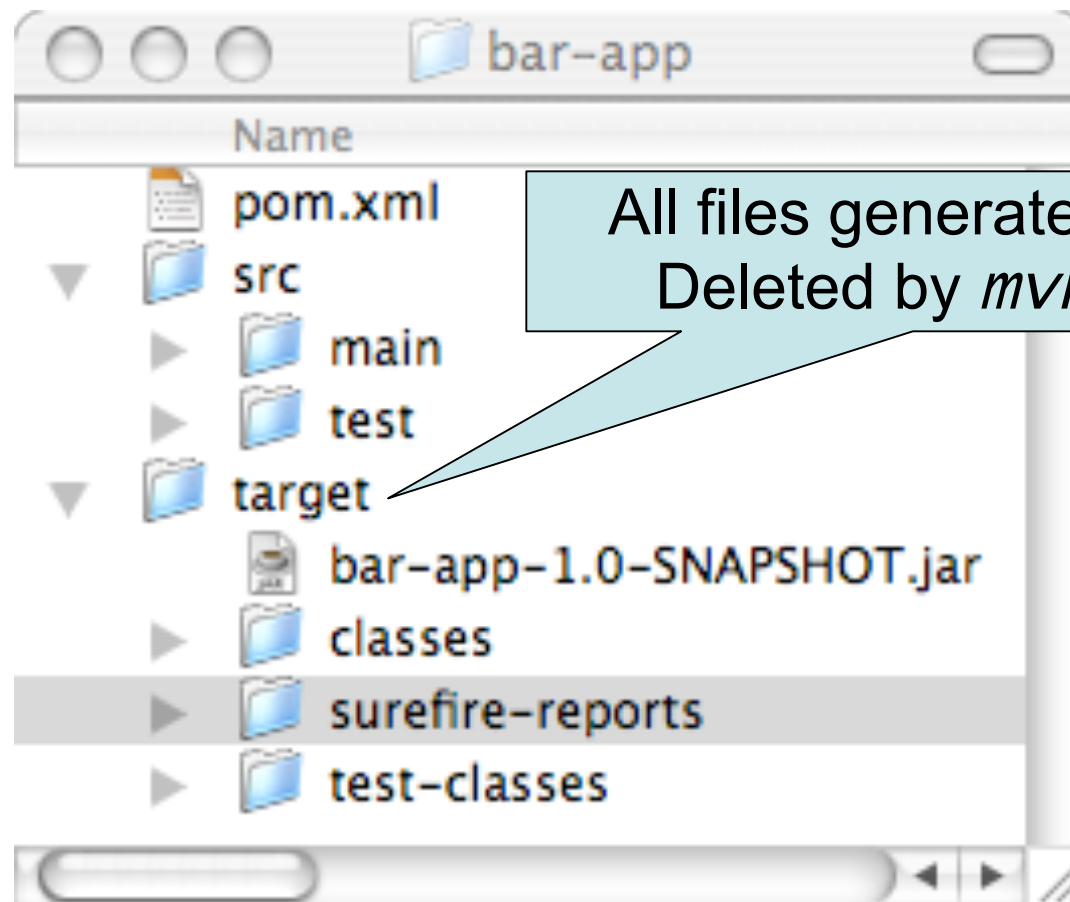


Getting Started - Building



All files used to generate build

Getting Started - Building



With Great Power Comes
Great Configuration

pom.xml

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  ... >
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.mbari.foo</groupId>
  <artifactId>bar-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>bar-app</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>c
```

pom.xml - Add Dependencies

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  ... >
  ...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>sl4j-api</artifactId>
      <version>1.3.0</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
</project>
```

You do not need to specify transitive dependencies. Maven takes care of them for you

