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/)

# The Java Plugin

Contents

[Usage](#4d34og8)

[Project layout](#2s8eyo1)

[Source sets](#17dp8vu)

[Tasks](#3rdcrjn)

[Dependency management](#26in1rg)

[Publishing](#lnxbz9)

[Convention properties](#35nkun2)

[Javadoc](#1ksv4uv)

[Clean](#44sinio)

[Resources](#2jxsxqh)

[CompileJava](#z337ya)

[Test](#3j2qqm3)

[Jar](#1y810tw)

The Java plugin adds Java compilation along with testing and bundling capabilities to a project. It serves as the basis for many of the other JVM language Gradle plugins. You can find a comprehensive introduction and overview to the Java Plugin in the [Building Java Projects](http://docs.google.com/building_java_projects.html#building_java_projects) chapter.

[Usage](#4d34og8)

To use the Java plugin, include the following in your build script:

[Example: Using the Java plugin](#4i7ojhp)

**build.gradle**

apply plugin: 'java'

[Project layout](#2s8eyo1)

The Java plugin assumes the project layout shown below. None of these directories need to exist or have anything in them. The Java plugin will compile whatever it finds, and handles anything which is missing.

src/main/java

Production Java source.

src/main/resources

Production resources, such as XML and properties files.

src/test/java

Test Java source.

src/test/resources

Test resources.

src/*sourceSet*/java

Java source for the source set named *sourceSet*.

src/*sourceSet*/resources

Resources for the source set named *sourceSet*.

[Changing the project layout](#2xcytpi)

You configure the project layout by configuring the appropriate source set. This is discussed in more detail in the following sections. Here is a brief example which changes the main Java and resource source directories.

[Example: Custom Java source layout](#1ci93xb)

**build.gradle**

sourceSets {  
 main {  
 java {  
 srcDirs = ['src/java']  
 }  
 resources {  
 srcDirs = ['src/resources']  
 }  
 }  
}

[Source sets](#17dp8vu)

The plugin adds the following [source sets](http://docs.google.com/building_java_projects.html#sec:java_source_sets):

main

Contains the production source code of the project, which is compiled and assembled into a JAR.

test

Contains your test source code, which is compiled and executed using JUnit or TestNG. These are typically unit tests, but you can include any test in this source set as long as they all share the same compilation and runtime classpaths.

[Source set properties](#3whwml4)

The following table lists some of the important properties of a source set. You can find more details in the API documentation for [SourceSet](http://docs.google.com/dsl/org.gradle.api.tasks.SourceSet.html).

name — (read-only) String

The name of the source set, used to identify it.

output — (read-only) [SourceSetOutput](http://docs.google.com/dsl/org.gradle.api.tasks.SourceSetOutput.html)

The output files of the source set, containing its compiled classes and resources.

output.classesDirs — (read-only) [FileCollection](http://docs.google.com/javadoc/org/gradle/api/file/FileCollection.html)

*Default value*: $buildDir/classes/java/$name, e.g. *build/classes/java/main*

The directories to generate the classes of this source set into. May contain directories for other JVM languages, e.g. *build/classes/kotlin/main*.

output.resourcesDir — File

*Default value*: $buildDir/resources/$name, e.g. *build/resources/main*

The directory to generate the resources of this source set into.

compileClasspath — [FileCollection](http://docs.google.com/javadoc/org/gradle/api/file/FileCollection.html)

*Default value*: ${name}CompileClasspath configuration

The classpath to use when compiling the source files of this source set.

annotationProcessorPath — [FileCollection](http://docs.google.com/javadoc/org/gradle/api/file/FileCollection.html)

*Default value*: ${name}AnnotationProcessor configuration

The processor path to use when compiling the source files of this source set.

runtimeClasspath — [FileCollection](http://docs.google.com/javadoc/org/gradle/api/file/FileCollection.html)

*Default value*: $output, ${name}RuntimeClasspath configuration

The classpath to use when executing the classes of this source set.

java — (read-only) [SourceDirectorySet](http://docs.google.com/dsl/org.gradle.api.file.SourceDirectorySet.html)

The Java source files of this source set. Contains only .java files found in the Java source directories, and excludes all other files.

