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/)

# Testing Build Logic with TestKit

Contents

[Usage](#4d34og8)

[Functional testing with the Gradle runner](#2s8eyo1)

[Getting the plugin-under-test into the test build](#17dp8vu)

[Controlling the build environment](#3rdcrjn)

[The Gradle version used to test](#26in1rg)

[Debugging build logic](#lnxbz9)

[Testing with the Build Cache](#35nkun2)

The Gradle TestKit (a.k.a. just TestKit) is a library that aids in testing Gradle plugins and build logic generally. At this time, it is focused on *functional* testing. That is, testing build logic by exercising it as part of a programmatically executed build. Over time, the TestKit will likely expand to facilitate other kinds of tests.

[Usage](#4d34og8)

To use the TestKit, include the following in your plugin’s build:

[Example: Declaring the TestKit dependency](#1ksv4uv)

**build.gradle**

dependencies {  
 testCompile gradleTestKit()  
}

The gradleTestKit() encompasses the classes of the TestKit, as well as the [Gradle Tooling API client](http://docs.google.com/embedding.html#embedding). It does not include a version of [JUnit](http://junit.org), [TestNG](http://testng.org), or any other test execution framework. Such a dependency must be explicitly declared.

[Example: Declaring the JUnit dependency](#44sinio)

**build.gradle**

dependencies {  
 testCompile 'junit:junit:4.12'  
}

[Functional testing with the Gradle runner](#2s8eyo1)

The [GradleRunner](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html) facilitates programmatically executing Gradle builds, and inspecting the result.

A contrived build can be created (e.g. programmatically, or from a template) that exercises the “logic under test”. The build can then be executed, potentially in a variety of ways (e.g. different combinations of tasks and arguments). The correctness of the logic can then be verified by asserting the following, potentially in combination:

* The build’s output;
* The build’s logging (i.e. console output);
* The set of tasks executed by the build and their results (e.g. FAILED, UP-TO-DATE etc.).

After creating and configuring a runner instance, the build can be executed via the [GradleRunner.build()](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html#build--) or [GradleRunner.buildAndFail()](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html#buildAndFail--) methods depending on the anticipated outcome.

The following demonstrates the usage of Gradle runner in a Java JUnit test:

[Example: Using GradleRunner with JUnit](#2jxsxqh)

**BuildLogicFunctionalTest.java**

import org.gradle.testkit.runner.BuildResult;  
import org.gradle.testkit.runner.GradleRunner;  
import org.junit.Before;  
import org.junit.Rule;  
import org.junit.Test;  
import org.junit.rules.TemporaryFolder;  
  
import java.io.BufferedWriter;  
import java.io.File;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.Collections;  
  
import static org.junit.Assert.assertEquals;  
import static org.junit.Assert.assertTrue;  
  
import static org.gradle.testkit.runner.TaskOutcome.\*;  
  
public class BuildLogicFunctionalTest {  
 @Rule public final TemporaryFolder testProjectDir = new TemporaryFolder();  
 private File buildFile;  
  
 @Before  
 public void setup() throws IOException {  
 buildFile = testProjectDir.newFile("build.gradle");  
 }  
  
 @Test  
 public void testHelloWorldTask() throws IOException {  
 String buildFileContent = "task helloWorld {" +  
 " doLast {" +  
 " println 'Hello world!'" +  
 " }" +  
 "}";  
 writeFile(buildFile, buildFileContent);  
  
 BuildResult result = GradleRunner.create()  
 .withProjectDir(testProjectDir.getRoot())  
 .withArguments("helloWorld")  
 .build();  
  
 assertTrue(result.getOutput().contains("Hello world!"));  
 assertEquals(SUCCESS, result.task(":helloWorld").getOutcome());  
 }  
  
 private void writeFile(File destination, String content) throws IOException {  
 BufferedWriter output = null;  
 try {  
 output = new BufferedWriter(new FileWriter(destination));  
 output.write(content);  
 } finally {  
 if (output != null) {  
 output.close();  
 }  
 }  
 }  
}

Any test execution framework can be used.

As Gradle build scripts are written in the Groovy programming language, and as many plugins are implemented in Groovy, it is often a productive choice to write Gradle functional tests in Groovy. Furthermore, it is recommended to use the (Groovy based) [Spock test execution framework](https://code.google.com/p/spock/) as it offers many compelling features over the use of JUnit.

