



Maven



Web Site:

www.soebes.com

E-Mail:

gearconf2011@soebes.com

Dipl.Ing.(FH) Karl Heinz Marbaise

Agenda

1. Overview
2. Basic Concepts
3. First Project
4. Configuration
5. Advanced Features
6. Release Module/Artifact

1. Overview

- Official Web Site
 - <http://maven.apache.org>
- Currently Maven 3.0.3 is most recent version
 - Maven 3.0 is taking over.
 - Maven 2.2.1 widely used.
 - Maven 2.0.11 existing but should be replaced with at least with 2.2.1 but better migrate to Maven 3.0.

1. Overview

- Installation Requirements
 - Sun Java JDK 1.5+ (JRE does not work!)
 - Recommended: Java Sun 1.6 JDK
 - Download from the Web-Site:
 - `apache-maven-3.0.3-bin.tar.gz` (Unix like)
 - `apache-maven-3.0.3-bin.zip` (Windows)
 - Unpack the archive and put the bin folder into your path.
 - May be you have to set `JAVA_HOME`



<http://maven.apache.org/download.html>

1. Overview

What is Maven?

- Build and deployment tool
- “Software Project Management and comprehension tool”.
- Unification of the build process
 - If you know one you know all maven projects.
- “Best Practices” in Java development

1. Overview

Maven Features

- Management of dependencies
 - Simple as well as transitive dependencies.
 - Your project uses the Tika library.
 - What about the dependencies of the Tika library?
 - This is handled by Maven which is called “**transitive dependencies**”.



Note: [Dependency Mechanism Guide](#)

1. Overview

Maven Features

```
org.apache.tika:tika-parsers:bundle:0.7
+- org.apache.tika:tika-core:jar:0.7:compile
+- org.apache.commons:commons-compress:jar:1.0:compile
+- org.apache.pdfbox:pdfbox:jar:1.1.0:compile
| +- org.apache.pdfbox:fontbox:jar:1.1.0:compile
| \- org.apache.pdfbox:jempbox:jar:1.1.0:compile
+- org.bouncycastle:bcmail-jdk15:jar:1.45:compile
+- org.bouncycastle:bcprov-jdk15:jar:1.45:compile
+- org.apache.poi:poi:jar:3.6:compile
+- org.apache.poi:poi-scratchpad:jar:3.6:compile
+- org.apache.poi:poi-ooxml:jar:3.6:compile
| +- org.apache.poi:poi-ooxml-schemas:jar:3.6:compile
| | \- org.apache.xmlbeans:xmlbeans:jar:2.3.0:compile
| \- dom4j:dom4j:jar:1.6.1:compile
|   \- xml-apis:xml-apis:jar:1.0.b2:compile
+- org.apache.geronimo.specs:geronimo-stax-api_1.0_spec:jar:1.0.1:compile
+- commons-logging:commons-logging:jar:1.1.1:compile
+- org.ccil.cowan.tagsoup:tagsoup:jar:1.2:compile
+- asm:asm:jar:3.1:compile
+- log4j:log4j:jar:1.2.14:compile
+- junit:junit:jar:3.8.1:test
+- org.mockito:mockito-core:jar:1.7:test
| +- org.hamcrest:hamcrest-core:jar:1.1:test
| \- org.objenesis:objenesis:jar:1.0:test
\- com.drewnoakes:metadata-extractor:jar:2.4.0-beta-1:compile
```



1. Overview

Maven Features

- Creation of a project site
- Dependencies, Reports, quality information
 - JavaDoc.
 - Support with change logs from VCS*.
 - Cross references.
 - Mailing list.
 - Unit Test reporting.
 - Etc.

* Version Control System

1. Overview

Maven Features - Site

Subversion Authentication Parse Module

Last Published: 2011-01-15 | Version: 0.3

Overview

- Introduction
- Examples
- Load File
- Features

Project Documentation

- Project Information
 - About
 - Continuous Integration
 - Dependencies
 - Distribution Management
 - Issue Tracking
 - Mailing Lists
 - Plugin Management
 - Project License
 - Project Plugins
 - Project Summary
 - Project Team
 - Source Repository
- Project Reports



Subversion Authentication Parser Module

The idea is to read the [Subversion authentication file](#) and convert it into a object tree which can be used for applications and to handle their permissions.

History

The original idea for this module based upon the [SupoSE project](#) which reads the contents of the whole repository but has to limit the results of queries to the appropriate information which are allowed for particular users.

