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/)

# Organizing Gradle Projects

Contents

[Separate language-specific source files](#4d34og8)

[Separate source files per test type](#2s8eyo1)

[Use standard conventions as much as possible](#17dp8vu)

[Always define a settings file](#3rdcrjn)

[Use buildSrc to abstract imperative logic](#26in1rg)

[Declare properties in gradle.properties file](#lnxbz9)

[Avoid overlapping task outputs](#35nkun2)

[Standardizing builds with a custom Gradle distribution](#1ksv4uv)

Source code and build logic of every software project should be organized in a meaningful way. This page lays out the best practices that lead to readable, maintainable projects. The following sections also touch on common problems and how to avoid them.

[Separate language-specific source files](#4d34og8)

Gradle’s language plugins establish conventions for discovering and compiling source code. For example, a project applying the [Java plugin](http://docs.google.com/java_plugin.html#java_plugin) will automatically compile the code in the directory src/main/java. Other language plugins follow the same pattern. The last portion of the directory path usually indicates the expected language of the source files.

Some compilers are capable of cross-compiling multiple languages in the same source directory. The Groovy compiler can handle the scenario of mixing Java and Groovy source files located in src/main/groovy. Gradle recommends that you place sources in directories according to their language, because builds are more performant and both the user and build can make stronger assumptions.

The following source tree contains Java and Kotlin source files. Java source files live in src/main/java, whereas Kotlin source files live in src/main/kotlin.

.  
├── build.gradle  
├── settings.gradle  
└── src  
 └── main  
 ├── java  
 │   └── HelloWorld.java  
 └── kotlin  
 └── Utils.kt

[Separate source files per test type](#2s8eyo1)

It’s very common that a project defines and executes different types of tests e.g. unit tests, integration tests, functional tests or smoke tests. Optimally, the test source code for each test type should be stored in dedicated source directories. Separated test source code has a positive impact on maintainability and separation of concerns as you can run test types independent from each other.

The following source tree demonstrates how to separate unit from integration tests in a Java-based project.

.  
├── build.gradle  
├── gradle  
│   └── integration-test.gradle  
├── settings.gradle  
└── src  
 ├── integTest  
 │   └── java  
 │   └── DefaultFileReaderIntegrationTest.java  
 ├── main  
 │   └── java  
 │   ├── DefaultFileReader.java  
 │   ├── FileReader.java  
 │   └── StringUtils.java  
 └── test  
 └── java  
 └── StringUtilsTest.java

Gradle models source code directories with the help of the [source set concept](http://docs.google.com/building_java_projects.html#sec:java_source_sets). By pointing an instance of a source set to one or many source code directories, Gradle will automatically create a corresponding compilation task out-of-the-box.

[Example: Integration test source set](#44sinio)

**gradle/integration-test.gradle**

sourceSets {  
 integTest {  
 java.srcDir file('src/integTest/java')  
 resources.srcDir file('src/integTest/resources')  
 compileClasspath += sourceSets.main.output + configurations.testRuntimeClasspath  
 runtimeClasspath += output + compileClasspath  
 }  
}

Source sets are only responsible for compiling source code, but do not deal with executing the byte code. For the purpose of test execution, a corresponding task of type [Test](http://docs.google.com/dsl/org.gradle.api.tasks.testing.Test.html) needs to be established.

[Example: Integration test task](#2jxsxqh)

**gradle/integration-test.gradle**

task integTest(type: Test) {  
 description = 'Runs the integration tests.'  
 group = 'verification'  
 testClassesDirs = sourceSets.integTest.output.classesDirs  
 classpath = sourceSets.integTest.runtimeClasspath  
 mustRunAfter test  
}  
  
check.dependsOn integTest

[Use standard conventions as much as possible](#17dp8vu)

All Gradle core plugins follow the software engineering paradigm [convention over configuration](https://en.wikipedia.org/wiki/Convention_over_configuration). The plugin logic provides users with sensible defaults and standards, the conventions, in a certain context. Let’s take the [Java plugin](http://docs.google.com/java_plugin.html#java_plugin) as an example.

* It defines the directory src/main/java as the default source directory for compilation.
* The output directory for compiled source code and other artifacts (like the JAR file) is build.

By sticking to the default conventions, new developers to the project immediately know how to find their way around. While those conventions can be reconfigured, it makes it harder to build script users and authors to manage the build logic and its outcome. Try to stick to the default conventions as much as possible except if you need to adapt to the layout of a legacy project. Refer to the reference page of the relevant plugin to learn about its default conventions.

[Always define a settings file](#3rdcrjn)

Gradle tries to locate a settings.gradle (Groovy DSL) or a settings.gradle.kts (Kotlin DSL) file with every invocation of the build. For that purpose, the runtime walks the hierarchy of the directory tree up to the root directory. The algorithm stops searching as soon as it finds the settings file.