java.srcDirs — Set<File>

*Default value*: src/$name/java, e.g. *src/main/java*

The source directories containing the Java source files of this source set. You can set this to any value that is described in sec:specifying\_multiple\_files[this section](http://docs.google.com/working_with_files.html#sec:specifying_multiple_files).

java.outputDir — File

*Default value*: $buildDir/classes/java/$name, e.g. *build/classes/java/main*

The directory to generate compiled Java sources into. You can set this to any value that is described in [this section](http://docs.google.com/working_with_files.html#sec:single_file_paths).

resources — (read-only) [SourceDirectorySet](http://docs.google.com/dsl/org.gradle.api.file.SourceDirectorySet.html)

The resources of this source set. Contains only resources, and excludes any .java files found in the resource directories. Other plugins, such as the [Groovy Plugin](http://docs.google.com/groovy_plugin.html#groovy_plugin), exclude additional types of files from this collection.

resources.srcDirs — Set<File>

*Default value*: [src/$name/resources]

The directories containing the resources of this source set. You can set this to any type of value that is described in [this section](http://docs.google.com/working_with_files.html#sec:specifying_multiple_files).

allJava — (read-only) [SourceDirectorySet](http://docs.google.com/dsl/org.gradle.api.file.SourceDirectorySet.html)

*Default value*: Same as java property

All Java files of this source set. Some plugins, such as the Groovy Plugin, add additional Java source files to this collection.

allSource — (read-only) [SourceDirectorySet](http://docs.google.com/dsl/org.gradle.api.file.SourceDirectorySet.html)

*Default value*: Sum of everything in the resources and java properties

All source files of this source set of any language. This includes all resource files and all Java source files. Some plugins, such as the Groovy Plugin, add additional source files to this collection.

[Defining new source sets](#2bn6wsx)

See the [integration test example](http://docs.google.com/java_testing.html#sec:configuring_java_integration_tests) in the *Testing in Java & JVM projects* chapter.

[Some other simple source set examples](#qsh70q)

Adding a JAR containing the classes of a source set:

[Example: Assembling a JAR for a source set](#3as4poj)

**build.gradle**

task intTestJar(type: Jar) {  
 from sourceSets.intTest.output  
}

Generating Javadoc for a source set:

[Example: Generating the Javadoc for a source set](#1pxezwc)

**build.gradle**

task intTestJavadoc(type: Javadoc) {  
 source sourceSets.intTest.allJava  
}

Adding a test suite to run the tests in a source set:

[Example: Running tests in a source set](#49x2ik5)

**build.gradle**

task intTest(type: Test) {  
 testClassesDirs = sourceSets.intTest.output.classesDirs  
 classpath = sourceSets.intTest.runtimeClasspath  
}

[Tasks](#3rdcrjn)

The Java plugin adds a number of tasks to your project, as shown below.

compileJava — [JavaCompile](http://docs.google.com/dsl/org.gradle.api.tasks.compile.JavaCompile.html)

*Depends on*: All tasks which contribute to the compilation classpath, including jar tasks from projects that are on the classpath via project dependencies

Compiles production Java source files using the JDK compiler.

processResources — [Copy](http://docs.google.com/dsl/org.gradle.api.tasks.Copy.html)

Copies production resources into the production resources directory.

classes

*Depends on*: compileJava, processResources

This is an aggregate task that just depends on other tasks. Other plugins may attach additional compilation tasks to it.

compileTestJava — [JavaCompile](http://docs.google.com/dsl/org.gradle.api.tasks.compile.JavaCompile.html)

*Depends on*: classes, and all tasks that contribute to the test compilation classpath

Compiles test Java source files using the JDK compiler.

processTestResources — [Copy](http://docs.google.com/dsl/org.gradle.api.tasks.Copy.html)

Copies test resources into the test resources directory.

testClasses

*Depends on*: compileTestJava, processTestResources

This is an aggregate task that just depends on other tasks. Other plugins may attach additional test compilation tasks to it.

jar — [Jar](http://docs.google.com/dsl/org.gradle.api.tasks.bundling.Jar.html)

*Depends on*: classes

Assembles the production JAR file, based on the classes and resources attached to the main source set.

javadoc — [Javadoc](http://docs.google.com/dsl/org.gradle.api.tasks.javadoc.Javadoc.html)

