Gradle Build Language Reference

Introduction

This reference guide describes the various types which make up the Gradle build language, or DSL.

Some basics

There are a few basic concepts that you should understand, which will help you write Gradle scripts.

First, Gradle scripts are *configuration scripts*. As the script executes, it configures an object of a particular type. For example, as a build script executes, it configures an object of type `Project`. This object is called the *delegate object* of the script. The following table shows the delegate for each type of Gradle script.

Type of script	Delegates to instance of
Build script	`Project`
Init script	`Gradle`
Settings script	`Settings`

The properties and methods of the delegate object are available for you to use in the script.

Second, each Gradle script implements the `Script` interface. This interface defines a number of properties and methods which you can use in the script.

Build script structure

A build script is made up of zero or more statements and script blocks. Statements can include method calls, property assignments, and local variable definitions. A script block is a method call which takes a closure as a parameter. The closure is treated as a *configuration closure* which configures some delegate object as it executes. The top level script blocks are listed below.

Block	Description
`allprojects { }`	Configures this project and each of its sub-projects.
`artifacts { }`	Configures the published artifacts for this project.
`buildscript { }`	Configures the build script classpath for this project.
`configurations { }`	Configures the dependency configurations for this project.
`dependencies { }`	Configures the dependencies for this project.
`repositories { }`	Configures the repositories for this project.
`sourceSets { }`	Configures the source sets of this project.
`subprojects { }`	Configures the sub-projects of this project.
`publishing { }`	Configures the `PublishingExtension` added by the publishing plugin.

A build script is also a Groovy script, and so can contain those elements allowed in a Groovy script, such as method definitions and class definitions.

Core types

Listed below are some of the central types which are used in Gradle scripts:

Type	Description
`Project`	This interface is the main API you use to interact with Gradle from your build file. From a `Project`, you have programmatic access to all of Gradle's features.
`Task`	A `Task` represents a single atomic piece of work for a build, such as compiling classes or generating javadoc.
`Gradle`	Represents an invocation of Gradle.
`Settings`	Declares the configuration required to instantiate and configure the hierarchy of `Project` instances which are to participate in a build.
`IncludedBuild`	A build that is included in the composite.
`Script`	This interface is implemented by all Gradle scripts to add in some Gradle-specific methods. As your compiled script class will implement this interface, you can use the methods and properties declared by this interface directly in your script.
`JavaToolChain`	A set of tools for building from Java source.

Task types

Type	Description
`SourceSet`	A `SourceSet` represents a logical group of Java source and resources.
`SourceSetOutput`	A collection of all output directories (compiled classes, processed resources, etc.) - notice that `SourceSetOutput` extends `FileCollection`.
`SourceDirectorySet`	A `SourceDirectorySet` represents a set of source files composed from a set of source directories, along with associated include and exclude patterns.
`IncrementalTaskInputs`	Provides access to any input files that need to be processed by an incremental task.
`Configuration`	A `Configuration` represents a group of artifacts and their dependencies. Find more information about declaring dependencies to a configuration or about managing configurations in docs for `ConfigurationContainer`
`ResolutionStrategy`	Defines the strategies around dependency resolution. For example, forcing certain dependency versions, substitutions, conflict resolutions or snapshot timeouts.
`ArtifactResolutionQuery`	A builder to construct a query that can resolve selected software artifacts of the specified components.

- JavaExec
- JDepend
- Pmd
- PublishToIvyRepository
- PublishToMavenRepository
- ScalaCompile
- ScalaDoc
- InitBuild
- Sign
- Sync
- Tar
- AbstractTestTask
- Test
- TestReport
- Upload
- War
- Wrapper
- WriteProperties
- Zip

Reporting types

- CustomizableHtmlReport
- SingleFileReport
- DirectoryReport
- FindBugsXmlReport
- Report
- Reporting
- ReportContainer