The following demonstrates the usage of Gradle runner in a Groovy Spock test:

[Example: Using GradleRunner with Spock](#z337ya)

**BuildLogicFunctionalTest.groovy**

import org.gradle.testkit.runner.GradleRunner  
import static org.gradle.testkit.runner.TaskOutcome.\*  
import org.junit.Rule  
import org.junit.rules.TemporaryFolder  
import spock.lang.Specification  
  
class BuildLogicFunctionalTest extends Specification {  
 @Rule final TemporaryFolder testProjectDir = new TemporaryFolder()  
 File buildFile  
  
 def setup() {  
 buildFile = testProjectDir.newFile('build.gradle')  
 }  
  
 def "hello world task prints hello world"() {  
 given:  
 buildFile << """  
 task helloWorld {  
 doLast {  
 println 'Hello world!'  
 }  
 }  
 """  
  
 when:  
 def result = GradleRunner.create()  
 .withProjectDir(testProjectDir.root)  
 .withArguments('helloWorld')  
 .build()  
  
 then:  
 result.output.contains('Hello world!')  
 result.task(":helloWorld").outcome == SUCCESS  
 }  
}

It is a common practice to implement any custom build logic (like plugins and task types) that is more complex in nature as external classes in a standalone project. The main driver behind this approach is bundle the compiled code into a JAR file, publish it to a binary repository and reuse it across various projects.

[Getting the plugin-under-test into the test build](#17dp8vu)

The GradleRunner uses the [Tooling API](http://docs.google.com/embedding.html#embedding) to execute builds. An implication of this is that the builds are executed in a separate process (i.e. not the same process executing the tests). Therefore, the test build does not share the same classpath or classloaders as the test process and the code under test is not implicitly available to the test build.

Starting with version 2.13, Gradle provides a [conventional mechanism to inject the code under test into the test build](#3j2qqm3).

For earlier versions of Gradle (before 2.13), it is possible to manually make the code under test available via some extra configuration. The following example demonstrates having the build generate a file containing the implementation classpath of the code under test, and making it available at test runtime.

[Example: Making the code under test classpath available to the tests](#1y810tw)

**build.gradle**

// Write the plugin's classpath to a file to share with the tests  
task createClasspathManifest {  
 def outputDir = file("$buildDir/$name")  
  
 inputs.files sourceSets.main.runtimeClasspath  
 outputs.dir outputDir  
  
 doLast {  
 outputDir.mkdirs()  
 file("$outputDir/plugin-classpath.txt").text = sourceSets.main.runtimeClasspath.join("\n")  
 }  
}  
  
// Add the classpath file to the test runtime classpath  
dependencies {  
 testRuntime files(createClasspathManifest)  
}

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

The tests can then read this value, and inject the classpath into the test build by using the method [GradleRunner.withPluginClasspath(java.lang.Iterable)](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html#withPluginClasspath-java.lang.Iterable-). This classpath is then available to use to locate plugins in a test build via the plugins DSL (see [Plugins](http://docs.google.com/plugins.html#plugins)). Applying plugins with the plugins DSL requires the definition of a plugin identifier. The following is an example (in Groovy) of doing this from within a Spock Framework setup() method, which is analogous to a JUnit @Before method.

[Example: Injecting the code under test classes into test builds](#4i7ojhp)

**src/test/groovy/org/gradle/sample/BuildLogicFunctionalTest.groovy**

List<File> pluginClasspath  
  
 def setup() {  
 buildFile = testProjectDir.newFile('build.gradle')  
  
 def pluginClasspathResource = getClass().classLoader.findResource("plugin-classpath.txt")  
 if (pluginClasspathResource == null) {  
 throw new IllegalStateException("Did not find plugin classpath resource, run `testClasses` build task.")  
 }  
  
 pluginClasspath = pluginClasspathResource.readLines().collect { new File(it) }  
 }  
  
 def "hello world task prints hello world"() {  
 given:  
 buildFile << """  
 plugins {  
 id 'org.gradle.sample.helloworld'  
 }  
 """  
  
 when:  
 def result = GradleRunner.create()  
 .withProjectDir(testProjectDir.root)  
 .withArguments('helloWorld')  
 .withPluginClasspath(pluginClasspath)  
 .build()  
  
 then:  
 result.output.contains('Hello world!')  
 result.task(":helloWorld").outcome == SUCCESS  
 }

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

This approach works well when executing the functional tests as part of the Gradle build. When executing the functional tests from an IDE, there are extra considerations. Namely, the classpath manifest file points to the class files etc. generated by Gradle and not the IDE. This means that after making a change to the source of the code under test, the source must be recompiled by Gradle. Similarly, if the effective classpath of the code under test changes, the manifest must be regenerated. In either case, executing the testClasses task of the build will ensure that things are up to date.

Some IDEs provide a convenience option to delegate the "test classpath generation and execution" to the build. In IntelliJ you can find this option under Preferences…​ > Build, Execution, Deployment > Build Tools > Gradle > Runner > Delegate IDE build/run actions to gradle. Please consult the documentation of your IDE for more information.

