Mastering Apache Maven 3

Maven is the number one build tool used by developers 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 has made Maven the de-facto tool used to manage and build Java projects.

This book is a technical guide to the difficult and complex concepts in Maven and build automation. It starts with the core Maven concepts and its architecture, and then explains how to build extensions such as plugins, archetypes, and lifecycles in depth.

This book is a step-by-step guide that shows you how to use Apache Maven in an optimal way to address your enterprise build requirements.

Who this book is written for

If you are working with Java or Java EE projects and you want to take full advantage of Maven in designing, executing, and maintaining your build system for optimal developer productivity, then this book is ideal for you. You should be well versed with Maven and its basic functionality if you wish to get the most out of the book.

What you will learn from this book

- Apply Maven best practices in designing a build system to improve developer productivity
- Customize the build process to suit your enterprise needs by developing custom Maven plugins, lifecycles, and archetypes
- Implement and deploy a Maven repository manager to manage the build process in a better and smoother way
- Design the build in a way that prevents any maintenance nightmares with proper dependency management
- Optimize Maven configuration settings
- Create your own distribution archive using Maven assemblies
- Build custom Maven lifecycles and lifecycle extensions


Prabath Siriwardena

Enhance developer productivity and address exact enterprise build requirements by extending Maven.
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 Mastering Apache Maven 3

About the Author

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.
Acknowledgments

I would first like to thank Subho Gupta, a senior acquisition editor at Packt Publishing, who came up with the idea of writing a book on mastering Apache Maven; then I would like to thank, Ruchita Bhansali, a content development editor at Packt Publishing, who I worked with closely throughout the project—thank you very much, Ruchita, for your patience and flexibility. Also, I would like to thank all the others at Packt who helped me throughout to make this book a reality from the initial idea. Thank you very much for all your continuous support.

Dr. Sanjiva Weerawarana, the CEO of WSO2, and Paul Fremantle, the CTO of WSO2, have always been my mentors. I am truly grateful to both of them for everything they have done for me.

I'd like to thank my beloved wife, Pavithra, and my loving little daughter, Dinadi. Pavithra wanted me to write this book even more than I wanted. If I say she is the driving force behind this book, I am not exaggerating. She simply went beyond by not only feeding me with all the encouragement, but also by helping immensely in reviewing the book and developing samples. She was always the first reader. Thank you very much, Pavithra. Also, thanks to little Dinadi for your patience—it was your time I spent on writing the book.

I would also like to thank all the technical reviewers of the book. All your suggestions and thoughts were extremely valuable and are much appreciated.

My parents and my sister have been the driving force behind me since my birth. If not for them, I wouldn't be who I am today. I am grateful to them for everything they have done for me. Last but not least, I'd like to thank my wife's parents; they were amazingly helpful in making sure that the only thing I had to do was to write this book, taking care of almost all the other things that I was supposed to do.

Although this would sounds like a one-man effort, it's actually a team effort. Thanks to everyone who supported me in different ways.
Mastering Apache Maven 3

Maven is the number one build tool used by developers, and it has been available for more than a decade. Maven stands out among other build tools due to its extensible architecture, which is built on the concept of convention over configuration. This has made Maven the de-facto tool to manage and build Java projects. It's being widely used by many open source Java projects under Apache Software Foundation, SourceForge, Google Code, and many more.

Mastering Apache Maven 3 provides a step-by-step guide that will show you how to use Apache Maven in an optimal way to address enterprise build requirements. After reading this book, you will be able to:

- Apply Maven's best practices in designing a build system to improve developers' productivity
- Customize the build process to suit your enterprise needs by developing custom Maven plugins, lifecycles, and archetypes
- Troubleshoot build issues with greater confidence
- Implement and deploy a Maven repository manager to manage the build process in a better way
- Design the build with proper dependency management, avoiding any maintenance nightmares
- Optimize the Maven configuration settings
- Build your own distribution archive using Maven assemblies
- Build custom Maven lifecycles and lifecycle extensions

What This Book Covers

Chapter 1, Apache Maven Quick Start, focuses on giving an introduction to Apache Maven. If you are an advanced Maven user, you can simply jump to the next chapter. It will show how to install and configure Maven on different operating systems such as Linux, Mac, and Microsoft Windows and tips and tricks to use Maven.

