Maven Essentials

Maven is the #1 build tool used by developers and has been around for more than a decade. It's widely used by many open source Java projects under the Apache Software Foundation, Sourceforge, Google Code, and many more.

Maven Essentials is a fast-paced guide to show you the key concepts of Maven and build automation. We get started by introducing you to Maven and exploring its core concepts and the architecture. Next, you will learn about the Project Object Model (POM) while creating your own Maven project. You will also discover ways and means of archetypes and plugins to establish the most common goals in build automation. After this, you'll get to know how to design the build to prevent any maintenance nightmares, with proper dependency management. We then explore Maven build lifecycles and Maven assemblies. Finally, you will discover how to apply best practices when designing a build system to improve developer productivity.

Who this book is written for

This book is ideal for experienced developers who are already familiar with build automation, but want to learn how to use Maven and apply its concepts to the most difficult scenarios in build automation.

What you will learn from this book

- Comprehend the key concepts of Apache Maven
- Get to know how Maven extension points are used
- Troubleshoot build issues with greater confidence
- Optimize Maven's configuration settings
- Understand Maven custom lifecycles and lifecycle extensions
- Get hands-on and create Maven assemblies
- Explore best practices to design a build system that improves developer productivity

Get started with the essentials of Apache Maven and get your build automation system up and running quickly.
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Apache Maven Quick Start'
- A synopsis of the book’s content
- More information on Maven Essentials
Prabath Siriwardena is the director of Security Architecture at WSO2 Inc., a company that produces a wide variety of open source software from data to screen. He is a member of OASIS Identity Metasystem Interoperability (IMI) TC, OASIS eXtensible Access Control Markup Language (XACML) TC, OASIS Security Services (SAML) TC, OASIS Identity in the Cloud TC, and OASIS Cloud Authorization (CloudAuthZ) TC. Prabath is also a member of PMC Apache Axis and has spoken at numerous international conferences, including OSCON, ApacheCon, WSO2Con, EIC, Identity Next, and OSDC. He has more than 10 years of industry experience and has worked with many Fortune 100 companies.
Maven is the number one build tool used by developers, and it has been there for more than a decade. Maven stands out among other build tools due to its extremely extensible architecture, which is built on top of the concept convention over configuration. This, in fact, has made Maven the de-facto tool to manage and build Java projects. It's being used widely by many open source Java projects under the Apache Software Foundation, SourceForge, Google Code, and many more.

This book provides a step-by-step guide, showing the readers how to use Apache Maven in an optimal way to address enterprise build requirements. Following the book, readers will be able to gain a thorough understanding of the following key areas:

- How to get started with Apache Maven, applying Maven best practices in order to design a build system to improve a developer's productivity
- How to customize the build process to suit it exactly to your enterprise's needs by using appropriate Maven plugins, lifecycles, and archetypes
- How to troubleshoot build issues with greater confidence
- How to design the build in a way that avoids any maintenance nightmares with proper dependency management
- How to optimize Maven configuration settings
- How to build your own distribution archive using Maven assemblies
What this book covers

Chapter 1, Apache Maven Quick Start, focuses on building a basic foundation around Maven to get started. It starts by explaining the basic steps to install and configure Maven on Ubuntu, Mac OS X, and Microsoft Windows operating systems. The latter part of the chapter covers some of the common useful Maven tips and tricks.

Chapter 2, Understanding the Project Object Model (POM), focuses on the Maven Project Object Model (POM) and how to adhere to the industry-wide accepted best practices to avoid maintenance nightmares. The key elements of a POM file, POM hierarchy and inheritance, managing dependencies, and related topics are covered here.

Chapter 3, Maven Archetypes, focuses on how Maven archetypes provide a way of reducing repetitive work in building Maven projects. There are thousands of archetypes out there that are available publicly to help you build different types of projects. This chapter covers a commonly used set of archetypes.

Chapter 4, Maven Plugins, covers some of the most commonly used Maven plugins and then explains how plugins are discovered and executed. Maven only provides a build framework, while the Maven plugins perform the actual tasks. Maven has a large, rich set of plugins, and the chances are very small that you will have to write your own custom plugin.

