Maven 3.2

Daniel Hinojosa

Maven 3.2

- During this course, we'll cover:
    - What is Maven
    - Introduction to Working with Maven
    - Key Maven concepts
    - Simple Maven Customizations
    - A Smattering of HelloWorlds
    - Working with Dependencies
    - Multimodule Projects
    - Simplifying the POM

What is Maven?

> "Software Project management and comprehension tool"

> "An attempt to apply patterns to a project's build infrastructure in order to promote comprehension and productivity" maven.apache.org

What is Maven?

- An automated build tool
- Focused on simplicity
    - Creation – generates intelligent "starters"
    - Management – assumes intelligence defaults
- Covers build-oriented phases in ALM
    - Build Management
    - Testing
    - Release Versioning
    - Deployment

What is Maven as a Project?

- An Apache Open Source project
- Development began in 2001
- Grew out of unwieldy Ant build files for other Apache projects
- Has gone through many iterations

- Current version is Maven 3.x

Key Maven Resources

- http://maven.apache.org – main site
- http://maven.apache.org/download.html – download site
- http://search.maven.org – “central” repository
- http://www.sonatype.com/books/mvnex-book/reference/public-book.html - Maven by Example*

Tenets of Maven

- Project oriented
- Convention over Configuration
- Dependency management
- Extensible through plug-ins
- Reuse through centralized repositories

Maven vs. Other Build Tools

- Simplicity
    - Maven assumes defaults for every project
    - Other’s assume every project is unique
- Project oriented
    - Maven assumes software project paradigm
    - Other’s assume task-oriented build scripts
- Lifecycle driven
    - Maven assumes standard build lifecycles
    - Other’s assume tasks

Project Oriented Builds

- Defined as XML in pom.xml
- Supports single-inheritance tree like Java
- Every pom extends the Super POM (bundled with Maven)

Maven vs. Ant Files

Maven vs. Ant File Comparison

Build Configuration Aspects	**Maven**	**Ant**
Build File Name	pom.xml	build.xml
Project Name	inside of pom.xml	inside of build.xml
Project Directory Structure	Automatic	Manual
Build Preparation	Automatic	Manual
Library Dependencies	Declarative/Automatic	Manual Download and Management
Compilation	Automatic	Manual
Testing	Automatic	Manual
Packaging	Automatic	Manual
Versioning	Automatic	Manual
Cleanup	Automatic	Manual

Introduction to Working with Maven

Topics

- Setting up your environment
- Key Maven Commands
- Your first Maven project

Setting up your environment

Preparing your environment

- Java Development Kit
  - Installed
  - Configured JAVA_HOME environment variable
- Internet Connection
  - Interacting with library repositories
  - Downloading Dependencies

Downloading and Installing Maven

- Download Maven
  - http://maven.apache.org/download
  - Choose .zip format
- Extract Maven
  - /usr/local/maven (for Linux/Unix/Mac)
  - C:\java\maven (for Windows)

Windows use may be easier if your folder does not contain spaces

Configuring Operating System Use for Maven

- Set M2_HOME Environment Variable
  - %> export M2_HOME=/usr/local/maven
  - C:\> set M2_HOME=C:\java\maven
- Modify/Append the PATH Variable
  - export PATH=$PATH:$M2_HOME/bin
  - set PATH=%PATH%;%M2_HOME%\bin

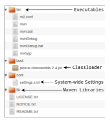
Note: You may want to configure these variables at a system level instead of each time you use the shell

Testing the Maven Installation

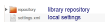
- %> mvn -v
- C:\ mvn -v

```
% mvn -v
Apache Maven 3.2.1 (ea8b2b07643dbb1b84b6d16e1f08391b666bc1e9; 2014-02-14T10:37:52-
07:00)
Maven home: /home/danno/java/apache-maven-3.2.1
Java version: 1.8.0_05, vendor: Oracle Corporation
Java home: /usr/lib/jvm/jdk1.8.0_05/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "3.13.0-27-generic", arch: "amd64", family: "unix"
```

Maven Installation ($M2_HOME)



Local Settings (USER_HOME/.m2 directory)



Lab: Install Maven

Key Maven Commands

Key Maven Commands

- Maven as an application has a limited number of commands
- “Commands” are a way of invoking the maven lifecycle
- Other “commands” are defined through plugins

Overview of Maven Plugins

- Core maven functionality is simplistic
    - Really just a **plugin execution framework**
    - Knows predefined lifecycle and how to execute plugins
    - Plugins are dynamically downloaded and installed
- Plugins encapsulate build related functionality
    - Defined by a name
    - Contains a set of goals (aka tasks or “commands”) Plugins are invoked using syntax:

```
%> mvn plugin_name:goal
```

Maven Help

If Maven is installed by a package manager like apt-get:

```
man mvn
```

For all other users:

```
mvn -h
```

Our first project

Quick overview of maven lifecycles

- Maven uses concept of a lifecycle for builds
- 4 key phases in a project’s lifecycle:
  - Creation – create a project
  - Compile – compile the project (and its test cases)
  - Test – perform automated testing
  - Package – bundle project into JAR, WAR, etc
- Phases
  - Build on each other
  - Phase name can be used like maven command

