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/)

# Composite builds

Contents

[What is a composite build?](#4d34og8)

[Defining a composite build](#2s8eyo1)

[Interacting with a composite build](#17dp8vu)

[Declaring the dependencies substituted by an included build](#3rdcrjn)

[Depending on tasks in an included build](#26in1rg)

[Current limitations and future plans for composite builds](#lnxbz9)

| **✨** | Composite build is an [incubating](http://docs.google.com/feature_lifecycle.html#feature_lifecycle) feature. While useful for many use cases, there are bugs to be discovered, rough edges to smooth, and enhancements we plan to make. Thanks for trying it out! |
| --- | --- |

[What is a composite build?](#4d34og8)

A composite build is simply a build that includes other builds. In many ways a composite build is similar to a Gradle multi-project build, except that instead of including single projects, complete builds are included.

Composite builds allow you to:

* combine builds that are usually developed independently, for instance when trying out a bug fix in a library that your application uses
* decompose a large multi-project build into smaller, more isolated chunks that can be worked in independently or together as needed

A build that is included in a composite build is referred to, naturally enough, as an "included build". Included builds do not share any configuration with the composite build, or the other included builds. Each included build is configured and executed in isolation.

Included builds interact with other builds via dependency substitution. If any build in the composite has a dependency that can be satisfied by the included build, then that dependency will be replaced by a project dependency on the included build.

By default, Gradle will attempt to determine the dependencies that can be substituted by an included build. However for more flexibility, it is possible to explicitly declare these substitutions if the default ones determined by Gradle are not correct for the composite. See [Declaring substitutions](#3rdcrjn).

As well as consuming outputs via project dependencies, a composite build can directly declare task dependencies on included builds. Included builds are isolated, and are not able to declare task dependencies on the composite build or on other included builds. See [Depending on tasks in an included build](#26in1rg).

[Defining a composite build](#2s8eyo1)

The following examples demonstrate the various ways that 2 Gradle builds that are normally developed separately can be combined into a composite build. For these examples, the my-utils multi-project build produces 2 different java libraries (number-utils and string-utils), and the my-app build produces an executable using functions from those libraries.

The my-app build does not have direct dependencies on my-utils. Instead, it declares binary dependencies on the libraries produced by my-utils.

[Example: Dependencies of my-app](#35nkun2)

**my-app/build.gradle**

apply plugin: 'java'  
apply plugin: 'application'  
apply plugin: 'idea'  
  
group "org.sample"  
version "1.0"  
  
mainClassName = "org.sample.myapp.Main"  
  
dependencies {  
 compile "org.sample:number-utils:1.0"  
 compile "org.sample:string-utils:1.0"  
}  
  
repositories {  
 jcenter()  
}

| **✨** | The code for this example can be found at samples/compositeBuilds/basic in the ‘-all’ distribution of Gradle. |
| --- | --- |

[Defining a composite build via --include-build](#1ksv4uv)

The --include-build command-line argument turns the executed build into a composite, substituting dependencies from the included build into the executed build.

[Example: Declaring a command-line composite](#44sinio)

**Output of** gradle --include-build ../my-utils run

> gradle --include-build ../my-utils run  
> Task :processResources NO-SOURCE  
> Task :my-utils:string-utils:compileJava  
> Task :my-utils:string-utils:processResources NO-SOURCE  
> Task :my-utils:string-utils:classes  
> Task :my-utils:string-utils:jar  
> Task :my-utils:number-utils:compileJava  
> Task :my-utils:number-utils:processResources NO-SOURCE  
> Task :my-utils:number-utils:classes  
> Task :my-utils:number-utils:jar  
> Task :compileJava  
> Task :classes  
  
> Task :run  
The answer is 42  
  
  
BUILD SUCCESSFUL in 0s  
2 actionable tasks: 2 executed

[Defining a composite build via settings.gradle](#2jxsxqh)

It’s possible to make the above arrangement persistent, by using [Settings.includeBuild(java.lang.Object)](http://docs.google.com/dsl/org.gradle.api.initialization.Settings.html#org.gradle.api.initialization.Settings:includeBuild(java.lang.Object)) to declare the included build in the settings.gradle file. The settings.gradle file can be used to add subprojects and included builds at the same time. Included builds are added by location. See the examples below for more details.

