Oracle® Fusion Middleware Developing Applications Using Continuous Integration

7 Understanding Maven Version Numbers

In a Maven environment, it is very important to understand the use of version numbers. A well thought out strategy can greatly simplify your dependency management workload. This chapter presents important concepts about how version numbers work in Maven in general, and also some specific details of how the Oracle-supplied artifacts use version numbers and how you should use them when referring to Oracle artifacts.

This chapter includes the following sections:

7.1 How Version Numbers Work in Maven

Maven's versioning scheme uses the following standards:

- MajorVersion
- MinorVersion
- IncrementalVersion
- BuildNumber
- Qualifier

For example:

- MajorVersion: 1.2.1
- MinorVersion: 2.0
- IncrementalVersion: 1.2-SNAPSHOT
- BuildNumber: 1.4.2-12
- Qualifier: 1.2-beta-2

All versions with a qualifier are older than the same version without a qualifier (release version).

For example:

1.2-beta-2 is older than 1.2.

Identical versions with different qualifier fields are compared by using basic string comparison.

For example:

1.2-beta-2 is newer than 1.2-alpha-6.

If you do not follow Maven versioning standards in your project versioning scheme, then for version comparison, Maven interprets the entire version as a simple string. Maven and its core plug-ins use version comparison for a number of tasks, most importantly, the release process.

If you use a nonstandard versioning scheme, Maven release and version plug-in goals might not yield the expected results. Because basic string comparison is performed on nonstandard versions, version comparison calculates the order of versions incorrectly in some cases.

For example, Maven arranges the version list in the following manner:

1.0.1.0
1.0.10.1
1.0.10.2
1.0.9.3
Version `1.0.9.3` should come before `1.0.10.1` and `1.0.10.2`, but the unexpected fourth field (`.3`) forced Maven to evaluate the version as a string.

An example of this effect on Maven is found in the Maven Versions plug-in. The Maven Versions plug-in provides goals to check your project dependencies for currency in a different ways. One useful goal is **`versions:dependency-updates-report`**. The **`versions:dependency-updates-report`** goal examines a project's dependency hierarchy and reports which ones have newer releases available. When you are coordinating a large release, this goal can help you to find stale references in dependency configuration. If Maven incorrectly identifies a newer release, then it is also reported incorrectly in the plug-in. Given the preceding example sequence, if your current reference was `1.0.10.2`, then the plug-in would report `1.0.9.3` as a newer release.

Version resolution is also very important if you intend to use version ranges in your dependency references. See Section 7.3 (#A1000686) for information about version changes.

7.2 The SNAPSHOT Qualifier

Maven treats the SNAPSHOT qualifier differently from all others. If a version number is followed by -SNAPSHOT, then Maven considers it the "as-yet-unreleased" version of the associated MajorVersion, MinorVersion, or IncrementalVersion.

In a continuous integration environment, the SNAPSHOT version plays a vital role in keeping the integration build up-to-date while minimizing the amount of rebuilding that is required for each integration step.

SNAPSHOT version references enable Maven to fetch the most recently deployed instance of the SNAPSHOT dependency at a dependent project build time. Note that the SNAPSHOT changes constantly. Whenever an agent deploys the artifact, it is updated in the shared repository. The SNAPSHOT dependency is refetched, on a developer's machine or it is updated in every build. This ensures that dependencies are updated and integrated with the latest changes without the need for changes to the project dependency reference configuration.

Usually, only the most recently deployed SNAPSHOT, for a particular version of an artifact is kept in the artifact repository. Although the repository can be configured to maintain a rolling archive with a number of the most recent deployments of a given artifact, the older instances are typically used only for troubleshooting purposes and do not play a role in integration.

Continuous build servers that include the ability to define and execute a job based on a Maven project, such as Hudson, can be configured to recognize when a SNAPSHOT artifact is updated and then rebuild projects that have a dependency on the updated artifact.

For example, a Hudson build configuration that maps to a Maven Project Object Model has a SNAPSHOT dependency. Hudson periodically checks the artifact repository for SNAPSHOT updates. When it detects the update of the project's dependency, it triggers a new build of the project to ensure that integration is performed with the most recent version of the dependency. If other projects have a dependency on this project, they too are rebuilt with updated dependencies.

7.3 Version Range References

Maven enables you to specify a range of versions that are acceptable to use as dependencies. Table 7-1 (#CJHDEHAB) shows a range of version specifications:

Table 7-1 Version Range References

Range	Meaning
(,1.0]	x <= 1.0
1.0	It generally means 1.0 or a later version, if 1.0 is not available. Various Maven plug-ins may interpret this differently, so it is safer to use one of the other, more specific options.

Range	Meaning
[1.0]	Exactly 1.0
[1.2,1.3]	1.2 <= x <= 1.3
[1.0,2.0)	1.0 <= x < 2.0
[1.5,)	x >= 1.5
(,1.0],[1.2,)	x <= 1.0 or x >= 1.2. Multiple sets are separated by a comma.
(,1.1),(1.1,)	This excludes 1.1 if it is known not to work in combination with the library.

