Docs

[User Manual](http://docs.google.com/userguide/userguide.html)

[Guides and Tutorials](https://guides.gradle.org)

[DSL Reference](http://docs.google.com/dsl/)

[Javadoc](http://docs.google.com/javadoc/)

[Release Notes](http://docs.google.com/release-notes.html)

[Forums](https://discuss.gradle.org/)

[Training](https://gradle.org/training/)

[Try Gradle Enterprise](https://gradle.com/enterprise)

[PDF](http://docs.google.com/userguide/userguide.pdf)

* [User Manual Home](http://docs.google.com/userguide/userguide.html)
* [Release Notes](http://docs.google.com/release-notes.html)
* [Installing Gradle](http://docs.google.com/userguide/installation.html)
* [Tutorials](https://guides.gradle.org/)

### Reference

* [Groovy DSL Reference](http://docs.google.com/dsl/)
* [Gradle API Javadoc](http://docs.google.com/javadoc/)
* [Core Plugins](http://docs.google.com/userguide/plugin_reference.html)
* [Gradle & Third-party Tools](http://docs.google.com/userguide/third_party_integration.html)

### Getting Started

* [Creating New Gradle Builds](https://guides.gradle.org/creating-new-gradle-builds/)
* [Creating Build Scans](https://guides.gradle.org/creating-build-scans/)
* [Migrating From Maven](https://guides.gradle.org/migrating-from-maven/)

### Running Gradle Builds

* [Command-Line Interface](http://docs.google.com/userguide/command_line_interface.html)
* [Customizing Execution](#gjdgxs)
  + [Configuring the Build Environment](http://docs.google.com/userguide/build_environment.html)
  + [Configuring the Gradle Daemon](http://docs.google.com/userguide/gradle_daemon.html)
  + [Initialization Scripts](http://docs.google.com/userguide/init_scripts.html)
* [Directory Layout](http://docs.google.com/userguide/directory_layout.html)
* [Executing Multi-Project Builds](http://docs.google.com/userguide/intro_multi_project_builds.html)
* [Gradle Wrapper](http://docs.google.com/userguide/gradle_wrapper.html)
* [Troubleshooting](http://docs.google.com/userguide/troubleshooting.html)
* [Using Build Scans](https://docs.gradle.com/build-scan-plugin)
* [Enabling and Configuring the Build Cache](http://docs.google.com/userguide/build_cache.html)
* [Integrating Separate Gradle Builds (Composite Builds)](http://docs.google.com/userguide/composite_builds.html)

