cmvn - Configured Maven User Manual

Tobias Roeser

Version 0.1.6

Contents

1. Introduction

`cmvn` is a helper tool for developers targeted at the Java Virtual Machine (JVM). Its main focus is to declaratively describe and configure the required development and build environment.

A most significant difference between Java-targeted build systems compared to those for C/C++[1] is the lack of a defined configuration step before executing the compiler. Although building a Java application is a lot more easy compared to platform dependent programming languages and tools, nevertheless a lot of the (configuration) tasks of those other build systems are still required. Often the lack of a configuration process in Java build systems results in very obscure setups.

A very popular build system for Java is Apache Maven[2], currently in version 3. It partially helps the developer with managing her dependencies but fails miserably at producing reliable builds, at least without support of a complex build infrastructure.[3]

`cmvn` tries to close this gap by providing a configuration step before the actual build system. Concrete, this goal will be reached by generating the build scripts for the underlying (nativ) build system. Whenever a configuration has changed `cmvn` will first regenerate the build scripts and after that executes the underlying build system with the right (configured) settings. To assist the developer but do not stay in his way, `cmvn` does not aim to replace existing build chains. Instead, its main focus is adding another (first) configuration step to the build chain to create more reliable and reproducable build environments.

In its first release `cmvn` supports Apache Maven 2.0 and above[4]. In later releases support for various other build chains will be added, e.g. Apache Ant + Ivy, SBT, JackBuild, or others. Although `cmvn` generates the build scripts (in Maven case: `pom.xml` files) is does not can and it wants not handle all aspects of the underlying build infrastructure. To leave the full power to the developer, `cmvn` supports templates for the underlying build scripts for those settings `cmvn` can not generate. But for common project setup, this is rarely needed.

2. Execution Modes

`cmvn` can be run in different execution modes:

- ▷ Configuration
- ▷ Build
- ▷ Cleanup

[1]e.g. Autotools + GNU Make, CMake, Scons.
[2]http://maven.apache.org
[3]...like a Repository Manager.
[4]Using Apache Maven 3.0 is highly recommended.

The execution mode is given as (first) parameter when executing `cmvn`.

2.1. Configuration

Simplest Example:

```
shell> cmvn --configure
```

Just generate the needed build scripts (if needed) of the underlying buildsystem.

For Maven this generates a `pom.xml` and a Maven settings file `.cmvn/settings.xml` in a local hidden directory.

2.1.1. Automatic re-configuration

To reconfigure an already configured project, e.g. because the `cmvn` config file has changed or a generated file is missing, one can use the option `--reconfigure` which does exactly the same as `--configure` except that the concrete initial configuration is preserved, thus only the files were recreated without changing the current configuration.

```
shell> cmvn --reconfigure
```

If `cmvn` detects, that the current project is not up-to-date, it must be used with `--reconfigure`. To avoid the burden of beeing forced to run a `cmvn --reconfigure` after each change of the project (or sub project) configuration, the option `--auto-reconfigure` can be used together with `--configure`.

```
shell> cmvn --configure --auto-reconfigure
```

Configured that way, `cmvn` will automatically reconfigure the project (and the whole project tree) before a build, if needed.

Since cmvn-0.1.2 `--autoreconfigure` is the default.

2.1.2. Maven Settings

By default, `cmvn --configure` initally created an new project-local Maven settings file and thus uses a project-local repository. This is intended to isolate projects from each other while still maintaining project-interoperability via (remotely) realeased dependencies. This default way enables the developer to easy build branches without fearing of interferences and inconsistencies caused by multiple projects (branches) that are releasing to the same local repository.

Of course, the newly created repository and Maven settings file is shared between all sub projects of the one you just configured.

In case, this default behavior is not desired, you can tell `cmvn` to use an alternative existing Maven settings file with the option `--maven-settings`. In this case, you may will loose the benefits of side-effect free development of multiple project on the same computer. Also this may limit the reproducability of the build process in different environments.

```
shell> cmvn --configure --maven-settings /home/user/.m2/settings.xml
```

Notice, that if you use an alternative Maven settings file, `cmvn` will not touch this file and the Local Maven Repository when running in cleanup execution mode.