[Defining a separate composite build](#z337ya)

One downside of the above approach is that it requires you to modify an existing build, rendering it less useful as a standalone build. One way to avoid this is to define a separate composite build, whose only purpose is to combine otherwise separate builds.

[Example: Declaring a separate composite](#3j2qqm3)

**settings.gradle**

rootProject.name='adhoc'  
  
includeBuild '../my-app'  
includeBuild '../my-utils'

In this scenario, the 'main' build that is executed is the composite, and it doesn’t define any useful tasks to execute itself. In order to execute the 'run' task in the 'my-app' build, the composite build must define a delegating task.

[Example: Depending on task from included build](#1y810tw)

**build.gradle**

task run {  
 dependsOn gradle.includedBuild('my-app').task(':run')  
}

More details tasks that depend on included build tasks below.

[Restrictions on included builds](#4i7ojhp)

Most builds can be included into a composite, including other composite builds. However there are some limitations.

Every included build:

* must not have a rootProject.name the same as another included build.
* must not have a rootProject.name the same as a top-level project of the composite build.
* must not have a rootProject.name the same as the composite build rootProject.name.

[Interacting with a composite build](#17dp8vu)

In general, interacting with a composite build is much the same as a regular multi-project build. Tasks can be executed, tests can be run, and builds can be imported into the IDE.

[Executing tasks](#2xcytpi)

Tasks from the composite build can be executed from the command line, or from you IDE. Executing a task will result in direct task dependencies being executed, as well as those tasks required to build dependency artifacts from included builds.

| **✨** | There is not (yet) any means to directly execute a task from an included build via the command line. Included build tasks are automatically executed in order to generate required dependency artifacts, or the [including build can declare a dependency on a task from an included build](#26in1rg). |
| --- | --- |

[Importing into the IDE](#1ci93xb)

One of the most useful features of composite builds is IDE integration. By applying the [idea](http://docs.google.com/idea_plugin.html#idea_plugin) or [eclipse](http://docs.google.com/eclipse_plugin.html#eclipse_plugin) plugin to your build, it is possible to generate a single IDEA or Eclipse project that permits all builds in the composite to be developed together.

In addition to these Gradle plugins, recent versions of [IntelliJ IDEA](https://www.jetbrains.com/idea/) and [Eclipse Buildship](https://projects.eclipse.org/projects/tools.buildship) support direct import of a composite build.

Importing a composite build permits sources from separate Gradle builds to be easily developed together. For every included build, each sub-project is included as an IDEA Module or Eclipse Project. Source dependencies are configured, providing cross-build navigation and refactoring.

[Declaring the dependencies substituted by an included build](#3rdcrjn)

By default, Gradle will configure each included build in order to determine the dependencies it can provide. The algorithm for doing this is very simple: Gradle will inspect the group and name for the projects in the included build, and substitute project dependencies for any external dependency matching ${project.group}:${project.name}.

There are cases when the default substitutions determined by Gradle are not sufficient, or they are not correct for a particular composite. For these cases it is possible to explicitly declare the substitutions for an included build. Take for example a single-project build 'unpublished', that produces a java utility library but does not declare a value for the group attribute:

[Example: Build that does not declare group attribute](#3whwml4)

**build.gradle**

apply plugin: 'java'

When this build is included in a composite, it will attempt to substitute for the dependency module "undefined:unpublished" ("undefined" being the default value for project.group, and 'unpublished' being the root project name). Clearly this isn’t going to be very useful in a composite build. To use the unpublished library unmodified in a composite build, the composing build can explicitly declare the substitutions that it provides:

[Example: Declaring the substitutions for an included build](#2bn6wsx)

**settings.gradle**

rootProject.name = 'app'  
  
includeBuild('../anonymous-library') {  
 dependencySubstitution {  
 substitute module('org.sample:number-utils') with project(':')  
 }  
}

With this configuration, the "my-app" composite build will substitute any dependency on org.sample:number-utils with a dependency on the root project of "unpublished".