Chapter 5, Build Lifecycles, explains how the three standard lifecycles work and how we can customize them. Later in the chapter, we discuss how to develop our own lifecycle extensions.

Chapter 6, Maven Assemblies, covers real-world examples of how to use the Maven assembly plugin in detail, and finally concludes with an end-to-end sample Maven project.

Chapter 7, Best Practices, looks at and highlights some of the best practices to be followed in a large-scale development project with Maven. It is always recommended to follow the best practices since it will drastically improve developer productivity and reduce any maintenance nightmares.
Apache Maven Quick Start

Apache Maven is popular as a build tool. However, in reality, it goes beyond being just a build tool. It provides a comprehensive build management platform. Prior to Maven, developers had to spend a lot of time in building a build system. There was no common interface. It differed from project to project, and each time a developer moved from one project to another, there was a learning curve. Maven filled this gap by introducing a common interface. It merely ended the era of the build engineer.

In this chapter, we will talk about the following topics:

- Installing and configuring Maven on Ubuntu, Mac OS X, and Microsoft Windows
- IDE integration
- Tips and tricks for using Maven effectively

Installing Apache Maven

Installing Maven on any platform is more than a straightforward task. At the time of writing this book, the latest version was 3.3.3, which is available for download at http://maven.apache.org/download.cgi. This version requires JDK 1.7.0 or above.

You should keep a note on the Java requirement for version 3.3.3, if you are planning to upgrade from versions 3.0.*, 3.1.*, or 3.2.*. Prior to Maven 3.3.x, the only requirement was JDK 1.5.0 or JDK 1.6.0 (for 3.2.*).

Apache Maven is an extremely lightweight distribution. It does not have any hard requirements on memory, disk space, or CPU. Maven itself is built on top of Java, and it would work on any operating system that runs Java virtual machine (JVM).
Installing Apache Maven on Ubuntu

Installing Maven on Ubuntu is a single line command. Proceed with the following steps:

1. Run the following `apt-get` command in the command prompt. You need to have the `sudo` privileges to execute this:
   
   ```
   $ sudo apt-get install maven
   ```

2. This takes a few minutes to complete. Upon the completion of the installation, you can run the following command to verify the installation:
   
   ```
   $ mvn -version
   ```

3. You should get an output similar to the following, if Apache Maven has been installed successfully:
   
   ```
   $ mvn -version
   Apache Maven 3.3.3
   Maven home: /usr/share/maven
   Java version: 1.7.0_60, vendor: Oracle Corporation
   Java home: /usr/lib/jvm/java-7-oracle/jre
   Default locale: en_US, platform encoding: UTF-8
   OS name: "linux", version: "3.13.0-24-generic", arch: "amd64",
   family: "unix"
   ```

4. Maven is installed under the `/usr/share/maven` directory. To check the directory structure behind the Maven installation directory, use the following command:
   
   ```
   $ ls /usr/share/maven
   bin  boot  conf  lib  man
   ```

5. Maven configuration files can be found at `/etc/maven`, which can be listed using the following command:
   
   ```
   $ ls /etc/maven
   m2.conf  settings.xml
   ```

If you don't want to work with the `apt-get` command, there is another way of installing Maven under any Unix-based operating system. We will discuss this in the next section. Since Mac OS X has a kernel built at the top of the Unix kernel, installing Maven on Mac OS X would be the same as installing it on any Unix-based operating system.
Installing Apache Maven on Mac OS X

Most of the OS X distributions prior to OS X Mavericks had Apache Maven preinstalled. To verify that you've got Maven installed in your system, try out the following command. If it does not result in a version, then it means you do not have it installed:

$ mvn -version

The following steps will guide you through the Maven installation process on Max OS X Yosemite:

1. First, we need to download the latest version of Maven. Throughout this book, we will use Maven 3.3.3, which was the latest version at the time of writing this book. Maven 3.3.3 ZIP distribution can be downloaded from http://maven.apache.org/download.cgi.

