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/)

# Build Lifecycle

Contents

[Build phases](#4d34og8)

[Settings file](#2s8eyo1)

[Multi-project builds](#17dp8vu)

[Initialization](#3rdcrjn)

[Configuration and execution of a single project build](#26in1rg)

[Responding to the lifecycle in the build script](#lnxbz9)

We said earlier that the core of Gradle is a language for dependency based programming. In Gradle terms this means that you can define tasks and dependencies between tasks. Gradle guarantees that these tasks are executed in the order of their dependencies, and that each task is executed only once. These tasks form a [Directed Acyclic Graph](http://en.wikipedia.org/wiki/Directed_acyclic_graph). There are build tools that build up such a dependency graph as they execute their tasks. Gradle builds the complete dependency graph *before* any task is executed. This lies at the heart of Gradle and makes many things possible which would not be possible otherwise.

Your build scripts configure this dependency graph. Therefore they are strictly speaking *build configuration scripts*.

[Build phases](#4d34og8)

A Gradle build has three distinct phases.

Initialization

Gradle supports single and multi-project builds. During the initialization phase, Gradle determines which projects are going to take part in the build, and creates a [Project](http://docs.google.com/dsl/org.gradle.api.Project.html) instance for each of these projects.

Configuration

During this phase the project objects are configured. The build scripts of *all* projects which are part of the build are executed.

Execution

Gradle determines the subset of the tasks, created and configured during the configuration phase, to be executed. The subset is determined by the task name arguments passed to the gradle command and the current directory. Gradle then executes each of the selected tasks.

[Settings file](#2s8eyo1)

Beside the build script files, Gradle defines a settings file. The settings file is determined by Gradle via a naming convention. The default name for this file is settings.gradle. Later in this chapter we explain how Gradle looks for a settings file.

The settings file is executed during the initialization phase. A multi-project build must have a settings.gradle file in the root project of the multi-project hierarchy. It is required because the settings file defines which projects are taking part in the multi-project build (see [Authoring Multi-Project Builds](http://docs.google.com/multi_project_builds.html#multi_project_builds)). For a single-project build, a settings file is optional. Besides defining the included projects, you might need it to add libraries to your build script classpath (see [Organizing Gradle Projects](http://docs.google.com/organizing_gradle_projects.html#organizing_gradle_projects)). Let’s first do some introspection with a single project build:

[Example: Single project build](#35nkun2)

**settings.gradle**

println 'This is executed during the initialization phase.'

**build.gradle**

println 'This is executed during the configuration phase.'  
  
task configured {  
 println 'This is also executed during the configuration phase.'  
}  
  
task test {  
 doLast {  
 println 'This is executed during the execution phase.'  
 }  
}  
  
task testBoth {  
 doFirst {  
 println 'This is executed first during the execution phase.'  
 }  
 doLast {  
 println 'This is executed last during the execution phase.'  
 }  
 println 'This is executed during the configuration phase as well.'  
}

**Output of gradle test testBoth**

> gradle test testBoth  
This is executed during the initialization phase.  
  
> Configure project :  
This is executed during the configuration phase.  
This is also executed during the configuration phase.  
This is executed during the configuration phase as well.  
  
> Task :test  
This is executed during the execution phase.  
  
> Task :testBoth  
This is executed first during the execution phase.  
This is executed last during the execution phase.  
  
BUILD SUCCESSFUL in 0s  
2 actionable tasks: 2 executed

For a build script, the property access and method calls are delegated to a project object. Similarly property access and method calls within the settings file is delegated to a settings object. Look at the [Settings](http://docs.google.com/dsl/org.gradle.api.initialization.Settings.html) class in the API documentation for more information.

[Multi-project builds](#17dp8vu)

A multi-project build is a build where you build more than one project during a single execution of Gradle. You have to declare the projects taking part in the multi-project build in the settings file. There is much more to say about multi-project builds in the chapter dedicated to this topic (see [Authoring Multi-Project Builds](http://docs.google.com/multi_project_builds.html#multi_project_builds)).

[Project locations](#1ksv4uv)

Multi-project builds are always represented by a tree with a single root. Each element in the tree represents a project. A project has a path which denotes the position of the project in the multi-project build tree. In most cases the project path is consistent with the physical location of the project in the file system. However, this behavior is configurable. The project tree is created in the settings.gradle file. By default it is assumed that the location of the settings file is also the location of the root project. But you can redefine the location of the root project in the settings file.

[Building the tree](#44sinio)

In the settings file you can use a set of methods to build the project tree. Hierarchical and flat physical layouts get special support.