pom.xml - Add Dependencies

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  ... >
  ...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>[1.2.0,)</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
</project>
```

Can specify version ranges. Here we say use version 1.2.0 or later.

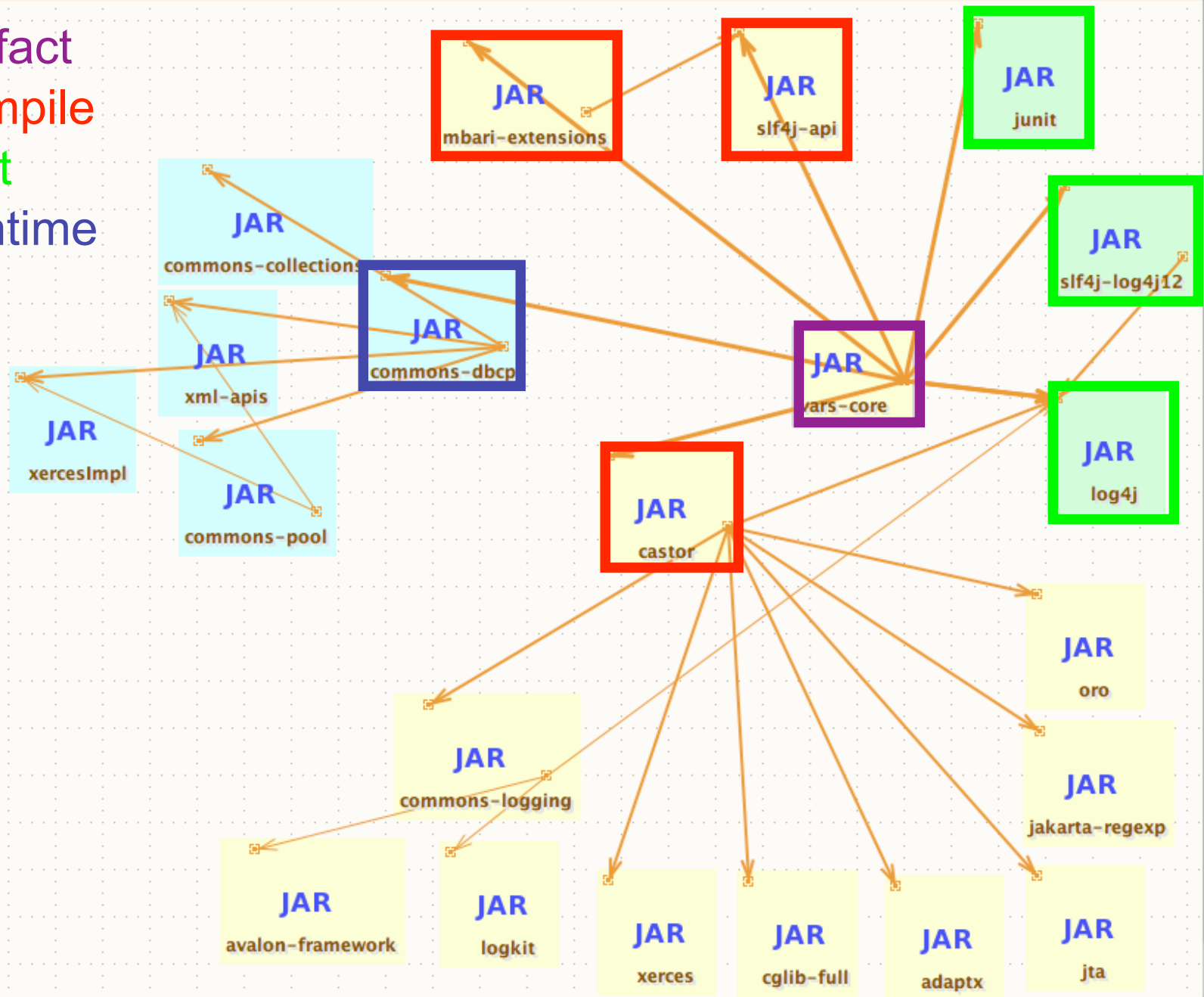
pom.xml - Add Dependencies

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  ... >
  ...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>1.3.0</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
</project>
```

Scopes can be:
compile [default]
runtime
test
provided

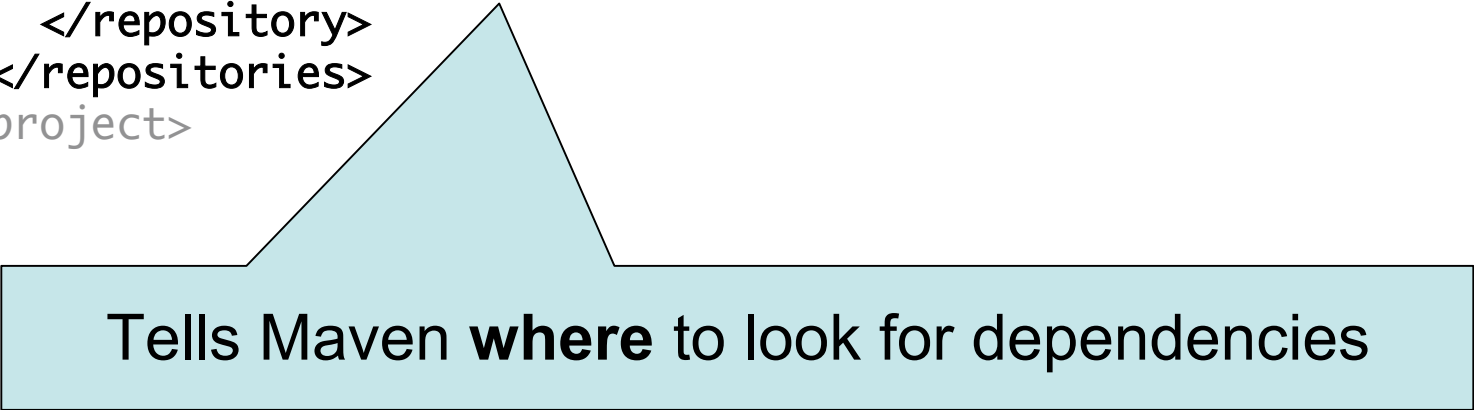
Transitive Dependencies for VARS Data Model and DAO (jar)

