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/)

# Declaring Dependencies

Contents

[Declaring a dependency to a module](#4d34og8)

[Declaring a file dependency](#2s8eyo1)

[Declaring a project dependency](#17dp8vu)

[Resolving specific artifacts from a module dependency](#3rdcrjn)

Gradle builds can declare dependencies on modules hosted in repositories, files and other Gradle projects. You can find examples for common scenarios in this section. For more information, see the [full reference on all types of dependencies](http://docs.google.com/dependency_types.html#dependency_types).

Every dependency needs to be assigned to a configuration when declared in a build script. For more information on the purpose and syntax of configurations, see [Managing Dependency Configurations](http://docs.google.com/managing_dependency_configurations.html#managing_dependency_configurations).

[Declaring a dependency to a module](#4d34og8)

Modern software projects rarely build code in isolation. Projects reference modules for the purpose of reusing existing and proven functionality. Upon resolution, selected versions of modules are downloaded from dedicated repositories and stored in the [dependency cache](http://docs.google.com/dependency_cache.html#dependency_cache) to avoid unnecessary network traffic.



*Figure 1. Resolving dependencies from remote repositories*

[Declaring a concrete version of a dependency](#26in1rg)

A typical example for such a library in a Java project is the [Spring framework](https://projects.spring.io/spring-framework/). The following code snippet declares a compile-time dependency on the Spring web module by its coordinates: org.springframework:spring-web:5.0.2.RELEASE. Gradle resolves the module including its transitive dependencies from the [Maven Central repository](https://search.maven.org/) and uses it to compile Java source code. The version attribute of the dependency coordinates points to a *concrete version* indicating that the underlying artifacts do not change over time. The use of concrete versions ensure reproducibility for the aspect of dependency resolution.

[Example: Declaring a dependency with a concrete version](#lnxbz9)

**build.gradle**

apply plugin: 'java-library'  
  
repositories {  
 mavenCentral()  
}  
  
dependencies {  
 implementation 'org.springframework:spring-web:5.0.2.RELEASE'  
}

A Gradle project can define other types of repositories hosting modules. You can learn more about the syntax and API in the section on [declaring repositories](http://docs.google.com/declaring_repositories.html#declaring_repositories). Refer to [the chapter on the Java Plugin](http://docs.google.com/java_plugin.html#java_plugin) for a deep dive on declaring dependencies for a Java project. The resolution behavior for dependencies is [highly customizable](http://docs.google.com/customizing_dependency_resolution_behavior.html#customizing_dependency_resolution_behavior).

[Declaring a dependency without version](#35nkun2)

A recommended practice for larger projects is to declare dependencies without versions and use [dependency constraints](http://docs.google.com/managing_transitive_dependencies.html#sec:dependency_constraints) for version declaration. The advantage is that dependency constrains allow you to manage versions of all dependencies, including transitive ones, in one place.

[Example: Declaring a dependency without version](#1ksv4uv)

**build.gradle**

dependencies {  
 implementation 'org.springframework:spring-web'  
}  
  
dependencies {  
 constraints {  
 implementation 'org.springframework:spring-web:5.0.2.RELEASE'  
 }  
}

[Declaring a dynamic version](#44sinio)

Projects might adopt a more aggressive approach for consuming dependencies to modules. For example you might want to always integrate the latest version of a dependency to consume cutting edge features at any given time. A *dynamic version* allows for resolving the latest version or the latest version of a version range for a given module.

| **✨** | Using dynamic versions in a build bears the risk of potentially breaking it. As soon as a new version of the dependency is released that contains an incompatible API change your source code might stop compiling. |
| --- | --- |

[Example: Declaring a dependency with a dynamic version](#2jxsxqh)

**build.gradle**

apply plugin: 'java-library'  
  
repositories {  
 mavenCentral()  
}  
  
dependencies {  
 implementation 'org.springframework:spring-web:5.+'  
}

A [build scan](https://scans.gradle.com/) can effectively visualize dynamic dependency versions and their respective, selected versions.