When Maven encounters multiple matches for a version reference, it uses the highest matching version. Generally, version references should be only as specific as required so that Maven is free to choose a new version of dependencies where appropriate, but knows when a specific version must be used. This enables Maven to choose the most appropriate version in cases where a dependency is specified at different points in the transitive dependency graph, with different versions. When a conflict like this occurs, Maven chooses the highest version from all references.

Given the option to use version ranges, you may wonder if there is still utility in using SNAPSHOT versions. Although you can achieve some of the same results by using a version range expression, a SNAPSHOT works better in a continuous build system for the following reasons:

- Maven artifact repository managers deal with SNAPSHOTs more efficiently than next version ranges. Because a single artifact can be deployed multiple times in a day, the number of unique instances maintained by the repository can increase very rapidly.
- Non-SNAPSHOT release versions are meant to be maintained indefinitely. If you are constantly releasing a new version and incrementing the build number or version, the storage requirements can quickly become unmanageable. Repository managers are designed to discard older SNAPSHOTs to make room for new instances so the amount of storage required stays constant.
- SNAPSHOTs are also recognized by Maven and Maven's release process, which affords you some benefits when performing a release build.

7.4 Understanding Maven Version Numbers in Oracle Provided Artifacts

The two important scenarios where Maven version numbers are used in Oracle provided artifacts are as follows:

- In the Maven coordinates of the artifact, that is, in the `project.version` of the artifact's POM
- In the dependency section of POMs to refer to other artifacts

This section provides details on how version numbers are defined for Oracle artifacts in both the scenarios.

7.4.1 Version Numbers in Maven Coordinates

The version number of the artifact defined in the POM file is the same as the version number of the released product, for example 12.1.2.0.0, expressed using five digits, as described in the following:

In `x.x.x-y-z` :

- `x.x.x` is the release version number, for example 12.1.2.
- `y` is the `PatchSet` number, for example `0,1,2,3,…` with no leading zeros.
- `z` is the `Bundle Patch` number, for example `0,1,2,3,…` with no leading zeros.
- The periods and hyphens are literals.

> **Note:**
> The version numbers of artifacts (as specified in `project.version` in the POM) use a different format than version number ranges used in dependencies (as specified in `project.dependencies.dependency.version` in the POM).

The release version number of Oracle-owned components do not change by a one-off patch. The release version number changes with a release and always matches the release, even if the component has not changed from the previous release.

The PatchSet (fourth position) changes when you apply a PatchSet. The Bundle Patch (fifth position) changes when you apply a `Bundle Patch` , `Patch set Update` , or equivalent (the name of this type of patch varies from product to product).

Following are the examples of valid version numbers:

```
12.1.2-0-0 12.1.2-1-0 12.1.2-2-0 12.1.2-0-1 12.1.2-1-1 12.1.2-2-1 ... 12.1.2-0-10 12.1.2-
1-1 12.1.2-2-1
```

7.4.2 Version Number Ranges in Dependencies

The two important scenarios where dependencies on Oracle-provided Maven artifacts are specified are as following:

- Inside the POM files of artifacts that are part of the Oracle product
- Inside POM files that you include in your own projects

The version number range should be specified in both the scenarios. This section describes how version number ranges are specified in Oracle-provided artifacts and when you are declaring a dependency on an Oracle-provided artifact.

When specifying dependencies on other artifacts, the most specific correct syntax should be used to ensure that the definition does not allow an incorrect or unsuitable version of the dependency to be used.

In **`[x.x.x,y.y.y)`** :

- **`x.x.x`** is the release version number, for example 12.1.2
- **`y.y.y`** is the next possible release version number, for example, 12.1.3
- Brackets, periods, commands and parenthesis are literals

An example of the correct way to specify a dependency is as follows:

[12.1.2,12.1.3)

As Table 7-1 (#CJHDEHAB) shows, the previous example means that the latest available version is 12.1.2 or greater, but less than 12.1.3.

The version number scheme used by Oracle-provided artifacts ensures correct sorting of version numbers, for example, Maven will resolve the following versions in the order shown (from oldest to newest):

12.1.2-0-0, 12.1.2-0-1, 12.1.2-0-2, 12.1.2-0-10, 12.1.2-1-0, 12.1.2-1-1, 12.1.2-1-2, 12.1.2-1-10, 12.1.2-0-0, 12.1.3-0-0

If it is necessary to specify a dependency which relies on a certain PatchSet or Bundle Patch, for example, when a new API is introduced, you must include the fourth or fourth and fifth digits respectively.

For example:

```
[12.1.2-2,12.1.3) depends on 12.1.2 with PatchSet 2 [12.1.2-2-5,12.1.3) depends on 12.1.2
with PatchSet 2 and Bundle Patch 5
```