[Working with Gradle versions prior to 2.8](#2xcytpi)

The [GradleRunner.withPluginClasspath(java.lang.Iterable)](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html#withPluginClasspath-java.lang.Iterable-) method will not work when executing the build with a Gradle version earlier than 2.8 (see [The version used to test](#26in1rg)), as this feature is not supported on such Gradle versions.

Instead, the code must be injected via the build script itself. The following sample demonstrates how this can be done.

[Example: Injecting the code under test classes into test builds for Gradle versions prior to 2.8](#1ci93xb)

**src/test/groovy/org/gradle/sample/BuildLogicFunctionalTest.groovy**

List<File> pluginClasspath  
  
 def setup() {  
 buildFile = testProjectDir.newFile('build.gradle')  
  
 def pluginClasspathResource = getClass().classLoader.findResource("plugin-classpath.txt")  
 if (pluginClasspathResource == null) {  
 throw new IllegalStateException("Did not find plugin classpath resource, run `testClasses` build task.")  
 }  
  
 pluginClasspath = pluginClasspathResource.readLines().collect { new File(it) }  
 }  
  
 def "hello world task prints hello world with pre Gradle 2.8"() {  
 given:  
 def classpathString = pluginClasspath  
 .collect { it.absolutePath.replace('\\', '\\\\') } // escape backslashes in Windows paths  
 .collect { "'$it'" }  
 .join(", ")  
  
 buildFile << """  
 buildscript {  
 dependencies {  
 classpath files($classpathString)  
 }  
 }  
 apply plugin: "org.gradle.sample.helloworld"  
 """  
  
 when:  
 def result = GradleRunner.create()  
 .withProjectDir(testProjectDir.root)  
 .withArguments('helloWorld')  
 .withGradleVersion("2.7")  
 .build()  
  
 then:  
 result.output.contains('Hello world!')  
 result.task(":helloWorld").outcome == SUCCESS  
 }

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

[Automatic injection with the Java Gradle Plugin Development plugin](#3j2qqm3)

The [Java Gradle Plugin development plugin](http://docs.google.com/java_gradle_plugin.html#java_gradle_plugin) can be used to assist in the development of Gradle plugins. Starting with Gradle version 2.13, the plugin provides a direct integration with TestKit. When applied to a project, the plugin automatically adds the gradleTestKit() dependency to the test compile configuration. Furthermore, it automatically generates the classpath for the code under test and injects it via [GradleRunner.withPluginClasspath()](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html#withPluginClasspath--) for any GradleRunner instance created by the user. It’s important to note that the mechanism currently *only* works if the plugin under test is applied using the [plugins DSL](http://docs.google.com/plugins.html#sec:plugins_block). If the [target Gradle version](#26in1rg) is prior to 2.8, automatic plugin classpath injection is not performed.

The plugin uses the following conventions for applying the TestKit dependency and injecting the classpath:

* Source set containing code under test: sourceSets.main
* Source set used for injecting the plugin classpath: sourceSets.test

Any of these conventions can be reconfigured with the help of the class[GradlePluginDevelopmentExtension](http://docs.google.com/javadoc/org/gradle/plugin/devel/GradlePluginDevelopmentExtension.html).

The following Groovy-based sample demonstrates how to automatically inject the plugin classpath by using the standard conventions applied by the Java Gradle Plugin Development plugin.

[Example: Using the Java Gradle Development plugin for generating the plugin metadata](#3whwml4)

**build.gradle**

apply plugin: 'groovy'  
apply plugin: 'java-gradle-plugin'  
  
dependencies {  
 testCompile('org.spockframework:spock-core:1.1-groovy-2.4') {  
 exclude module: 'groovy-all'  
 }  
}

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

[Example: Automatically injecting the code under test classes into test builds](#2bn6wsx)

**src/test/groovy/org/gradle/sample/BuildLogicFunctionalTest.groovy**

def "hello world task prints hello world"() {  
 given:  
 buildFile << """  
 plugins {  
 id 'org.gradle.sample.helloworld'  
 }  
 """  
  
 when:  
 def result = GradleRunner.create()  
 .withProjectDir(testProjectDir.root)  
 .withArguments('helloWorld')  
 .withPluginClasspath()  
 .build()  
  
 then:  
 result.output.contains('Hello world!')  
 result.task(":helloWorld").outcome == SUCCESS  
}

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

The following build script demonstrates how to reconfigure the conventions provided by the Java Gradle Plugin Development plugin for a project that uses a custom Test source set.