Artifact
Compile
Test
Runtime



pom.xml - Add Repositories

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  ... >
  ...
  <repositories>
    <repository>
      <id>mbari</id>
      <name>MBARI Repository</name>
      <url>http://oceana.shore.mbari.org/maven2</url>
    </repository>
  </repositories>
</project>
```



Tells Maven **where** to look for dependencies

pom.xml - Resolving Artifacts

Repository URL

<http://oceana/maven2>

GroupID

org.mbari.foo

ArtifactID

bar

Version

1.0

Artifact URL

<http://oceana/maven2/org/mbari/foo/bar/1.0>

About Repositories

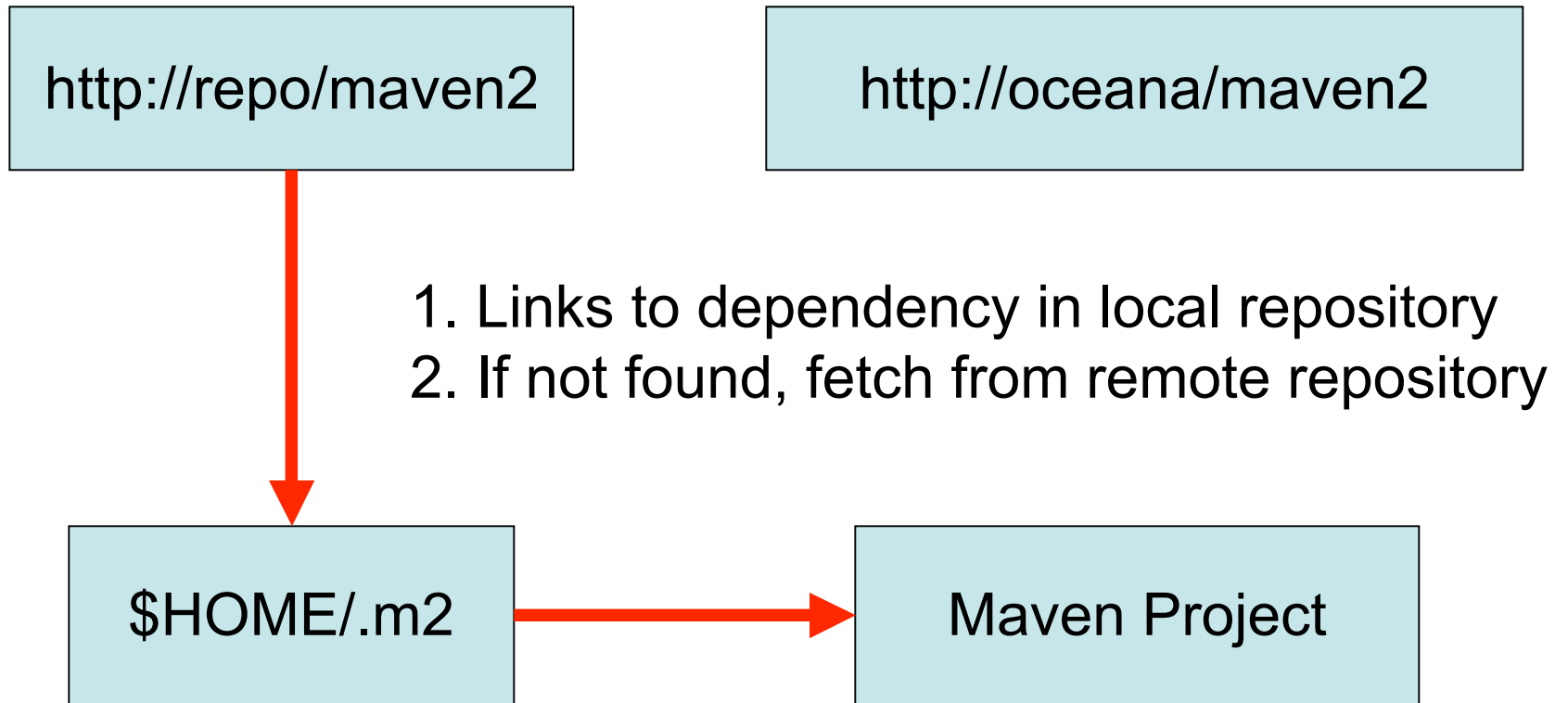
`http://repo/maven2`

`http://oceana/maven2`

`$HOME/.m2`

Maven Project

About Repositories



`mvn compile`

About Repositories

`http://repo/maven2`

`http://oceana/maven2`

1. Installs dependency in local repository

It's now available for other local projects to use



`mvn install`

About Repositories