*Figure 2. Dynamic dependencies in build scan*

By default, Gradle caches dynamic versions of dependencies for 24 hours. Within this time frame, Gradle does not try to resolve newer versions from the declared repositories. The [threshold can be configured](http://docs.google.com/troubleshooting_dependency_resolution.html#sub:dynamic_versions_and_changing_modules) as needed for example if you want to resolve new versions earlier.

[Declaring a changing version](#z337ya)

A team might decide to implement a series of features before releasing a new version of the application or library. A common strategy to allow consumers to integrate an unfinished version of their artifacts early and often is to release a module with a so-called *changing version*. A changing version indicates that the feature set is still under active development and hasn’t released a stable version for general availability yet.

In Maven repositories, changing versions are commonly referred to as [snapshot versions](https://maven.apache.org/guides/getting-started/index.html#What_is_a_SNAPSHOT_version). Snapshot versions contain the suffix -SNAPSHOT. The following example demonstrates how to declare a snapshot version on the Spring dependency.

[Example: Declaring a dependency with a changing version](#3j2qqm3)

**build.gradle**

apply plugin: 'java-library'  
  
repositories {  
 mavenCentral()  
 maven {  
 url 'https://repo.spring.io/snapshot/'  
 }  
}  
  
dependencies {  
 implementation 'org.springframework:spring-web:5.0.3.BUILD-SNAPSHOT'  
}

By default, Gradle caches changing versions of dependencies for 24 hours. Within this time frame, Gradle does not try to resolve newer versions from the declared repositories. The [threshold can be configured](http://docs.google.com/troubleshooting_dependency_resolution.html#sub:dynamic_versions_and_changing_modules) as needed for example if you want to resolve new snapshot versions earlier.

Gradle is flexible enough to treat any version as changing version e.g. if you wanted to model snapshot behavior for an Ivy module. All you need to do is to set the property [ExternalModuleDependency.setChanging(boolean)](http://docs.google.com/javadoc/org/gradle/api/artifacts/ExternalModuleDependency.html#setChanging-boolean-) to true.

[Declaring a file dependency](#2s8eyo1)

Projects sometimes do not rely on a binary repository product e.g. JFrog Artifactory or Sonatype Nexus for hosting and resolving external dependencies. It’s common practice to host those dependencies on a shared drive or check them into version control alongside the project source code. Those dependencies are referred to as *file dependencies*, the reason being that they represent a file without any [metadata](http://docs.google.com/dependency_management_terminology.html#sub:terminology_module_metadata) (like information about transitive dependencies, the origin or its author) attached to them.



*Figure 3. Resolving file dependencies from the local file system and a shared drive*

The following example resolves file dependencies from the directories ant, libs and tools.

[Example: Declaring multiple file dependencies](#1y810tw)

**build.gradle**

configurations {  
 antContrib  
 externalLibs  
 deploymentTools  
}  
  
dependencies {  
 antContrib files('ant/antcontrib.jar')  
 externalLibs files('libs/commons-lang.jar', 'libs/log4j.jar')  
 deploymentTools fileTree(dir: 'tools', include: '\*.exe')  
}

As you can see in the code example, every dependency has to define its exact location in the file system. The most prominent methods for creating a file reference are [Project.files(java.lang.Object…​)](http://docs.google.com/dsl/org.gradle.api.Project.html#org.gradle.api.Project:files(java.lang.Object%5B%5D)), [ProjectLayout.files(java.lang.Object…​)](http://docs.google.com/javadoc/org/gradle/api/file/ProjectLayout.html#files-java.lang.Object...-), [ProjectLayout.configurableFiles(java.lang.Object…​)](http://docs.google.com/javadoc/org/gradle/api/file/ProjectLayout.html#configurableFiles-java.lang.Object...-), and [Project.fileTree(java.lang.Object)](http://docs.google.com/dsl/org.gradle.api.Project.html#org.gradle.api.Project:fileTree(java.lang.Object)) Alternatively, you can also define the source directory of one or many file dependencies in the form of a [flat directory repository](http://docs.google.com/repository_types.html#sec:flat_dir_resolver).