### Authoring Gradle Builds

* [Fundamentals](#30j0zll)
  + [Introducing the Basics of Build Scripts](http://docs.google.com/userguide/tutorial_using_tasks.html)
  + [Working with Tasks](http://docs.google.com/userguide/more_about_tasks.html)
  + [Learning More About Build Scripts](http://docs.google.com/userguide/writing_build_scripts.html)
  + [Working with Files](http://docs.google.com/userguide/working_with_files.html)
  + [Creating Custom Task Types](http://docs.google.com/userguide/custom_tasks.html)
  + [Using Gradle Plugins](http://docs.google.com/userguide/plugins.html)
  + [The Standard Gradle Plugins](http://docs.google.com/userguide/standard_plugins.html)
  + [Understanding the Build Lifecycle](http://docs.google.com/userguide/build_lifecycle.html)
  + [Working with Logging](http://docs.google.com/userguide/logging.html)
  + [Configuring Multi-Project Builds](http://docs.google.com/userguide/multi_project_builds.html)
* [Best Practices](#1fob9te)
  + [Authoring Maintainable Build Scripts](http://docs.google.com/userguide/authoring_maintainable_build_scripts.html)
  + [Organizing Gradle Projects](http://docs.google.com/userguide/organizing_gradle_projects.html)
  + [Optimizing Build Performance](https://guides.gradle.org/performance/)
  + [Using the Build Cache](https://guides.gradle.org/using-build-cache/)
* [Dependency Management](#3znysh7)
  + [Introduction to Dependency Management](http://docs.google.com/userguide/introduction_dependency_management.html)
  + [Dependency Management Terminology](http://docs.google.com/userguide/dependency_management_terminology.html)
  + [Dependency Types](http://docs.google.com/userguide/dependency_types.html)
  + [Repository Types](http://docs.google.com/userguide/repository_types.html)
  + [Declaring Dependencies](http://docs.google.com/userguide/declaring_dependencies.html)
  + [Declaring Repositories](http://docs.google.com/userguide/declaring_repositories.html)
  + [Inspecting Dependencies](http://docs.google.com/userguide/inspecting_dependencies.html)
  + [Managing Dependency Configurations](http://docs.google.com/userguide/managing_dependency_configurations.html)
  + [Managing Transitive Dependencies](http://docs.google.com/userguide/managing_transitive_dependencies.html)
  + [Dependency Locking](http://docs.google.com/userguide/dependency_locking.html)
  + [Troubleshooting Dependency Resolution](http://docs.google.com/userguide/troubleshooting_dependency_resolution.html)
  + [Customizing Dependency Resolution Behavior](http://docs.google.com/userguide/customizing_dependency_resolution_behavior.html)
  + [Dependency Cache Internals](http://docs.google.com/userguide/dependency_cache.html)
  + [Working with Dependencies](http://docs.google.com/userguide/working_with_dependencies.html)
* [Publishing Artifacts](http://docs.google.com/userguide/artifact_management.html)
* [C++ Projects](#2et92p0)
  + [Building Native Software](http://docs.google.com/userguide/native_software.html)
  + [Software Model Concepts](http://docs.google.com/userguide/software_model_concepts.html)
  + [Rule-based Model Configuration](http://docs.google.com/userguide/software_model.html)
  + [Implementing Model Rules in a Plugin](http://docs.google.com/userguide/rule_source.html)
  + [Extending the Software Model](http://docs.google.com/userguide/software_model_extend.html)
* [Java Projects](#tyjcwt)
  + [Building Java & JVM projects](http://docs.google.com/userguide/building_java_projects.html)
  + [Testing Java & JVM projects](http://docs.google.com/userguide/java_testing.html)
* [Advanced Techniques](#3dy6vkm)
  + [Configuring Tasks Lazily](http://docs.google.com/userguide/lazy_configuration.html)
  + [Developing Parallel Tasks](https://guides.gradle.org/using-the-worker-api/)
  + [Testing Your Build with TestKit](http://docs.google.com/userguide/test_kit.html)
  + [Using Ant from Gradle](http://docs.google.com/userguide/ant.html)
* [Sample Gradle builds](#1t3h5sf)
  + [Groovy DSL Samples](https://github.com/gradle/gradle/tree/master/subprojects/docs/src/samples)
  + [Kotlin DSL Samples](https://github.com/gradle/kotlin-dsl/tree/master/samples)

### Extending Gradle

* [Writing Custom Plugins](http://docs.google.com/userguide/custom_plugins.html)
* [Plugin Development Guides](https://gradle.org/guides/?q=Plugin+Development)

[Edit this page](https://github.com/gradle/gradle/edit/master/subprojects/docs/src/docs/userguide/)

# Managing Transitive Dependencies

Contents

[Managing versions of transitive dependencies with dependency constraints](#4d34og8)

[Excluding transitive module dependencies](#2s8eyo1)

[Enforcing a particular dependency version](#17dp8vu)

[Disabling resolution of transitive dependencies](#3rdcrjn)

[Importing version recommendations from a Maven BOM](#26in1rg)

Resolution behavior for transitive dependencies can be customized to a high degree to meet enterprise requirements.

[Managing versions of transitive dependencies with dependency constraints](#4d34og8)

Dependency constraints allow you to define the version or the version range of both dependencies declared in the build script and transitive dependencies. It is the preferred method to express constraints that should be applied to all dependencies of a configuration. When Gradle attempts to resolve a dependency to a module version, all [dependency declarations with version](http://docs.google.com/declaring_dependencies.html#sub:declaring_dependency_with_version), all transitive dependencies and all dependency constraints for that module are taken into consideration. The highest version that matches all conditions is selected. If no such version is found, Gradle fails with an error showing the conflicting declarations. If this happens you can adjust your dependencies or dependency constraints declarations, or [make other adjustments to the transitive dependencies](#lnxbz9) if needed. Similar to dependency declarations, dependency constraint declarations are [scoped by configurations](http://docs.google.com/managing_dependency_configurations.html#managing_dependency_configurations) and can therefore be selectively defined for parts of a build. If a dependency constraint influenced the resolution result, any type of [dependency resolve rules](http://docs.google.com/customizing_dependency_resolution_behavior.html#sec:dependency_resolve_rules) may still be applied afterwards.