Group Id, Artifact Id, Version (GAV)

- [groupId]
  - Which group is responsible for the project?
  - Typically reverse domain name
  - (e.g. org.springframework, org.jboss, gov.treasury, com.h2database, com.oracle)
  - Some projects violate the rule
- [artifactId]
  - Name of the project
  - Typically a one word term
  - (e.g. spring-orm, joda-time, guava, jboss-seam)
- [version]
  - What version are you working on?
  - Default is 1.0-SNAPSHOT
  - Versions (X.X-SNAPSHOT, X.X-RELEASE) have meaning when releasing to Nexus or Artifactory

How to use http://search.maven.org/

Create command doesn’t exist

- No built in command for creating a project

```
% mvn create
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.062 s
[INFO] Finished at: 2014-06-09T17:01:41-07:00
[INFO] Final Memory: 5M/269M
[INFO] ------------------------------------------------------------------------
[ERROR] The goal you specified requires a project to execute but there is no POM in
this directory (/home/danno/java/apache-maven-3.2.1/bin). Please verify you invoked
Maven from the correct directory. -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the
following articles:
[ERROR] [Help 1]
http://cwiki.apache.org/confluence/display/MAVEN/MissingProjectException
```

Creation with archetype plugin

- Projects are created using the archetype plugin
  - Archetype has prototypes for many Java related projects
  - Archetype creates project structure and necessary starter files based on project type
- Archetypes have a number of different goals

Primary goal for project creation is generate:

```
%> mvn archetype:generate
C:\> mvn archetype:generate
```

NOTE mvn archetype:create goal has been deprecated

mvn archetype:generate goal

Basic call:

```
mvn archetype:generate
```

If you know what you exactly what you are looking for:

```
mvn archetype:generate
 -DarchetypeGroupId=...
 -DarchetypeArtifactId=...
 -DarchetypeVersion=...
```

If you have a fuzzy name of what you are looking for:

```
mvn archetype:generate -Dfilter=quickstart
```

Even easier way to find your favorite archetype

1. mvn archetype:generate
2. Wait for the list to complete
3. Type a fuzzy term of what you are looking for
4. Hit Enter
5. View the shorter filtered list
6. Select number of archetype you wish to use

Find your archetype

```
% mvn archetype:generate
[INFO] Scanning for projects...
[INFO]
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
Choose archetype:
1: remote -> br.com.ingenieux:elasticbeanstalk-service-webapp-archetype (A Maven
Archetype Encompassing R
estAssured, Jetty, Jackson, Guice and Jersey for Publishing JAX-RS-based Services on
AWS' Elastic Beansta
lk Service)
2: remote -> br.com.otavio.vraptor.archetypes:vraptor-archetype-blank (A simple
project to start with VRa
ptor 4)
....
1038: remote -> se.vgregion.javg.maven.archetypes:javg-minimal-archetype (-)
1039: remote -> sk.seges.sesam:sesam-annotation-archetype (-)
1040: remote -> tk.skuro:clojure-maven-archetype (A simple Maven archetype for
Clojure)
1041: remote -> tr.com.lucidcode:kite-archetype (A Maven Archetype that allows users
to create a Fresh Ki
te project)
1042: remote -> uk.ac.rdg.resc:edal-ncwms-based-webapp (-)
```

Choose your archetype

```
1038: remote -> se.vgregion.javg.maven.archetypes:javg-minimal-archetype (-)
1039: remote -> sk.seges.sesam:sesam-annotation-archetype (-)
1040: remote -> tk.skuro:clojure-maven-archetype (A simple Maven archetype for
Clojure)
1041: remote -> tr.com.lucidcode:kite-archetype (A Maven Archetype that allows users
to create a Fresh Kite project)
1042: remote -> uk.ac.rdg.resc:edal-ncwms-based-webapp (-)
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive
contains): 399:
```

The number provided is typically the quickstart project

Enter information about your project

```
Choose org.apache.maven.archetypes:maven-archetype-quickstart version:
1: 1.0-alpha-1
2: 1.0-alpha-2
3: 1.0-alpha-3
4: 1.0-alpha-4
5: 1.0
6: 1.1
Choose a number: 6:
Define value for property 'groupId': : com.xyzcorp
Define value for property 'artifactId': : basicmaven
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  com.xyzcorp: :
Confirm properties configuration:
groupId: com.xyzcorp
artifactId: basicmaven
version: 1.0-SNAPSHOT
package: com.xyzcorp
Y:
```

Enjoying Success

```
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
maven-archetype-quickstart:1.1
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: /home/danno/java/apache-maven-3.2.1/bin
[INFO] Parameter: package, Value: com.xyzcorp
[INFO] Parameter: groupId, Value: com.xyzcorp
[INFO] Parameter: artifactId, Value: basicmaven
[INFO] Parameter: packageName, Value: com.xyzcorp
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: /home/danno/java/apache-maven-
3.2.1/bin/basicmaven
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 02:32 min
[INFO] Finished at: 2014-06-09T17:22:09-07:00
[INFO] Final Memory: 12M/307M
[INFO] ------------------------------------------------------------------------
```

What just happened?