2. Unzip the downloaded ZIP file into /usr/share/java. You need to have the sudo privileges to execute this:

   $ sudo unzip apache-maven-3.3.3-bin.zip -d /usr/share/java/

3. In case you already have Maven installed in your system, use the following command to unlink. /usr/share/maven is only a symlink to the directory where Maven is installed:

   $ sudo unlink /usr/share/maven

4. Use the following command to create a symlink to the latest Maven distribution that you just unzipped. You need to have the sudo privileges to execute this:

   $ sudo ln -s /usr/share/java/apache-maven-3.3.3 /usr/share/maven

5. Use the following command to update the value of the PATH environment variable:

   $ export PATH=$PATH:/usr/share/maven/bin

6. Use the following command to update (or set) the value of the M2_HOME environment variable:

   $ export M2_HOME=/usr/share/maven

7. Verify the Maven installation with the following command:

   $ mvn -version

   Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T04:57:37-07:00)
   Maven home: /usr/share/maven
   Java version: 1.7.0_75, vendor: Oracle Corporation
Apache Maven Quick Start

Java home: /Library/Java/JavaVirtualMachines/jdk1.7.0_75.jdk/Contents/Home/jre

Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "10.10.2", arch: "x86_64",
family: "mac"

8. If you get the following error while running the preceding command, it means you have another version of Maven running in your system, and the PATH system variable includes the path to its bin directory. If that is the case, you need to clean out the value of the PATH system variable by removing the path to the old Maven installation:

-Dmaven.multiModuleProjectDirectory system property is not set. Check $M2_HOME environment variable and mvn script match.

Maven can also be installed on Mac OS X with Homebrew. This video explains the installation process in detail—https://www.youtube.com/watch?v=xTzLGcqUf8k

Installing Apache Maven on Microsoft Windows

First, we need to download the latest version of Maven. Apache Maven 3.3.3 ZIP distribution can be downloaded from http://maven.apache.org/download.cgi. Then, we need to perform the following steps:

1. Unzip the downloaded ZIP file into the C:\Program Files\ASF directory.
2. Set the M2_HOME environment variable and point it to C:\Program Files\ASF\apache-maven-3.3.3.
3. Verify the Maven installation with the following command on the command prompt:

mvn -version

To know more about how to set the environment variables on Microsoft Windows, refer to http://www.computerhope.com/issues/ch000549.htm.
Chapter 1

Configuring the heap size
Once you have Maven installed in your system, the very next step is to fine-tune it for an optimal performance. By default, the maximum heap allocation is 512 MB, which starts from 256 MB (-Xms256m to -Xmx512m). This default limit is not good enough to build a large, complex Java project, and it is recommended that you have at least 1024 MB of the maximum heap.

If you encounter java.lang.OutOfMemoryError at any point during a Maven build, then it is mostly due to a lack of memory. You can use the MAVEN_OPTS environment variable to set the maximum allowed heap size for Maven at a global level. The following command will set the heap size in any Unix-based operating system, including Linux and Mac OS X. Make sure that the value set as the maximum heap size does not exceed your system memory of the machine, which runs Maven:

$ export MAVEN_OPTS="-Xmx1024m -XX:MaxPermSize=128m"

If you are on Microsoft Windows, use the following command:

$ set MAVEN_OPTS=-Xmx1024m -XX:MaxPermSize=128m

Here, -Xmx takes the maximum heap size and -XX:MaxPermSize takes the maximum Permanent Generation (PermGen) size.

Maven runs as a Java process on JVM. As it proceeds with a build, it keeps on creating Java objects. These objects are stored in the memory allocated to Maven. This area of memory where Java objects are stored is known as heap. Heap is created at the JVM start and it increases as more and more objects are created up to the defined maximum limit. The -Xms JVM flag is used to instruct JVM about the minimum value that it should set at the time of creating the heap. The -Xmx JVM flag sets the maximum heap size.

PermGen is an area of memory managed by JVM, which stores the internal representations of Java classes. The maximum size of PermGen can be set by the -XX:MaxPermSize JVM flag.

