Introduction to Maven

Outline

- Introduce Maven
- Basic Maven Pom File and Project Structure
- Dependencies

Maven Background

- Is a Java build tool
  - "project management and comprehension tool"
- An Apache Project
- History
  - Maven 1 (2003)
    - Very Ugly
  - Maven 2 (2005)
    - Complete rewrite
    - Not backwards Compatible
  - Maven 3 (2010)
    - Same as Maven 2 but more stable

Maven Features

- Dependency System
- Consistent project structure
- Consistent build model
- Plugin oriented

The Maven Mindset

- All build systems are essentially the same:
  - Compile Source code
  - Run Tests
  - Package Project
  - Deploy Project
  - Cleanup

Other Java Build Tools

- Ant (2000)
  - Granddaddy of Java Build Tools
  - Scripting in XML
  - Very flexible
- Ant+Ivy (2004)
  - Ant but with Dependency Management
- Gradle (2008)
  - Attempt to combine Maven structure with Groovy Scripting
  - Easily extensible
  - Immature

Learning Resources

- Maven Homepage
  - http://maven.apache.org
    - Reference Documentation for Maven
    - Reference Documentation for core Plugins

Maven POM

- Stands for Project Object Model
- Describes a project
  - Name and Version
  - Artifact Type
  - Source Code Locations
  - Dependencies
  - Plugins
- Uses XML by Default

Project Name

- Maven uniquely identifies a project using:
  - groupID: Arbitrary project grouping identifier (no spaces or colons)
    - Usually loosely based on Java package
  - artfiactId: Arbitrary name of project (no spaces or colons)
  - version: Version of project
    - Format {Major}.{Minor}.{Maintanence}
    - Add '-SNAPSHOT ' to identify in development
- Syntax: groupId:artifactId:version

Packaging

- Build type identified using the “packaging” element
- Tells Maven how to build the project
- Example packaging types:
  - pom, jar, war, ear, custom
  - Default is jar

Maven Conventions

- target: Default work directory
- src: All project source files go in this directory
- src/main: All sources that go into primary artifact
- src/test: All sources contributing to testing project
- src/main/java: All java source files
- src/test/java: All java test source files

Maven Build Lifecycle

- A Maven build follow a lifecycle
- Default lifecycle
  - generate-sources/generate-resources
  - compile
  - test
  - package
  - integration-test (pre and post)
  - Install
  - deploy
- There is also a Clean lifecycle

Example Maven Goals

- To invoke a Maven build you set a lifecycle “goal”
- mvn install
  - Invokes generate* and compile, test, package, integration-test, install
- mvn clean
  - Invokes just clean
- mvn clean compile
  - Clean old builds and execute generate*, compile
- mvn compile install
  - Invokes generate*, compile, test, integration-test, package, install
- mvn test clean
  - Invokes generate*, compile, test then cleans

Summary

- Maven is a different kind of build tool
- It is easy to create multi-module builds
- Dependencies are awesome