2.1.3. Using Templates – `pom.xml.cmvn`

When `cmvn` detects the presence of a file `pom.xml.cmvn` it will use it as template when generating the `pom.xml` file for Maven. `cmvn` will first read the template file and afterward extend it with the settings found in `cmvn.conf`. You can use this to easily migrate existing Maven projects or if you need complex setups and Maven features (like `<profile>`).

2.2. Build

Maven Example: Clean project build and install the build jar file into the local Maven repository.

```
shell> cmvn --build clean install
```

The build execution mode is automatically enabled if no other mode was requested and at least one non-option argument was given to `cmvn`. So the example above could also be written as:

```
shell> cmvn clean install
```

If `cmvn` is run without any option and parameter but the project was configured with the `reconfigure`-option, all necessary project files will be regenerated automatically if needed.[5]

[5]Without the `auto-reconfigure` setting the same behavior can be achieved by running `cmvn --reconfigure`.

2.3. Cleanup

The execution mode cleanup is used to remove all generated files and the configuration data. Currently there are two variants to enable the cleanup mode: one version enabled with `--clean` removes only the generated native build scripts, the other variant `--distclean` cleans also the configuration state and any other generated environment setup, e.g. a hidden project local Maven repository.

```
shell> cmvn --clean
```

Cleans up all generated native build scripts.

```
shell> cmvn --distclean
```

Cleans up all generated files including configured state.

3. The configuration file `cmvn.conf`

3.1. Config file syntax

The config file has a very simplistic human readable and editable format:

1. *empty lines* were ignored
2. the hash sign (#) starts a *comment* until end of line
3. each non-comment line consists of a pair of *key* and *value* delimited by a colon (`:`)
4. keys starting with a hyphen (−) are *directives* all other keys were *settings*
5. values may have *options*, in which case options are separated by a semicolon (`;`)
6. value-options are themselves key-value pairs delimited by equal sign (=)
7. if an option-value is ommitted (an option without an equal sign) it is evaluated to '`true`'
8. non-comment lines ending with a backslash (\) were *continued* on the next line

3.2. Config file example

The following is an example project config file `cmvn.conf`:

```
# Include directive
-include: ../common/cmvncommon.conf

# Immutable variable directive
-val: EXAMPLE_VERSION=0.0.1

# project settings using a variable
# cmvn uses a short syntax for projects and dependencies
# group:artifact:version (GAV) or org:name:rev
project: de.tototec:de.tototec.example:$${EXAMPLE_VERSION}

# a dependency with option spreading two lines
compile: de.tototec:de.tototec.example.utils:$${EXAMPLE_VERSION}; \
 classifier=jdk15

# compile-scope dependency
compile: org.slf4j:slf4j-api:1.6.1

# optional runtime-dependency
runtime: ch.qos.logback:logback-classic:0.9.26;optional

# test-scope dependency
test: org.testng:testng:5.14.6
```

3.3. Legend

The following sections contain tables with uses the following keyword in the format column:

BOOLEAN	A boolean value: '`true`' or '`false`'
DIR	A directory in the local file system
FILE	A file in the local file system
GAV	*groupId*:*artifactId*:*version* (analog to Maven) or *org*:*name*:*rev* (analog to Ivy)
GA	Same as GAV, but without a version
LIST[X]	A semicolon delimited list of X (if ommitted, than text)
OPTION	A *key*=*value* pair
TEXT	Text
URL	A URL
XML	A XML fragement

3.4. Directives

Directives are instructions to `cmvn` to do something special.

Directive	*Format*	*Description*
`-configClass`	LIST[OPTION]	Generate a Java class as source code containing static methods.
`-include`	FILE	Include the content of the given file. The content will be threated as if it was in the actual file.
`-val`	OPTION	Create an immutable variable *key* with content *value*. All occurences of this variable were expanded in the value-part of all succeeding lines (except `-include`).

3.5. Settings

Settings are used to generate the underlying (native) build scripts. Currently the only supported buildsystem is Maven 2 or greater.