Usage

If you like to use the Subversion Authentication Parser Module (SAPM) inside your application the simplest way is (if you are using [Maven](#)) to define the dependency in your [pom.xml](#) and use the classes of the module otherwise you have to download the jar file and put it into your classpath.

```
<project>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>com.soebes.subversion.sapm</groupId>
        <artifactId>sapm</artifactId>
        <version>0.3</version>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

Copyright © 2010-2011 SoftwareEntwicklung Beratung Schulung. All Rights Reserved.

1. Overview

Maven Features - Site

Subversion Authentication Parse Module

Last Published: 2011-01-15 | Version: 0.3

Overview

[Introduction](#)
[Examples](#)
[Load File](#)
[Features](#)

Project Documentation

[Project Information](#)
▼ **Project Reports**
 [Changes Report](#)
 [Cobertura Test Coverage](#)
 [CPD Report](#)
 [JavaDocs](#)
 [JDepend](#)
 [PMD Report](#)
 [Source Xref](#)
 [Surefire Report](#)
 [Test JavaDocs](#)
 [Test Source Xref](#)



Generated Reports

This document provides an overview of the various reports that are automatically generated by [Maven](#) . Each report is briefly described below.

Overview

Document	Description
Changes Report	Changes Report on Releases of the Project.
Cobertura Test Coverage	Cobertura Test Coverage Report.
CPD Report	Duplicate code detection.
JavaDocs	JavaDoc API documentation.
JDepend	JDepend traverses Java class file directories and generates design quality metrics for each Java package. JDepend allows you to automatically measure the quality of a design in terms of its extensibility, reusability, and maintainability to manage package dependencies effectively.
PMD Report	Verification of coding rules.
Source Xref	HTML based, cross-reference version of Java source code.
Surefire Report	Report on the test results of the project.
Test JavaDocs	Test JavaDoc API documentation.
Test Source Xref	HTML based, cross-reference version of Java test source code.

Copyright © 2010-2011 SoftwareEntwicklung Beratung Schulung. All Rights Reserved.

1. Overview

Maven Features

- The “Best Practice” for Java projects
 - Separate locations for production- and test code
 - Naming conventions for unit tests
 - Support for unit and integration tests
 - Separate test setup

1. Overview

Maven Features

- Distribution of artifacts via:
 - Maven Central Repository
 - Your company Repository
- Defined release cycle via:

`mvn release:prepare release:perform`

1. Overview

Maven Features

- Maven can be enhanced via plugin's:
- There are already existing a large number of plugins* to support different things.
 - EAR, License Header Checks, SCM, Versions, RPM, ejb, WAR, cobertura, changes, cargo etc.
 - If you really need to you can write your own plugin(s).

*Plugin list:

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

<http://mojo.codehaus.org/plugins.html>

Don't forget Google and friends ;-)

1. Overview

Maven Features

- Maven is written 100% in pure Java so it's platform independent.
- Works on
 - Windows
 - Linux
 - Mac OS X
 - And much more.....

2. Basic Concepts

Conventions over Configuration

- “**Convention over Configuration**” paradigm
- Configure only in those cases where it's really needed and if you have a good reason **not** to use the defaults.
 - Not so good reasons are*:
 - Different locations for production and test
 - Using different databases for prod, test and dev.
 - Etc.

* We will later discuss how to solve this in a more “*Mavenized*” way.

2. Basic Concepts

The Maven Coordinates

- Identification of artifacts in Maven is based on the Maven coordinates which comprises of the groupId, artifactId and version (GAV)
- **groupId**
 - Group, company, organization
 - Typically the reverse domain of the organization or company
- **artifactId**
 - Name of the artifact must be unique within the groupId
- **version**
 - Version of the artifact

2. Basic Concepts

The Maven Coordinates

- **packaging**
 - Packaging type like jar (default), war, ear, pom etc.
- **classifier**
 - Things like jdk15, bin etc.

2. Basic Concepts

The Maven Coordinates

- Version
 - The pattern
<major-version>.<minor-version>.<incremental version>-<qualifier>
 - We have versions like:
 - 1.3.2, 2.5.6, 0.1.4, 1.0, 0.7
 - 1.3.2-alpha-1, 3.0.2-RC1
 - These are released versions.

2. Basic Concepts

The Maven Coordinates

- Version
 - And we have versions like
 - 1.3.2-SNAPSHOT, 2.5.6-SNAPSHOT
 - 0.1.4-SNAPSHOT, 1.0-SNAPSHOT
 - 0.7-SNAPSHOT
 - These versions are currently under development.
 - They are going towards the versions without the “-SNAPSHOT”

2. Basic Concepts

The Maven Coordinates

- This means that an artifact will be identified based on the following pattern:

groupId:artifactId-classifier:packaging:version

Examples:

org.bouncycastle:bcprov-jdk15:jar:1.45

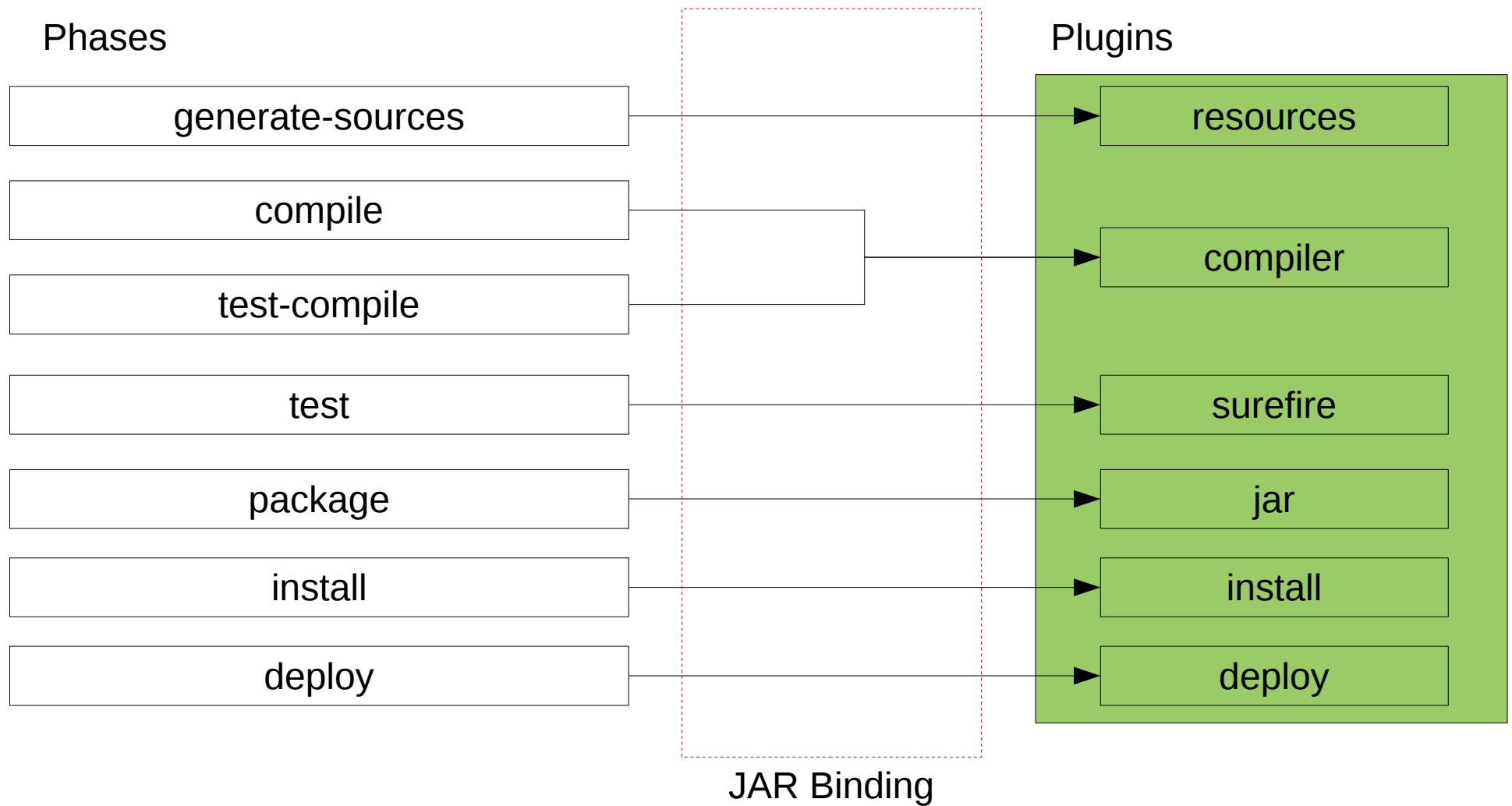
2. Basic Concepts Build Life-Cycle

- This simplified life cycle (default) contains the following phases:
 - validate validates the project.
 - compile compile the code.
 - test unit test the code.
 - package create the package.
 - verify checking of quality.
 - install installation into local repository.
 - deploy installation into remote repository.

Note: [Life cycle documentation](#)

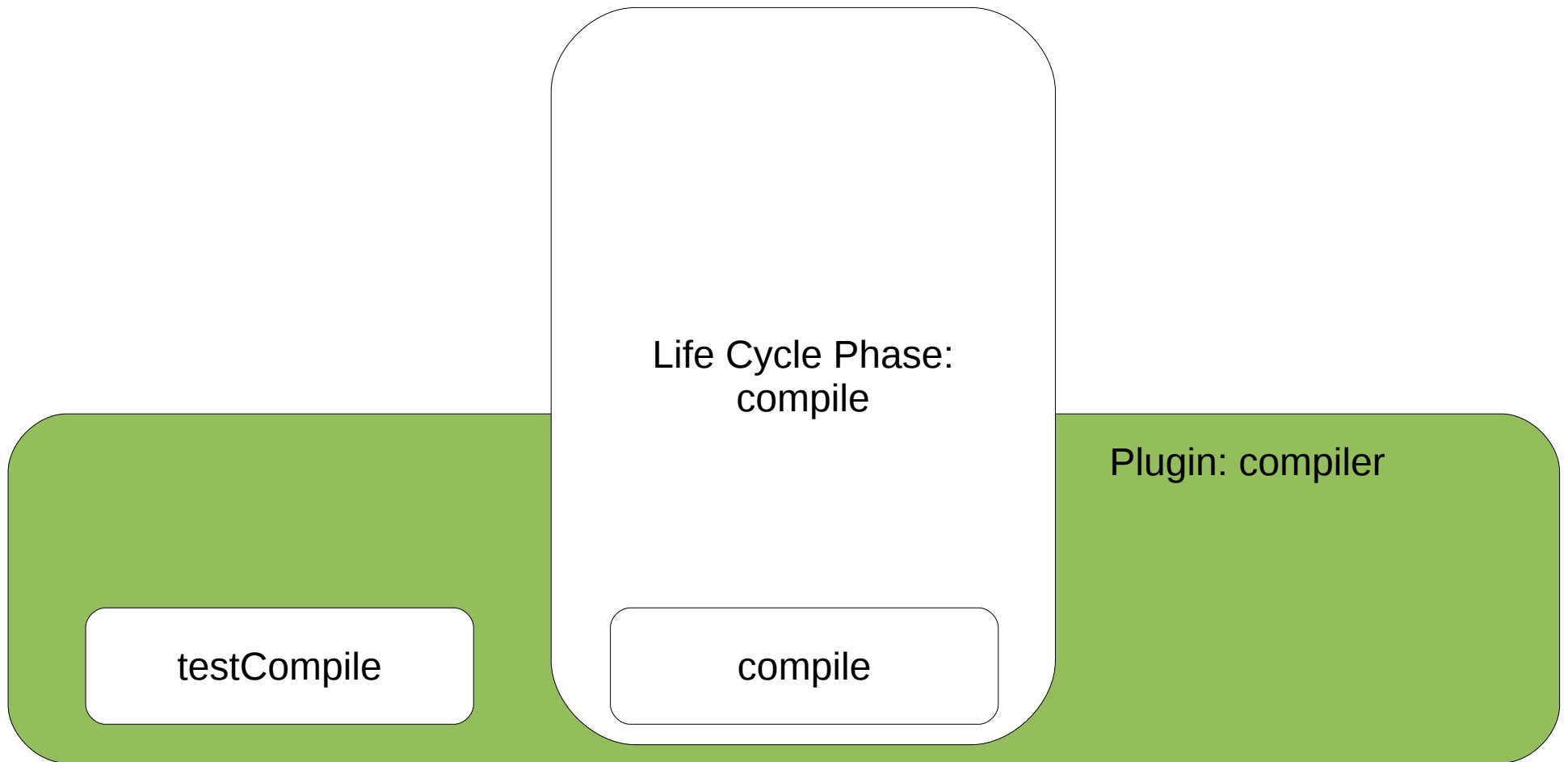
2. Basic Concepts

Plugins and the Life-Cycle



2. Basic Concepts

Plugin Goals Relationship



2. Basic Concepts Life-Cycle Calling

- Usually you will call a Maven life cycle phases

For example:

