Maven 3.x Compatibility Notes

This document is intended for end users of Apache Maven that look into upgrading from version 2.x to 3.x and provides an overview of potential effects on existing builds.

- General Changes
  - profiles.xml
  - Site and Reporting
  - Legacy-style Repositories
  - Caching of Artifact Resolution Errors
  - Snapshot Updates
  - Non-unique Snapshot Deployments
  - Unique Snapshot Versions and Classifiers
  - Automatic Plugin Version Resolution
  - Plugin Metaversion Resolution
  - Plugin Repositories
  - Resolution from Local Repository
  - Dependency Resolution
  - Version Comparison
  - Legacy-style Reactor Mode using Includes/Excludes
  - Stricter POM Validation
  - Parent POM Resolution
  - Extension Loading
  - Transport Protocols (Wagons)
  - Error Reporting

- Plugin Compatibility Matrix

General Changes

profiles.xml

The support for profiles outside of the POM or the settings.xml has been removed in version 3.x and users that previously used this feature are advised to move the affected profiles into their settings.xml.

Site and Reporting

A major aim of the refactoring in Maven 3.x was to decouple the Maven core from Doxia and to allow arbitrary reporting systems to be developed. For this reason, all reporting related code has been removed from the core of Maven 3. As a result, the 2.x versions of the Maven Site Plugin will no longer work with Maven 3. Development of an updated version of the Maven Site Plugin has already been started. See Maven 3.x and Site Plugin for further information.