[Example: Define dependency constraints](#35nkun2)

**build.gradle**

dependencies {  
 implementation 'org.apache.httpcomponents:httpclient'  
 constraints {  
 implementation('org.apache.httpcomponents:httpclient:4.5.3') {  
 because 'previous versions have a bug impacting this application'  
 }  
 implementation('commons-codec:commons-codec:1.11') {  
 because 'version 1.9 pulled from httpclient has bugs affecting this application'  
 }  
 }  
}

In the example, all versions are omitted from the dependency declaration. Instead, the versions are defined in the constraints block. The version definition for commons-codec:1.11 is only taken into account if commons-codec is brought in as transitive dependency, since commons-codec is not defined as dependency in the project. Otherwise, the constraint has no effect.

| **✨** | Dependency constraints are not yet published, but that will be added in a future release. This means that their use currently only targets builds that do not publish artifacts to maven or ivy repositories. |
| --- | --- |

Dependency constraints themselves can also be added transitively. If a modules’s metadata is defined in a .pom file that contains dependency entries with <optional>true</optional>, Gradle will create a dependency constraint for each of these so-called *optional dependencies*. This leads to a similar resolution behavior as provided by Maven: if the corresponding module is brought in by another, non-optional dependency declaration, then the constraint will apply when choosing the version for that module (e.g., if the optional dependency defines a higher version, that one is chosen).

| **✨** | Support for *optional dependencies* from pom files is active by default with Gradle 5.0+. For using it in Gradle 4.6+, you need to activate it by adding enableFeaturePreview('IMPROVED\_POM\_SUPPORT') in *settings.gradle*. |
| --- | --- |

[Excluding transitive module dependencies](#2s8eyo1)

Declared dependencies in a build script can pull in a lot of transitive dependencies. You might decide that you do not want a particular transitive dependency as part of the dependency graph for a good reason.

* The dependency is undesired due to licensing constraints.
* The dependency is not available in any of the declared repositories.
* The metadata for the dependency exists but the artifact does not.
* The metadata provides incorrect coordinates for a transitive dependency.

Transitive dependencies can be excluded on the level of a declared dependency or a configuration. Let’s demonstrate both use cases. In the following two examples the build script declares a dependency on Log4J, a popular logging framework in the Java world. The metadata of the particular version of Log4J also defines transitive dependencies.

[Example: Unresolved artifacts for transitive dependencies](#1ksv4uv)

**build.gradle**

apply plugin: 'java'  
  
repositories {  
 mavenCentral()  
}  
  
dependencies {  
 implementation 'log4j:log4j:1.2.15'  
}

If resolved from Maven Central some of the transitive dependencies provide metadata but not the corresponding binary artifact. As a result any task requiring the binary files will fail e.g. a compilation task.

> gradle -q compileJava  
  
\* What went wrong:  
Could not resolve all files for configuration ':compileClasspath'.  
> Could not find jms.jar (javax.jms:jms:1.1).  
 Searched in the following locations:  
 https://repo.maven.apache.org/maven2/javax/jms/jms/1.1/jms-1.1.jar  
> Could not find jmxtools.jar (com.sun.jdmk:jmxtools:1.2.1).  
 Searched in the following locations:  
 https://repo.maven.apache.org/maven2/com/sun/jdmk/jmxtools/1.2.1/jmxtools-1.2.1.jar  
> Could not find jmxri.jar (com.sun.jmx:jmxri:1.2.1).  
 Searched in the following locations:  
 https://repo.maven.apache.org/maven2/com/sun/jmx/jmxri/1.2.1/jmxri-1.2.1.jar

The situation can be fixed by adding a repository containing those dependencies. In the given example project, the source code does not actually use any of Log4J’s functionality that require the JMS (e.g. [JMSAppender](https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/net/JMSAppender.html)) or JMX libraries. It’s safe to exclude them from the dependency declaration.

Exclusions need to spelled out as a key/value pair via the attributes group and/or module. For more information, refer to [ModuleDependency.exclude(java.util.Map)](http://docs.google.com/javadoc/org/gradle/api/artifacts/ModuleDependency.html#exclude-java.util.Map-).