```
mvn clean
```

```
mvn package
```

```
mvn compile
```

```
mvn verify
```

```
mvn site
```


2. Basic Concepts

Calling Goals

- You can call goals of plugins if you really need to but usually you don't.

```
mvn plugin:goal
```

```
mvn compiler:compile
```

```
mvn compiler:testCompile
```

```
mvn jar:jar
```

- **Exceptions:**

```
mvn release:prepare release:perform
```

```
mvn help:help
```

2. Basic Concepts

The folder layout (convention)

- The folder layout is defined like the following:

```
MyProject
+-- pom.xml
+-- src
    +-- main
        !     +-- java
        !     +-- resources
        !     +-- javadoc
        !     +-- webapp
        !
    +-- test
        !     +-- java
        !     +-- resources
        !     +-- javadoc
        !
    +-- site
        +-- apt
        +-- xdoc
```

2. Basic Concepts

Project Object Model

- The “Project Object Model” (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/maven-v4_0_0.xsd">
```

```
  <modelVersion>4.0.0</modelVersion>
```

```
  <groupId>com.soebes.training.maven</groupId>
```

```
  <artifactId>first-artifact</artifactId>
```

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

```
  <name>The name of the component</name>
```

```
  <url>http://www.soebes.com</url>
```

```
  .
```

```
  .
```

```
</project>
```

2. Basic Concepts

Project Object Model

```
<groupId>...</groupId>  
<artifactId>...</artifactId>  
<version>...</version>  
<packaging>...</packaging>  
<dependencies>...</dependencies>  
<parent>...</parent>  
<dependencyManagement>...</dependencyManagement>  
<modules>...</modules>
```