Setting	*Format*	*Description*
`artifactrepo`	URL	Alias for `repository` with option `plugins=false`
`build`	LIST[OPTION]	List of options for the `<build>`-block
`plugin`	GAV	Maven plugin configuration
`project`	GAV	Project coordinates
`compile`	GAV	Alias for `dependency` with option `scope=compile`
`dependency`	GAV	A project/package dependency
`eclispeClasspath`	LIST[OPTION]	List of options to generate a `.classpath` file used by Eclipse
`dependencyManagement`	GAV	Managed dependency in `dependencyManagement`-block
`exclude`	GA	Exclude the given dependency in transitive dependencies. Will generate exclusion-blocks in any dependency-block.
`module`	DIR	The path of a sub project
`pluginrepo`	URL	Alias for `repository` with option `artifacts=false`
`property`	OPTION	Definition of property *key* with value *value*
`repo`	URL	Alias for `repository`
`repository`	URL	Maven Repository
`runtime`	GAV	Alias for `dependency` with option `scope=runtime`
`system`	GAV	alias for `dependency` with option `scope=system`
`test`	GAV	Alias for `dependency` with option `scope=test`

3.5.1. `project`

Essential project information mandatory for Maven.

Format: GAV[;OPTION]*

Options:

Option	*Format*	*Desciption*
`packaging`	TEXT	The packaging of the project, if ommitted, than '`jar`'

Example:

```
# using maven-bundle-plugin
project: org.example:org.example.osgibundle:1.0.0;packaging=bundle
```

3.5.2. `module`

Definition of a sub project.

Format: DIR[;OPTION]*

Options:

Option	*Format*	*Desciption*
`skipCmvn`	BOOLEAN	This sub project is a pure Maven project. Do not try to find a `cmvn.conf` file.
`skipEmvn`	BOOLEAN	Alias for `skipCmvn` (for compatibility).

Example:

```
module: org.example.domain
module: org.example.service
module: org.example.service.impl.legacy;skipCmvn
```

3.5.3. `dependency`

A dependency referencing a project in a Maven repository (in most cases a `*.jar` file).

Format: GAV[;OPTION]*

Options:

Option	*Format*	*Desciption*
`scope`	TEXT	The scope of the dependency. One of '`compile`', '`runtime`', `test`', `system`' or '`provided`'.
`systemPath`	FILE	The local file path to the jar file. Only valid if `scope` is `system`. In contrast to Maven specification, this path can be also relative.
`classifier`	TEXT	The classifier, e.g. '`sources`'.
`type`	TEXT	The type.
`optional`	BOOLEAN	An optional dependency is not optional for the current project but will be ignored in a transitive dependency resolution. (In an ideal world any compile type dependency should be optional!)
`exclude`	GA	Excluded dependency from transitive resolved dependency tree.
`forceversion`	BOOLEAN	Additionally the dependency will be added to the `<dependencyManagement>`-block. This enforces the given version and is somethimes an alternative to the `exclude` option (and vice versa).

Aliases:

`compile`	A dependency with option `scope=compile`.
`test`	A dependency with option `scope=test`.
`runtime`	A dependency with option `scope=runtime`.
`system`	A dependency with option `scope=system`.
`dependencyManagement`	A managed dependency only in `dependencyManagement`-block.

Example:

```
compile: org.slf4j:slf4j-api:1.6.1;optional
compile: org.slf4j:jcl-over-slf4j:1.6.1;optional;forceversion
test: org.testng:testng:6.0.1
```

3.5.4. `property`

Define a property in a `<properties>`-block.

Format: OPTION

Example:

```
property: maven.compiler.source=1.6
property: maven.compiler.target=1.6
property: project.build.sourceEncoding=UTF-8
```

3.5.5. `repository`

A remote Maven repository used to download dependencies.

Format: URL

Options:

Option	*Format*	*Desciption*
`plugins`	BOOLEAN	Can be used to download Maven plugins (default: `true`).
`artifacts`	BOOLEAN	Can be used to download Maven artifacts (default: `true`).
`releases`	BOOLEAN	Can be used to download released dependencies.
`snapshots`	BOOLEAN	Can be used to download snapshot dependencies.

Aliases:

`repo`	Same as `repository`.
`pluginrepo`	A repository with option `artifacts=false`.
`artifactrepo`	A repository with option `plugins=false`.

3.5.6. `plugin`

A Maven plugin contribution to the Maven lifecycle.

Format: GAV

Options: Any option has the format OPTION and is added to the `<configuration>`-block of the plugin definition.