Type	Description
`ComponentSelection`	Represents a tuple of the component selector of a module and a candidate version to be evaluated in a component selection rule.
`ComponentSelectionRules`	Represents a container for component selection rules. Rules can be applied as part of the resolutionStrategy of a configuration and individual components can be explicitly accepted or rejected by rule. Components that are neither accepted or rejected will be subject to the default version matching strategies.
`ExtensionAware`	Objects that can be extended at runtime with other objects.
`ExtraPropertiesExtension`	Additional, ad-hoc, properties for Gradle domain objects.
`PublishingExtension`	The configuration of how to “publish” the different components of a project.
`IvyPublication`	A `IvyPublication` is the representation/configuration of how Gradle should publish something in Ivy format, to an Ivy repository. You directly add a named Ivy Publication the project's `publishing.publications` container by providing `IvyPublication` as the type.
`IvyArtifact`	An artifact published as part of a `IvyPublication`.

Type	Description
`IvyArtifactSet`	A Collection of `IvyArtifact`s to be included in an `IvyPublication`. Being a `DomainObjectSet`, a `IvyArtifactSet` provides convenient methods for querying, filtering, and applying actions to the set of `IvyArtifact`s.
`IvyModuleDescriptorSpec`	The descriptor of any Ivy publication.
`MavenPublication`	A `MavenPublication` is the representation/configuration of how Gradle should publish something in Maven format. You directly add a named Maven Publication the project's `publishing.publications` container by providing `MavenPublication` as the type.
`MavenArtifact`	An artifact published as part of a `MavenPublication`.
`MavenArtifactSet`	A Collection of `MavenArtifact`s to be included in a `MavenPublication`. Being a `DomainObjectSet`, a `MavenArtifactSet` provides convenient methods for querying, filtering, and applying actions to the set of `MavenArtifact`s.
`MavenPom`	The POM for a Maven publication. The `MavenPom.withXml(org.gradle.api.Action)` method can be used to modify the descriptor after it has been generated according to the publication data.
`PluginDependenciesSpec`	The DSL for declaring plugins to use in a script.

- PrebuiltStaticLibraryBinary
- NativeComponentSpec
- NativeExecutableSpec
- NativeLibrarySpec
- NativeTestSuiteSpec
- CUnitTestSuiteSpec
- GoogleTestTestSuiteSpec
- NativeBinarySpec
- NativeExecutableBinarySpec
- NativeLibraryBinarySpec
- SharedLibraryBinarySpec
- StaticLibraryBinarySpec
- NativeTestSuiteBinarySpec
- CUnitTestSuiteBinarySpec
- GoogleTestTestSuiteBinarySpec
- NativePlatform
- BuildType
- Flavor
- Gcc
- Clang
- VisualCpp
- AssemblerSourceSet
- CSourceSet
- CppSourceSet
- ObjectiveCSourceSet
- ObjectiveCppSourceSet
- WindowsResourceSet

Type	Description
`PluginDependencySpec`	A mutable specification of a dependency on a plugin.
`PluginManagementSpec`	Configures how plugins are resolved.
`ResourceHandler`	Provides access to resource-specific utility methods, for example factory methods that create various resources.
`TextResourceFactory`	Creates `TextResource` s backed by sources such as strings, files, and archive entries.

Container types

Container types that handle various declarative elements (e.g. dependencies, configurations, artifacts, etc.):

Type	Description
`ConfigurationContainer`	A `ConfigurationContainer` is responsible for declaring and managing configurations. See also `Configuration` .
`RepositoryHandler`	A `RepositoryHandler` manages a set of repositories, allowing repositories to be defined and queried.
`DependencyHandler`	A `DependencyHandler` is used to declare dependencies. Dependencies are grouped into configurations (see `Configuration`).

Type	Description
`ArtifactHandler`	This class is for defining artifacts to be published and adding them to configurations. Creating publish artifacts does not mean to create an archive. What is created is a domain object which represents a file to be published and information on how it should be published (e.g. the name).

Build Cache types

Types used to connect to and configure the build cache:

Type	Description
`BuildCacheConfiguration`	Configuration for the build cache for an entire Gradle build.
`DirectoryBuildCache`	Configuration object for the local directory build cache.
`HttpBuildCache`	Configuration object for the HTTP build cache. The build cache only supports BASIC authentication currently.