- Maven looks for required plugins and dependencies
  - Downloads if missing

    - Updates if needed
    - Places dependencies into local repository ~/.m2/repository
- Executes archetype:generate goal
    - Uses values entered or specified on command line
    - Creates artifactId directory (basic-maven)
    - Creates project files based on select artifact

Generated 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/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.xyzcorp</groupId>
  <artifactId>basicmaven</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>basicmaven</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Generated Code

```
package com.xyzcorp;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
    }
}
```

Generated Test

```
package com.xyzcorp;

import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

/**
 * Unit test for simple App.
 */
public class AppTest
    extends TestCase
{
    /**
     * Create the test case
     *
     * @param testName name of the test case
     */
    public AppTest( String testName )
    {
        super( testName );
    }

    /**
     * @return the suite of tests being tested
     */
    public static Test suite()
    {
        return new TestSuite( AppTest.class );
    }

    /**
     * Rigourous Test :-)
     */
    public void testApp()
    {
        assertTrue( true );
    }
}
```

Lab: Create basic maven app

Compile Phase

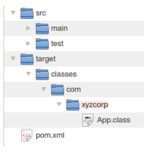
Seventh Phase of Project Lifecycle is Compilation

```
% mvn compile
```

What just happened?

- Maven looks for required plugins
    - Downloads them if neeeded
    - Updates if needed
- Executes compile lifecycle
    - Compiles source code found in src/main/java
    - Places compiled code in target/classes

What was created?



NOTE Test classes are not compiled

Running com.xyzcorp.basicmaven.App

Now that the application is built it can also be run:

```
java -cp target/classes com.xyzcorp.basicmaven.App
```

Alternate way of running com.xyzcorp.basicmaven.App

Run with the exec plugin

```
mvn:exec -Dexec.mainClass=com.xyzcorp.basicmaven.App
```

Lab: Compile and Run

Test Phase

Fifteenth phase in the project lifecycle is the testing phase

```
mvn test
```

What just happened

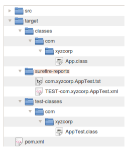
- Looked for and downloaded required plugins
- Looked for and downloaded required dependencies
- Executes the compile lifecycle phase
- Executes the test-compile lifecycle phase
    - Compiles code from src/test/java
    - Output placed in the target/test-classes
- Ran JUnit Tests (as default)

Where did JUnit come from?

pom.xml

```
<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>
  ...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

What was created?



Lab: Test

Package Phase

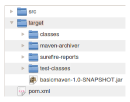
Seventeeth phase in the project lifecycle is packaging

```
mvn package
```

What just happened?

- Looked for and downloaded required plugins
- Looked for and downloaded required dependencies
- Executes the compile lifecycle phase
- Executes the test-compile lifecycle phase
- Executes the test lifecycle phase
- Ran JUnit Tests (as default)
- Creates project artifact (jar)
    - Uses target/classes
    - Creates a jar

What was created?



Where did the jar come from?

pom.xml

```
<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.xyzcorp</groupId>
  <artifactId>basicmaven</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  ...

</project>
```

Running com.xyzcorp.basicmaven.App as a jar

Now that the application is built as a it can also be run:

```
java -cp target/basic-maven-1.0-SNAPSHOT.jar com.xyzcorp.basicmaven.App
```

Lab: Run app as a jar

Understanding the Maven Lifecycle

Topics

In this section we will cover:

- Clean lifecycle
- Site lifecycle
- Default Build Lifecycle

Overview of the Maven Lifecycle

What is the Maven Lifecycle?

Process a build goes through:

- Lifecycle executed in terms of phases
    - Maven steps through a set of phases
    - Executions associated with phases
    - Executions defined in terms of plugin goals
- Lifecycle completes when all phases execute successfully

Lifecycle Definition

- Lifecyle built into Maven
    - Described in components.xml of Maven
    - Described in terms of <lifecycle>
- 3 standard life cycles defined

    - clean - attempts to clean up project build directory
    - site - used to generate site documentation
    - default - attempts to build the project
- Can modify standard lifecycles or create your own

Clean Lifecycle

- Simple lifecycle used to clean up build crumbs
- Contains 3 phases
    - pre-clean - preparation tasks
    - clean - deletion of artifacts
    - post-clean - post deletion tasks
- Invoked by performing mvn clean

clean phase

