(/)

The settings.xml File in Maven

Last updated: May 21, 2022

Written by: Daniel Strmecki
(https://www.baeldung.com/author/danielstrmecki)

Maven (https://www.baeldung.com/category/maven)

Maven Basics (https://www.baeldung.com/tag/maven-basics)

1. Overview

While using Maven, we keep most of the project-specific configuration in the *pom.xml*.

Maven provides a settings file, *settings.xml,* which allows us to specify which local and remote repositories it will use. We can also use it to store settings that we don't want in our source code, such as credentials.

In this tutorial, we'll learn how to use the *settings.xml*. We'll look at proxies, mirroring, and profiles. We'll also discuss how to determine the current settings that apply to our project.

Further reading:

Default Values for Maven Properties (/maven-properties-defaults)

In this short article, we'll go through how to configure Maven properties default values, and how to use them.

Read more (/maven-properties-defaults) →

Additional Source Directories in Maven (/maven-add-src-directories)

Learn how to configure additional source directories in Maven.

Read more (/maven-add-src-directories) →

Maven Packaging Types (/maven-packaging-types)

In this article, we explore the different packaging types available in Maven.

Read more (/maven-packaging-types) →

2. Configurations

The *settings.xml* file configures a Maven (/maven) installation. It's similar to a *pom.xml* file but is defined globally or per user.

Let's explore the elements we can configure in the *settings.xml* file. The main *settings* element of the *settings.xml* file can contain nine possible predefined child elements:

```
<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
https://maven.apache.org/xsd/settings-1.0.0.xsd">
    <localRepository/>
    <interactiveMode/>
    <offline/>
    <pluginGroups/>
    <servers/>
    <mirrors/>
    <proxies/>
    <profiles/>
    <activeProfiles/>
</settings>
```

2.1. Simple Values

Some of the top-level configuration elements contain simple values:

```
<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
https://maven.apache.org/xsd/settings-1.0.0.xsd">
    <localRepository>${user.home}/.m2/repository</localRepository>
    <interactiveMode>true</interactiveMode>
    <offline>false</offline>
</settings>
```

The *localRepository* element points to the path of the system's local repository. The **local repository (/maven-local-repository) is where all the dependencies from our projects get cached**. The default is to use the user's home directory. However, we could change it to allow all logged-in users to build from a common local repository.

The *interactiveMode* flag defines if we allow Maven to interact with the user asking for input. This flag defaults to *true*.

The *offline* flag determines if the build system may operate in offline mode. This defaults to *false;* however, we can switch it to *true* in cases where the build servers cannot connect to a remote repository.

2.2. Plugin Groups

The *pluginGroups* element contains a list of child elements that specify a *groupId*. A *groupId* is the unique identifier of the organization that created a specific Maven artifact:

```
<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
https://maven.apache.org/xsd/settings-1.0.0.xsd">
    <pluginGroups>
        <pluginGroup>org.apache.tomcat.maven</pluginGroup>
    </pluginGroups>
</settings>
```

Maven searches the list of plugin groups when a plugin is used without a groupId provided at the command line. The list contains the groups *org.apache.maven.plugins* and *org.codehaus.mojo* by default.

The *settings.xml* file defined above allows us to execute truncated Tomcat plugin commands:

```
mvn tomcat7:help
mvn tomcat7:deploy
mvn tomcat7:run
```

2.3. Proxies

We can configure a proxy for some or all of Maven's HTTP requests. The *proxies* element allows a list of child proxy elements, but **only one proxy can be active at a time**:

```
<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
https://maven.apache.org/xsd/settings-1.0.0.xsd">
    <proxies>
        <proxy>
            <id>baeldung-proxy</id>
            <active>true</active>
            <protocol>http</protocol>
            <host>baeldung.proxy.com</host>
            <port>8080</port>
            <username>demo-user</username>
            <password>dummy-password</password>

<nonProxyHosts>*.baeldung.com|*.apache.org</nonProxyHosts>
        </proxy>
    </proxies>
</settings>
```

We define the currently active proxy via the *active* flag. Then with the *nonProxyHosts* element, we specify which hosts are not proxied. The delimiter used depends on the specific proxy server. The most common delimiters are pipe and comma.

2.4. Mirrors

Repositories can be declared inside a project *pom.xml*. This means that the developers sharing the project code get the right repository settings out of the box.

We can use mirrors in cases where we want to **define an alternative mirror for a particular repository**. This overrides what's in the *pom.xml*.