In the same context, the default execution of site:attach-descriptor has been removed from the built-in lifecycle bindings for projects with packaging "pom". Users that actually use those projects to provide a common site descriptor for sub modules will need to explicitly define the following goal execution to restore the intended behavior:

```
...<build>
  <plugins>
    <plugin>
      <artifactId>maven-site-plugin</artifactId>
      <version>3.0-beta-2</version>
      <executions>
        <execution>
          <id>attach-descriptor</id>
          <goals>
            <goal>attach-descriptor</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
...```

Last but not least, plugin configuration in the <reporting> section no longer affects plugins in the <build> section of the POM. Users that employed this style of configuration will need to copy the desired configuration parameters for a plugin from the <reporting> section to the <build> section. When Maven 3 executes a plugin during the build lifecycle or directly from the command line, only plugin configuration from the <build> section is considered.
Legacy-style Repositories

Maven 3.x no longer supports repositories using `<layout>legacy</layout>`. Users that need to access repositories created with Maven 1.x are advised to use a repository manager that is capable of providing a Maven 2.x compatible view of the legacy repository.

Caching of Artifact Resolution Errors

To improve network performance, Maven 3 remembers when an artifact could not be resolved (by means of a marker file in the local repository). This cache expires according to the update policy of the respective repository and gets automatically cleared when the repository URL etc. changes. In some situations, e.g. when a repository manager is used and its configuration gets changed to proxy another repository, users might need to manually invalidate the cache by adding the flag `-U` to the command line, thereby forcing Maven to update the artifact status and retry to resolve it.

Snapshot Updates

Due to implementation differences, Maven 3 and Maven 2 use different files in the local repository to track metadata. For users that employ Maven 2 and Maven 3 (or integrators of Maven 3 like M2Eclipse) side by side on the same projects, this causes both Maven versions to independently check for updates of snapshot artifacts. In other words, even if one Maven version just checked for snapshot updates, the other Maven version will do again.

Non-unique Snapshot Deployments

The setting `<uniqueVersion>false</uniqueVersion>` for a distribution repository has no effect in version 3.x, snapshot artifacts will always be deployed using a timestamped version.

Unique Snapshot Versions and Classifiers

As described in [MNG-4452](https://issues.apache.org/), Maven 3 now supports deploying artifacts with different classifiers from a different build or source and having them properly resolved.

This may impact Maven 2 or other clients that use the repository metadata to determine the version, since the version of the POM would be out of sync with the latest version of some artifacts. While this behavior matches the result of attempting to do the same in Maven 2, please be aware that you will not be able to successfully make subsequent deployments from Maven 3 if you have Maven 2 clients accessing the repository.

Additionally, some repository managers and versions of Maven prior to 2.0.10 may not cope with the additional XML elements in the metadata file. If this is the case in your environment, you can disable the additional information by setting the `maven.metadata.legacy` system property to `true` when running Maven.

Automatic Plugin Version Resolution

When a plugin was invoked without an explicit version given in the POM or on the command line, Maven 2.x used to pick the latest version available where the latest version could either be a release or a snapshot. For the sake of stability, Maven 3.x prefers the latest release version over the latest snapshot version.

Given the threat of non-reproducible builds imposed by automatic plugin version resolution, this feature is scheduled for removal as far as plugin declarations in the POM are concerned. Users of Maven 3.x will find it output a warning when missing plugin versions are detected to encourage the addition of plugin versions to the POM or one of its parent POMs. The Enforcer rule `requirePluginVersions` can be used additionally check for missing plugin versions in the POM.

Plugin Metaversion Resolution

Internally, Maven 2.x used the special version markers `RELEASE` and `LATEST` to support automatic plugin version resolution. These metaversions were also recognized in the `<version>` element for a `<plugin>` declaration. For the sake of reproducible builds, Maven 3.x no longer supports usage of these metaversions in the POM. As a result, users will need to replace occurrences of these metaversions with a concrete version.

Plugin Repositories

Maven 3 aims at supporting a stricter separation between the compile/runtime/test dependencies of a project and the plugins used to build the project. For this reason, build extensions, plugins and plugin dependencies are no longer resolved from the `<repositories>` of a project but only from its `<pluginRepositories>`.

Resolution from Local Repository

The local repository used by Maven 3.x has been enhanced to keep track from what remote repositories an artifact was resolved from. This information about an artifact's origin is used to ensure builds can only access locally cached artifacts if they have the proper repositories configured. Hence, projects that lack required remote repositories in their POM might fail when being built with Maven 3. This improves reproducibility of a build by eliminating the effects of unintended artifact sharing via the local repository.

Dependency Resolution

The core of Maven 3 uses Aether for dependency resolution. Aether employs a different approach to calculate the transitive dependencies and is meant to fix some of the trickier (conflict) resolution issues within Maven 2.x.
In practice, this change can result in different class paths especially for projects with big/complex dependency graphs. For instance, projects that erroneously have conflicting dependencies on their classpath might encounter build issues depending on the classpath order. See JETTY-1257 for a concrete example of this issue.

Furthermore, not all parts of the Maven 2.x resolution API could be bridged onto Aether. Most notably the maven-dependency-tree shared component which is used for mvn dependency:tree still uses the legacy resolution code. As such, the output from mvn dependency:tree can differ from the actual dependency tree used by Maven itself to derive the classpaths of a project (see MSHARE-167 for an example of such a discrepancy). For now, the actual dependency trees can be inspected when running Maven with debug logging enabled.

Last but not least, Maven 3 inspects the POMs of all matching versions when processing version ranges to enable sophisticated conflict resolution.

**Version Comparison**

Maven 3 incorporates a revised version comparison logic to address some anomalies found with the version ordering in Maven 2.x, e.g. Maven 3 considers 1.0-alpha-1 < 1.0-beta-1 < 1.0-SNAPSHOT < 1.0. This can affect builds using version ranges that previously relied on the corner cases.

**Legacy-style Reactor Mode using Includes/Excludes**

Maven 3 dropped support for the command-line parameter --reactor to build sub projects selected by glob patterns. Future development will focus on improvements to the make-like reactor mode that provides similar functionality but is aware of the project hierarchy.

**Stricter POM Validation**

The validation of the POM has been extended in Maven 3.x and will report more errors than Maven 2.x. Potentially common problems are the duplicate declaration of project dependencies or bad scopes for plugin dependencies.

**Parent POM Resolution**

Maven 3 no longer resolves parent POMs from the local project checkout unless the child POM's <relativePath> element is accurate, whether explicitly given or using the default value. This improves consistency regarding the build result when building the child project in isolation and when performing a reactor build that includes the parent project. In Maven 2, building the child project in isolation could fail while the reactor build would succeed to resolve the parent.

**Extension Loading**

In Maven 2.x, build extensions defined by one module had global effects and as such affected other modules in the reactor. In contrast, Maven 3 keeps the visibility of build extensions limited to the modules that actually declare them. As a consequence, modules that require build extensions, e.g. to properly resolve dependencies with custom types, and did not declare the extension but relied on some previous module in the reactor to load and share the extension will now fail to build. This is consistent with the build result one would experience when trying to build such a module on its own outside of a reactor build.

**Transport Protocols (Wagons)**

Unlike Maven 2, Maven 3 supports out of the box only http:, https: and file: as transport protocols. To use other transport protocols like scp:, the appropriate wagons have to be explicitly declared in the POM as a build extension. If the wagon in question is only used for deployment to a repository, it can alternatively be declared as a dependency of the Maven Deploy Plugin.

For more information, see Guide to Using Extensions.

**Error Reporting**

Maven now always shows a reactor summary when an error occurs, then follows with an explanation based on the exception that occurred. If your plugin relies on reporting errors that are not included in the MojoFailureException or MojoExecutionException (eg through the logger), these may not be as visible to your users. Sufficient detail (or encouragement to scrollback) should be included in the exception message.

**Plugin Compatibility Matrix**

The Maven 3.x Plugin Compatibility Matrix lists known issues regarding the usage of existing plugins with Maven 3.x. Please consult this table before filling an issue in our issue tracker.