[Cases where included build substitutions must be declared](#qsh70q)

Many builds that use the uploadArchives task to publish artifacts will function automatically as an included build, without declared substitutions. Here are some common cases where declared substitutions are required:

* When the archivesBaseName property is used to set the name of the published artifact.
* When a configuration other than default is published: this usually means a task other than uploadArchives is used.
* When the MavenPom.addFilter() is used to publish artifacts that don’t match the project name.
* When the maven-publish or ivy-publish plugins are used for publishing, and the publication coordinates don’t match ${project.group}:${project.name}.

[Cases where composite build substitutions won’t work](#3as4poj)

Some builds won’t function correctly when included in a composite, even when dependency substitutions are explicitly declared. This limitation is due to the fact that a project dependency that is substituted will always point to the default configuration of the target project. Any time that the artifacts and dependencies specified for the default configuration of a project don’t match what is actually published to a repository, then the composite build may exhibit different behaviour.

Here are some cases where the publish module metadata may be different from the project default configuration:

* When a configuration other than default is published.
* When the maven-publish or ivy-publish plugins are used.
* When the POM or ivy.xml file is tweaked as part of publication.

Builds using these features function incorrectly when included in a composite build. We plan to improve this in the future.

[Depending on tasks in an included build](#26in1rg)

While included builds are isolated from one another and cannot declare direct dependencies, a composite build is able to declare task dependencies on its included builds. The included builds are accessed using [Gradle.getIncludedBuilds()](http://docs.google.com/dsl/org.gradle.api.invocation.Gradle.html#org.gradle.api.invocation.Gradle:includedBuilds) or [Gradle.includedBuild(java.lang.String)](http://docs.google.com/dsl/org.gradle.api.invocation.Gradle.html#org.gradle.api.invocation.Gradle:includedBuild(java.lang.String)), and a task reference is obtained via the [IncludedBuild.task(java.lang.String)](http://docs.google.com/dsl/org.gradle.api.initialization.IncludedBuild.html#org.gradle.api.initialization.IncludedBuild:task(java.lang.String)) method.

Using these APIs, it is possible to declare a dependency on a task in a particular included build, or tasks with a certain path in all or some of the included builds.

[Example: Depending on a single task from an included build](#1pxezwc)

**build.gradle**

task run {  
 dependsOn gradle.includedBuild('my-app').task(':run')  
}

[Example: Depending on a tasks with path in all included builds](#49x2ik5)

**build.gradle**

task publishDeps {  
 dependsOn gradle.includedBuilds\*.task(':uploadArchives')  
}

[Current limitations and future plans for composite builds](#lnxbz9)

We think composite builds are pretty useful already. However, there are some things that don’t yet work the way we’d like, and other improvements that we think will make things work even better.

Limitations of the current implementation include:

* No support for included builds that have publications that don’t mirror the project default configuration. See [Cases where composite builds won’t work](#3as4poj).
* Native builds are not supported. (Binary dependencies are not yet supported for native builds).
* Substituting plugins only works with the buildscript block but not with the plugins block.

Improvements we have planned for upcoming releases include:

* Better detection of dependency substitution, for build that publish with custom coordinates, builds that produce multiple components, etc. This will reduce the cases where dependency substitution needs to be explicitly declared for an included build.
* The ability to target a task or tasks in an included build directly from the command line. We are currently exploring syntax options for allowing this functionality, which will remove many cases where a delegating task is required in the composite.
* Make the plugins {} block consider included builds when locating plugins and their dependencies.
* Making the implicit buildSrc project an included build.

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/)