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/)

# Authoring Maintainable Build Scripts

Contents

[Avoiding imperative logic in scripts](#4d34og8)

[Avoiding Gradle internal APIs](#2s8eyo1)

[Declaring tasks in a build script](#17dp8vu)

[Improving task discoverability](#3rdcrjn)

[Minimize logic executed during the configuration phase](#26in1rg)

[Avoiding the use of GradleBuild](#lnxbz9)

[Avoiding inter-project configuration](#35nkun2)

[Avoiding passwords in plain text](#1ksv4uv)

Gradle build scripts combine the qualities of declarative build logic, expressiveness as well as flexibility and rigidity as needed. As a build script author it is easy to fall into the trap of striking the wrong balance or applying poor coding habits. This chapter describes best practices for writing your build script in a meaningful, yet flexible and efficient way.

| **✨** | The third-party [Gradle lint plugin](https://github.com/nebula-plugins/gradle-lint-plugin) helps with enforcing a desired code style in a build script if you are looking for appropriate linting automation. |
| --- | --- |

[Avoiding imperative logic in scripts](#4d34og8)

The Gradle runtime does not enforce a specific style for build logic. For that very reason, it’s easy to end up with a build script that mixes declarative DSL elements with imperative, procedural code. Let’s talk about some concrete examples.

* *Declarative code:* Built-in, language-agnostic DSL elements (e.g. [Project.dependencies{}](http://docs.google.com/dsl/org.gradle.api.Project.html#org.gradle.api.Project:dependencies(groovy.lang.Closure)) or [Project.repositories{}](http://docs.google.com/dsl/org.gradle.api.Project.html#org.gradle.api.Project:repositories(groovy.lang.Closure))) or DSLs exposed by plugins
* *Imperative code:* Conditional logic or very complex task action implementations

The end goal of every build script should be to only contain declarative language elements which makes the code easier to understand and maintain. Imperative logic should live in binary plugins and which in turn is applied to the build script. As a side product, you automatically enable your team to [reuse the plugin logic in other projects](https://guides.gradle.org/designing-gradle-plugins/#reusable_logic_should_be_written_as_binary_plugin) if you publish the artifact to a binary repository.

The following sample build shows a negative example of using conditional logic directly in the build script. While this code snippet is small, it is easy to imagine a full-blown build script using numerous procedural statements and the impact it would have on readability and maintainability. By moving the code into a class [testability](https://guides.gradle.org/testing-gradle-plugins/) also becomes a valid option.

[Example: A build script using conditional logic to create a task](#44sinio)

**build.gradle**

if (project.findProperty('releaseEngineer')) {  
 task release {  
 doLast {  
 logger.quiet 'Releasing to production...'  
  
 // release the artifact to production  
 }  
 }  
}

Let’s compare the build script with the same logic implemented as a binary plugin. The code might look more involved at first but clearly looks more like typical application code. This particular plugin class lives in the [buildSrc directory](http://docs.google.com/organizing_gradle_projects.html#sec:build_sources) which makes it available to the build script automatically.

[Example: A binary plugin implementing imperative logic](#2jxsxqh)

**ReleasePlugin.java**

package com.enterprise;  
  
import org.gradle.api.Action;  
import org.gradle.api.Plugin;  
import org.gradle.api.Project;  
import org.gradle.api.Task;  
  
public class ReleasePlugin implements Plugin<Project> {  
 private static final String RELEASE\_ENG\_ROLE\_PROP = "releaseEngineer";  
 private static final String RELEASE\_TASK\_NAME = "release";  
  
 @Override  
 public void apply(Project project) {  
 if (project.findProperty(RELEASE\_ENG\_ROLE\_PROP) != null) {  
 Task task = project.getTasks().create(RELEASE\_TASK\_NAME);  
  
 task.doLast(new Action<Task>() {  
 @Override  
 public void execute(Task task) {  
 task.getLogger().quiet("Releasing to production...");  
  
 // release the artifact to production  
 }  
 });  
 }  
 }  
}

Now that the build logic has been translated into a plugin, you can apply it in the build script. The build script has been shrunk from 8 lines of code to a one liner.

[Example: A build script applying a plugin that encapsulates imperative logic](#z337ya)

**build.gradle**

apply plugin: 'com.enterprise.release'

[Avoiding Gradle internal APIs](#2s8eyo1)

Use of Gradle internal APIs in plugins and build scripts has the potential to break builds when either Gradle or plugins change.