When the Java virtual machine cannot allocate enough memory to Maven, it could result in an OutOfMemoryError. To know more about the Maven OutOfMemoryError, refer to https://cwiki.apache.org/confluence/display/MAVEN/OutOfMemoryError.
Hello Maven!

The easiest way to get started with a Maven project is to use the `generate` goal of the archetype plugin to generate a simple Maven project. Maven archetypes are discussed in detail in Chapter 3, *Maven Archetypes*, and plugins are covered in Chapter 4, *Maven Plugins*.

Let's start with a simple example:

```bash
$ mvn archetype:generate
    -DgroupId=com.packt.samples
    -DartifactId=com.packt.samples.archetype
    -Dversion=1.0.0
    -DinteractiveMode=false
```

This command will invoke the `generate` goal of the Maven archetype plugin to create a simple Java project. You will see that the following project structure is created with a sample POM file. The name of the root or the base directory is derived from the value of the `artifactId` parameter:

```
com.packt.samples.archetype
    |-pom.xml
    |-src
    |   |-main/java/com/packt/samples/App.java
    |   |-test/java/com/packt/samples/AppTest.java
```

The sample POM file will only have a dependency to the junit JAR file with test as the scope:

```xml
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.packt.samples</groupId>
    <artifactId>com.packt.samples.archetype</artifactId>
    <packaging>jar</packaging>
    <version>1.0.0</version>
    <name>com.packt.samples.archetype</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>
```
The generated `App.java` class will have the following template code. The name of the package is derived from the provided `groupId` parameter. If you want to have a different value as the package name, then you need to pass this value in the command itself as `-Dpackage=com.packt.samples.application`:

```java
package com.packt.samples;

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

To build the sample project, run the following command from the `com.packt.samples.archetype` directory, where the `pom.xml` file exists:

```bash
$ mvn clean install
```

## Convention over configuration

Convention over configuration is one of the main design philosophies behind Apache Maven. Let's go through a few examples.

A complete Maven project can be created using the following configuration file (`pom.xml`):

```xml
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.packt</groupId>
    <artifactId>sample-one</artifactId>
    <version>1.0.0</version>
</project>
```

The Maven POM file starts with the `<project>` element. Always define the `<project>` element with the schema. Some tools can't validate the file without it:

```xml
<project xmlns=http://maven.apache.org/POM/4.0.0
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:schemaLocation="...">
```
Apache Maven Quick Start

The \( \text{pom.xml} \) file is the heart of any Maven project and is discussed in detail in Chapter 2, Understanding the Project Object Model (POM). Copy the previous configuration element and create a \( \text{pom.xml} \) file out of it. Then, place it in a directory called chapter-01, and then create the following child directories under it:

- chapter-01/src/main/java
- chapter-01/src/test/java

Now, you can place your Java code under \( \text{chapter-01/src/main/java} \) and test cases under \( \text{chapter-01/src/test/java} \). Use the following command to run the Maven build from where the \( \text{pom.xml} \) is:

```
$ mvn clean install
```

This little configuration that you found in the sample \( \text{pom.xml} \) file is tied up with many conventions:

- Java source code is available at \{base-dir\}/src/main/java
- Test cases are available at \{base-dir\}/src/test/java
- The type of the artifact produced is a JAR file
- Compiled class files are copied to \{base-dir\}/target/classes
- The final artifact is copied to \{base-dir\}/target
- \http://repo.maven.apache.org/maven2\, is used as the repository URL.

If someone needs to override the default, conventional behavior of Maven, then it is possible too. The following sample \( \text{pom.xml} \) file shows how to override some of the preceding default values:

```xml
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.packt</groupId>
    <artifactId>sample-one</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <build>
        <sourceDirectory>${basedir}/src/main/java</sourceDirectory>
        <testSourceDirectory>${basedir}/src/test/java
        <outputDirectory>${basedir}/target/classes
    </build>