Basic configuration
coordinates and modules.

```
<name>...</name>  
<description>...</description>  
<url>...</url>  
<inceptionYear>...</inceptionYear>  
<licenses>...</licenses>  
<organization>...</organization>  
<developers>...</developers>  
<contributors>...</contributors>  
<mailingLists>...</mailingLists>
```

```
<build>...</build>  
<reporting>...</reporting>
```

```
<issueManagement>...</issueManagement>  
<ciManagement>...</ciManagement>  
<scm>...</scm>
```

```
<prerequisites>...</prerequisites>  
<distributionManagement>...</distributionManagement>  
<profiles>...</profiles>  
<properties>...</properties>  
<repositories>...</repositories>  
<pluginRepositories>...</pluginRepositories>
```

2. Basic Concepts

Project Object Model

```
<groupId>...</groupId>  
<artifactId>...</artifactId>  
<version>...</version>  
<packaging>...</packaging>  
<dependencies>...</dependencies>  
<parent>...</parent>  
<dependencyManagement>...</dependencyManagement>  
<modules>...</modules>
```

```
<name>...</name>  
<description>...</description>  
<url>...</url>  
<inceptionYear>...</inceptionYear>  
<licenses>...</licenses>  
<organization>...</organization>  
<developers>...</developers>  
<contributors>...</contributors>  
<mailingLists>...</mailingLists>
```

```
<build>...</build>  
<reporting>...</reporting>
```

```
<issueManagement>...</issueManagement>  
<ciManagement>...</ciManagement>  
<scm>...</scm>
```

```
<prerequisites>...</prerequisites>  
<distributionManagement>...</distributionManagement>  
<profiles>...</profiles>  
<properties>...</properties>  
<repositories>...</repositories>  
<pluginRepositories>...</pluginRepositories>
```

Project information, licenses,
developers, mailing list etc.

2. Basic Concepts

Project Object Model

```
<groupId>...</groupId>  
<artifactId>...</artifactId>  
<version>...</version>  
<packaging>...</packaging>  
<dependencies>...</dependencies>  
<parent>...</parent>  
<dependencyManagement>...</dependencyManagement>  
<modules>...</modules>
```

```
<name>...</name>  
<description>...</description>  
<url>...</url>  
<inceptionYear>...</inceptionYear>  
<licenses>...</licenses>  
<organization>...</organization>  
<developers>...</developers>  
<contributors>...</contributors>  
<mailingLists>...</mailingLists>
```

```
<build>...</build>  
<reporting>...</reporting>
```

```
<issueManagement>...</issueManagement>  
<ciManagement>...</ciManagement>  
<scm>...</scm>
```

```
<prerequisites>...</prerequisites>  
<distributionManagement>...</distributionManagement>  
<profiles>...</profiles>  
<properties>...</properties>  
<repositories>...</repositories>  
<pluginRepositories>...</pluginRepositories>
```

build configuration, directories,
plugins, reporting

2. Basic Concepts

Project Object Model

```
<groupId>...</groupId>  
<artifactId>...</artifactId>  
<version>...</version>  
<packaging>...</packaging>  
<dependencies>...</dependencies>  
<parent>...</parent>  
<dependencyManagement>...</dependencyManagement>  
<modules>...</modules>
```

```
<name>...</name>  
<description>...</description>  
<url>...</url>  
<inceptionYear>...</inceptionYear>  
<licenses>...</licenses>  
<organization>...</organization>  
<developers>...</developers>  
<contributors>...</contributors>  
<mailingLists>...</mailingLists>
```

```
<build>...</build>  
<reporting>...</reporting>
```

```
<issueManagement>...</issueManagement>  
<ciManagement>...</ciManagement>  
<scm>...</scm>
```

```
<prerequisites>...</prerequisites>  
<distributionManagement>...</distributionManagement>  
<profiles>...</profiles>  
<properties>...</properties>  
<repositories>...</repositories>  
<pluginRepositories>...</pluginRepositories>
```

Issue management, ci management,
scm configuration.

2. Basic Concepts

Project Object Model

```
<groupId>...</groupId>  
<artifactId>...</artifactId>  
<version>...</version>  
<packaging>...</packaging>  
<dependencies>...</dependencies>  
<parent>...</parent>  
<dependencyManagement>...</dependencyManagement>  
<modules>...</modules>
```

```
<name>...</name>  
<description>...</description>  
<url>...</url>  
<inceptionYear>...</inceptionYear>  
<licenses>...</licenses>  
<organization>...</organization>  
<developers>...</developers>  
<contributors>...</contributors>  
<mailingLists>...</mailingLists>
```

```
<build>...</build>  
<reporting>...</reporting>
```

```
<issueManagement>...</issueManagement>  
<ciManagement>...</ciManagement>  
<scm>...</scm>
```