[Hierarchical layouts](#2jxsxqh)

[Example: Hierarchical layout](#z337ya)

**settings.gradle**

include 'project1', 'project2:child', 'project3:child1'

The include method takes project paths as arguments. The project path is assumed to be equal to the relative physical file system path. For example, a path 'services:api' is mapped by default to a folder 'services/api' (relative from the project root). You only need to specify the leaves of the tree. This means that the inclusion of the path 'services:hotels:api' will result in creating 3 projects: 'services', 'services:hotels' and 'services:hotels:api'. More examples of how to work with the project path can be found in the DSL documentation of [Settings.include(java.lang.String[])](http://docs.google.com/dsl/org.gradle.api.initialization.Settings.html#org.gradle.api.initialization.Settings:include(java.lang.String%5B%5D)).

[Flat layouts](#3j2qqm3)

[Example: Flat layout](#1y810tw)

**settings.gradle**

includeFlat 'project3', 'project4'

The includeFlat method takes directory names as an argument. These directories need to exist as siblings of the root project directory. The location of these directories are considered as child projects of the root project in the multi-project tree.

[Modifying elements of the project tree](#4i7ojhp)

The multi-project tree created in the settings file is made up of so called *project descriptors*. You can modify these descriptors in the settings file at any time. To access a descriptor you can do:

[Example: Lookup of elements of the project tree](#2xcytpi)

**settings.gradle**

println rootProject.name  
println project(':projectA').name

Using this descriptor you can change the name, project directory and build file of a project.

[Example: Modification of elements of the project tree](#1ci93xb)

**settings.gradle**

rootProject.name = 'main'  
project(':projectA').projectDir = new File(settingsDir, '../my-project-a')  
project(':projectA').buildFileName = 'projectA.gradle'

Look at the [ProjectDescriptor](http://docs.google.com/javadoc/org/gradle/api/initialization/ProjectDescriptor.html) class in the API documentation for more information.

[Initialization](#3rdcrjn)

How does Gradle know whether to do a single or multi-project build? If you trigger a multi-project build from a directory with a settings file, things are easy. But Gradle also allows you to execute the build from within any subproject taking part in the build.[[1](#3whwml4)] If you execute Gradle from within a project with no settings.gradle file, Gradle looks for a settings.gradle file in the following way:

* It looks in a directory called master which has the same nesting level as the current dir.
* If not found yet, it searches parent directories.
* If not found yet, the build is executed as a single project build.
* If a settings.gradle file is found, Gradle checks if the current project is part of the multi-project hierarchy defined in the found settings.gradle file. If not, the build is executed as a single project build. Otherwise a multi-project build is executed.

What is the purpose of this behavior? Gradle needs to determine whether the project you are in is a subproject of a multi-project build or not. Of course, if it is a subproject, only the subproject and its dependent projects are built, but Gradle needs to create the build configuration for the whole multi-project build (see [Authoring Multi-Project Builds](http://docs.google.com/multi_project_builds.html#multi_project_builds)). You can use the -u command line option to tell Gradle not to look in the parent hierarchy for a settings.gradle file. The current project is then always built as a single project build. If the current project contains a settings.gradle file, the -u option has no meaning. Such a build is always executed as:

* a single project build, if the settings.gradle file does not define a multi-project hierarchy
* a multi-project build, if the settings.gradle file does define a multi-project hierarchy.

The automatic search for a settings.gradle file only works for multi-project builds with a physical hierarchical or flat layout. For a flat layout you must additionally follow the naming convention described above (“master”). Gradle supports arbitrary physical layouts for a multi-project build, but for such arbitrary layouts you need to execute the build from the directory where the settings file is located. For information on how to run partial builds from the root, see [Running tasks by their absolute path](http://docs.google.com/multi_project_builds.html#sec:running_partial_build_from_the_root).

Gradle creates a Project object for every project taking part in the build. For a multi-project build these are the projects specified in the Settings object (plus the root project). Each project object has by default a name equal to the name of its top level directory, and every project except the root project has a parent project. Any project may have child projects.

[Configuration and execution of a single project build](#26in1rg)

For a single project build, the workflow of the *after initialization* phases are pretty simple. The build script is executed against the project object that was created during the initialization phase. Then Gradle looks for tasks with names equal to those passed as command line arguments. If these task names exist, they are executed as a separate build in the order you have passed them. The configuration and execution for multi-project builds is discussed in [Authoring Multi-Project Builds](http://docs.google.com/multi_project_builds.html#multi_project_builds).

[Responding to the lifecycle in the build script](#lnxbz9)

Your build script can receive notifications as the build progresses through its lifecycle. These notifications generally take two forms: You can either implement a particular listener interface, or you can provide a closure to execute when the notification is fired. The examples below use closures. For details on how to use the listener interfaces, refer to the API documentation.

[Project evaluation](#2bn6wsx)

You can receive a notification immediately before and after a project is evaluated. This can be used to do things like performing additional configuration once all the definitions in a build script have been applied, or for some custom logging or profiling.

Below is an example which adds a test task to each project which has a hasTests property value of true.