*Depends on*: classes

Generates API documentation for the production Java source using Javadoc.

test — [Test](http://docs.google.com/dsl/org.gradle.api.tasks.testing.Test.html)

*Depends on*: testClasses, and all tasks which produce the test runtime classpath

Runs the unit tests using JUnit or TestNG.

uploadArchives — [Upload](http://docs.google.com/dsl/org.gradle.api.tasks.Upload.html)

*Depends on*: jar, and any other task that produces an artifact attached to the archives configuration

Uploads artifacts in the archives configuration — including the production JAR file — to the configured repositories.

clean — [Delete](http://docs.google.com/dsl/org.gradle.api.tasks.Delete.html)

Deletes the project build directory.

clean*TaskName* — [Delete](http://docs.google.com/dsl/org.gradle.api.tasks.Delete.html)

Deletes files created by the specified task. For example, cleanJar will delete the JAR file created by the jar task and cleanTest will delete the test results created by the test task.

[SourceSet Tasks](#2p2csry)

For each source set you add to the project, the Java plugin adds the following tasks:

compile*SourceSet*Java — [JavaCompile](http://docs.google.com/dsl/org.gradle.api.tasks.compile.JavaCompile.html)

*Depends on*: All tasks which contribute to the source set’s compilation classpath

Compiles the given source set’s Java source files using the JDK compiler.

process*SourceSet*Resources — [Copy](http://docs.google.com/dsl/org.gradle.api.tasks.Copy.html)

Copies the given source set’s resources into the resources directory.

*sourceSet*Classes — [Task](http://docs.google.com/dsl/org.gradle.api.Task.html)

*Depends on*: compile*SourceSet*Java, process*SourceSet*Resources

Prepares the given source set’s classes and resources for packaging and execution. Some plugins may add additional compilation tasks for the source set.

[Lifecycle Tasks](#147n2zr)

The Java plugin attaches some of its tasks to the lifecycle tasks defined by the [Base Plugin](http://docs.google.com/base_plugin.html#sec:base_tasks) — which the Java Plugin applies automatically — and it also adds a few other lifecycle tasks:

assemble

*Depends on*: jar, and all other tasks that create artifacts attached to the archives configuration

Aggregate task that assembles all the archives in the project. This task is added by the Base Plugin.

check

*Depends on*: test

Aggregate task that performs verification tasks, such as running the tests. Some plugins add their own verification tasks to check. You should also attach any custom Test tasks to this lifecycle task if you want them to execute for a full build. This task is added by the Base Plugin.

build

*Depends on*: check, assemble

Aggregate tasks that performs a full build of the project. This task is added by the Base Plugin.

buildNeeded

*Depends on*: build, and buildNeeded tasks in all projects that are dependencies in the testRuntimeClasspath configuration.

Performs a full build of the project and all projects it depends on.

buildDependents

*Depends on*: build, and buildDependents tasks in all projects that have this project as a dependency in theeir testRuntimeClasspath configurations

Performs a full build of the project and all projects which depend upon it.

build*ConfigName* — *task rule*

*Depends on*: all tasks that generate the artifacts attached to the named — *ConfigName* — configuration

Assembles the artifacts for the specified configuration. This rule is added by the Base Plugin.

upload*ConfigName* — *task rule*, type: [Upload](http://docs.google.com/dsl/org.gradle.api.tasks.Upload.html)

*Depends on*: all tasks that generate the artifacts attached to the named — *ConfigName* — configuration

Assembles and uploads the artifacts in the specified configuration. This rule is added by the Base Plugin.

The following diagram shows the relationships between these tasks.



*Figure 1. Java plugin - tasks*

[Dependency management](#26in1rg)

The Java plugin adds a number of [dependency configurations](http://docs.google.com/managing_dependency_configurations.html#managing_dependency_configurations) to your project, as shown below. It assigns those configurations to tasks such as compileJava and test.