`http://repo/maven2`

`http://oceana/maven2`

1. Installs dependency in deployment repository

It's now available for all projects to use

`$HOME/.m2`

Maven Project

`mvn deploy`



pom.xml - Modifying Build

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ... >
  ...
  <build>
    <plugins>
      <!-- Use Java 5.0; default is Java 1.3 -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.5</source>
          <target>1.5</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

pom.xml - Modifying Build

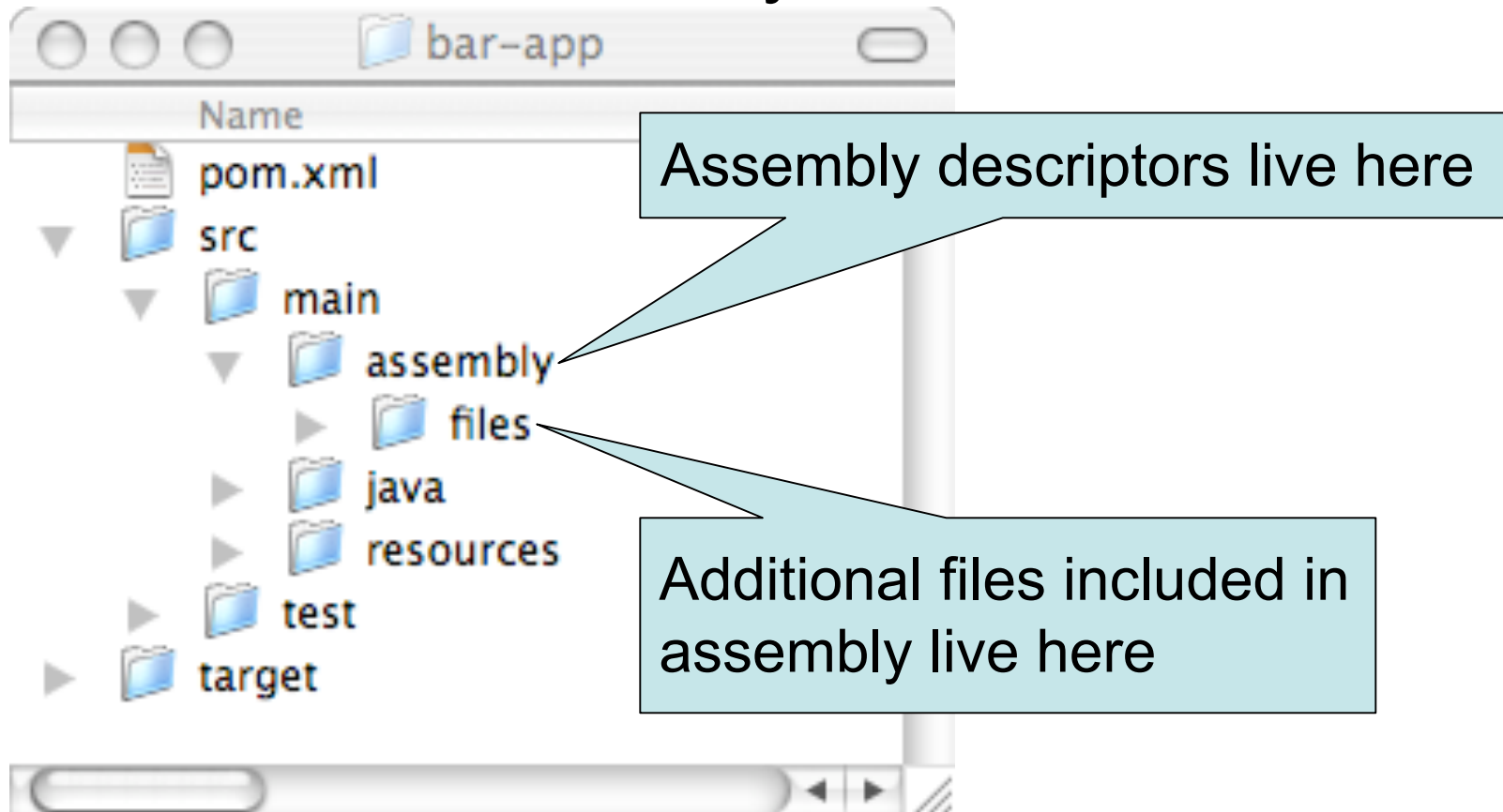
```
<project xmlns="http://maven.apache.org/POM/4.0.0" ... >
  ...
  <build>
    <plugins>
      <!-- Create source and javadoc jars when deploying -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-source-plugin</artifactId>
        <executions>
          <execution>
            <goals>
              <goal>jar</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