[Example: Excluding transitive dependency for a particular dependency declaration](#44sinio)

**build.gradle**

dependencies {  
 implementation('log4j:log4j:1.2.15') {  
 exclude group: 'javax.jms', module: 'jms'  
 exclude group: 'com.sun.jdmk', module: 'jmxtools'  
 exclude group: 'com.sun.jmx', module: 'jmxri'  
 }  
}

You may find that other dependencies will want to pull in the same transitive dependency that misses the artifacts. Alternatively, you can exclude the transitive dependencies for a particular configuration by calling the method [Configuration.exclude(java.util.Map)](http://docs.google.com/dsl/org.gradle.api.artifacts.Configuration.html#org.gradle.api.artifacts.Configuration:exclude(java.util.Map)).

[Example: Excluding transitive dependency for a particular configuration](#2jxsxqh)

**build.gradle**

configurations {  
 implementation {  
 exclude group: 'javax.jms', module: 'jms'  
 exclude group: 'com.sun.jdmk', module: 'jmxtools'  
 exclude group: 'com.sun.jmx', module: 'jmxri'  
 }  
}  
  
dependencies {  
 implementation 'log4j:log4j:1.2.15'  
}

| **✨** | As a build script author you often times know that you want to exclude a dependency for all configurations available in the project. You can use the method [DomainObjectCollection.all(org.gradle.api.Action)](http://docs.google.com/javadoc/org/gradle/api/DomainObjectCollection.html#all-org.gradle.api.Action-) to define a global rule. |
| --- | --- |

You might encounter other use cases that don’t quite fit the bill of an exclude rule. For example you want to automatically select a version for a dependency with a specific requested version or you want to select a different group for a requested dependency to react to a relocation. Those use cases are better solved by the [ResolutionStrategy](http://docs.google.com/dsl/org.gradle.api.artifacts.ResolutionStrategy.html) API. Some of these use cases are covered in [Customizing Dependency Resolution Behavior](http://docs.google.com/customizing_dependency_resolution_behavior.html#customizing_dependency_resolution_behavior).

[Enforcing a particular dependency version](#17dp8vu)

Gradle resolves any dependency version conflicts by selecting the latest version found in the dependency graph. Some projects might need to divert from the default behavior and enforce an earlier version of a dependency e.g. if the source code of the project depends on an older API of a dependency than some of the external libraries.

| **✨** | Enforcing a version of a dependency requires a conscious decision. Changing the version of a transitive dependency might lead to runtime errors if external libraries do not properly function without them. Consider upgrading your source code to use a newer version of the library as an alternative approach. |
| --- | --- |

Let’s say a project uses the [HttpClient library](https://hc.apache.org/httpcomponents-client-ga/) for performing HTTP calls. HttpClient pulls in [Commons Codec](https://commons.apache.org/proper/commons-codec/) as transitive dependency with version 1.10. However, the production source code of the project requires an API from Commons Codec 1.9 which is not available in 1.10 anymore. A dependency version can be enforced by declaring it in the build script and setting [ExternalDependency.setForce(boolean)](http://docs.google.com/javadoc/org/gradle/api/artifacts/ExternalDependency.html#setForce-boolean-) to true.

[Example: Enforcing a dependency version](#z337ya)

**build.gradle**

dependencies {  
 implementation 'org.apache.httpcomponents:httpclient:4.5.4'  
 implementation('commons-codec:commons-codec:1.9') {  
 force = true  
 }  
}

If the project requires a specific version of a dependency on a configuration-level then it can be achieved by calling the method [ResolutionStrategy.force(java.lang.Object[])](http://docs.google.com/dsl/org.gradle.api.artifacts.ResolutionStrategy.html#org.gradle.api.artifacts.ResolutionStrategy:force(java.lang.Object%5B%5D)).

[Example: Enforcing a dependency version on the configuration-level](#3j2qqm3)

**build.gradle**

configurations {  
 compileClasspath {  
 resolutionStrategy.force 'commons-codec:commons-codec:1.9'  
 }  
}  
  
dependencies {  
 implementation 'org.apache.httpcomponents:httpclient:4.5.4'  
}

[Disabling resolution of transitive dependencies](#3rdcrjn)

By default Gradle resolves all transitive dependencies specified by the dependency metadata. Sometimes this behavior may not be desirable e.g. if the metadata is incorrect or defines a large graph of transitive dependencies. You can tell Gradle to disable transitive dependency management for a dependency by setting [ModuleDependency.setTransitive(boolean)](http://docs.google.com/javadoc/org/gradle/api/artifacts/ModuleDependency.html#setTransitive-boolean-) to true. As a result only the main artifact will be resolved for the declared dependency.