[Dependency configurations](#3o7alnk)

| **✨** | To find information on the api configuration, please consult the [Java Library Plugin](http://docs.google.com/java_library_plugin.html#sec:java_library_separation) reference documentation and the [Dependency Management Tutorial](http://docs.google.com/dependency_management_for_java_projects.html#dependency_management_for_java_projects). |
| --- | --- |

~~compile~~(Deprecated)

Compile time dependencies. Superseded by implementation.

implementation extends compile

Implementation only dependencies.

compileOnly

Compile time only dependencies, not used at runtime.

compileClasspath extends compile, compileOnly, implementation

Compile classpath, used when compiling source. Used by task compileJava.

annotationProcessor

Annotation processors used during compilation.

~~runtime~~(Deprecated) extends compile

Runtime dependencies. Superseded by runtimeOnly.

runtimeOnly

Runtime only dependencies.

runtimeClasspath extends runtimeOnly, runtime, implementation

Runtime classpath contains elements of the implementation, as well as runtime only elements.

~~testCompile~~(Deprecated) extends compile

Additional dependencies for compiling tests. Superseded by testImplementation.

testImplementation extends testCompile, implementation

Implementation only dependencies for tests.

testCompileOnly

Additional dependencies only for compiling tests, not used at runtime.

testCompileClasspath extends testCompile, testCompileOnly, testImplementation

Test compile classpath, used when compiling test sources. Used by task compileTestJava.

~~testRuntime~~(Deprecated) extends runtime, testCompile

Additional dependencies for running tests only. Used by task test. Superseded by testRuntimeOnly.

testRuntimeOnly extends runtimeOnly

Runtime only dependencies for running tests. Used by task test.

testRuntimeClasspath extends testRuntimeOnly, testRuntime, testImplementation

Runtime classpath for running tests.

archives

Artifacts (e.g. jars) produced by this project. Used by tasks uploadArchives.

default extends runtime

The default configuration used by a project dependency on this project. Contains the artifacts and dependencies required by this project at runtime.



*Figure 2. Java plugin - dependency configurations*

For each source set you add to the project, the Java plugins adds the following dependency configurations:

[SourceSet dependency configurations](#23ckvvd)

*~~sourceSet~~*~~Compile~~(Deprecated)

Compile time dependencies for the given source set. Superseded by *sourceSet*Implementation.

*sourceSet*Implementation extends *sourceSet*Compile

Compile time dependencies for the given source set. Used by *sourceSet*CompileClasspath, *sourceSet*RuntimeClasspath.

*sourceSet*CompileOnly

Compile time only dependencies for the given source set, not used at runtime.

*sourceSet*CompileClasspath extends compile*SourceSet*Java

Compile classpath, used when compiling source. Used by *sourceSet*Compile, *sourceSet*CompileOnly, *sourceSet*Implementation.

*sourceSet*AnnotationProcessor

Annotation processors used during compilation of this source set.

*~~sourceSet~~*~~Runtime~~(Deprecated)

Runtime dependencies for the given source set. Used by *sourceSet*Compile. Superseded by *sourceSet*RuntimeOnly.

*sourceSet*RuntimeOnly

Runtime only dependencies for the given source set.

*sourceSet*RuntimeClasspath extends *sourceSet*RuntimeOnly, *sourceSet*Runtime, *sourceSet*Implementation

Runtime classpath contains elements of the implementation, as well as runtime only elements.

[Publishing](#lnxbz9)

components.java

A [SoftwareComponent](http://docs.google.com/javadoc/org/gradle/api/component/SoftwareComponent.html) for [publishing](http://docs.google.com/publishing_overview.html#publishing_overview) the production JAR created by the jar task. This component includes the runtime dependency information for the JAR.

[Convention properties](#35nkun2)

The Java Plugin adds a number of convention properties to the project, shown below. You can use these properties in your build script as though they were properties of the project object.