```
mvn clean
```

What just happened

- By default the target folder was deleted
- Default behavior can be changed

Invoking clean plugin

- Executed in the clean phase of clean lifecycle
    - Relies on clean plugin
    - Invokes the clean clean goal
- Invoking direct does not invoke any lifecycle process

Lab: Clean Lifecycle

Default Lifecycle

Default Lifecycle

Represents general build process for software

- Most comprehensive lifecycle
  - By default, not all phases have bound plugins
  - Different project types require different plugin goals
- Typically project packaging defines which plugins are bound to which phase

components.xml

- http://svn.apache.org/repos/asf/maven/maven-3/trunk/maven-core/src/main/resources/META-INF/plexus/components.xml
- Also inside of '$M2_HOME/lib/maven-core-3.2.1.jar' in 'META-INF/components.xml'

Each phase attached to a plugin

process-resources	resources:resources
compile	compiler:compile
process-test-resources	resources:testResources
test-compile	compiler:testCompile
test	surefire:test
package	jar:jar
install	install:install
deploy	deploy:deploy

Process Resources Phase

- Processes resources and copies them to the target folder
- Typically handled by the resources:resources goal
- Supports variable substitutions during processing through filters – turned off by default

Compile Phase

- Compiles source code and copies bytecode to output directory (target/classes)
- Typically handled by the compiler plugin
  - Uses the javac compiler

    - Compiles with -source and -target options
    - Most likely will need to modify build elements to support new versions of Java

Invoking the Compiler Plugin

- Executed in the compile lifecycle phase
    - Relies on the compiler plugin
    - Invokes the compiler:compile plugin and goal
    - Direct invocation does not invoke any lifecycle phases

```
mvn compiler:compile
```

Process Test Resources Phase

- Processes test resources and copies them to the target/test-classes folder
- Typically handled by the resources:testResources goal
- Supports variable substitutions during processing through filters – *turned off by default*

Test Compile Phase

- Compiles source code and copies bytecode to output directory (target/test-classes)
- Typically handled by the compiler:testCompile plugin and goal
    - Uses the javac compiler
    - Compiles with -source and -target options
    - Most likely will need to modify build elements to support new versions of Java

Test Phase

- Executes Automated Tests
    - Executes the surefire:test goal
        - Use JUnit by default; supports JUnit 3&4
        - Can be configured to use TestNG
        - Finds tests based on *Test naming convention
        - Build stops on failed test
    - Can skip test phase by invoking maven with -Dmaven.test.skip=true

Invoking the Surefire Plugin

- Executes in the test lifecycle phase
    - Executes the surefire:test goal
    - Direct invocation does not invoke any lifecycle phases

```
mvn surefire:test
```

Package Phase

- Packages build into an artifact (.jar, .war, .ear, etc)
- Typically uses the package plugin
    - Goal depends on packaging artifact type
    - Default configuration not specify Main-Class attribute when build a jar
    - Jar filename follows Maven convention

Invoking the Jar Plugin

- Executed in the package lifecycle phase
    - Relies on the jar:jar plugin and goal
    - Direct invocation does not invoke any lifecycle phases

Install Phase

- Installs project artifact (.jar, .war, .ear, etc) into the local repository
    - Makes the artifact available for other projects to use
    - Not the same as deploying to a “server”
- Typically uses the install:install goal

Deploy Phase

- “Releases” project artifact to remote Maven repository (for sharing)
- Typically uses deploy:deploy plugin
    - Not configured by default
    - Need to specify server settings in either pom.xml or settings.xml

Site Lifecycle (To be discussed later)

Summary

- Maven has three main lifecycles
  - Clean
  - Site
  - Default (build)
- Default lifecycle has different configurations based on packaging type

Lab: Install Basic Maven

Simple Maven Customizations

Topics

In this section we'll cover:

- General Overview of POM
- Working with Versions
- Configuring basic project information
- Using project properties
- Configuring build settings
- Configuring JAR Packaging

Super POM

- All maven projects extend the Super POM
  - Similar in concept to java.lang.Object
  - Describes a set of default behaviors and assumptions shared by all projects
  - Part of the maven "install"
- Super POM defines 4 key things:
  - Main maven repository – central repository
  - Main plugin repository – no automatic updates
  - Default build settings – project structure settings
  - Plugin management – list of "core" plugins

Lab: Review Super POM

General Overview of a POM

- POM stands for Project Object Model
- Describes a project in terms of identity, structure, dependencies, lifecycles, etc.
- Focused on descriptions not tasks
- Encapsulated in <project> … </project> tags of pom.xml, found in project directory
- Contains project-specific values that over-ride Super POM’s configuration

General Overview of a POM (Continued)

POM contains 4 categories of descriptions

- Basic project information – coordinates, dependencies, etc.
- Build settings – compiler configuration, etc.
- Optional project information – team information, licenses, etc.
- Environment settings – continuous integration, repositories, etc.

Customizing Your Project

Three way to customize your POM

- Modify the Super POM - **Bad Idea**
    - Unbundle 'maven-3.x.x-uber.jar'
    - Modify 'org/apache/maven/project/pom-4.0.0.xml'
    - Rebuild 'maven-3.x.x-uber.jar'
- Modify the project POM
    - Define project specific settings
    - Modify 'project_dir/pom.xml'
- Modify “global” local settings
    - '~/.m2/settings.xml'
    - Shared configuration across all user maven projects

Effective POM

Description of POM, including all inherited values

- Used by maven when performing goals

- Viewed using: mvn help:effective-pom

Lab: View the Effective POM

Modifying the Basics

General Project Information

Description of the Project

- Functions as a container of all project information
- Coordinates
- Packaging
- Dependencies
- Inheritance and project relationships
- Properties

POM Coordinates

UUID for project to maven and other projects

- Coordinates are used for things like:
    - Installing project in repository
    - Specifying dependencies
- Project coordinates are defined by:
    - groupId – company name or group working on project
    - artifactId – project name associated with group
    - version – specific release of project
    - packaging – the structure of the release artifact
- Coordinates written as colon delimited string

Coordinates in basic maven application