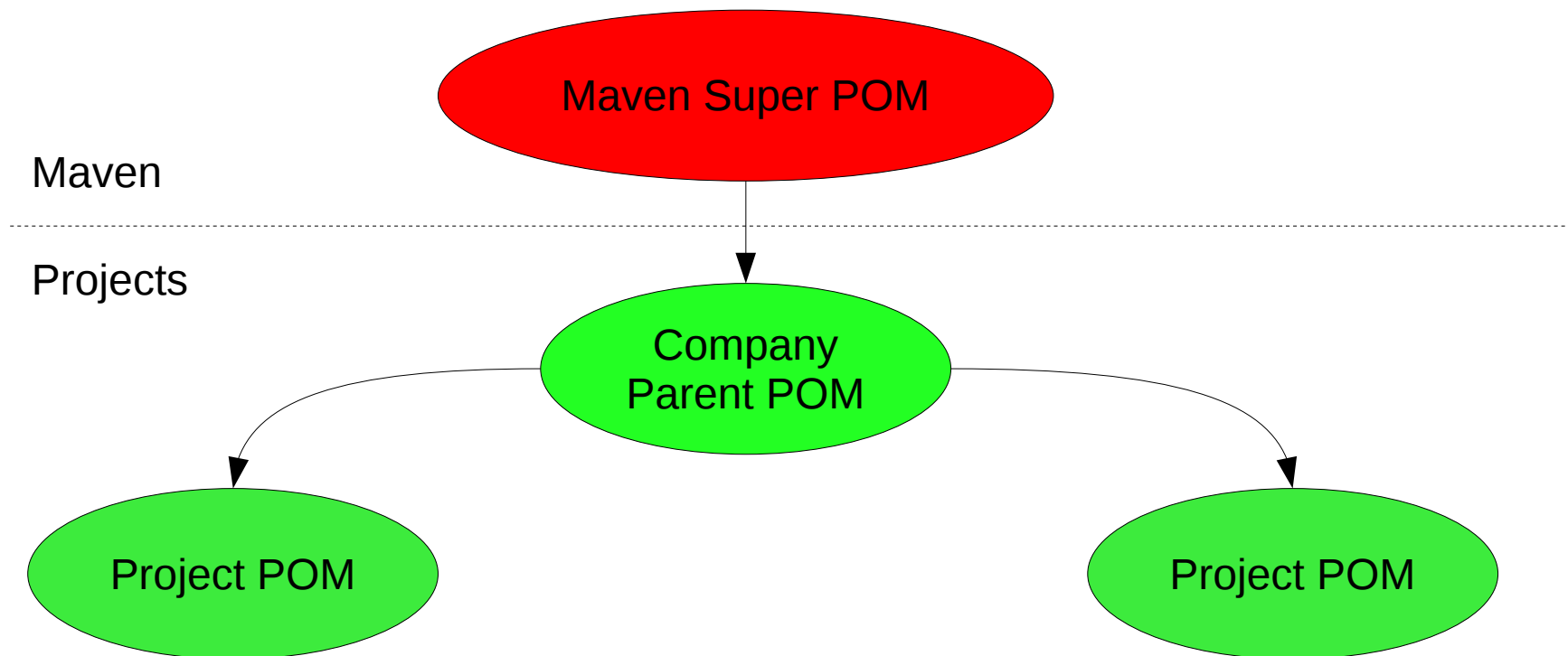
```
<prerequisites>...</prerequisites>  
<distributionManagement>...</distributionManagement>  
<profiles>...</profiles>  
<properties>...</properties>  
<repositories>...</repositories>  
<pluginRepositories>...</pluginRepositories>
```

Distribution management, profiles, repositories, pluginRepositories.

2. Basic Concepts

Project Inheritance

- The inheritance of informations between projects.



2. Basic Concepts

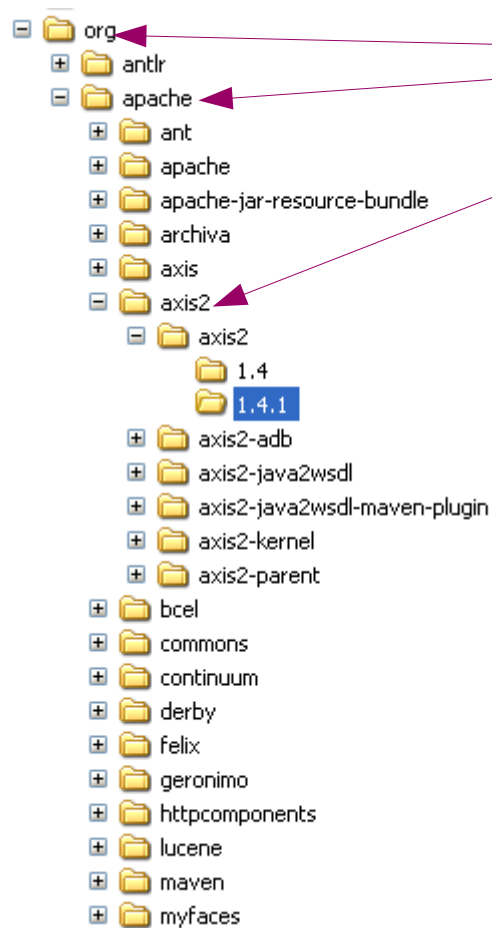
Project Inheritance

- The following parts will be inherited*:
 - dependencies, dependencyManagement
 - developers and contributors
 - plugin lists, pluginManagement
 - reports lists
 - plugin executions with matching ids
 - plugin configuration

[*Inheritance Guide](#)

2. Basic Concepts

Repositories

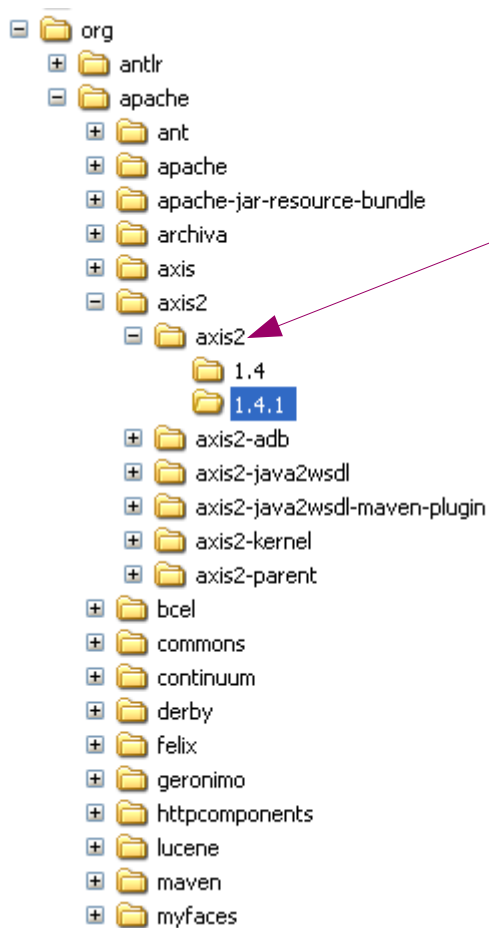


```
<groupId>org.apache.axis2</groupId>  
<artifactId>axis2</artifactId>  
<version>1.4.1</version>
```

axis2-1.4.1.jar	2,555 KB
axis2-1.4.1.jar.sha1	1 KB
axis2-1.4.1.pom	14 KB
axis2-1.4.1.pom.sha1	1 KB

2. Basic Concepts

Repositories



```
<groupId>org.apache.axis2</groupId>  
<artifactId>axis2</artifactId>  
<version>1.4.1</version>
```

axis2-1.4.1.jar	2,555 KB
axis2-1.4.1.jar.sha1	1 KB
axis2-1.4.1.pom	14 KB
axis2-1.4.1.pom.sha1	1 KB