[Directory properties](#ihv636)

String reporting.baseDir

The name of the directory to generate reports into, relative to the build directory. Default value: reports

(read-only) File reportsDir

The directory to generate reports into. Default value: *buildDir*/*reporting.baseDir*

String testResultsDirName

The name of the directory to generate test result .xml files into, relative to the build directory. Default value: test-results

(read-only) File testResultsDir

The directory to generate test result .xml files into. Default value: *buildDir*/*testResultsDirName*

String testReportDirName

The name of the directory to generate the test report into, relative to the reports directory. Default value: tests

(read-only) File testReportDir

The directory to generate the test report into. Default value: *reportsDir*/testReportDirName

String libsDirName

The name of the directory to generate libraries into, relative to the build directory. Default value: libs

(read-only) File libsDir

The directory to generate libraries into. Default value: *buildDir*/*libsDirName*

String distsDirName

The name of the directory to generate distributions into, relative to the build directory. Default value: distributions

(read-only) File distsDir

The directory to generate distributions into. Default value: *buildDir*/*distsDirName*

String docsDirName

The name of the directory to generate documentation into, relative to the build directory. Default value: docs

(read-only) File docsDir

The directory to generate documentation into. Default value: *buildDir*/*docsDirName*

String dependencyCacheDirName

The name of the directory to use to cache source dependency information, relative to the build directory. Default value: dependency-cache

[Other convention properties](#32hioqz)

(read-only) [SourceSetContainer](http://docs.google.com/javadoc/org/gradle/api/tasks/SourceSetContainer.html) sourceSets

Contains the project’s source sets. Default value: Not null [SourceSetContainer](http://docs.google.com/javadoc/org/gradle/api/tasks/SourceSetContainer.html)

[JavaVersion](http://docs.google.com/javadoc/org/gradle/api/JavaVersion.html) sourceCompatibility

Java version compatibility to use when compiling Java source. Default value: version of the current JVM in use [JavaVersion](http://docs.google.com/javadoc/org/gradle/api/JavaVersion.html). Can also set using a String or a Number, e.g. '1.5' or 1.5.

[JavaVersion](http://docs.google.com/javadoc/org/gradle/api/JavaVersion.html) targetCompatibility

Java version to generate classes for. Default value: *sourceCompatibility*. Can also set using a String or Number, e.g. '1.5' or 1.5.

String archivesBaseName

The basename to use for archives, such as JAR or ZIP files. Default value: *projectName*

[Manifest](http://docs.google.com/javadoc/org/gradle/api/java/archives/Manifest.html) manifest

The manifest to include in all JAR files. Default value: an empty manifest.

These properties are provided by convention objects of type [JavaPluginConvention](http://docs.google.com/javadoc/org/gradle/api/plugins/JavaPluginConvention.html), and [BasePluginConvention](http://docs.google.com/javadoc/org/gradle/api/plugins/BasePluginConvention.html).

[Javadoc](#1ksv4uv)

The javadoc task is an instance of [Javadoc](http://docs.google.com/dsl/org.gradle.api.tasks.javadoc.Javadoc.html). It supports the core Javadoc options and the options of the standard doclet described in the [reference documentation](http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html) of the Javadoc executable. For a complete list of supported Javadoc options consult the API documentation of the following classes: [CoreJavadocOptions](http://docs.google.com/javadoc/org/gradle/external/javadoc/CoreJavadocOptions.html) and [StandardJavadocDocletOptions](http://docs.google.com/javadoc/org/gradle/external/javadoc/StandardJavadocDocletOptions.html).

[Javadoc properties](#1hmsyys)

[FileCollection](http://docs.google.com/javadoc/org/gradle/api/file/FileCollection.html) classpath

Default value: sourceSets.main.output + sourceSets.main.compileClasspath

[FileTree](http://docs.google.com/javadoc/org/gradle/api/file/FileTree.html) source

Default value: sourceSets.main.allJava. Can set using anything described in [Understanding implicit conversion to file collections](http://docs.google.com/working_with_files.html#sec:specifying_multiple_files).

File destinationDir

Default value: *docsDir*/javadoc

String title

Default value: The name and version of the project

[Clean](#44sinio)

The clean task is an instance of [Delete](http://docs.google.com/dsl/org.gradle.api.tasks.Delete.html). It simply removes the directory denoted by its dir property.

[Clean properties](#41mghml)

File dir

Default value: *buildDir*

[Resources](#2jxsxqh)

The Java plugin uses the [Copy](http://docs.google.com/dsl/org.gradle.api.tasks.Copy.html) task for resource handling. It adds an instance for each source set in the project. You can find out more about the copy task in [File copying in depth](http://docs.google.com/working_with_files.html#sec:copying_files).