[Example: Disabling transitive dependency resolution for a declared dependency](#1y810tw)

**build.gradle**

dependencies {  
 implementation('com.google.guava:guava:23.0') {  
 transitive = false  
 }  
}

| **✨** | Disabling transitive dependency resolution will likely require you to declare the necessary runtime dependencies in your build script which otherwise would have been resolved automatically. Not doing so might lead to runtime classpath issues. |
| --- | --- |

A project can decide to disable transitive dependency resolution completely. You either don’t want to rely on the metadata published to the consumed repositories or you want to gain full control over the dependencies in your graph. For more information, see [Configuration.setTransitive(boolean)](http://docs.google.com/javadoc/org/gradle/api/artifacts/Configuration.html#setTransitive-boolean-).

[Example: Disabling transitive dependency resolution on the configuration-level](#4i7ojhp)

**build.gradle**

configurations.all {  
 transitive = false  
}  
  
dependencies {  
 implementation 'com.google.guava:guava:23.0'  
}

[Importing version recommendations from a Maven BOM](#26in1rg)

Gradle provides support for importing [bill of materials (BOM) files](https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html#Importing_Dependencies), which are effectively .pom files that use <dependencyManagement> to control the dependency versions of direct and transitive dependencies. The BOM support in Gradle works similar to using <scope>import</scope> when depending on a BOM in Maven. In Gradle however, it is done via a regular dependency declaration on the BOM:

[Example: Depending on a BOM to import its dependency constraints](#2xcytpi)

**build.gradle**

dependencies {  
 // import a BOM  
 implementation 'org.springframework.boot:spring-boot-dependencies:1.5.8.RELEASE'  
  
 // define dependencies without versions  
 implementation 'com.google.code.gson:gson'  
 implementation 'dom4j:dom4j'  
}

In the example, the versions of gson and dom4j are provided by the Spring Boot BOM. This way, if you are developing for a platform like Spring Boot, you do not have to declare any versions yourself but can rely on the versions the platform provides.

Gradle treats all entries in the <dependencyManagement> block of a BOM similar to [Gradle’s dependency constraints](#4d34og8). This means that any version defined in the <dependencyManagement> block can impact the dependency resolution result. In order to qualify as a BOM, a .pom file needs to have <packaging>pom</packaging> set.

| **✨** | Importing dependency constraints from Maven BOMs is active by default with Gradle 5.0+. For using it in Gradle 4.6+, you need to activate it by adding enableFeaturePreview('IMPROVED\_POM\_SUPPORT') in *settings.gradle*. |
| --- | --- |

Docs

* [User Manual](http://docs.google.com/userguide/userguide.html)
* [DSL Reference](http://docs.google.com/dsl/)
* [Release Notes](http://docs.google.com/release-notes.html)
* [Javadoc](http://docs.google.com/javadoc/)

News

* [Blog](https://blog.gradle.org/)
* [Newsletter](https://newsletter.gradle.com/)
* [Twitter](https://twitter.com/gradle)

Products

* [Build Scans](https://gradle.com/build-scans)
* [Build Cache](https://gradle.com/build-cache)
* [Enterprise Docs](https://gradle.com/enterprise/resources)

Get Help

* [Forums](https://discuss.gradle.org/c/help-discuss)
* [GitHub](https://github.com/gradle/)
* [Training](https://gradle.org/training/)
* [Services](https://gradle.org/services/)

Subscribe for important Gradle updates and news

Subscribe

By entering your email, you agree to our [Terms](https://gradle.org/terms/) and [Privacy Policy](https://gradle.org/privacy/), including receipt of emails. You can unsubscribe at any time.

© [Gradle Inc.](https://gradle.com) 2018 All rights reserved.

[Careers](https://gradle.com/careers) | [Privacy](https://gradle.org/privacy) | [Terms of Service](https://gradle.org/terms) | [Contact](https://gradle.org/contact/)