The following packages are listed in the [Gradle public API definition](https://github.com/gradle/gradle/blob/180b9d3fa84b91768364c603380e82947437eda1/buildSrc/subprojects/configuration/src/main/kotlin/org/gradle/gradlebuild/public-api.kt), with the exception of any subpackage with internal in the name:

org/gradle/\*  
org/gradle/api/\*\*  
org/gradle/authentication/\*\*  
org/gradle/buildinit/\*\*  
org/gradle/caching/\*\*  
org/gradle/concurrent/\*\*  
org/gradle/deployment/\*\*  
org/gradle/external/javadoc/\*\*  
org/gradle/ide/\*\*  
org/gradle/includedbuild/\*\*  
org/gradle/ivy/\*\*  
org/gradle/jvm/\*\*  
org/gradle/language/\*\*  
org/gradle/maven/\*\*  
org/gradle/nativeplatform/\*\*  
org/gradle/normalization/\*\*  
org/gradle/platform/\*\*  
org/gradle/play/\*\*  
org/gradle/plugin/devel/\*\*  
org/gradle/plugin/repository/\*  
org/gradle/plugin/use/\*  
org/gradle/plugin/management/\*  
org/gradle/plugins/\*\*  
org/gradle/process/\*\*  
org/gradle/testfixtures/\*\*  
org/gradle/testing/jacoco/\*\*  
org/gradle/tooling/\*\*  
org/gradle/swiftpm/\*\*  
org/gradle/model/\*\*  
org/gradle/testkit/\*\*  
org/gradle/testing/\*\*  
org/gradle/vcs/\*\*  
org/gradle/workers/\*\*

[Alternatives for oft-used internal APIs](#3j2qqm3)

To provide a nested DSL for your custom task, don’t use org.gradle.internal.reflect.Instantiator; use [ObjectFactory](http://docs.google.com/javadoc/org/gradle/api/model/ObjectFactory.html) instead. It may also be helpful to read [the chapter on lazy configuration](http://docs.google.com/lazy_configuration.html#lazy_configuration).

Don’t use org.gradle.api.internal.ConventionMapping. Use [Provider](http://docs.google.com/javadoc/org/gradle/api/provider/Provider.html) and/or [Property](http://docs.google.com/javadoc/org/gradle/api/provider/Property.html). You can find an example for capturing user input to configure runtime behavior in the [implementing plugins guide](https://guides.gradle.org/implementing-gradle-plugins/#capturing_user_input_to_configure_plugin_runtime_behavior).

Instead of org.gradle.internal.os.OperatingSystem, use another method to detect operating system, such as [Apache commons-lang SystemUtils](https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/SystemUtils.html) or System.getProperty("os.name").

Use other collections or I/O frameworks instead of org.gradle.util.CollectionUtils, org.gradle.util.GFileUtils, and other classes under org.gradle.util.\*.

Gradle plugin authors may find the Designing Gradle Plugins subsection on [restricting the plugin implementation to Gradle’s public API](https://guides.gradle.org/designing-gradle-plugins/#restricting_the_plugin_implementation_to_gradle_s_public_api) helpful.

[Declaring tasks in a build script](#17dp8vu)

The task API gives a build author a lot of flexibility to declare tasks in a build script. For optimal readability and maintainability follow these rules:

* The task type should be the only key-value pair that belongs within the parentheses after the task name.
* Any other configuration logic for a task should be defined as part of [Project.task(java.lang.String, groovy.lang.Closure)](http://docs.google.com/dsl/org.gradle.api.Project.html#org.gradle.api.Project:task(java.lang.String,%20groovy.lang.Closure)) if possible.
* [Task actions](http://docs.google.com/tutorial_using_tasks.html#sec:hello_world) should only be declared with the methods [Task.doFirst{}](http://docs.google.com/dsl/org.gradle.api.Task.html#org.gradle.api.Task:doFirst(org.gradle.api.Action)) or [Task.doLast{}](http://docs.google.com/dsl/org.gradle.api.Task.html#org.gradle.api.Task:doLast(org.gradle.api.Action)).
* [Task.doLast{}](http://docs.google.com/dsl/org.gradle.api.Task.html#org.gradle.api.Task:doLast(org.gradle.api.Action)) should be used if the task only defines a single action.
* A task should [define a group and description](#3rdcrjn).

[Example: Definition of tasks following best practices](#1y810tw)

**build.gradle**

import com.enterprise.DocsGenerate  
  
task generateHtmlDocs(type: DocsGenerate) {  
 group = JavaBasePlugin.DOCUMENTATION\_GROUP  
 description = 'Generates the HTML documentation for this project.'  
 title = 'Project docs'  
 outputDir = file("${buildDir}/docs")  
}  
  
task allDocs {  
 group = JavaBasePlugin.DOCUMENTATION\_GROUP  
 description = 'Generates all documentation for this project.'  
 dependsOn generateHtmlDocs  
  
 doLast {  
 logger.quiet('Generating all documentation...')  
 }  
}

[Improving task discoverability](#3rdcrjn)

Even new users to a build should to be able to find crucial information quickly and effortlessly. In Gradle you can declare a [group](http://docs.google.com/dsl/org.gradle.api.Project.html#org.gradle.api.Project:group) and a [description](http://docs.google.com/dsl/org.gradle.api.Project.html#org.gradle.api.Project:description) for any task of the build. The [tasks report](http://docs.google.com/command_line_interface.html#sec:listing_tasks) uses the assigned values to organize and render the task for easy discoverability. Assigning a group and description is most helpful for any task that you expect build users to invoke.