[ProcessResources properties](#2grqrue)

Object srcDirs

Default value: *sourceSet*.resources. Can set using anything described in [Understanding implicit conversion to file collections](http://docs.google.com/working_with_files.html#sec:specifying_multiple_files).

File destinationDir

Default value: *sourceSet*.output.resourcesDir. Can set using anything described in [file paths in depth](http://docs.google.com/working_with_files.html#sec:locating_files).

[CompileJava](#z337ya)

The Java plugin adds a [JavaCompile](http://docs.google.com/dsl/org.gradle.api.tasks.compile.JavaCompile.html) instance for each source set in the project. Some of the most common configuration options are shown below.

[Compile properties](#vx1227)

[FileCollection](http://docs.google.com/javadoc/org/gradle/api/file/FileCollection.html) classpath

Default value: *sourceSet*.compileClasspath

[FileTree](http://docs.google.com/javadoc/org/gradle/api/file/FileTree.html) source

Default value: *sourceSet*.java. Can set using anything described in [Understanding implicit conversion to file collections](http://docs.google.com/working_with_files.html#sec:specifying_multiple_files).

File destinationDir

Default value: *sourceSet*.java.outputDir

By default, the Java compiler runs in the Gradle process. Setting options.fork to true causes compilation to occur in a separate process. In the case of the Ant javac task, this means that a new process will be forked for each compile task, which can slow down compilation. Conversely, Gradle’s direct compiler integration (see above) will reuse the same compiler process as much as possible. In both cases, all fork options specified with options.forkOptions will be honored.

[Incremental Java compilation](#3fwokq0)

Gradle comes with a sophisticated incremental Java compiler that is active by default.

This gives you the following benefits

* Incremental builds are much faster.
* The smallest possible number of class files are changed. Classes that don’t need to be recompiled remain unchanged in the output directory. An example scenario when this is really useful is using JRebel - the fewer output classes are changed the quicker the JVM can use refreshed classes.

To help you understand how incremental compilation works, the following provides a high-level overview:

* Gradle will recompile all classes *affected* by a change.
* A class is *affected* if it has been changed or if it depends on another affected class. This works no matter if the other class is defined in the same project, another project or even an external library.
* A class’s dependencies are determined from type references in its bytecode.
* Since constants can be inlined, any change to a constant will result in Gradle recompiling all source files. For that reason, you should try to minimize the use of constants in your source code and replace them with static methods where possible.
* Since source-retention annotations are not visible in bytecode, changes to a source-retention annotation will result in full recompilation.
* You can improve incremental compilation performance by applying good software desing principles like loose coupling. For instance, if you put an interface between a concrete class and its dependents, the dependent classes are only recompiled when the interface changes, but not when the implementation changes.
* The class analysis is cached in the project directory, so the first build after a clean checkout can be slower. Consider turning off the incremental compiler on your build server.

[Known issues](#1v1yuxt)

* If a compile task fails due to a compile error, it will do a full compilation again the next time it is invoked.

[Incremental annotation processing](#4f1mdlm)

Starting with Gradle 4.7, the incremental compiler also supports incremental annotation processing. Annotation processors need to opt in to this feature, otherwise they will trigger a full recompilation.

As a user you can see which annotation processors are triggering full recompilations in the --info log. Incremental annotation processing will be deactivated if a custom executable or javaHome is configured on the compile task.

[Making an annotation processor incremental](#2u6wntf)

Please first have a look at [incremental Java compilation](#3fwokq0), as incremental annotation processing builds on top of it.

Gradle supports incremental compilation for two common categories of annotation processors: "isolating" and "aggregating". Please consult the information below to decide which category fits your processor.

You can then register your processor for incremental compilation using a file in the processor’s META-INF directory. The format is one line per processor, with the fully qualified name of the processor class and its category separated by a comma.

[Example: Registering incremental annotation processors](#19c6y18)

**processor/src/main/resources/META-INF/gradle/incremental.annotation.processors**

EntityProcessor,isolating  
ServiceRegistryProcessor,dynamic

If your processor can only decide at runtime whether it is incremental or not, you can declare it as "dynamic" in the META-INF descriptor and return its true type at runtime using the [Processor#getSupportedOptions()](https://docs.oracle.com/javase/10/docs/api/javax/annotation/processing/Processor.html#getSupportedOptions()) method.