2. Basic Concepts

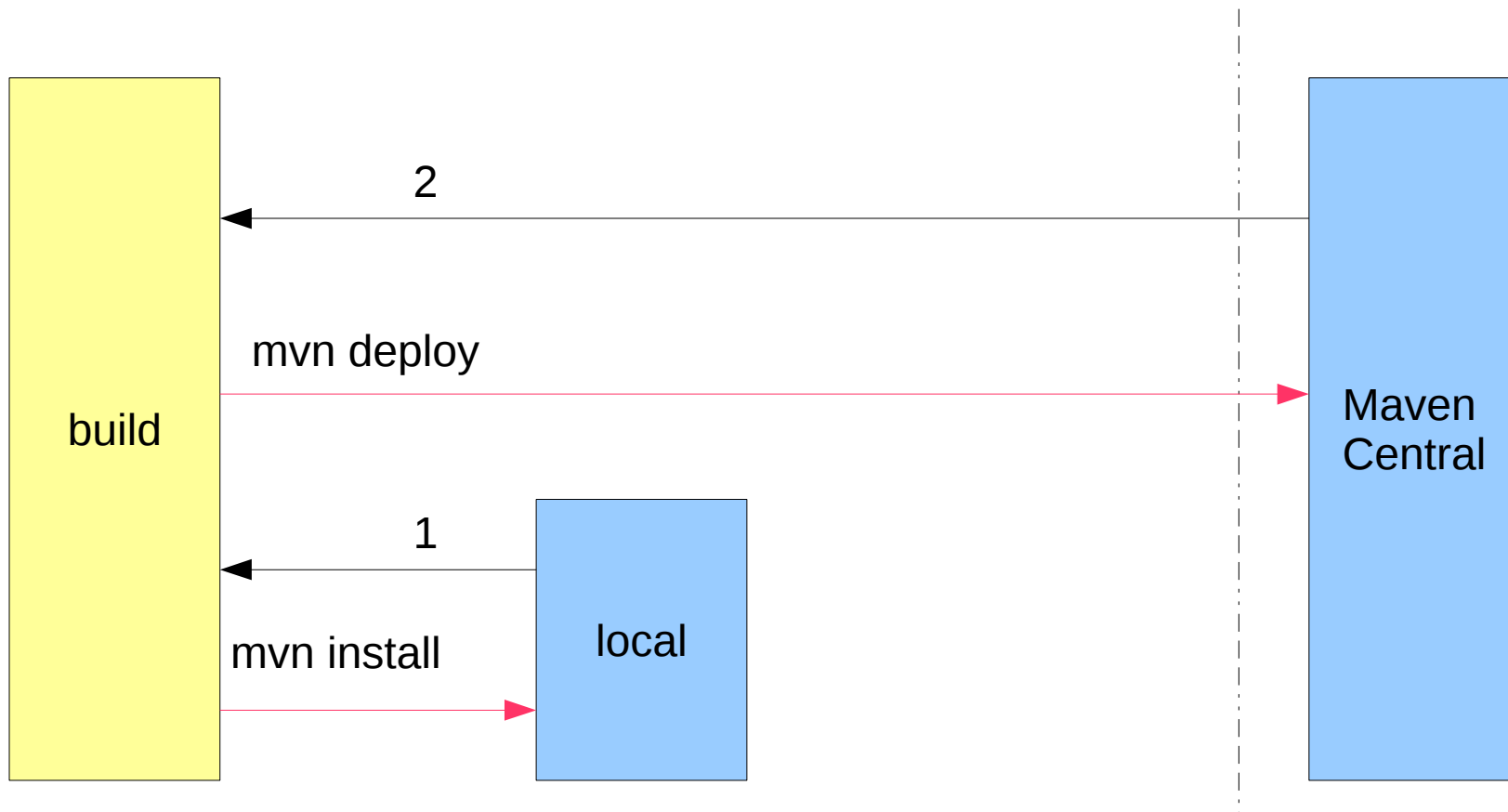
Repositories

`<groupId>org.apache.axis2</groupId>`
`<artifactId>axis2</artifactId>`
`<version>1.4.1</version>`

axis2-1.4.1.jar	2,555 KB
axis2-1.4.1.jar.sha1	1 KB
axis2-1.4.1.pom	14 KB
axis2-1.4.1.pom.sha1	1 KB

2. Basic Concepts Repositories

- The simple use case (training or private)