Directives:

Directive	*Format*	*Desciption*
`-extension`	BOOLEAN	Specify if this plugin is a extensions-plugin (and thus e.g. can contribute new project packaging types).
`-execution`	XML	A free XML fragement that will be placed inside the `<executions>`-block of this plugin.
`-xml:`*anyOption*	XML	Can be used if the option-value is XML and not text.

Example:

```
plugin: org.apache.maven.plugins:maven-assembly-plugin:2.2-beta-5; \
 appendAssemblyId=false; \
 -xml:descriptorRefs= \
    <descriptorRef>jar-with-dependencies</descriptorRef>; \
 -xml:archive= \
    <manifest> \
      <mainClass>org.example.Main</mainClass> \
    </manifest>
```

3.5.7. `build`

Redefine some project default settings.

Format: LIST[OPTION]

Options:

Option	*Format*	*Desciption*
`sources`	DIR	The directory containing the source files.
`testSources`	DIR	The directory containing the test source files.
`finalName`	FILE	The name of the final build JAR file.
`targetDir`	DIR	The directory containing the build output files (e.g. `target`).

3.5.8. `eclipseClasspath`

Generate a .classpath file which can be used by Eclipse to generate the project classpath container.

Format: LIST[OPTION]

Options:

Option	*Format*	*Desciption*
`autoGenerate`	TEXT	Auto-generate lib-entries for project dependencies of the given scope. Supported scopes are: compile (includes provided and system), test, runtime.
`optional`	BOOLEAN	Generate an optional-marker for the actual entry.
key	TEXT	A free «key» added as attribute in the classpathentry-element. Known supported attributes are, e.g.: kind, path, output, sourcepath, ...

Example:

Multiple given eclipseClasspath settings to configure a Java 6 project with tests.

```
eclipseClasspath: kind=src;path=src/main/java
eclipseClasspath: kind=src;path=src/main/resources
eclipseClasspath: kind=output;path=target/classes
eclipseClasspath:
   kind=src;output=target/test-classes;path=src/test/java
eclipseClasspath:
   kind=con;path=org.eclipse.jdt.launching.JRE_CONTAINER/\\
org.eclipse.jdt.internal.debug.ui.launcher.StandardVMType/JavaSE-1.6
eclipseClasspath: autoGenerate=compile
eclipseClasspath: autoGenerate=test
```

4. Terms of Use (License)

`cmvn` is published under the Apache License, Version 2.0.

http://www.apache.org/licenses/LICENSE-2.0

A. Shell Wrapper

cmvn is distributed as executable jar including all its required dependencies.

For convenience, you may want to create a simple shell script cmvn as an executable wrapper around the program:

Listing 1: Shell wrapper: mvu

```
#!/bin/sh
# pass all arguments to cmvn with $@
exec java -jar cmvn-executable-0.1.3.jar $@
```

B. Command Shell Wrapper (Windows)

Listing 2: Windows Command Shell wrapper: cmvn.bat

```
:init
@REM Decide how to startup depending on the version of windows

@REM -- Windows NT with Novell Login
if "%OS%"=="WINNT" goto WinNTNovell

@REM -- Win98ME
if NOT "%OS%"=="Windows_NT" goto Win9xArg

:WinNTNovell

@REM -- 4NT shell
if "%@eval[2+2]" == "4" goto 4NTArgs

@REM -- Regular WinNT shell
set CMVN_CMD_LINE_ARGS=%*
goto endInit

@REM The 4NT Shell from jp software
:4NTArgs
set CMVN_CMD_LINE_ARGS=%$
goto endInit

:Win9xArg
@REM Slurp the command line arguments.  This loop allows for an
   unlimited number
@REM of agruments (up to the command line limit, anyway).
set CMVN_CMD_LINE_ARGS=
:Win9xApp
if %1a==a goto endInit
set CMVN_CMD_LINE_ARGS=%CMVN_CMD_LINE_ARGS% %1
```

```
shift
goto Win9xApp

@REM Reaching here means variables are defined and arguments have been
   captured
:endInit
SET CMVN_JAVA_EXE="%JAVA_HOME%\bin\java.exe"

%CMVN_JAVA_EXE% -jar cmvn-executable.jar %CMVN_CMD_LINE_ARGS%

set CMVN_JAVA_EXE=
set CMVN_CMD_LINE_ARGS=
```