pom.xml - Deploying

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ... >
...
<distributionManagement>
  <!-- Web site deployment location -->
  <site>
    <id>oceana.shore.mbari.org</id>
    <url>scp://oceana/var/www/html/projects/${project.artifactId}</url>
  </site>
  <!-- Artifact deployment location -->
  <repository>
    <id>mbari-repository</id>
    <name>MBARI Repository</name>
    <url>scp://oceana/var/www/html/maven2</url>
  </repository>
</distributionManagement>
</project>
```

What if I need to build more than just a Jar?

- Create an Assembly



```
<?xml version="1.0" encoding="UTF-8"?>
<assembly>
  <id>standalone</id>
  <formats>
    <format>zip</format>
  </formats>
  <includeBaseDirectory>false</includeBaseDirectory>
  <!-- copy assembly files -->
  <fileSets>
    <fileSet>
      <directory>src/main/assembly/files</directory>
      <outputDirectory></outputDirectory>
      <includes>
        <include>**/*</include>
      </includes>
    </fileSet>
  </fileSets>
  <dependencySets>
    <!-- Grab dependencies libraries -->
    <dependencySet>
      <outputDirectory>lib</outputDirectory>
      <unpack>false</unpack>
      <scope>runtime</scope>
    </dependencySet>
  </dependencySets>
</assembly>
```

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

```
<assembly>
```

```
  <id>standalone</id>
```

```
  <formats>
```

```
    <format>zip</format>
```

```
  </formats>
```

```
  <includeBaseDirectory>>false</includeBaseDirectory>
```

```
  <!-- copy assembly files -->
```

```
  <fileSets>
```

```
    <fileSet>
```

```
      <directory>src/main/assembly/files</directory>
```

```
      <outputDirectory></outputDirectory>
```

```
      <includes>
```

```
        <include>**/*</include>
```

```
      </includes>
```

```
    </fileSet>
```

```
  </fileSets>
```

```
  <dependencySets>
```

```
    <!-- Grab dependencies libraries -->
```

```
    <dependencySet>
```

```
      <outputDirectory>lib</outputDirectory>
```

```
      <unpack>>false</unpack>
```

```
      <scope>runtime</scope>
```

```
    </dependencySet>
```

```
  </dependencySets>
```

```
</assembly>
```

Assembly format:
zip, gzip, tar, etc.

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

```
<assembly>
```

```
  <id>standalone</id>
```

```
  <formats>
```

```
    <format>zip</format>
```

```
  </formats>
```

```
  <includeBaseDirectory>false</includeBaseDirectory>
```

```
  <!-- copy assembly -->
```

```
  <fileSets>
```

```
    <fileSet>
```

```
      <directory>src/main/assembly/files</directory>
```

```
      <outputDirectory></outputDirectory>
```

```
      <includes>
```

```
        <include>**/*</include>
```

```
      </includes>
```

```
    </fileSet>
```

```
  </fileSets>
```

```
  <dependencySets>
```

```
    <!-- Grab dependencies libraries -->
```

```
    <dependencySet>
```

```
      <outputDirectory>lib</outputDirectory>
```

```
      <unpack>>false</unpack>
```

```
      <scope>runtime</scope>
```

```
    </dependencySet>
```

```
  </dependencySets>
```

```
</assembly>
```

Files to include in the assembly.
Here we are including all files
in src/main/assembly/files;
preserving directory structure

