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/)

# Extending the software model

Contents

[Introduction](#4d34og8)

[Concepts](#2s8eyo1)

[Components](#17dp8vu)

[Binaries](#3rdcrjn)

[Source sets](#26in1rg)

[Putting it all together](#lnxbz9)

[About internal views](#35nkun2)

| **⚠** | Rule based configuration [will be deprecated](https://blog.gradle.org/state-and-future-of-the-gradle-software-model). New plugins should not use this concept. |
| --- | --- |

[Introduction](#4d34og8)

One of the strengths of Gradle has always been its extensibility, and its adaptability to new domains. The software model takes this extensibility to a new level, enabling the deep modeling of specific domains via richly typed DSLs. The following chapter describes how the model and the corresponding DSLs can be extended to support domains like the [Play Framework](http://docs.google.com/play_plugin.html#play_plugin) or [native software development](http://docs.google.com/native_software.html#native_software). Before reading this you should be familiar with the Gradle software model [rule based configuration](http://docs.google.com/software_model.html#software_model) and [concepts](http://docs.google.com/software_model_concepts.html#software_model_concepts).

The following build script is an example of using a custom software model for building Markdown based documentation:

[Example: an example of using a custom software model](#1ksv4uv)

**build.gradle**

import sample.documentation.DocumentationComponent  
import sample.documentation.TextSourceSet  
import sample.markdown.MarkdownSourceSet  
  
apply plugin:sample.documentation.DocumentationPlugin  
apply plugin:sample.markdown.MarkdownPlugin  
  
model {  
 components {  
 docs(DocumentationComponent) {  
 sources {  
 reference(TextSourceSet)  
 userguide(MarkdownSourceSet) {  
 generateIndex = true  
 smartQuotes = true  
 }  
 }  
 }  
 }  
}

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

The rest of this chapter is dedicated to explaining what is going on behind this build script.

[Concepts](#2s8eyo1)

A custom software model type has a public type, a base interface and internal views. Multiple such types then collaborate to define a custom software model.

[Public type and base interfaces](#44sinio)

Extended types declare a *public type* that extends a *base interface*:

* Components extend the [ComponentSpec](http://docs.google.com/javadoc/org/gradle/platform/base/ComponentSpec.html) base interface
* Binaries extend the [BinarySpec](http://docs.google.com/javadoc/org/gradle/platform/base/BinarySpec.html) base interface
* Source sets extend the [LanguageSourceSet](http://docs.google.com/javadoc/org/gradle/language/base/LanguageSourceSet.html) base interface

The *public type* is exposed to build logic.

[Internal views](#2jxsxqh)

Adding internal views to your model type, you can make some data visible to build logic via a public type, while hiding the rest of the data behind the internal view types. This is covered in a [dedicated section](#35nkun2) below.

[Components all the way down](#z337ya)

Components are composed of other components. A source set is just a special kind of component representing sources. It might be that the sources are provided, or generated. Similarly, some components are composed of different binaries, which are built by tasks. All buildable components are built by tasks. In the software model, you will write rules to generate both binaries from components and tasks from binaries.

[Components](#17dp8vu)

To declare a custom component type one must extend [ComponentSpec](http://docs.google.com/javadoc/org/gradle/platform/base/ComponentSpec.html), or one of the following, depending on the use case:

* [SourceComponentSpec](http://docs.google.com/javadoc/org/gradle/platform/base/SourceComponentSpec.html) represents a component which has sources
* [VariantComponentSpec](http://docs.google.com/javadoc/org/gradle/platform/base/VariantComponentSpec.html) represents a component which generates different binaries based on context (target platforms, build flavors, …​). Such a component generally produces multiple binaries.
* [GeneralComponentSpec](http://docs.google.com/javadoc/org/gradle/platform/base/GeneralComponentSpec.html) is a convenient base interface for components that are built from sources and variant-aware. This is the typical case for a lot of software components, and therefore it should be in most of the cases the base type to be extended.

The core software model includes more types that can be used as base for extension. For example: [LibrarySpec](http://docs.google.com/javadoc/org/gradle/platform/base/LibrarySpec.html) and [ApplicationSpec](http://docs.google.com/javadoc/org/gradle/platform/base/ApplicationSpec.html) can also be extended in this manner. Theses are no-op extensions of GeneralComponentSpec used to describe a software model better by distinguishing libraries and applications components. [TestSuiteSpec](http://docs.google.com/javadoc/org/gradle/testing/base/TestSuiteSpec.html) should be used for all components that describe a test suite.