Chapter 2, Demystifying Project Object Model, focuses on core concepts and best practices related to Project Object Model (POM) in building a large-scale multimodule Maven project.

Chapter 3, Maven Configuration, discusses how to customize the Maven configuration at three different levels: the global level, the user level, and the project level for optimal use.

Chapter 4, Build Lifecycles, discusses the Maven build lifecycles in detail. A Maven build lifecycle consists of a set of well-defined phases. Each phase groups a set of goals defined by Maven plugins and the lifecycle defines the order of execution of the phases.
Chapter 5, Maven Plugins, explains the usage of key Maven plugins and demonstrates how to build custom plugins. All the useful functionalities in the build process are developed as Maven plugins. One could also easily call Maven, a plugin execution framework.

Chapter 6, Maven Assemblies, explains how to build custom assemblies with the Maven assembly plugin. The Maven assembly plugin produces a custom archive, which adheres to a user-defined layout. This custom archive is also known as the Maven assembly. In other words, it's a distribution unit that is built according to a custom layout.

Chapter 7, Maven Archetypes, explains how to use existing archetypes and how to build custom Maven archetypes. Maven archetypes provide a way of reducing repetitive work in building Maven projects. There are thousands of archetypes out there that are available freely to assist you in building different types of projects.

Chapter 8, Maven Repository Management, discusses the pros and cons of using a Maven repository manager. This chapter further explains how to use Nexus as a repository manager and configure it as a hosted, proxied, and group repository.

Chapter 9, 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 best practices, as they will drastically improve developers’ productivity and reduce maintenance nightmares.
This chapter will introduce Apache Maven. If you are an advanced Maven user, you can simply jump into the next chapter. Even for an advanced user, it is highly recommended that you at least brush through this chapter, as it will be helpful to make sure that we are on the same page as we proceed.

In this chapter, we will be discussing about the following topics:

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

A quick introduction
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—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 ended the era of "the build engineer."

Installing Apache Maven
Installing Maven on any platform is a straightforward task. At the time of writing this book, the latest version is 3.2.3, which is available to download from http://maven.apache.org/download.cgi. This version requires JDK 1.6.0 or above. You should keep a note of the Java requirement for version 3.2.3 if you are planning to upgrade from version 3.0.0 family or 3.1.0 family. Prior to Maven 3.2.1, the only requirement was JDK 1.5.0.
Apache Maven is an extremely lightweight distribution. It does not have any hard requirements in terms of memory, disk space, or CPU. Maven is built on top of Java and will work on any operating system that runs a Java Virtual Machine (JVM).

**Installing Apache Maven on Ubuntu**

Installing Maven on Ubuntu just needs 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. The installation 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 one if Apache Maven has been installed successfully:
   
   ```
   $ mvn -version
   Apache Maven 3.2.3
   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 under the `/etc/maven` directory 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 on 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:

```
$ mvn -version
```

If it does not result in a version, this means you do not have Apache Maven installed.

The following steps will guide you through the Maven installation process:

1. First, we need to download the latest version of Maven. Throughout this book, we will use Maven 3.2.3, which is the latest version at the time of writing this book. The Maven 3.2.3 ZIP distribution can be downloaded from [http://maven.apache.org/download.cgi](http://maven.apache.org/download.cgi).

2. Unzip the downloaded ZIP file and extract it to `/usr/share/java` directory. You need to have the `sudo` privileges to execute the following command:

   ```
   $ sudo unzip apache-maven-3.2.3-bin.zip -d /usr/share/java/
   ```

3. If you already have Maven installed in your system, use the following command to unlink:

   ```
   $ sudo unlink /usr/share/maven
   ```

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

   ```
   $ sudo ln -s /usr/share/java/apache-maven-3.2.3 /usr/share/maven
   ```

5. Verify the Maven installation with the following command:

   ```
   $ mvn -version
   Apache Maven 3.2.3 (33f8c3e1027c3ddde99d3cdebad2656a31e8f5f4;
   2014-08-12T02:28:10+05:30)
   Maven home: /usr/share/maven
   Java version: 1.6.0_65, vendor: Apple Inc.
   ```
Apache Maven Quick Start

Java home: /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home
Default locale: en_US, platform encoding: MacRoman
OS name: "mac os x", version: "10.8.5", arch: "x86_64",
family: "mac"

Maven can also be installed on Mac OS X with Homebrew. Check out the video at this link, https://www.youtube.com/watch?v=xTzLGcqUf8k, which explains the installation process in detail.

Installing Apache Maven on Microsoft Windows

First, we need to download the latest version of Maven. The Apache Maven 3.2.3 ZIP distribution can be downloaded from http://maven.apache.org/download.cgi. Next, perform the following steps:

1. Unzip the downloaded ZIP file and extract it to \Program Files\ASF folder.
2. Set the M2_HOME environment variable and point it to \Program Files\ASF\apache-maven-3.2.3.
3. Verify the Maven installation with the following command on the command prompt:
   mvn -version

To learn how to set the environment variables on Microsoft Windows, you can refer http://www.computerhope.com/issues/ch000549.htm.

Configuring the heap size

Once you have installed Maven in your system, the next step is to fine-tune it for optimal performance. By default, the maximum heap allocation is 256 - 512 MB (-Xms256m to -Xmx512m). This default limit does not work while building a large, complex Java project, and it is recommended that you have at least 1024 MB of maximum heap. If you encounter the java.lang.OutOfMemoryError error at any point during a Maven build, it is mostly due to the 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 Linux. Make sure that the value set as the maximum heap size does not exceed your system memory of the machine that runs Maven.

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

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

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

Here `-Xmx` takes the maximum heap size and `-XX:MaxPermSize` takes the maximum PermGen size.

Maven runs as a Java process on JVM. As Java 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 the minimum value it should set at the time it creates the heap. The `-Xmx` JVM flag sets the maximum heap size.

Permanent Generation (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.

To learn about the Maven OutOfMemoryError error, check out the information at this link: https://cwiki.apache.org/confluence/display/MAVEN/OutOfMemoryError.

**Monitoring the build**

The most popular way of starting a Maven build is by using the `mvn clean install` command. This will build all the Maven modules under your project and install the artifacts to your local repository. For a simple project, the entire build process will take less than a minute. However, for a large project, to create an online build with a clean repository could even take more than 3 hours; this is not an exaggeration. If you look at the WSO2 Carbon complete code base, the complete build process takes more than four hours to run with all the test cases. During a long-running build process, it is extremely important that we monitor the build properly.
WSO2 Carbon is a framework that is written on top of OSGi to build servers. All WSO2 products, which are 100 percent open source and released under Apache 2.0 license, are built on top of WSO2 Carbon. WSO2 Carbon code base is available at https://svn.wso2.org/repos/wso2/carbon/.

The following screenshot shows an overview of the JVisualVM tool running a Maven build:

JVisualVM is a Java virtual machine monitoring, troubleshooting, and profiling tool. To learn more about it, refer http://docs.oracle.com/javase/6/docs/technotes/tools/share/jvisualvm.html.

The JVisualVM tool that comes with the JDK distribution can be used to monitor a running Maven build. First, we need to start the Maven build and then start JVisualVM using the following command:

$ jvisualvm
This command will start the JVisualVM tool. Once the tool gets started, select org.codehaus.plexus.classworlds.launcher.Launcher from the Applications tab to monitor the running Maven build. You can gather many important statistics using JVisualVM, and based on that you can optimize your system resources for an optimal Maven build.

The following screenshot shows JVisualVM statistics of a running Maven build:

![JVisualVM screenshot]

**Remote debugging**

For a developer, remote debugging is a must-have feature for any build system. Why do we need remote debugging for a build system? This is extremely useful when you run your tests through the build itself. If any of the tests fail during the build, you should be able to debug and pinpoint the problem. The following command will run Maven in the debugging mode:

```bash
$ mvn clean install -Dmaven.surefire.debug
```
When the build starts to execute tests, it will be paused to connect with an IDE. You can connect Eclipse, NetBeans, or your favorite IDE to port 5005 in order to start remote debugging. By default, Maven opens up port 5005 for remote debugging.

Listening for transport dt_socket at address: 5005

The default listening port number can be changed by setting the value of address appropriately. When you set the value of the suspend variable to y, the Maven build will stop until an IDE gets connected to it. If you want the build to continue and connect the IDE later, then set the value to n. To get full control over the debugging options, you can use the following command:

$ mvn clean install -Dmaven.surefire.debug=-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000 -Xnoagent -Djava.compiler=NONE

Refer to the corresponding IDE documentation to see how it can be remotely connected to an externally running process for remote debugging.

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 code snippet in pom.xml file:

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

Downloading the example code

You can download the example code files from your account at http://www.packtpub.com for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.
The Maven POM file starts with the `<project>` element. Always define the `<project/>` element with the corresponding 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="http://maven.apache.org/POM/4.0.0
doc/maven-4.0.0.xsd">
```

Copy the previous configuration element and create a pom.xml file out of it. Then, place it in a directory called chapter-01 and 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 chapter-01/src/main/java and test cases under chapter-01/src/test/java. Use the following command to run the Maven build:

```
$ mvn clean install
```

This little configuration is tied up with many conventions:

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

If someone needs to override the default, conventional behavior of Maven, that is possible too. The following sample 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>
```
Apache Maven Quick Start

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

IDE integration

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

NetBeans integration

NetBeans 6.7 or newer ships with inbuilt Maven integration, while NetBeans 7.0 and newer versions bundle a complete copy of Maven 3 and run 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 that. 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.

Eclipse integration

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

Troubleshooting
If everything works fine, we don't have to worry about troubleshooting. However, most of the time this is not the case. A Maven build could fail for many reasons—some are under your control, while others are beyond your control. Knowing proper troubleshooting tips helps you pinpoint the exact problem. The following sections list out some of the commonly used troubleshooting tips. We will expand the list as we proceed in this book.

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

```bash
$ mvn clean install -X
```

Building dependency tree
If you find any issues with any dependencies 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:

```bash
$ mvn dependency:tree
```

The following result 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:tree (default-cli) @ rampart-trust ---
```

[INFO] org.apache.ws.commons.axiom:axiom-api:jar:1.2.11-wso2v4:compile (version managed from 1.2.11)
Apache Maven Quick Start

[INFO] | +- org.apache.ws.commons.axiom:axiom-impl:jar:1.2.11-wso2v4:compile (version managed from 1.2.11)
[INFO] | +- javax.servlet:servlet-api:jar:2.3:compile

Viewing all 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 result is the truncated output of the previous command:

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

java.runtime.name=Java(TM) SE Runtime Environment
sun.boot.library.path=/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Libraries
java.vm.version=20.65-b04-462
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=9E4F0D49-180D-45F6-B6FB-DFA2DCBF4B77
M2_HOME=/usr/share/maven/maven-3.2.3/
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 the configuration parameters when those are not overridden at the project level configuration. This is exactly what we discussed under the *convention over configuration* section. If we take the same sample POM file we used before in this chapter, we can see how the effective POM file would look using the following command.

$ mvn help:effective-pom

This is also the best way to see what default values are being used by Maven. More details about the `effective-pom` command are discussed in *Chapter 2, Demystifying Project Object Model*. 
Viewing the dependency classpath
The following command lists all the JAR files and directories in the build classpath:

$ mvn dependency:build-classpath

The following result 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: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 of Maven to bring all the readers to a common ground. We discussed the basic steps to install and configure Maven in Ubuntu, Mac OS X, and Microsoft Windows operating systems. Then, we covered some of the common, useful Maven tips and tricks. As we proceed with the book, some of the concepts that we touched on in this chapter will be discussed in detail later.

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

You can buy Mastering Apache Maven 3 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.