</project>
```

[8]
Maven repositories

The magic behind how Maven finds and loads dependent jars for a given Maven project is Maven repositories. In the corresponding pom.xml file of your Maven project, under the <dependencies> element, you can define references to all the dependent jar files required to build your project successfully. Each dependency defined in the pom.xml file is identified uniquely using Maven coordinates. Maven coordinates uniquely identify a project, a dependency, or a plugin defined in a POM. Each entity is uniquely identified by the combination of a group identifier, an artifact identifier, and version (and, of course, with the packaging and the classifier). Maven coordinates are discussed in detail in Chapter 2, Understanding the Project Object Model (POM). Once Maven finds out all the required dependencies for a given project, it loads them to the local file system of Maven repositories, and adds them to the project classpath.

By convention, Maven uses http://repo.maven.apache.org/maven2 as the repository. If all the artifacts required to build the project are present in this repository, then those will be loaded into the local file system or the local Maven repository, which is, by default, at USER_HOME/.m2/repository. You can add custom repositories at the project level under the <repositories> element of the pom.xml file or at the global level under the MAVEN_HOME/conf/settings.xml file.

IDE integration

Most of the hardcore developers never want to leave their IDE. Not just for coding, but for building, deploying, testing, and for everything if possible - they would happily do these from the IDE itself. Most of the popular IDEs support Maven integration, and they have developed their own plugins to support Maven.

NetBeans integration

NetBeans 6.7 or newer ships with in-built Maven integration, while NetBeans 7.0 has newer versions that bundle a complete copy of Maven 3 and runs it for builds just like you would from the command line. For version 6.9 or older, you have to download a Maven build and configure the IDE to run this. More information corresponding to Maven and NetBeans integration is available at http://wiki.netbeans.org/MavenBestPractices.
IntelliJ IDEA integration
IntelliJ IDEA has inbuilt support for Maven; hence, you don't need to perform any additional steps to install it. More information corresponding to Maven and IntelliJ IDEA integration is available at [http://wiki.jetbrains.net/intellij/Creating_and_importing_Maven_projects](http://wiki.jetbrains.net/intellij/Creating_and_importing_Maven_projects).

Eclipse integration
The M2Eclipse project provides first class Maven support through the Eclipse IDE. More information corresponding to Maven and the Eclipse integration is available at [https://www.eclipse.org/m2e/](https://www.eclipse.org/m2e/).


Troubleshooting
If everything works fine, we should never have to worry about troubleshooting. However, most of the time this is not the case. A Maven build can fail for many reasons, some of which are under your control and also out of your control. Knowing proper troubleshooting tips helps you to pinpoint the exact problem. The following section lists some of the most used troubleshooting tips. We will expand the list as we proceed in this book.

Enabling Maven debug level logs
Once the Maven debug level logging is enabled, it will print all the actions that it takes during the build process. To enable debug level logging, use the following command:

$ mvn clean install -X

Building a dependency tree
If you find any issue with any dependency in your Maven project, the first step is to build a dependency tree. This shows where each dependency comes from. To build the dependency tree, run the following command against your project **pom** file:

$ mvn dependency:tree
The following shows the truncated output of the previous command executed against the Apache Rampart project:

[INFO] --------------------------------------------------------------
[INFO] Building Rampart - Trust 1.6.1-wso2v12
[INFO] --------------------------------------------------------------
[INFO] [INFO] --- maven-dependency-plugin:2.1:tree (default-cli) @ rampart-trust ---
[INFO] [INFO] org.apache.rampart:rampart-trust:jar:1.6.1-wso2v12
[INFO] [INFO] |  |  +- org.apache.ws.commons.axiom:axiom-api:jar:1.2.11-wso2v4:compile (version managed from 1.2.11)
[INFO] [INFO] |  |  |  \- jaxen:jaxen:jar:1.1.1:compile
[INFO] [INFO] |  +- org.apache.ws.commons.axiom:axiom-impl:jar:1.2.11-wso2v4:compile (version managed from 1.2.11)
[INFO] [INFO] |  |  +- org.apache.geronimo.specs:geronimo-ws-metadata_2.0_spec:jar:1.1.2:compile
[INFO] [INFO] |  |  +- org.apache.geronimo.specs:geronimo-jta_1.1_spec:jar:1.1:compile
[INFO] [INFO] |  |  +- javax.servlet:servlet-api:jar:2.3:compile
[INFO] [INFO] |  |  |  \- commons-httpclient:commons-httpclient:jar:3.1:compile
[INFO] [INFO] |  |  |  \- commons-codec:commons-codec:jar:1.2:compile
[INFO] [INFO] |  |  +- commons-fileupload:commons-fileupload:jar:1.2:compile

Viewing all the environment variables and system properties

If you have multiple JDKs installed in your system, you may wonder what is being used by Maven. The following command will display all the environment variables and system properties set for a given Maven project:

$ mvn help:system

The following is the truncated output of the previous command:

================================================================================
Platform Properties Details
================================================================================
System Properties

[11]
Apache Maven Quick Start

=====================================================================
java.runtime.name=Java(TM) SE Runtime Environment
sun.boot.library.path= /Library/Java/JavaVirtualMachines/jdk1.7.0_75.jdk/
    Contents/Home/jre/lib
java.vm.version= 24.75-b04
awt.nativeDoubleBuffering=true
gopherProxySet=false
mrj.build=11M4609
java.vm.vendor=Apple Inc.
java.vendor.url=http://www.apple.com/
guice.disable.misplaced.annotation.check=true
path.separator=: 
java.vm.name=Java HotSpot(TM) 64-Bit Server VM
file.encoding.pkg=sun.io
sun.java.launcher=SUN_STANDARD
user.country=US
sun.os.patch.level=unknown

 =========================================================
Environment Variables
 =========================================================
JAVA_HOME=/System/Library/Frameworks/JavaVM.framework/Versions/
    CurrentJDK/Home
HOME=/Users/prabath
TERM_SESSION_ID=C2CEFB58-4705-4C67-BE1F-9E4179F96391
M2_HOME=/usr/share/maven/maven-3.3.3/
COMMAND_MODE=unix2003
Apple_PubSub_Socket_Render=/tmp/launch-w7NZbG/Render
LOGNAME=prabath
USER=prabath

Viewing the effective POM file

Maven uses default values for configuration parameters when they are not
overridden in the configuration. This is exactly what we discussed under the
Convention over configuration section. If we take the same sample POM file that we used
before in this chapter, we can see how the effective POM file would look using the
following command. This is also the best way to see what default values are being
used by Maven:

$ mvn help:effective-pom
More details about the effective-pom command are discussed in Chapter 2, Understanding the Project Object Model (POM).

**Viewing the dependency classpath**

The following command will list all the JAR files and directories in the build classpath:

```bash
$ mvn dependency:build-classpath
```

The following shows the truncated output of the previous command executed against the Apache Rampart project:

```
[INFO] --------------------------------------------------------------
[INFO] Building Rampart - Trust 1.6.1-wso2v12
[INFO] --------------------------------------------------------------
[INFO] --- maven-dependency-plugin:2.1:build-classpath (default-cli) @ rampart-trust ---
[INFO] Dependencies classpath:
/Users/prabath/.m2/repository/bouncycastle/bcprov-jdk14/140/bcprov-jdk14-140.jar:/Users/prabath/.m2/repository/commons-cli/commons-cli/1.0/commons-cli-1.0.jar:/Users/prabath/.m2/repository/commons-codec/commons-codec/1.2/commons-codec-1.2.jar:/Users/prabath/.m2/repository/commons-collections/commons-collections/3.1/commons-collections-3.1.jar
```

**Summary**

This chapter focused on building a basic foundation around Maven to bring all the readers into a common ground. It started with explaining the basic steps to install and configure Maven under Ubuntu, Mac OS X, and Microsoft Windows operating systems. The latter part of the chapter covered some of the common useful Maven tips and tricks. As we proceed with the book, some of the concepts touched in this chapter will be discussed in detail.

In the next chapter, we will discuss **Maven Project Object Model (POM)** in detail.
Where to buy this book

You can buy Maven Essentials from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.