[Example: Declare a custom component](#3j2qqm3)

**DocumentationComponent.groovy**

@Managed  
interface DocumentationComponent extends GeneralComponentSpec {}

Types extending ComponentSpec are registered via a rule annotated with [ComponentType](http://docs.google.com/javadoc/org/gradle/platform/base/ComponentType.html):

[Example: Register a custom component](#1y810tw)

**DocumentationPlugin.groovy**

class DocumentationPlugin extends RuleSource {  
 @ComponentType  
 void registerComponent(TypeBuilder<DocumentationComponent> builder) {}  
}

[Binaries](#3rdcrjn)

To declare a custom binary type one must extend [BinarySpec](http://docs.google.com/javadoc/org/gradle/platform/base/BinarySpec.html).

[Example: Declare a custom binary](#4i7ojhp)

**DocumentationBinary.groovy**

@Managed  
interface DocumentationBinary extends BinarySpec {  
 File getOutputDir()  
 void setOutputDir(File outputDir)  
}

Types extending BinarySpec are registered via a rule annotated with [ComponentType](http://docs.google.com/javadoc/org/gradle/platform/base/ComponentType.html):

[Example: Register a custom binary](#2xcytpi)

**DocumentationPlugin.groovy**

class DocumentationPlugin extends RuleSource {  
 @ComponentType  
 void registerBinary(TypeBuilder<DocumentationBinary> builder) {}  
}

[Source sets](#26in1rg)

To declare a custom source set type one must extend [LanguageSourceSet](http://docs.google.com/javadoc/org/gradle/language/base/LanguageSourceSet.html).

[Example: Declare a custom source set](#1ci93xb)

**MarkdownSourceSet.groovy**

@Managed  
interface MarkdownSourceSet extends LanguageSourceSet {  
 boolean isGenerateIndex()  
 void setGenerateIndex(boolean generateIndex)  
  
 boolean isSmartQuotes()  
 void setSmartQuotes(boolean smartQuotes)  
}

Types extending LanguageSourceSet are registered via a rule annotated with [ComponentType](http://docs.google.com/javadoc/org/gradle/platform/base/ComponentType.html):

[Example: Register a custom source set](#3whwml4)

**MarkdownPlugin.groovy**

class MarkdownPlugin extends RuleSource {  
 @ComponentType  
 void registerMarkdownLanguage(TypeBuilder<MarkdownSourceSet> builder) {}  
}

Setting the *language name* is mandatory.

[Putting it all together](#lnxbz9)

[Generating binaries from components](#2bn6wsx)

Binaries generation from components is done via rules annotated with [ComponentBinaries](http://docs.google.com/javadoc/org/gradle/platform/base/ComponentBinaries.html). This rule generates a DocumentationBinary named exploded for each DocumentationComponent and sets its outputDir property:

[Example: Generates documentation binaries](#qsh70q)

**DocumentationPlugin.groovy**

class DocumentationPlugin extends RuleSource {  
 @ComponentBinaries  
 void generateDocBinaries(ModelMap<DocumentationBinary> binaries, VariantComponentSpec component, @Path("buildDir") File buildDir) {  
 binaries.create("exploded") { binary ->  
 outputDir = new File(buildDir, "${component.name}/${binary.name}")  
 }  
 }  
}

[Generating tasks from binaries](#3as4poj)

Tasks generation from binaries is done via rules annotated with [BinaryTasks](http://docs.google.com/javadoc/org/gradle/platform/base/BinaryTasks.html). This rule generates a Copy task for each TextSourceSet of each DocumentationBinary:

[Example: Generates tasks for text source sets](#1pxezwc)

**DocumentationPlugin.groovy**

class DocumentationPlugin extends RuleSource {  
 @BinaryTasks  
 void generateTextTasks(ModelMap<Task> tasks, final DocumentationBinary binary) {  
 binary.inputs.withType(TextSourceSet) { textSourceSet ->  
 def taskName = binary.tasks.taskName("compile", textSourceSet.name)  
 def outputDir = new File(binary.outputDir, textSourceSet.name)  
 tasks.create(taskName, Copy) {  
 from textSourceSet.source  
 destinationDir = outputDir  
 }  
 }  
 }  
}