Input Normalization types

Types used to configure input normalization

Type	Description
`InputNormalizationHandler`	Used to configure input normalization. Currently, it is only possible to configure runtime classpath normalization.
`InputNormalization`	Input normalization configuration. Input normalization is used when Gradle tries to determine if two task inputs are different. Gradle normalizes both inputs and the inputs are considered different if and only if the normalizations are different.
`RuntimeClasspathNormalization`	Configuration of runtime classpath normalization.

Help Task types

Below are the task types that are available for every Gradle project. Those task types can also be declared and configured directly in the build script.

Type	Description
`TaskReportTask`	Displays a list of tasks in the project. An instance of this type is used when you execute the `tasks` task from the command-line.

Type	Description
`ProjectReportTask`	Displays a list of projects in the build. An instance of this type is used when you execute the `projects` task from the command-line.
`DependencyReportTask`	Displays the dependency tree for a project. An instance of this type is used when you execute the `dependencies` task from the command-line.
`DependencyInsightReportTask`	Generates a report that attempts to answer questions like:
`PropertyReportTask`	Displays the properties of a project. An instance of this type is used when you execute the `properties` task from the command-line.
`ComponentReport`	Displays some details about the software components produced by the project.
`DependentComponentsReport`	Displays dependent components.
`ModelReport`	Displays some details about the configuration model of the project. An instance of this type is used when you execute the `model` task from the command-line.

Task types

Listed below are the various task types which are available for use in your build script:

Type	Description
`AntlrTask`	Generates parsers from Antlr grammars.
`BuildEnvironmentReportTask`	Provides information about the build environment for the project that the task is associated with.
`Checkstyle`	Runs Checkstyle against some source files.
`CodeNarc`	Runs CodeNarc against some source files.
`CompareGradleBuilds`	Executes two Gradle builds (that can be the same build) with specified versions and compares the outcomes. Please see the “Comparing Builds” chapter of the Gradle User Guide for more information.
`Copy`	Copies files into a destination directory. This task can also rename and filter files as it copies. The task implements `CopySpec` for specifying what to copy.
`CreateStartScripts`	Creates start scripts for launching JVM applications.
`Delete`	Deletes files or directories. Example:
`Ear`	Assembles an EAR archive.
`Exec`	Executes a command line process. Example:
`FindBugs`	Analyzes code with FindBugs. See the FindBugs Manual for additional information on configuration options.
`GenerateIvyDescriptor`	Generates an Ivy XML Module Descriptor file.

Type	Description
`GenerateMavenPom`	Generates a Maven module descriptor (POM) file.
`GenerateBuildDashboard`	Generates build dashboard report.
`GradleBuild`	Executes a Gradle build.
`GroovyCompile`	Compiles Groovy source files, and optionally, Java source files.
`Groovydoc`	Generates HTML API documentation for Groovy source, and optionally, Java source.
`HtmlDependencyReportTask`	Generates an HTML dependency report. This report combines the features of the ASCII dependency report and those of the ASCII dependency insight report. For a given project, it generates a tree of the dependencies of every configuration, and each dependency can be clicked to show the insight of this dependency.
`JacocoReport`	Task to generate HTML, Xml and CSV reports of Jacoco coverage data.
`JacocoMerge`	Task to merge multiple execution data files into one.
`JacocoCoverageVerification`	Task for verifying code coverage metrics. Fails the task if violations are detected based on specified rules.
`Jar`	Assembles a JAR archive.

Type	Description
`JavaCompile`	Compiles Java source files.
`Javadoc`	Generates HTML API documentation for Java classes.
`JavaExec`	Executes a Java application in a child process.
`JDepend`	Analyzes code with JDepend.
`Pmd`	Runs a set of static code analysis rules on Java source code files and generates a report of problems found.
`PublishToIvyRepository`	Publishes an IvyPublication to an IvyArtifactRepository.
`PublishToMavenRepository`	Publishes a `MavenPublication` to a `MavenArtifactRepository`.
`ScalaCompile`	Compiles Scala source files, and optionally, Java source files.
`ScalaDoc`	Generates HTML API documentation for Scala source files.
`InitBuild`	Generates a Gradle project structure.
`Sign`	A task for creating digital signature files for one or more; tasks, files, publishable artifacts or configurations.