```
<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.xyzcorp.basicmaven</groupId>
      <artifactId>basicmaven</artifactId>
      <version>1.0-SNAPSHOT</version>
      <packaging>jar</packaging>
</project>
```

Versioning

- Snapshot versioning
  - Represents a project under development; not formally released
  - Version # contains SNAPSHOT
  - When a project has a dependency on a SNAPSHOT, maven will try to access most recent version
- Release versioning
  - Anything that does not contain SNAPSHOT
  - Project installed in a local or remote repository
  - When a project has a dependency on a released version, maven will not use newer or older versions

Modified Version

```
<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.xyzcorp.basicmaven</groupId>
      <artifactId>basicmaven</artifactId>
      <version>1.0</version>  <!--modified-->
      <packaging>jar</packaging>
</project>
```

Lab: Versioning

Adding Project Information

Organization Information

```
<organization>
  <name>DevelopIntelligence</name>
  <url>http://www.DevelopIntelligence.com</url>
</organization>
```

Team Information

```
<developers>
  <developer>
     <id>zorgdra1</id>
     <name>Kelby</name>
     <email>kelby@developintelligence.com</email>
     <roles><role>developer</role></roles>
  </developer>
</developers>
```

Adding New Project Information

```xml
<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.xyzcorp.basicmaven</groupId>
   <artifactId>basicmaven</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>jar</packaging>

   <name>basicmaven</name>
   <url>http://maven.apache.org</url>

   <organization>
      <name>DevelopIntelligence</name>
      <url>http://www.DevelopIntelligence.com</url>
   </organization>

   <developers>
      <developer>
        <id>zorgdra1</id>
        <name>Kelby</name>
        <email>kelby@developintelligence.com</email>
        <roles><role>developer</role></roles>
      </developer>
   </developers>
</project>
```

Lab: Adding Project Information

Working with Project Properties

Build Properties

- Properties are used to define build settings
  - Referenced using bean shell notation
  - If property can not be evaluated, its treated as literal string
- Five types of properties associated with a build
  - Project - ${project}
  - Environment - ${env}
  - Settings - ${settings}
  - Java System Properties – ${java.home}, ${os.name}

    - User Defined

Project Properties

- Settings associated with project’s POM
    - Implicit properties defined in super and parent poms
    - Explicit properties defined in project POM
- Referenced using ${project}
    - ${project.groupId}
    - ${project.artifactId}
    - ${project.organization.name}

Environment Properties

- Environment variables associated with OS
    - Maven evaluates notation to actual OS environment variable
    - Can be used to access shell or system level variables
- Referenced using ${env}
    - ${env.PATH}
    - ${env.M2_HOME}
    - ${env.HOSTNAME}

Settings Properties

- Settings defined in two places
    - '~/.m2/settings.xml'
    - 'M2_HOME/conf/settings.xml'
- Used to configure repositories, plugins, and profiles
- Referenced using ${settings}

Java System Properties

- Properties associated with java.lang.System
    - Contains all properties returned by System.getProperties()
    - Not to be confused with command line arguments
- Referenced using ${?} (depends on property)
    - ${java.home}

    - ${os.name}
    - ${jdbc.drivers}
    - ${myapp.myprop}

User Defined Properties

- Arbitrary Properties defined by you
    - Can be defined in 'pom.xml' or '~/.m2/settings.xml'
    - Defined using <properties> element
- Referenced using ${?} (depends on property)
- ${company-url}

User defined properties

```
<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.xyzcorp.basicmaven</groupId>
   <artifactId>basicmaven</artifactId>
   <version>1.0</version>
   <packaging>jar</packaging>

   <name>${project.artifactId}</name>
   <url>${company-url}</url>
   <description>
      ${project.name} is a simple maven project.
      The current version is ${project.version} and
      was built by ${user.name} on host ${env.HOSTNAME}
      The project was built with maven ${env.M2_HOME}
      and with version ${java.version}
   </description>

   <organization>...</organization>

   <developers>...</developers>

   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <company-name>DevelopIntelligence</company-name>
      <company-url>http://www.developintelligence.com</company-url>
   </properties>
</project>
```

Lab: Incorporate Properties

Dependencies

Dependencies

- Projects rely on other projects
- Maven simplifies dependency management
  - Manages internal (local) and external (remote) dependencies
  - Relies on repositories to locate dependencies
  - Uses transitive dependency management

Dependencies are declared in <dependencies> … </dependencies>

Dependency Definition

Default junit dependency