This rule generates a MarkdownCompileTask task for each MarkdownSourceSet of each DocumentationBinary:

[Example: Register a custom source set](#49x2ik5)

**MarkdownPlugin.groovy**

class MarkdownPlugin extends RuleSource {  
 @BinaryTasks  
 void processMarkdownDocumentation(ModelMap<Task> tasks, final DocumentationBinary binary) {  
 binary.inputs.withType(MarkdownSourceSet) { markdownSourceSet ->  
 def taskName = binary.tasks.taskName("compile", markdownSourceSet.name)  
 def outputDir = new File(binary.outputDir, markdownSourceSet.name)  
 tasks.create(taskName, MarkdownHtmlCompile) { compileTask ->  
 compileTask.source = markdownSourceSet.source  
 compileTask.destinationDir = outputDir  
 compileTask.smartQuotes = markdownSourceSet.smartQuotes  
 compileTask.generateIndex = markdownSourceSet.generateIndex  
 }  
 }  
 }  
}

See the sample source for more on the MarkdownCompileTask task.

[Using your custom model](#2p2csry)

This build script demonstrate usage of the custom model defined in the sections above:

[Example: an example of using a custom software model](#147n2zr)

**build.gradle**

import sample.documentation.DocumentationComponent  
import sample.documentation.TextSourceSet  
import sample.markdown.MarkdownSourceSet  
  
apply plugin:sample.documentation.DocumentationPlugin  
apply plugin:sample.markdown.MarkdownPlugin  
  
model {  
 components {  
 docs(DocumentationComponent) {  
 sources {  
 reference(TextSourceSet)  
 userguide(MarkdownSourceSet) {  
 generateIndex = true  
 smartQuotes = true  
 }  
 }  
 }  
 }  
}

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

And in the components reports for such a build script we can see our model types properly registered:

[Example: components report](#3o7alnk)

**Output of** gradle -q components

> gradle -q components  
  
------------------------------------------------------------  
Root project  
------------------------------------------------------------  
  
DocumentationComponent 'docs'  
-----------------------------  
  
Source sets  
 Markdown source 'docs:userguide'  
 srcDir: src/docs/userguide  
 Text source 'docs:reference'  
 srcDir: src/docs/reference  
  
Binaries  
 DocumentationBinary 'docs:exploded'  
 build using task: :docsExploded  
  
Note: currently not all plugins register their components, so some components may not be visible here.

[About internal views](#35nkun2)

Internal views can be added to an already registered type or to a new custom type. In other words, using internal views, you can attach extra properties to already registered components, binaries and source sets types like JvmLibrarySpec, JarBinarySpec or JavaSourceSet and to the custom types you write.

Let’s start with a simple component public type and its internal view declarations:

[Example: public type and internal view declaration](#23ckvvd)

**build.gradle**

@Managed interface MyComponent extends ComponentSpec {  
 String getPublicData()  
 void setPublicData(String data)  
}  
@Managed interface MyComponentInternal extends MyComponent {  
 String getInternalData()  
 void setInternalData(String internal)  
}

The type registration is as follows:

[Example: type registration](#ihv636)

**build.gradle**

class MyPlugin extends RuleSource {  
 @ComponentType  
 void registerMyComponent(TypeBuilder<MyComponent> builder) {  
 builder.internalView(MyComponentInternal)  
 }  
}

The internalView(type) method of the type builder can be called several times. This is how you would add several internal views to a type.