Always add a settings.gradle to the root directory of your build to avoid the initial performance impact. This recommendation applies to single project builds as well as multi-project builds. The file can either be empty or define the desired name of the project.

A typical Gradle project with a settings file look as such:

.  
├── build.gradle  
└── settings.gradle

[Use buildSrc to abstract imperative logic](#26in1rg)

Complex build logic is usually a good candidate for being encapsulated either as custom task or binary plugin. Custom task and plugin implementations should not live in the build script. It is very convenient to use buildSrc for that purpose as long as the code does not need to be shared among multiple, independent projects.

The directory buildSrc is treated as an [included build](http://docs.google.com/composite_builds.html#composite_build_intro). Upon discovery of the directory, Gradle automatically compiles and tests this code and puts it in the classpath of your build script. For multi-project builds there can be only one buildSrc directory, which has to sit in the root project directory. buildSrc should be preferred over [script plugins](http://docs.google.com/plugins.html#sec:script_plugins) as it is easier to maintain, refactor and test the code.

buildSrc uses the same [source code conventions](http://docs.google.com/java_plugin.html#javalayout) applicable to Java and Groovy projects. It also provides direct access to the Gradle API. Additional dependencies can be declared in a dedicated build.gradle under buildSrc.

[Example: Custom buildSrc build script](#z337ya)

**buildSrc/build.gradle**

repositories {  
 mavenCentral()  
}  
  
dependencies {  
 testCompile 'junit:junit:4.12'  
}

A typical project including buildSrc has the following layout. Any code under buildSrc should use a package similar to application code. Optionally, the buildSrc directory can host a build script if additional configuration is needed (e.g. to apply plugins or to declare dependencies).

.  
├── build.gradle  
├── buildSrc  
│   ├── build.gradle  
│   └── src  
│   ├── main  
│   │   └── java  
│   │   └── com  
│   │   └── enterprise  
│   │   ├── Deploy.java  
│   │   └── DeploymentPlugin.java  
│   └── test  
│   └── java  
│   └── com  
│   └── enterprise  
│   └── DeploymentPluginTest.java  
└── settings.gradle

[Declare properties in gradle.properties file](#lnxbz9)

In Gradle, properties can be define in the build script, in a gradle.properties file or as parameters on the command line.

It’s common to declare properties on the command line for ad-hoc scenarios. For example you may want to pass in a specific property value to control runtime behavior just for this one invocation of the build. Properties in a build script can easily become a maintenance headache and convolute the build script logic. The gradle.properties helps with keeping properties separate from the build script and should be explored as viable option. It’s a good location for placing [properties that control the build environment](http://docs.google.com/build_environment.html#sec:gradle_configuration_properties).

A typical project setup places the gradle.properties file in the root directory of the build. Alternatively, the file can also live in the GRADLE\_USER\_HOME directory if you want to it apply to all builds on your machine.

.  
├── build.gradle  
├── gradle.properties  
└── settings.gradle

[Avoid overlapping task outputs](#35nkun2)

Tasks should define inputs and outputs to get the performance benefits of [incremental build functionality](http://docs.google.com/more_about_tasks.html#sec:up_to_date_checks). When declaring the outputs of a task, make sure that the directory for writing outputs is unique among all the tasks in your project.

Intermingling or overwriting output files produced by different tasks compromises up-to-date checking causing slower builds. In turn, these filesystem changes may prevent Gradle’s [build cache](http://docs.google.com/build_cache.html#build_cache) from properly identifying and caching what would otherwise be cacheable tasks.

[Standardizing builds with a custom Gradle distribution](#1ksv4uv)

Often enterprises want to standardize the build platform for all projects in the organization by defining common conventions or rules. You can achieve that with the help of initialization scripts. [Initialization scripts](http://docs.google.com/init_scripts.html#init_scripts) make it extremely easy to apply build logic across all projects on a single machine. For example, to declare a in-house repository and its credentials.

There are some drawbacks to the approach. First of all, you will have to communicate the setup process across all developers in the company. Furthermore, updating the initialization script logic uniformly can prove challenging.

Custom Gradle distributions are a practical solution to this very problem. A custom Gradle distribution is comprised of the standard Gradle distribution plus one or many custom initialization scripts. The initialization scripts come bundled with the distribution and are applied every time the build is run. Developers only need to point their checked-in [Wrapper](http://docs.google.com/gradle_wrapper.html#gradle_wrapper) files to the URL of the custom Gradle distribution.

The following steps are typical for creating a custom Gradle distribution:

1. Implement logic for downloading and repackaging a Gradle distribution.
2. Define one or many initialization scripts with the desired logic.
3. Bundle the initialization scripts with the Gradle distribution.
4. Upload the Gradle distribution archive to a HTTP server.
5. Change the Wrapper files of all projects to point to the URL of the custom Gradle distribution.

You can find a sample project that covers steps one to three in the samples directory of the standard -all Gradle distribution.

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/)