```
<dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>3.8.1</version>
   <scope>test</scope>
</dependency>
```

- Type
  - Typically associated with packaging type
  - Defaults to jar
- Scope – relates to classpath
  - compile– packaged with artifact(jar, war, ear, etc.)
  - provided – used for compile / test, but not packaged, available from server
  - runtime – needed only for execution, not packaged
  - test – needed only for testing; not packaged
  - system – not accessed from a repository but within the filesystem

Dependency Version Ranges

Qualifiers

- (,) - Exclusive Qualifier

- [,] - Inclusive Qualifier

Range	Meaning
(1.0]	x ⇐ 1.0
[1.2, 1.3]	1.2 ⇐ x ⇐ 1.3
[1.0, 2.0)	1.0 ⇐ x ⇐ 2.0
[1.5)	x >= 1.5

Upgrading a dependency

Updated junit dependency

```
<dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.10</version>
   <scope>test</scope>
</dependency>
```

Lab: Upgrade JUnit Dependency

Transitive Dependencies

- Dependency of a dependency
- Includes automatically the dependencies that your project relies

Including a Dependency that has Dependencies

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>4.0.5.RELEASE</version>
</dependency>
```

View a dependency list or tree with the dependency plugin

```
mvn dependency:tree
```

or

```
mvn dependency:list
```

Excluding Transitive Dependencies

- Dependency that a project depends on is unfit for the project
- Perhaps due to
  - License issues
  - Poor performance
  - Upgrade Available

Excluding Transitive Dependencies

- To exclude a Transitive Dependency
  - add <exclusion>..</exclusion>
  - add the projectId and artifactId that you need to exclude
  - no version is required

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>4.0.5.RELEASE</version>
    <exclusions>
        <exclusion>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

Verify that the transitive dependency is gone

```
mvn dependency:tree
```

or

```
mvn dependency:list
```

Lab: Excluding Dependency

Adding our own preferred dependency

- spring-core requires commons-logging
- We can provide a downgrade in case there is was something we didn't like about the latest version

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>4.0.5.RELEASE</version>
    <exclusions>
        <exclusion>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.1.2</version>
</dependency>
```

Lab: Add our own preferred dependency

Optional Dependencies

- Any dependencies that are not required when another project depends on your project
- "excluded by default"
- Marked with either:
    - <optional>true</optional>
    - ++<optional>false</optional>

Configuring Build Settings

Default Build Settings

- Maven assumes default build settings
  - Represent the “safest” bet
  - Probably need to “modify” for real project development
- Modify build settings in either:
  - Project’s 'pom.xml'
  - '~/.m2/settings.xml'
- Build settings over-ride settings found in Super POM

Basic Project Build Settings

- Configuration is done inside of <build>…</build>
- Can configure certain things:
  - <defaultGoal> - default behavior when type mvn
  - <sourceDirectory> - source directory if differs from src
  - <directory> - output directory, if differs from target
  - <finalName> - artifact output name
  - and more…

Specifying Default Goal

```
<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.xyzcorp.basicmaven</groupId>
   <artifactId>basicmaven</artifactId>
   <version>1.0</version>
   <packaging>jar</packaging>

   <name>${project.artifactId}</name>
   <url>${company-url}</url>
   <description>
      ${project.name} is a simple maven project.
      The current version is ${project.version} and was built
      by ${user.name} on host ${env.HOSTNAME}
      The project was built with maven ${env.M2_HOME}
      and with version ${java.version}
   </description>
   <build>
     <defaultGoal>install</defaultGoal>
   </build>
   ...
</project>
```

Lab: Modifying the Default Goal

Basic Project Build Settings

- Build configuration of project is done within <build>..</build>
- Along with some complex configurations
  - <plugins>
  - <resources>
  - <extensions> (Used with Wagon plugin)

Plugin Configuration

- Override plugin configuration inherited from Super POM
  - Allows you to define project specific behaviors
  - Or define project specific configurations
- Or, include other plugins as part of the build

Configuring a Plugin

- Plugin configuration uses:
    - <plugins> - container tag for all plugins
    - <plugin> - container tag for plugin
- Plugins are referenced by their coordinates
    - <groupId>
    - <artifactId>
    - <version>
- Configurations are specified with in the tags <configuration>..<configuration>

Configuring the Compiler

- Default compiler behavior:
    - javac –source 1.5 –target 1.5
    - Contains assertions, generics, annotations, etc.
- Can determine bytecode version of class using:
    - javap –verbose com.xyzcorp.basicmaven.App
    - file 'App.class'

Major Minor Compiler Versions

Major Version	Java Version
45	1.1
46	1.2
47	1.3
48	1.4
49	1.5
50	1.6
51	1.7
52	1.8

Compiler Plugin Example

```
<build>
  <defaultGoal>install</defaultGoal>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
           <artifactId>maven-compiler-plugin</artifactId>
           <version>2.5.1</version>
           <configuration>
             <source>1.8</source>
             <target>1.8</target>
           </configuration>
      </plugin>
   </plugins>
</build>
```

Configuring a test behavior

- Default behavior for maven lifecycle test phase
  - Tests are executed using the surefire plugin
  - If a test fails, build stops
- May want to modify this behavior to not fail
  - Will allow maven to execute all lifecycles
  - Including the generation of an artifact

Surefire Configuration Example