Type	Description
`Sync`	Synchronizes the contents of a destination directory with some source directories and files.
`Tar`	Assembles a TAR archive.
`AbstractTestTask`	Abstract class for all test task.
`Test`	Executes JUnit (3.8.x or 4.x) or TestNG tests. Test are always run in (one or more) separate JVMs. The sample below shows various configuration options.
`TestReport`	Generates an HTML test report from the results of one or more `Test` tasks.
`Upload`	Uploads the artifacts of a `Configuration` to a set of repositories.
`War`	Assembles a WAR archive.
`Wrapper`	Generates scripts (for *nix and windows) which allow you to build your project with Gradle, without having to install Gradle.
`WriteProperties`	Writes a `Properties` in a way that the results can be expected to be reproducible.
`Zip`	Assembles a ZIP archive. The default is to compress the contents of the zip.

Reporting types

Listed below are some of the types which are used when generating reports:

Type	Description
`CustomizableHtmlReport`	A HTML Report whose generation can be customized with a XSLT stylesheet.
`SingleFileReport`	A report that is a single file.
`DirectoryReport`	A directory based report to be created.
`FindBugsXmlReport`	The single file XML report for FindBugs.
`Report`	A file based report to be created.
`Reporting`	An object that provides reporting options.
`ReportContainer`	A container of `Report` objects, that represent potential reports.
`ReportingExtension`	A project extension named "reporting" that provides basic reporting settings and utilities.

Eclipse/IDEA model types

Used to configure Eclipse or IDEA plugins

Type	Description
`EclipseModel`	DSL-friendly model of the Eclipse project information. First point of entry for customizing Eclipse project generation.

Type	Description
`EclipseProject`	Enables fine-tuning project details (.project file) of the Eclipse plugin
`EclipseClasspath`	The build path settings for the generated Eclipse project. Used by the `GenerateEclipseClasspath` task to generate an Eclipse .classpath file.
`EclipseJdt`	Enables fine-tuning jdt details of the Eclipse plugin
`EclipseWtp`	Enables fine-tuning wtp/wst details of the Eclipse plugin
`EclipseWtpComponent`	Enables fine-tuning wtp component details of the Eclipse plugin
`EclipseWtpFacet`	Enables fine-tuning wtp facet details of the Eclipse plugin
`IdeaModel`	DSL-friendly model of the IDEA project information. First point of entry when it comes to customizing the IDEA generation.
`IdeaProject`	Enables fine-tuning project details (*.ipr file) of the IDEA plugin.
`IdeaModule`	Enables fine-tuning module details (*.iml file) of the IDEA plugin.
`IdeaWorkspace`	Enables fine-tuning workspace details (*.iws file) of the IDEA plugin.
`XmlFileContentMerger`	Models the generation/parsing/merging capabilities. Adds XML-related hooks.

Type	Description
`FileContentMerger`	Models the generation/parsing/merging capabilities.

Eclipse/IDEA task types

Tasks contributed by IDE plugins. To configure IDE plugins please use IDE model types.

Type	Description
`GenerateEclipseProject`	Generates an Eclipse `.project` file. If you want to fine tune the eclipse configuration
`GenerateEclipseClasspath`	Generates an Eclipse `.classpath` file. If you want to fine tune the eclipse configuration
`GenerateEclipseJdt`	Generates the Eclipse JDT configuration file. If you want to fine tune the eclipse configuration
`GenerateEclipseWtpComponent`	Generates the org.eclipse.wst.common.component settings file for Eclipse WTP. If you want to fine tune the eclipse configuration
`GenerateEclipseWtpFacet`	Generates the org.eclipse.wst.common.project.facet.core settings file for Eclipse WTP. If you want to fine tune the eclipse configuration
`GenerateIdeaModule`	Generates an IDEA module file. If you want to fine tune the idea configuration

Type	Description
`GenerateIdeaProject`	Generates an IDEA project file for root project *only*. If you want to fine tune the idea configuration
`GenerateIdeaWorkspace`	Generates an IDEA workspace file *only* for root project. There's little you can configure about workspace generation at the moment.