```
<?xml version="1.0" encoding="UTF-8"?>
<assembly>
  <id>standalone</id>
  <formats>
    <format>zip</format>
  </formats>
  <includeBaseDirectory>false</includeBaseDirectory>
  <!-- copy assembly files -->
  <fileSets>
    <fileSet>
      <directory>src/main</directory>
      <outputDirectory>target</outputDirectory>
      <includes>
        <include>**/*.class</include>
      </includes>
    </fileSet>
  </fileSets>
  <dependencySets>
    <!-- Grab dependencies libraries -->
    <dependencySet>
      <outputDirectory>lib</outputDirectory>
      <unpack>false</unpack>
      <scope>runtime</scope>
    </dependencySet>
  </dependencySets>
</assembly>
```

Here we are including the Runtime dependencies. They are put into *./lib* in the assembly

```
<?xml version="1.0" encoding="UTF-8"?>
<assembly>
  <id>standalone</id>
  <formats>
    <format>zip</format>
  </formats>
  <includeBaseDirectory>false</includeBaseDirectory>
  <!-- copy assembly files -->
  <fileSets>
    <fileSet>
      <directory>src/main/assembly/files</directory>
      <outputDirectory></outputDirectory>
      <includes>
        <include>**/*</include>
      </includes>
    </fileSet>
  </fileSets>
  <dependencySets>
    <!-- Grab dependencies libraries -->
    <dependencySet>
      <outputDirectory>lib</outputDirectory>
      <unpack>false</unpack>
      <scope>runtime</scope>
    </dependencySet>
  </dependencySets>
</assembly>
```

Executing Assembly

```
<build>
  <!-- Package assemblies -->
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <configuration>
      <descriptors>
        <descriptor>src/main/assembly/standalone.xml</descriptor>
      </descriptors>
    </configuration>
    <executions>
      <execution>
        <id>make-assembly</id>
        <phase>package</phase> <!-- append to the packaging phase -->
        <goals>
          <goal>attached</goal> <!-- goals == mojos -->
        </goals>
      </execution>
    </executions>
  </plugin>
  ...
```

Common Commands

- `mvn clean`
 - Clean project; delete target directory
- `mvn compile`
 - Just compiles the class
- `mvn test`
 - Compiles class and tests; executes tests
- `mvn test -Dtest=MyTestClass`
 - Execute a single named test

Common Commands

- `mvn install`
 - Install artifact in local repository. Makes artifact available to other local projects
- `mvn clean deploy`
 - Deploy to remote repository. Makes it artifact available to non-local projects
- `mvn site:site site:deploy`
 - Create and deploy web site
 - <http://www.javaworld.com/javaworld/jw-02-2006/jw-0227-maven.html>

Common Commands

- `mvn javadoc:javadoc`
 - Generate javadocs
- `mvn eclipse:eclipse`
 - Create eclipse project for artifact
- `mvn idea:idea`
 - Create IntelliJ IDEA project for artifact

Common Commands

- `mvn exec:java -Dexec.mainClass=foo.Class -Dexec.keepAlive=true`
 - Execute a java class.
 - Optional arguments
 - `-Dexec.arguments="arg1 arg2"`

Many Many Plugins Available

- <http://maven.apache.org/plugins/index.html>
- <http://mojo.codehaus.org/>

Profiles and Filters

- Maven supports filters and profiles.
Allowing for distinct builds when certain conditions are met.
- Good example at:

http://sujitpal.blogspot.com/2006_10_01_archive.html

Can use Maven from Ant

- Useful for managing dependencies.
 - I.e. Don't need to add 3rd party jars to your source repository.
 - Maven manages transitive dependencies
 - Can query Maven POM in order to construct dependency paths for scripts.

SNAPSHOTS

- Maven supports SNAPSHOT builds.

`<version>1.0-SNAPSHOT</version>`

Releases

- Maven has a release command that doesn't do the following:
- Removes SNAPSHOT
 - 1.0-SNAPSHOT becomes 1.0
- Tags release in version control system
- Deploys new release

IDE Integration

- Netbeans
 - MevenIDE
 - Best integration
- Eclipse
 - M2eclipse
- IDEA
 - Maven Reloaded

Site Generation

- Not much there by default
- Easy to add lots of good reports
- See

<http://www.javaworld.com/javaworld/jw-02-2006/jw-0227-maven.html>