[Declaring a project dependency](#17dp8vu)

Software projects often break up software components into modules to improve maintainability and prevent strong coupling. Modules can define dependencies between each other to reuse code within the same project.

Gradle can model dependencies between modules. Those dependencies are called *project dependencies* because each module is represented by a Gradle project. At runtime, the build automatically ensures that project dependencies are built in the correct order and added to the classpath for compilation. The chapter [Authoring Multi-Project Builds](http://docs.google.com/multi_project_builds.html#multi_project_builds) discusses how to set up and configure multi-project builds in more detail.



*Figure 4. Dependencies between projects*

The following example declares the dependencies on the utils and api project from the web-service project. The method [Project.project(java.lang.String)](http://docs.google.com/dsl/org.gradle.api.Project.html#org.gradle.api.Project:project(java.lang.String)) creates a reference to a specific subproject by path.

[Example: Declaring project dependencies](#4i7ojhp)

**build.gradle**

project(':web-service') {  
 dependencies {  
 implementation project(':utils')  
 implementation project(':api')  
 }  
}

[Resolving specific artifacts from a module dependency](#3rdcrjn)

Whenever Gradle tries to resolve a module from a Maven or Ivy repository, it looks for a metadata file and the default artifact file, a JAR. The build fails if none of these artifact files can be resolved. Under certain conditions, you might want to tweak the way Gradle resolves artifacts for a dependency.

* The dependency only provides a non-standard artifact without any metadata e.g. a ZIP file.
* The module metadata declares more than one artifact e.g. as part of an Ivy dependency descriptor.
* You only want to download a specific artifact without any of the transitive dependencies declared in the metadata.

Gradle is a polyglot build tool and not limited to just resolving Java libraries. Let’s assume you wanted to build a web application using JavaScript as the client technology. Most projects check in external JavaScript libraries into version control. An external JavaScript library is no different than a reusable Java library so why not download it from a repository instead?

[Google Hosted Libraries](https://developers.google.com/speed/libraries/) is a distribution platform for popular, open-source JavaScript libraries. With the help of the artifact-only notation you can download a JavaScript library file e.g. JQuery. The @ character separates the dependency’s coordinates from the artifact’s file extension.

[Example: Resolving a JavaScript artifact for a declared dependency](#2xcytpi)

**build.gradle**

repositories {  
 ivy {  
 url 'https://ajax.googleapis.com/ajax/libs'  
 layout 'pattern', {  
 artifact '[organization]/[revision]/[module].[ext]'  
 }  
 }  
}  
  
configurations {  
 js  
}  
  
dependencies {  
 js 'jquery:jquery:3.2.1@js'  
}

Some modules ship different "flavors" of the same artifact or they publish multiple artifacts that belong to a specific module version but have a different purpose. It’s common for a Java library to publish the artifact with the compiled class files, another one with just the source code in it and a third one containing the Javadocs.

In JavaScript, a library may exist as uncompressed or minified artifact. In Gradle, a specific artifact identifier is called *classifier*, a term generally used in Maven and Ivy dependency management.

Let’s say we wanted to download the minified artifact of the JQuery library instead of the uncompressed file. You can provide the classifier min as part of the dependency declaration.

[Example: Resolving a JavaScript artifact with classifier for a declared dependency](#1ci93xb)

**build.gradle**

repositories {  
 ivy {  
 url 'https://ajax.googleapis.com/ajax/libs'  
 layout 'pattern', {  
 artifact '[organization]/[revision]/[module](.[classifier]).[ext]'  
 }  
 }  
}  
  
configurations {  
 js  
}  
  
dependencies {  
 js 'jquery:jquery:3.2.1:min@js'  
}

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/)