[Example: Adding of test task to each project which has certain property set](#qsh70q)

**build.gradle**

allprojects {  
 afterEvaluate { project ->  
 if (project.hasTests) {  
 println "Adding test task to $project"  
 project.task('test') {  
 doLast {  
 println "Running tests for $project"  
 }  
 }  
 }  
 }  
}

**projectA.gradle**

hasTests = true

**Output of gradle -q test**

> gradle -q test  
Adding test task to project ':projectA'  
Running tests for project ':projectA'

This example uses method Project.afterEvaluate() to add a closure which is executed after the project is evaluated.

It is also possible to receive notifications when any project is evaluated. This example performs some custom logging of project evaluation. Notice that the afterProject notification is received regardless of whether the project evaluates successfully or fails with an exception.

[Example: Notifications](#3as4poj)

**build.gradle**

gradle.afterProject {project, projectState ->  
 if (projectState.failure) {  
 println "Evaluation of $project FAILED"  
 } else {  
 println "Evaluation of $project succeeded"  
 }  
}

**Output of** gradle -q test

> gradle -q test  
Evaluation of root project 'buildProjectEvaluateEvents' succeeded  
Evaluation of project ':projectA' succeeded  
Evaluation of project ':projectB' FAILED  
  
FAILURE: Build failed with an exception.  
  
\* Where:  
Build file '/home/user/gradle/samples/projectB.gradle' line: 1  
  
\* What went wrong:  
A problem occurred evaluating project ':projectB'.  
> broken  
  
\* Try:  
Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get more log output. Run with --scan to get full insights.  
  
\* Get more help at https://help.gradle.org  
  
BUILD FAILED in 0s

You can also add a [ProjectEvaluationListener](http://docs.google.com/javadoc/org/gradle/api/ProjectEvaluationListener.html) to the [Gradle](http://docs.google.com/dsl/org.gradle.api.invocation.Gradle.html) to receive these events.

[Task creation](#1pxezwc)

You can receive a notification immediately after a task is added to a project. This can be used to set some default values or add behaviour before the task is made available in the build file.

The following example sets the srcDir property of each task as it is created.

[Example: Setting of certain property to all tasks](#49x2ik5)

**build.gradle**

tasks.whenTaskAdded { task ->  
 task.ext.srcDir = 'src/main/java'  
}  
  
task a  
  
println "source dir is $a.srcDir"

**Output of** gradle -q a

> gradle -q a  
source dir is src/main/java

You can also add an [Action](http://docs.google.com/javadoc/org/gradle/api/Action.html) to a [TaskContainer](http://docs.google.com/javadoc/org/gradle/api/tasks/TaskContainer.html) to receive these events.

[Task execution graph ready](#2p2csry)

You can receive a notification immediately after the task execution graph has been populated (See [Configure by DAG](http://docs.google.com/tutorial_using_tasks.html#configure-by-dag)).

You can also add a [TaskExecutionGraphListener](http://docs.google.com/javadoc/org/gradle/api/execution/TaskExecutionGraphListener.html) to the [TaskExecutionGraph](http://docs.google.com/javadoc/org/gradle/api/execution/TaskExecutionGraph.html) to receive these events.

[Task execution](#147n2zr)

You can receive a notification immediately before and after any task is executed.

The following example logs the start and end of each task execution. Notice that the afterTask notification is received regardless of whether the task completes successfully or fails with an exception.

[Example: Logging of start and end of each task execution](#3o7alnk)

**build.gradle**

task ok  
  
task broken(dependsOn: ok) {  
 doLast {  
 throw new RuntimeException('broken')  
 }  
}  
  
gradle.taskGraph.beforeTask { Task task ->  
 println "executing $task ..."  
}  
  
gradle.taskGraph.afterTask { Task task, TaskState state ->  
 if (state.failure) {  
 println "FAILED"  
 }  
 else {  
 println "done"  
 }  
}

**Output of** gradle -q broken

> gradle -q broken  
executing task ':ok' ...  
done  
executing task ':broken' ...  
FAILED  
  
FAILURE: Build failed with an exception.  
  
\* Where:  
Build file '/home/user/gradle/samples/build.gradle' line: 5  
  
\* What went wrong:  
Execution failed for task ':broken'.  
> broken  
  
\* Try:  
Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get more log output. Run with --scan to get full insights.  
  
\* Get more help at https://help.gradle.org  
  
BUILD FAILED in 0s

You can also use a [TaskExecutionListener](http://docs.google.com/javadoc/org/gradle/api/execution/TaskExecutionListener.html) to the [TaskExecutionGraph](http://docs.google.com/javadoc/org/gradle/api/execution/TaskExecutionGraph.html) to receive these events.

[1](#23ckvvd). Gradle supports partial multi-project builds (see [Authoring Multi-Project Builds](http://docs.google.com/multi_project_builds.html#multi_project_builds)).

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/)