For example, we can force Maven to use a single repository by mirroring all repository requests:

```
<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
https://maven.apache.org/xsd/settings-1.0.0.xsd">
    <mirrors>
        <mirror>
            <id>internal-baeldung-repository</id>
            <name>Baeldung Internal Repo</name>
            <url>https://baeldung.com/repo/maven2/</url>
            <mirrorOf>*</mirrorOf>
        </mirror>
    </mirrors>
</settings>
```

We may define only one mirror for a given repository and Maven will pick the first match. Normally, **we should use the official repository** distributed worldwide via CDN.

2.5. Servers

Defining repositories in the project *pom.xml* is a good practice. However, we shouldn't put security settings, such as credentials, into our source code repository with the *pom.xml*. Instead, we **define this secure information in the *settings.xml* file**:

```
<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
https://maven.apache.org/xsd/settings-1.0.0.xsd">
    <servers>
        <server>
            <id>internal-baeldung-repository</id>
            <username>demo-user</username>
            <password>dummy-password</password>
            <privateKey>${user.home}/.ssh/bael_key</privateKey>
            <passphrase>dummy-passphrase</passphrase>
            <filePermissions>664</filePermissions>
            <directoryPermissions>775</directoryPermissions>
            <configuration></configuration>
        </server>
    </servers>
</settings>
```

We should note that the *ID* of the server in the *settings.xml* needs to match the *ID* element of the repository mentioned in the *pom.xml*. The XML also allows us to use placeholders to pick up credentials from environment variables.

3. Profiles

The *profiles* element enables us to create multiple *profile* (/maven-profiles) child elements differentiated by their *ID* child element. The *profile* element in the *settings.xml* is a truncated version of the same element available in the *pom.xml*.

It can contain only four child elements:
activation, *repositories*, *pluginRepositories,* and *properties.* These elements configure the build system as a whole, instead of any particular project.

It's important to note that values from an active profile in *settings.xml* will **override any equivalent profile values in a *pom.xml*** or *profiles.xml* file. Profiles are matched by *ID*.

3.1. Activation

We can use profiles to modify certain values only under given circumstances. We can specify those circumstances using the *activation* element. Consequently, profile **activation occurs when all specified criteria are met**:

```
<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
https://maven.apache.org/xsd/settings-1.0.0.xsd">
    <profiles>
        <profile>
            <id>baeldung-test</id>
            <activation>
                <activeByDefault>false</activeByDefault>
                <jdk>1.8</jdk>
                <os>
                    <name>Windows 10</name>
                    <family>Windows</family>
                    <arch>amd64</arch>
                    <version>10.0</version>
                </os>
                <property>
                    <name>mavenVersion</name>
                    <value>3.0.7</value>
                </property>
                <file>
                    <exists>${basedir}/activation-
file.properties</exists>
                    <missing>${basedir}/deactivation-
file.properties</missing>
                </file>
            </activation>
        </profile>
    </profiles>
</settings>
```

There are four possible activators and not all of them need to be specified:

- *jdk:* activates based on the JDK version specified (ranges are supported)
- *os:* activates based on operating system properties
- *property:* activates the profile if Maven detects a specific property value
- *file:* activates the profile if a given filename exists or is missing

In order to check which profile will activate a certain build, we can use the Maven help plugin:

```
mvn help:active-profiles
```

The output will display currently active profiles for a given project:

```
[INFO] --- maven-help-plugin:3.2.0:active-profiles (default-cli) @
core-java-streams-3 ---
[INFO]
Active Profiles for Project 'com.baeldung.core-java-modules:core-
java-streams-3:jar:0.1.0-SNAPSHOT':
The following profiles are active:
 - baeldung-test (source: com.baeldung.core-java-modules:core-java-
streams-3:0.1.0-SNAPSHOT)
```

3.2. Properties

Maven properties can be thought of as named placeholders for a certain value. The values are **accessible within a *pom.xml* file using the *${property_name}* notation**:

```
<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
https://maven.apache.org/xsd/settings-1.0.0.xsd">
    <profiles>
        <profile>
            <id>baeldung-test</id>
            <properties>
                <user.project.folder>${user.home}/baeldung-
tutorials</user.project.folder>
            </properties>
        </profile>
    </profiles>
</settings>
```

Four different types of properties are available in *pom.xml* files:

- Properties using the *env* prefix return an environment variable value, such as *${env.*PATH}.
- Properties using the *project* prefix return a property value set in the *project* element of the *pom.xml*, such as *${project.version}.*
- Properties using the *settings* prefix return the corresponding element's value from the *settings.xml*, such as *${settings.localRepository}.*
- We may reference all properties available via the *System.getProperties* method in Java directly, such as *${java.home}.*
- We may use properties set within a *properties* element without a prefix, such as *${junit.version}.*