```
<build>
    <defaultGoal>install</defaultGoal>
    <plugins>
       <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>2.5.1</version>
          <configuration>
             <source>1.8</source>
             <target>1.8</target>
          </configuration>
       </plugin>

       <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.12.4</version>
          <configuration>
             <testFailureIgnore>true</testFailureIgnore>
          </configuration>
       </plugin>
    </plugins>
</build>
```

Configuring Jar Behavior

- Default behavior for maven lifecycle jar phase
    - Creates a non-transitive jar
    - “Library” oriented
- May want to modify this behavior to
    - Generate executable jar
    - Need to specify main class attribute

Site Lifecycle

- Four phases of site lifecycle
    - pre-site
    - site
    - post-site
    - site-deploy
- Site plugin is bound to site lifecycle
    - site - site:site

    - site-deploy - site:deploy
- As of version 3.0 site plugin is required

Setting up the Site in Maven 3.0

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-site-plugin</artifactId>
      <version>3.2</version>
    </plugin>
  </plugins>
</build>
```

NOTE At this point it isn’t going to do much

Adding project-info-reports-plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-site-plugin</artifactId>
      <version>3.2</version>
      <configuration>
        <reportPlugins>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-project-info-reports-plugin</artifactId>
            <version>2.5.1</version>
            <reports>
               <!--place favorite reports here -->
            </reports>
          </plugin>
       </reportPlugins>
      </configuration>
    </plugin>
  </plugins>
</build>
```

What kind of reports are available for maven-project-info-reports-plugin

- index
- plugin-management
- distribution-management
- dependency-info
- dependency-convergence
- scm
- mailing-list
- issue-tracking
- cim
- plugins
- license
- modules
- dependency-management
- project-team
- summary
- dependencies

Information

Most of the reports required for the project-info-reports-plugin require that certain project information be filled in for it to result in something meaningful

Adding more reports into project-info-reports-plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-site-plugin</artifactId>
      <version>3.2</version>
      <configuration>
        <reportPlugins>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-project-info-reports-plugin</artifactId>
            <version>2.5.1</version>
            <reports>
              <report>index</report>
              <report>dependencies</report>
              <report>summary</report>
            </reports>
          </plugin>
       </reportPlugins>
      </configuration>
    </plugin>
  </plugins>
</build>
```

What kinds of other report plugins are available directly by Maven?

Plugin	Version
maven-javadoc-plugin	2.9
maven-project-info-reports-plugin	2.5.1
maven-surefire-report-plugin	2.12.4
maven-jxr-plugin	2.3
maven-changelog-plugin	2.2
maven-changes-plugin	2.8
maven-checkstyle-plugin	2.9.1
maven-plugin-plugin	3.1
maven-pmd-plugin	2.7.1

What kinds of reports are available from Mojo Community?

Plugin	Version
cobertura-maven-plugin	2.5.2
emma-maven-plugin	1.0-alpha-3

Where to apply the other report plugins?

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-site-plugin</artifactId>
      <version>3.2</version>
      <configuration>
        <reportPlugins>
          <plugin>...</plugin>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-javadoc-plugin</artifactId>
            <version>2.9</version>
          </plugin>
        </reportPlugins>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Lab:Site

Resource Filtering

What is resource filtering?

- Process of performing variable replacement on project resources
- Resources can be:
  - Properties files
  - Meta description files ('web.xml', 'application.xml', etc.)
  - And more

- Variable substitutions based on property values specified in POM

How to perform resource filtering?

- Define resource location
- Default locations for resources
  - src/main/resources
  - src/site/resources
  - src/test/resources
- Use bean shell notation ${bean.property}
- Create user-defined properties in POM

How to perform resource filtering (continued)?

- Enable resource filtering in POM

```
<build>
  <resources>
    <resource>src/main/resources</resource>
    <filtering>true</filtering>
  </resources>
</build>
```

- Perform any phase mvn process-resources or later

'pom.xml' with resource filtering

```
<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>
   ...
   <build>
      ...
      <resources>
         <resource>
            <directory>src/main/resources</directory>
            <filtering>true</filtering>
         </resource>
      </resources>
      ...
   </build>
</project>
```

Including Resources

- Current resource configuration includes all files.
- To include a subset of files

```
<resource>
  <directory>src/my-resources</directory>
  <includes>
    <include>**/*.txt</include>
    <include>**/*.rtf</include>
  </includes>
</resource>
```

Excluding Resources

To exclude a subset of files

```
<resource>
  <directory>src/my-resources</directory>
  <excludes>
    <exclude>**/*.java</exclude>
    <exclude>**/*.class</exclude>
  </excludes>
</resource>
```

Can use both <includes> and <excludes>

Lab: Simple Resource Filtering

Profiles

What is a Build Profile?