[Example: Registering incremental annotation processors dynamically](#3tbugp1)

**processor/src/main/java/ServiceRegistryProcessor.java**

@Override  
public Set<String> getSupportedOptions() {  
 return Collections.singleton("org.gradle.annotation.processing.aggregating");  
}

Both categories have the following limitations:

* They must generate their files using the [Filer API](https://docs.oracle.com/javase/10/docs/api/javax/annotation/processing/Filer.html). Writing files any other way will result in silent failures later on, as these files won’t be cleaned up correctly. If your processor does this, it cannot be incremental.
* They must not depend on compiler-specific APIs like [com.sun.source.util.Trees](https://docs.oracle.com/javase/8/docs/jdk/api/javac/tree/com/sun/source/util/Trees.html). Gradle wraps the processing APIs, so attempts to cast to compiler-specific types will fail. If your processor does this, it cannot be incremental, unless you have some fallback mechanism.
* If they use [Filer#getResource](https://docs.oracle.com/javase/10/docs/api/javax/annotation/processing/Filer.html#getResource(javax.tools.JavaFileManager.Location,java.lang.CharSequence,java.lang.CharSequence)), Gradle will recompile all source files. See [gradle/issues/4701](https://github.com/gradle/gradle/issues/4701)
* If they use [Filer#createResource](https://docs.oracle.com/javase/10/docs/api/javax/annotation/processing/Filer.html#createResource(javax.tools.JavaFileManager.Location,java.lang.CharSequence,java.lang.CharSequence,javax.lang.model.element.Element...)), Gradle will recompile all source files. See [gradle/issues/4702](https://github.com/gradle/gradle/issues/4702)

["Isolating" annotation processors](#28h4qwu)

The fastest category, these look at each annotated element in isolation, creating generated files or validation messages for it. For instance an EntityProcessor could create a <TypeName>Repository for each type annotated with @Entity.

[Example: An isolated annotation processor](#nmf14n)

**processor/src/main/java/EntityProcessor.java**

Set<? extends Element> entities = roundEnv.getElementsAnnotatedWith(entityAnnotation);  
for (Element entity : entities) {  
 createRepository((TypeElement) entity);  
}

"Isolating" processors have the following limitations:

* They must make all decisions (code generation, validation messages) for an annotated type based on information reachable from its AST. This means you can analyze the types' super-class, method return types, annotations etc., even transitively. But you cannot make decisions based on unrelated elements in the RoundEnvironment. Doing so will result in silent failures because too few files will be recompiled later. If your processor needs to make decisions based on a combination of otherwise unrelated elements, mark it as "aggregating" instead.
* They must provide exactly one originating element for each file generated with the Filer API. If zero or many originating elements are provided, Gradle will recompile all source files.

When a source file is recompiled, Gradle will recompile all files generated from it. When a source file is deleted, the files generated from it are deleted.

["Aggregating" annotation processors](#37m2jsg)

These can aggregate several source files into one ore more output files or validation messages. For instance, a ServiceRegistryProcessor could create a single ServiceRegistry with one method for each type annotated with @Service

[Example: An aggregating annotation processor](#1mrcu09)

**processor/src/main/java/ServiceRegistryProcessor.java**

JavaFileObject serviceRegistry = filer.createSourceFile("ServiceRegistry");  
Writer writer = serviceRegistry.openWriter();  
writer.write("public class ServiceRegistry {");  
for (Element service : roundEnv.getElementsAnnotatedWith(serviceAnnotation)) {  
 addServiceCreationMethod(writer, (TypeElement) service);  
}  
writer.write("}");  
writer.close();

"Aggregating" processors have the following limitations:

* They can only read CLASS or RUNTIME retention annotations
* They can only read parameter names if the user passes the -parameters compiler argument.

Gradle will always reprocess (but not recompile) all annotated files that the processor was registered for. Gradle will always recompile any files the processor generates.

[Compile avoidance](#46r0co2)

If a dependent project has changed in an [ABI](https://en.wikipedia.org/wiki/Application_binary_interface)-compatible way (only its private API has changed), then Java compilation tasks will be up-to-date. This means that if project A depends on project B and a class in B is changed in an ABI-compatible way (typically, changing only the body of a method), then Gradle won’t recompile A.