The example task generateDocs generates documentation for a project in the form of HTML pages. The task should be organized underneath the bucket Documentation. The description should express its intent.

[Example: A task declaring the group and description](#4i7ojhp)

**build.gradle**

task generateDocs {  
 group = 'Documentation'  
 description = 'Generates the HTML documentation for this project.'  
  
 doLast {  
 // action implementation  
 }  
}

The output of the tasks report reflects the assigned values.

> gradle tasks  
  
> Task :tasks  
  
Documentation tasks  
-------------------  
generateDocs - Generates the HTML documentation for this project.

[Minimize logic executed during the configuration phase](#26in1rg)

It’s important for every build script developer to understand the different phases of the [build lifecycle](http://docs.google.com/build_lifecycle.html#build_lifecycle) and their implications on performance and evaluation order of build logic. During the configuration phase the project and its domain objects should be *configured*, whereas the execution phase only executes the actions of the task(s) requested on the command line plus their dependencies. Be aware that any code that is not part of a task action will be executed with *every single run* of the build. A [build scan](https://scans.gradle.com/get-started) can help you with identifying the time spent during each of the lifecycle phases. It’s an invaluable tool for diagnosing common performance issues.

Let’s consider the following incantation of the anti-pattern described above. In the build script you can see that the dependencies assigned to the configuration printArtifactNames are resolved outside of the task action.

[Example: Executing logic during configuration should be avoided](#2xcytpi)

**build.gradle**

dependencies {  
 implementation 'log4j:log4j:1.2.17'  
}  
  
task printArtifactNames {  
 // always executed  
 def libraryNames = configurations.compileClasspath.collect { it.name }  
  
 doLast {  
 logger.quiet libraryNames  
 }  
}

The code for resolving the dependencies should be moved into the task action to avoid the performance impact of resolving the dependencies before they are actually needed.

[Example: Executing logic during execution phase is preferred](#1ci93xb)

**build.gradle**

dependencies {  
 implementation 'log4j:log4j:1.2.17'  
}  
  
task printArtifactNames {  
 doLast {  
 def libraryNames = configurations.compileClasspath.collect { it.name }  
 logger.quiet libraryNames  
 }  
}

[Avoiding the use of GradleBuild](#lnxbz9)

The [GradleBuild](http://docs.google.com/dsl/org.gradle.api.tasks.GradleBuild.html) task type allows a build script to define a task that invokes another Gradle build. The use of this type is generally discouraged. There are some corner cases where the invoked build doesn’t expose the same runtime behavior as from the command line or through the Tooling API leading to unexpected results.

Usually, there’s a better way to model the requirement. The appropriate approach depends on the problem at hand. Here’re some options:

* Model the build as [multi-project build](http://docs.google.com/multi_project_builds.html#multi_project_builds) if the intention is to execute tasks from different modules as unified build.
* Use [composite builds](http://docs.google.com/composite_builds.html#composite_builds) for projects that are physically separated but should occasionally be built as a single unit.

[Avoiding inter-project configuration](#35nkun2)

Gradle does not restrict build script authors from reaching into the domain model from one project into another one in a [multi-project build](http://docs.google.com/multi_project_builds.html#multi_project_builds). Strongly-coupled projects hurts [build execution performance](http://docs.google.com/multi_project_builds.html#sec:parallel_execution) as well as readability and maintainability of code.

The following practices should be avoided:

* Explicitly depending on a task from another project via [Task.dependsOn(java.lang.Object...)](http://docs.google.com/dsl/org.gradle.api.Task.html#org.gradle.api.Task:dependsOn(java.lang.Object%5B%5D)).
* Setting property values or calling methods on domain objects from another project.
* Executing another portion of the build with [GradleBuild](#lnxbz9).
* Declaring unnecessary [project dependencies](http://docs.google.com/declaring_dependencies.html#sec:declaring_project_dependency).

[Avoiding passwords in plain text](#1ksv4uv)

Most builds need to consume one or many passwords. The reasons for this need may vary. Some builds need a password for publishing artifacts to a secured binary repository, other builds need a password for downloading binary files. Passwords should always kept safe to prevent fraud. Under no circumstance should you add the password to the build script as property in plain text or declare it in a gradle.properties. Those files usually live in a version control repository and can be viewed by anyone that has access to it.

Passwords should be stored in encrypted fashion. At the moment Gradle does not provide a built-in mechanism for encrypting, storing and accessing passwords. A good solution for solving this problem is the [Gradle Credentials plugin](https://github.com/etiennestuder/gradle-credentials-plugin).

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/)