- Additional build settings for project
    - Can still define overall settings
    - Build profile will override settings
- Used to define specific settings for specific environments
    - Development server settings
    - Test server settings
    - Production server settings
- Support configuration reuse (portability)

How do you define a Build Profile?

```
<profiles>
  <profile>
   <id>dev</id>
     ...
  </profile>
  <profile>
   <id>prod</id>
     ...
  </profile>
</profiles>
```

Note: Every profile requires an id

What can a profile define?

- Nearly everything found in a <project> and <build>
    - <dependencies>
    - <properties>
    - <plugins>
    - and more

Where can I put a profile?

- Can be specified in:
    - The project pom.xml
    - Your personal settings.xml
    - Global settings.xml (Not Recommended)

Building a Profile

- Project is not built with a profile by default
- Need to activate a profile to use it
  - Use the –P flag when invoking a lifecycle
  - mvn package –Pcmd-line (with profile)
  - mvn install (without profile)
- Profiles can be automatically activated

Lab: Build a simple profile

Automatic Profile Activation

- Invoking with -P requires explicit activation
- Less ideal
- Build profiles can be activated automatically

Defining Automatic Activation

- Automatic <activation> can be based on:
- Simple configuration:
  - JDK values
  - OS values
  - Existence of properties and/or files
  - Any Combination of the above (or relationship)

Simple Automatic Activation

Include <activation> element in profile

```
<profile>
  <activation>
     <activeByDefault>true</activeByDefault>
  </activation>
</profile>
```

JDK Activation

Activates automatically on JDK version

```
<profile>
 ...
 <activation>
   <activeByDefault>false</activeByDefault>
   <jdk>1.6</jdk>
 </activation>
 ...
</profile>
```

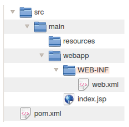
Multimodule Projects Live Project Example

Generating a Webapp

Using an Archetype to Generate a Webapp

```
mvn archetype:generate -Dfilter=maven-archetype-webapp
```

Web App Folder Contents



What happened?

- Generates project based on basic web template
- Contains webapp directory in src/main
- webapp contains all html, css, image resources

War Packaging Type

```
<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">
  ...
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  ...
</project>
```

Working with a web application

Default lifecycles are available

- process-resources
- compile
- process-test-resources
- test-compile
- test
- mvn package (creates war file instead)
- mvn install
- mvn deploy

Running a simple webapp

- WARs require a servlet container to run:
  - Can copy 'target/simple-webapp.war' to webapps directory of servlet container
  - Configure a servlet container plugin with the build
- Two common servlet containers:
  - Jetty
  - Tomcat

Configuration of the Jetty Plugin

Need to add a plugin to the <build>

- Find Jetty plugin information in Maven Central

```
<groupId>org.mortbay.jetty</groupId>
<artifactId>maven-jetty-plugin</artifactId>
<version>6.1.10</version>
```

Add <plugin> elements

```
<plugin>
   <groupId>org.mortbay.jetty</groupId>
   <artifactId>maven-jetty-plugin</artifactId>
   <version>6.1.10</version>
</plugin>
```

Configuration of the Jetty Plugin with Enhancements

```
<plugin>
   <groupId>org.mortbay.jetty</groupId>
   <artifactId>maven-jetty-plugin</artifactId>
   <version>6.1.10</version>
   <configuration>
           <scanIntervalSeconds>10</scanIntervalSeconds>
   </configuration>
</plugin>
```

Five Jetty Plugin Goals

- deploy-war
- run
- run-exploded
- run-war
- stop

Tomcat Plugin

Tomcat Plugin

Currently two plugins for Tomcat 6 and 7

```
<plugin>
     <groupId>org.apache.tomcat.maven</groupId>
     <artifactId>tomcat6-maven-plugin</artifactId>
     <version>2.1</version>
</plugin>
```

```
<plugin>
     <groupId>org.apache.tomcat.maven</groupId>
     <artifactId>tomcat7-maven-plugin</artifactId>
     <version>2.1</version>
</plugin>
```

Plugin Groups

Plugin Groups:

- Contains a list of pluginGroup elements, each contains a groupId.
- The list is searched when a plugin is used and the groupId is not provided in the command line
- Automatically contains:
  - org.apache.maven.plugins
  - org.codehaus.mojo

Plugin Groups in 'settings.xml'

In 'settings.xml'

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  ...
  <pluginGroups>
    <pluginGroup>org.mortbay.jetty</pluginGroup>
    <pluginGroup>org.apache.tomcat.maven</pluginGroup>
  </pluginGroups>
  ...
</settings>
```

This gives the user the ability to make the call jetty:run or tomcat6:run or tomcat7:run at the command line.

Using the Jetty/Tomcat Plugins

The full command to run any plugin and a goal is:

```
mvn <groupId>:<artifactId>:<version>:<goal>
```

For example:

```
mvn org.apache.maven.plugins:maven-jar-plugin:2.2:jar
```

Extended Lab: Generating Ear File, War File, Jar File, Deploy to Web Logic

Extended Lab: Deploying to Artifactory or Nexus

Integration with Jenkins