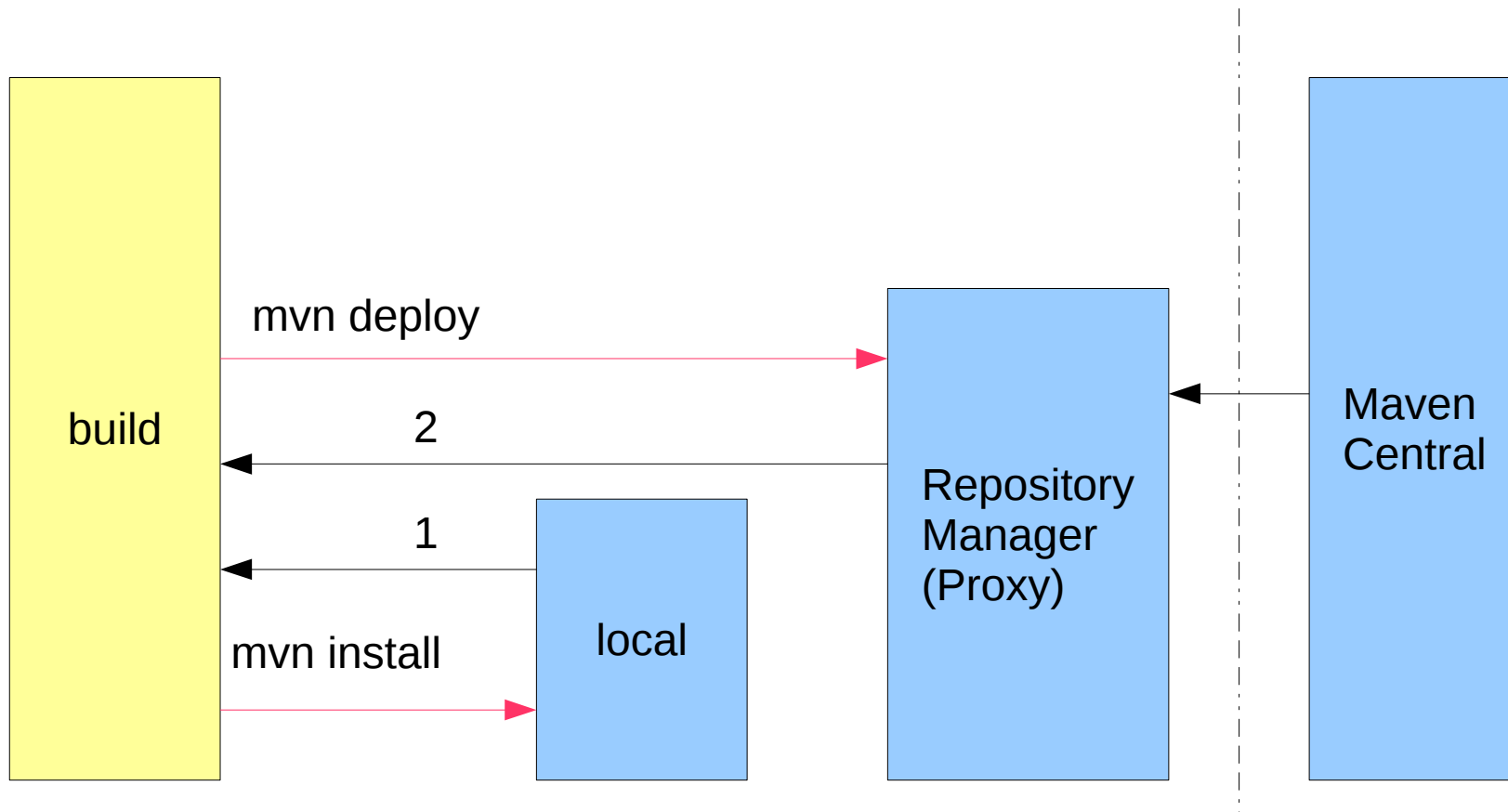
2. Basic Concepts Repositories

- The local repository is located at **`${HOME}/.m2/repository`**.
- This serves as a cache which improves the build speed.
- Maven will search first in your local repository and if it doesn't find the artifact there it will check the remote repository and download it from there.

[*Repository Guide](#)

2. Basic Concepts Repositories

- A real use case (company setup)



2. Basic Concepts Repositories

- For a real setup:
 - Use always a Repository Manager
 - Improvement of local builds.
 - Deployment of artifacts which should not be distributed into the public (e.g. Maven Central).
 - Store artifacts which are not available via Maven repositories.
 - Repository Managers:
 - Nexus (OSS)
 - Nexus Commercial
 - Artifactory (OSS)
 - Artifactory Commercial
 - Archiva

2. Basic Concepts

First project

- You can create the needed folders and files etc. by hand but it's simpler to use Maven itself for this.

```
mvn archetype:generate
-DgroupId=com.soebes.maven.training.first
-DartifactId=first-project
-Dversion=0.1.0-SNAPSHOT
```

- or if you don't like typing just use the following:

```
mvn archetype:generate
```

2. Basic Concepts

Dependencies

- If you like to use a particular library in your project just define the dependency.

```
<dependencies>
  ...
  <dependency>
    <groupId>org antlr</groupId>
    <artifactId>antlr-runtime</artifactId>
    <version>3.3</version>
  </dependency>
  ...
</dependencies>
```

Note: [Dependency Mechanism](#)

2. Basic Concepts

Dependency Scope

- If you define a dependency it has the default scope “compile” which means it will be used during compile time and run time.

```
<dependencies>
  ...
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.8.1</version>
    <scope>test</scope>
  </dependency>
  ...
</dependencies>
```

2. Basic Concepts

Dependency Scope

- The following scopes exist:
 - **compile**
 - Compile dependencies are available in all classpath's. Furthermore, those dependencies are propagated to dependent projects.
 - **test**
 - this scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases.
 - **provided**
 - This is much like compile, but indicates you expect the JDK or a container to provide it at runtime. It is only available on the compilation and test classpath, and it is not transitive.

2. Basic Concepts

Dependency Scope

– **runtime**

- This scope indicates that the dependency is not required for compilation, but is for execution. It is in the runtime and test classpaths, but not the compile classpath.

2. Basic Concepts

Dependency Scope

- **system** (use this with care, cause it might be removed in later versions of Maven).
 - This scope is similar to provided except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.
 - `systemPath`:
 - Only with system scope otherwise, the build will fail if this element is set. The path must be absolute, so it is recommended to use a property to specify the machine-specific path (more on properties below), such as `${java.home}/lib`. Maven checks to ensure that the file exists. If not, Maven will fail the build and suggest that you download and install it manually.

3. First Project

Simplest POM

- Based on the simplest POM you can take a look into the Super-POM by using the following command (inside the project which contains the pom.xml file):

```
mvn help:effective-pom
```


3. First Project Simplest POM

- Why is such a simple POM not recommended?
 - No definition of Maven Plugins only via Super-POM
 - Warning messages
 - “[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!”.
 - Reliability of the build is not given if you change the Maven version.

3. First Project Properties

- You will often see things like the following in pom.xml files or filtered files etc.
 - `${project.groupId}`, `${project.artifactId}`,
`${project.version}`
 - `${project.build.directory}`
 - This represents the **target** folder.
 - `${project.build.outputDirectory}`
 - This represents the *target/classes* folder.
- Never use literals use references where ever possible instead!

3. First Project Properties

- `env.X`
 - Prefixing a variable with `env.` will return the shell's environment variable. For example, `${env.PATH}` contains the `$path` environment variable. (`%PATH%` in Windows.)
- `project.x`
 - A dot-notated (`.`) path in the POM will contain the corresponding elements value.
 - Examples:
`${project.version}` or `${project.build.directory}`

3. First Project Properties

- settings.x
 - A dot-notated (.) path in the settings.xml will contain the corresponding elements value.
- Java system properties
 - All properties accessible via `java.lang.System.getProperties()` are available as POM properties, such as `${java.home}.x`

Note: [Reference for Properties Property Guide](#)

4. Configuration Overview

- There are two locations for the configuration of Maven:
 - System Configuration:
 - `${M2_HOME}/conf/settings.xml` *
 - User Based configuration:
 - `${HOME}/.m2/settings.xml` *
 - Project based configuration:
 - `${basedir}/profiles.xml` (removed in Maven 3).