Now, let’s mutate both public and internal data using some rule:

[Example: public and internal data mutation](#32hioqz)

**build.gradle**

class MyPlugin extends RuleSource {  
 @Mutate  
 void mutateMyComponents(ModelMap<MyComponentInternal> components) {  
 components.all { component ->  
 component.publicData = "Some PUBLIC data"  
 component.internalData = "Some INTERNAL data"  
 }  
 }  
}

Our internalData property should not be exposed to build logic. Let’s check this using the model task on the following build file:

[Example: Build script and model report output](#1hmsyys)

**build.gradle**

apply plugin: MyPlugin  
model {  
 components {  
 my(MyComponent)  
 }  
}

**Output of gradle -q model**

> gradle -q model  
  
------------------------------------------------------------  
Root project  
------------------------------------------------------------  
  
+ components  
 | Type: org.gradle.platform.base.ComponentSpecContainer  
 | Creator: ComponentBasePlugin.PluginRules#components(ComponentSpecContainer)  
 | Rules:  
 ⤷ components { ... } @ build.gradle line 53, column 5  
 ⤷ MyPlugin#mutateMyComponents(ModelMap<MyComponentInternal>)  
 + my  
 | Type: MyComponent  
 | Creator: components { ... } @ build.gradle line 53, column 5 > create(my)  
 | Rules:  
 ⤷ MyPlugin#mutateMyComponents(ModelMap<MyComponentInternal>) > all()  
 + publicData  
 | Type: java.lang.String  
 | Value: Some PUBLIC data  
 | Creator: components { ... } @ build.gradle line 53, column 5 > create(my)  
+ tasks  
 | Type: org.gradle.model.ModelMap<org.gradle.api.Task>  
 | Creator: Project.<init>.tasks()  
 + assemble  
 | Type: org.gradle.api.DefaultTask  
 | Value: task ':assemble'  
 | Creator: Project.<init>.tasks.assemble()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + build  
 | Type: org.gradle.api.DefaultTask  
 | Value: task ':build'  
 | Creator: Project.<init>.tasks.build()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + buildEnvironment  
 | Type: org.gradle.api.tasks.diagnostics.BuildEnvironmentReportTask  
 | Value: task ':buildEnvironment'  
 | Creator: Project.<init>.tasks.buildEnvironment()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + check  
 | Type: org.gradle.api.DefaultTask  
 | Value: task ':check'  
 | Creator: Project.<init>.tasks.check()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + clean  
 | Type: org.gradle.api.tasks.Delete  
 | Value: task ':clean'  
 | Creator: Project.<init>.tasks.clean()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + components  
 | Type: org.gradle.api.reporting.components.ComponentReport  
 | Value: task ':components'  
 | Creator: Project.<init>.tasks.components()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + dependencies  
 | Type: org.gradle.api.tasks.diagnostics.DependencyReportTask  
 | Value: task ':dependencies'  
 | Creator: Project.<init>.tasks.dependencies()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + dependencyInsight  
 | Type: org.gradle.api.tasks.diagnostics.DependencyInsightReportTask  
 | Value: task ':dependencyInsight'  
 | Creator: Project.<init>.tasks.dependencyInsight()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + dependentComponents  
 | Type: org.gradle.api.reporting.dependents.DependentComponentsReport  
 | Value: task ':dependentComponents'  
 | Creator: Project.<init>.tasks.dependentComponents()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + help  
 | Type: org.gradle.configuration.Help  
 | Value: task ':help'  
 | Creator: Project.<init>.tasks.help()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + init  
 | Type: org.gradle.buildinit.tasks.InitBuild  
 | Value: task ':init'  
 | Creator: Project.<init>.tasks.init()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + model  
 | Type: org.gradle.api.reporting.model.ModelReport  
 | Value: task ':model'  
 | Creator: Project.<init>.tasks.model()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + projects  
 | Type: org.gradle.api.tasks.diagnostics.ProjectReportTask  
 | Value: task ':projects'  
 | Creator: Project.<init>.tasks.projects()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + properties  
 | Type: org.gradle.api.tasks.diagnostics.PropertyReportTask  
 | Value: task ':properties'  
 | Creator: Project.<init>.tasks.properties()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + tasks  
 | Type: org.gradle.api.tasks.diagnostics.TaskReportTask  
 | Value: task ':tasks'  
 | Creator: Project.<init>.tasks.tasks()  
 | Rules:  
 ⤷ copyToTaskContainer  
 + wrapper  
 | Type: org.gradle.api.tasks.wrapper.Wrapper  
 | Value: task ':wrapper'  
 | Creator: Project.<init>.tasks.wrapper()  
 | Rules:  
 ⤷ copyToTaskContainer

We can see in this report that publicData is present and that internalData is not.

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/)