Native software model types

Used to configure software components developed with native code.

Type	Description
`PrebuiltLibrary`	A library component that is not built by gradle.
`PrebuiltSharedLibraryBinary`	A shared library that exists at a known location on the filesystem.
`PrebuiltStaticLibraryBinary`	A static library that exists at a known location on the filesystem.
`NativeComponentSpec`	Definition of a software component that is to be built by Gradle to run a on JVM platform.
`NativeExecutableSpec`	Definition of a native executable component that is to be built by Gradle.

Type	Description
`NativeLibrarySpec`	Definition of a native library component that is to be built by Gradle.
`NativeTestSuiteSpec`	A component representing a suite of tests that will be executed together.
`CUnitTestSuiteSpec`	Test suite of CUnit tests.
`GoogleTestTestSuiteSpec`	Test suite of Google Test tests.
`NativeBinarySpec`	Represents a binary artifact that is the result of building a native component.
`NativeExecutableBinarySpec`	An binary built by Gradle for a native application.
`NativeLibraryBinarySpec`	Represents a binary artifact that is the result of building a native library component.
`SharedLibraryBinarySpec`	A shared library binary built by Gradle for a native library.
`StaticLibraryBinarySpec`	A static library binary built by Gradle for a native library.
`NativeTestSuiteBinarySpec`	An executable which runs a suite of tests.
`CUnitTestSuiteBinarySpec`	An executable which run a CUnit test suite.
`GoogleTestTestSuiteBinarySpec`	An executable which run a Google Test test suite.

Type	Description
`NativePlatform`	A target platform for building native binaries. Each target platform is given a name, and may optionally be given a specific `Architecture` and/or `OperatingSystem` to target.
`BuildType`	Specifies a build-type for a native binary. Common build types are 'debug' and 'release', but others may be defined.
`Flavor`	Defines a custom variant that differentiate a `NativeBinary`.
`Gcc`	The GNU GCC tool chain.
`Clang`	The Clang tool chain.
`VisualCpp`	The Visual C++ tool chain.
`AssemblerSourceSet`	A set of assembly language sources.
`CSourceSet`	A set of C source files.
`CppSourceSet`	A set of C++ source files.
`ObjectiveCSourceSet`	A set of Objective-C source files.
`ObjectiveCppSourceSet`	A set of Objective-C++ source files.
`WindowsResourceSet`	A set of Windows Resource definition files.
`VisualStudioProject`	A visual studio project, created from one or more `NativeBinary` instances.

Type	Description
`VisualStudioSolution`	A visual studio solution, representing one or more `NativeBinarySpec` instances from the same `NativeComponentSpec`.
`NativeExecutable`	An executable native component that is built by Gradle.
`NativeLibrary`	A library component that is built by a gradle project.
`NativeBinary`	Represents a particular binary artifact.
`NativeExecutableBinary`	A binary artifact for a `NativeExecutable`, targeted at a particular platform with specific configuration.
`SharedLibraryBinary`	A `NativeLibrary` that has been compiled and linked as a shared library.
`StaticLibraryBinary`	A `NativeLibrary` that has been compiled and archived into a static library.

Native binary task types

Tasks used to build native binaries.

Type	Description
`CppCompile`	Compiles C++ source files into object files.

Type	Description
`CCompile`	Compiles C source files into object files.
`Assemble`	Translates Assembly language source files into object files.
`ObjectiveCCompile`	Compiles Objective-C source files into object files.
`ObjectiveCppCompile`	Compiles Objective-C++ source files into object files.
`WindowsResourceCompile`	Compiles Windows Resource scripts into .res files.
`LinkExecutable`	Links a binary executable from object files and libraries.
`LinkSharedLibrary`	Links a binary shared library from object files and imported libraries.
`CreateStaticLibrary`	Assembles a static library from object files.
`InstallExecutable`	Installs an executable with it's dependent libraries so it can be easily executed.
`RunTestExecutable`	Runs a compiled and installed test executable.