*[Settings Reference Manual](#)

*[Profiles for Maven 2.2.1](#)

4. Configuration Overview

- Overview of the settings.xml contents:

```
<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">
  <localRepository/>
  <interactiveMode/>
  <usePluginRegistry/>
  <offline/>
  <pluginGroups/>
  <servers/>
  <mirrors/>
  <proxies/>
  <profiles/>
  <activeProfiles/>
</settings>
```

Note: [Settings Reference](#)

4. Configuration Using Repository Manager

```
<mirrors>
  <mirror>
    <id>nexus</id>
    <mirrorOf>*</mirrorOf>
    <url>http://localhost:8081/nexus/content/groups/public</url>
  </mirror>
</mirrors>
<profiles>
  <profile>
    <id>nexus</id>
    <repositories>
      <repository>
        <id>central</id>
        <url>http://central</url>
        <releases><enabled>true</enabled></releases>
        <snapshots><enabled>true</enabled></snapshots>
      </repository>
    </repositories>
    <pluginRepositories>
      <pluginRepository>
        <id>central</id>
        <url>http://central</url>
        <releases><enabled>true</enabled></releases>
        <snapshots><enabled>true</enabled></snapshots>
      </pluginRepository>
    </pluginRepositories>
  </profile>
</profiles>
<activeProfiles>
  <activeProfile>nexus</activeProfile>
</activeProfiles>
```

Note: [Nexus: Configuration for Maven](#)
[Artifactory: Configuration for Maven](#)
[Archiva: Configuration for Maven](#)

4. Configuration

Proxy Example

- Proxy settings in the settings.xml file:

```
...
<proxies>
  <proxy>
    <id>myproxy</id>
    <active>true</active>
    <protocol>http</protocol>
    <host>proxy.somewhere.com</host>
    <port>8080</port>
    <username>proxyuser</username>
    <password>somepassword</password>
    <nonProxyHosts>*.google.com|*.internal.com</nonProxyHosts>
  </proxy>
</proxies>
...
```

Note: [Settings Reference](#)

5. More Advanced Features

Multi-Module Builds

- What is a multi-module build?
 - The mechanism in Maven that handles multi-module projects is referred to as the ***reactor***.
 - Collects all the available modules to build
 - Sorts the projects into the correct build order
 - Builds the selected projects in order

Note: [Multi Module Guide](#)

5. More Advanced Features

Multi-Module Builds

- A multi-module build contains usually a list of modules in the root pom:

```
<modules>
  <module>module-a</module>
  <module>module-b</module>
  <module>module-c</module>
</modules>
```

- The “module” does not mean inheritance.

6. Release a Project / Artifact

Maven Release Plugin

- Fulfill all requirements of the Release Plugin
- First test with the release plugin

```
mvn -DdryRun=true release:prepare
```

- This should run without any error message.
- If something goes wrong first

```
mvn release:rollback
```

Note: [Release Guide](#)

6. Release a Project / Artifact

Maven Release Plugin

- The final step within a release cycle is to call:

`mvn release:perform`
- This should run without any error message as well.
 - If something goes wrong first

`mvn release:rollback`

Note: [Release Guide](#)

A. Profiles

- Profiles can be used to make different configurations possible:
 - Typical use cases are:
 - Different machines for production, test and development etc.
 - Configure different databases for production, test and development.
 - Special things which have to be done in case of a release
 - Signing of artifacts via pgp etc.
 - etc.

A. Profiles

- Profiles can be activated via command line or by properties.
- You can activate a single profile or multiple profiles within a single call:

```
mvn -Pprofile-x,profile-y package
```

A. Profiles

Automatic activated

- activated by a property which is propagated by the maven release plugin.

```
<profile>
  <id>release</id>
  <activation>
    <property>
      <name>performRelease</name>
      <value>>true</value>
    </property>
  </activation>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-gpg-plugin</artifactId>
        <version>1.1</version>
        <executions>
          <execution>
            <id>sign-artifacts</id>
            <phase>verify</phase>
            <goals>
              <goal>sign</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
```

A. Profiles

Production / Development

- This can be activated by command line
 - mvn -Pprod

```
<project>
...
<properties>
  <jdbc.driverClassName>com.mysql.jdbc.Driver</jdbc.driverClassName>
  <jdbc.url>jdbc:mysql://localhost:3306/development_db</jdbc.url>
  <jdbc.username>dev_user</jdbc.username>
  <jdbc.password>dev</jdbc.password>
</properties>
...
<build>
  <resources>
    <resource>
      <directory>src/main/resources</directory>
      <filtering>>true</filtering>
    </resource>
  </resources>
</build>
...
<profiles>
  <profile>
    <id>prod</id>
    <properties>
      <jdbc.driverClassName>oracle.jdbc.driver.OracleDriver</jdbc.driverClassName>
      <jdbc.url>jdbc:oracle:thin:@proddb01:1521:PROD</jdbc.url>
      <jdbc.username>prod_user</jdbc.username>
      <jdbc.password>secret</jdbc.password>
    </properties>
  </profile>
</profiles>
</project>
```


A. Profiles

Compatibility Note

- Until Maven 2.2.1 it is possible to define a separate profiles.xml file.
- Starting with Maven 3.0 this not possible any more.

B. Migration to Maven 3

- Maven 3 has been implemented with the idea to be a replacement for Maven 2
- This means usually you can simply replace Maven 2 with Maven 3

Note: **Compatibility**

- One Exception:
 - Site generation
 - Can be handled by a profile
 - **Maven 3 Site Plugin**

C. Special Plugins

Maven License Plugin

- The maven-license-plugin supports you to have all files the correct license header.

Note: **Maven License Plugin**

C. Special Plugins

Maven Versions Plugin

- The Maven Versions Plugin can help you with versions etc. within a multi-module build or help you with reporting of dependencies etc.

Note: **Maven Versions Plugin**

C. Special Plugins

Maven Changes Plugin

- The Maven Changes Plugin supports you in maintaining a `changes.xml` file with all your changes.

Note: **Maven Changes Plugin**

References

- Maven References
 - <http://www.sonatype.com/Support/Books>
 - “Maven: The Complete Reference” Book
 - <https://github.com/sonatype/maven-reference-en>
 - “Maven by Example” Book
 - <https://github.com/sonatype/maven-example-en>
 - Maven Cookbook (work in progress)
 - <https://github.com/sonatype/maven-cookbook>
 - Developing with Eclipse and Maven
 - <https://github.com/sonatype/m2eclipse-book>

References

- Nexus Book
 - <https://github.com/sonatype/nexus-book>
- Maven Homepage
 - <http://maven.apache.org>
- Mailing lists (User)
 - <http://maven.apache.org/mail-lists.html>
- IRC
 - `irc.freenode.net #maven`
 - `irc.codehaus.org #maven`

References

- IDE Support
 - Eclipse
 - <http://m2eclipse.sonatype.org/>