3.3. Repositories

Remote repositories contain collections of artifacts that Maven uses to populate our local repository. Different remote repositories may be needed for particular artifacts. Maven **searches the repositories enabled under the active profile**:

```
<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
https://maven.apache.org/xsd/settings-1.0.0.xsd">
    <profiles>
        <profile>
            <id>adobe-public</id>
            <repositories>
                <repository>
                    <id>adobe-public-releases</id>
                    <name>Adobe Public Repository</name>

<url>https://repo.adobe.com/nexus/content/groups/public</url>
                    <releases>
                        <enabled>true</enabled>
                        <updatePolicy>never</updatePolicy>
                    </releases>
                    <snapshots>
                        <enabled>false</enabled>
                    </snapshots>
                </repository>
            </repositories>
        </profile>
    </profiles>
</settings>
```

We can use the *repository* element to enable only release or snapshots versions of artifacts from a specific repository.

3.4. Plugin Repositories

There are two standard types of Maven artifacts, dependencies and plugins. As Maven plugins are a special type of artifact, we may **separate plugin repositories from the others**:

```
<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
https://maven.apache.org/xsd/settings-1.0.0.xsd">
    <profiles>
        <profile>
            <id>adobe-public</id>
            <pluginRepositories>
               <pluginRepository>
                  <id>adobe-public-releases</id>
                  <name>Adobe Public Repository</name>

<url>https://repo.adobe.com/nexus/content/groups/public</url>
                  <releases>
                      <enabled>true</enabled>
                      <updatePolicy>never</updatePolicy>
                  </releases>
                  <snapshots>
                      <enabled>false</enabled>
                  </snapshots>
               </pluginRepository>
            </pluginRepositories>
        </profile>
    </profiles>
</settings>
```

Notably, the structure of the *pluginRepositories* element is very similar to the *repositories* element.

3.5. Active Profiles

The *activeProfiles* element contains child elements that refer to a specific profile *ID*. **Maven automatically activates any profile referenced here**:

```
<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
https://maven.apache.org/xsd/settings-1.0.0.xsd">
    <activeProfiles>
        <activeProfile>baeldung-test</activeProfile>
        <activeProfile>adobe-public</activeProfile>
    </activeProfiles>
</settings>
```

In this example, every invocation of *mvn* is run as though we've added *-P baeldung-test,adobe-public* to the command line.

4. Settings Level

A *settings.xml* file is usually found in a couple of places:

- Global settings in Mavens home directory: *${maven.home}/conf/settings.xml*
- User settings in the user's home: *${user.home}/.m2/settings.xml*

If both files exist, their contents are merged. **Configurations from the user settings take precedence**.

4.1. Determine File Location

In order to determine the location of global and user settings, we can run Maven using the debug flag and search for *"settings"* in the output:

```
$ mvn -X clean | grep "settings"


[DEBUG] Reading global settings from C:\Program Files
(x86)\Apache\apache-maven-3.6.3\bin\..\conf\settings.xml
[DEBUG] Reading user settings from
C:\Users\Baeldung\.m2\settings.xml
```

4.2. Determine Effective Settings

We can use the Maven help plugin to **find out the contents of the combined global and user settings**:

```
mvn help:effective-settings
```

This describes the settings in XML format:

```
<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
https://maven.apache.org/xsd/settings-1.0.0.xsd">

<localRepository>C:\Users\Baeldung\.m2\repository</localRepository>
    <pluginGroups>
        <pluginGroup>org.apache.tomcat.maven</pluginGroup>
        <pluginGroup>org.apache.maven.plugins</pluginGroup>
        <pluginGroup>org.codehaus.mojo</pluginGroup>
    </pluginGroups>
</settings>
```

4.3. Override the Default Location

Maven also allows us to override the location of the global and user settings via the command line:

```
$ mvn clean --settings c:\user\user-settings.xml --global-settings
c:\user\global-settings.xml
```

We can also use the shorter *–s* version of the same command:

```
$ mvn clean --s c:\user\user-settings.xml --gs c:\user\global-
settings.xml
```

5. Conclusion

In this article, we **explored the configurations available in Maven's *settings.xml* file**.

We learned how to configure proxies, repositories and profiles. Next, we looked at the difference between global and user settings files, and how to determine which are in use.

Finally, we looked at determining the effective settings used, and overriding default file locations.

Comments are closed on this article!

THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA “BACK TO BASICS” TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
APACHE HTTPCLIENT TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)
SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE FULL ARCHIVE (/FULL_ARCHIVE)
EDITORS (/EDITORS)
JOBS (/TAG/ACTIVE-JOB/)
OUR PARTNERS (/PARTNERS)
PARTNER WITH BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)