Some of the types of changes that do not affect the public API and are ignored:

* Changing a method body
* Changing a comment
* Adding, removing or changing private methods, fields, or inner classes
* Adding, removing or changing a resource
* Changing the name of jars or directories in the classpath
* Renaming a parameter

Compile-avoidance is deactivated if annotation processors are found on the compile classpath, because for annotation processors the implementation details matter. Annotation processors should be declared on the annotation processor path instead. Gradle 5.0 will ignore processors on the compile classpath.

[Example: Declaring annotation processors](#2lwamvv)

**build.gradle**

dependencies {  
 // The dagger compiler and its transitive dependencies will only be found on annotation processing classpath  
 annotationProcessor 'com.google.dagger:dagger-compiler:2.8'  
  
 // And we still need the Dagger library on the compile classpath itself  
 implementation 'com.google.dagger:dagger:2.8'  
}

[Test](#3j2qqm3)

The test task is an instance of [Test](http://docs.google.com/dsl/org.gradle.api.tasks.testing.Test.html). It automatically detects and executes all unit tests in the test source set. It also generates a report once test execution is complete. JUnit and TestNG are both supported. Have a look at [Test](http://docs.google.com/dsl/org.gradle.api.tasks.testing.Test.html) for the complete API.

See the [Testing in Java & JVM projects](http://docs.google.com/java_testing.html#java_testing) chapter for more details.

[Jar](#1y810tw)

The jar task creates a JAR file containing the class files and resources of the project. The JAR file is declared as an artifact in the archives dependency configuration. This means that the JAR is available in the classpath of a dependent project. If you upload your project into a repository, this JAR is declared as part of the dependency descriptor. You can learn more about how to work with archives in [Archive creation in depth](http://docs.google.com/working_with_files.html#sec:archives) and artifact configurations in [Legacy Publishing](http://docs.google.com/artifact_management.html#artifact_management).

[Manifest](#111kx3o)

Each jar or war object has a manifest property with a separate instance of [Manifest](http://docs.google.com/javadoc/org/gradle/api/java/archives/Manifest.html). When the archive is generated, a corresponding MANIFEST.MF file is written into the archive.

[Example: Customization of MANIFEST.MF](#3l18frh)

**build.gradle**

jar {  
 manifest {  
 attributes("Implementation-Title": "Gradle",  
 "Implementation-Version": version)  
 }  
}

You can create stand-alone instances of a Manifest. You can use that for example, to share manifest information between jars.

[Example: Creating a manifest object.](#206ipza)

**build.gradle**

ext.sharedManifest = manifest {  
 attributes("Implementation-Title": "Gradle",  
 "Implementation-Version": version)  
}  
task fooJar(type: Jar) {  
 manifest = project.manifest {  
 from sharedManifest  
 }  
}

You can merge other manifests into any Manifest object. The other manifests might be either described by a file path or, like in the example above, by a reference to another Manifest object.

[Example: Separate MANIFEST.MF for a particular archive](#4k668n3)

**build.gradle**

task barJar(type: Jar) {  
 manifest {  
 attributes key1: 'value1'  
 from sharedManifest, 'src/config/basemanifest.txt'  
 from('src/config/javabasemanifest.txt',  
 'src/config/libbasemanifest.txt') {  
 eachEntry { details ->  
 if (details.baseValue != details.mergeValue) {  
 details.value = baseValue  
 }  
 if (details.key == 'foo') {  
 details.exclude()  
 }  
 }  
 }  
 }  
}

Manifests are merged in the order they are declared by the from statement. If the base manifest and the merged manifest both define values for the same key, the merged manifest wins by default. You can fully customize the merge behavior by adding eachEntry actions in which you have access to a [ManifestMergeDetails](http://docs.google.com/javadoc/org/gradle/api/java/archives/ManifestMergeDetails.html) instance for each entry of the resulting manifest. The merge is not immediately triggered by the from statement. It is done lazily, either when generating the jar, or by calling writeTo or effectiveManifest

You can easily write a manifest to disk.

[Example: Saving a MANIFEST.MF to disk](#2zbgiuw)

**build.gradle**

jar.manifest.writeTo("$buildDir/mymanifest.mf")

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/)