[Example: Reconfiguring the classpath generation conventions of the Java Gradle Development plugin](#qsh70q)

**build.gradle**

apply plugin: 'groovy'  
apply plugin: 'java-gradle-plugin'  
  
sourceSets {  
 functionalTest {  
 groovy {  
 srcDir file('src/functionalTest/groovy')  
 }  
 resources {  
 srcDir file('src/functionalTest/resources')  
 }  
 compileClasspath += sourceSets.main.output + configurations.testRuntime  
 runtimeClasspath += output + compileClasspath  
 }  
}  
  
task functionalTest(type: Test) {  
 testClassesDirs = sourceSets.functionalTest.output.classesDirs  
 classpath = sourceSets.functionalTest.runtimeClasspath  
}  
  
check.dependsOn functionalTest  
  
gradlePlugin {  
 testSourceSets sourceSets.functionalTest  
}  
  
dependencies {  
 functionalTestCompile('org.spockframework:spock-core:1.1-groovy-2.4') {  
 exclude module: 'groovy-all'  
 }  
}

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

[Controlling the build environment](#3rdcrjn)

The runner executes the test builds in an isolated environment by specifying a dedicated "working directory" in a directory inside the JVM’s temp directory (i.e. the location specified by the java.io.tmpdir system property, typically /tmp). Any configuration in the default Gradle user home directory (e.g. ~/.gradle/gradle.properties) is not used for test execution. The TestKit does not expose a mechanism for fine grained control of environment variables etc. Future versions of the TestKit will provide improved configuration options.

The TestKit uses dedicated daemon processes that are automatically shut down after test execution.

[The Gradle version used to test](#26in1rg)

The Gradle runner requires a Gradle distribution in order to execute the build. The TestKit does not depend on all of Gradle’s implementation.

By default, the runner will attempt to find a Gradle distribution based on where the GradleRunner class was loaded from. That is, it is expected that the class was loaded from a Gradle distribution, as is the case when using the gradleTestKit() dependency declaration.

When using the runner as part of tests *being executed by Gradle* (e.g. executing the test task of a plugin project), the same distribution used to execute the tests will be used by the runner. When using the runner as part of tests *being executed by an IDE*, the same distribution of Gradle that was used when importing the project will be used. This means that the plugin will effectively be tested with the same version of Gradle that it is being built with.

Alternatively, a different and specific version of Gradle to use can be specified by the any of the following GradleRunner methods:

* [GradleRunner.withGradleVersion(java.lang.String)](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html#withGradleVersion-java.lang.String-)
* [GradleRunner.withGradleInstallation(java.io.File)](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html#withGradleInstallation-java.io.File-)
* [GradleRunner.withGradleDistribution(java.net.URI)](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html#withGradleDistribution-java.net.URI-)

This can potentially be used to test build logic across Gradle versions. The following demonstrates a cross-version compatibility test written as Groovy Spock test:

[Example: Specifying a Gradle version for test execution](#3as4poj)

**BuildLogicFunctionalTest.groovy**

import org.gradle.testkit.runner.GradleRunner  
import static org.gradle.testkit.runner.TaskOutcome.\*  
import org.junit.Rule  
import org.junit.rules.TemporaryFolder  
import spock.lang.Specification  
import spock.lang.Unroll  
  
class BuildLogicFunctionalTest extends Specification {  
 @Rule final TemporaryFolder testProjectDir = new TemporaryFolder()  
 File buildFile  
  
 def setup() {  
 buildFile = testProjectDir.newFile('build.gradle')  
 }  
  
 @Unroll  
 def "can execute hello world task with Gradle version #gradleVersion"() {  
 given:  
 buildFile << """  
 task helloWorld {  
 doLast {  
 logger.quiet 'Hello world!'  
 }  
 }  
 """  
  
 when:  
 def result = GradleRunner.create()  
 .withGradleVersion(gradleVersion)  
 .withProjectDir(testProjectDir.root)  
 .withArguments('helloWorld')  
 .build()  
  
 then:  
 result.output.contains('Hello world!')  
 result.task(":helloWorld").outcome == SUCCESS  
  
 where:  
 gradleVersion << ['2.6', '2.7']  
 }  
}

[Feature support when testing with different Gradle versions](#1pxezwc)

It is possible to use the GradleRunner to execute builds with Gradle 1.0 and later. However, some runner features are not supported on earlier versions. In such cases, the runner will throw an exception when attempting to use the feature.

The following table lists the features that are sensitive to the Gradle version being used.

Table 1. Gradle version compatibility

| **Feature** | **Minimum Version** | **Description** |
| --- | --- | --- |
| Inspecting executed tasks | 2.5 | Inspecting the executed tasks, using [BuildResult.getTasks()](http://docs.google.com/javadoc/org/gradle/testkit/runner/BuildResult.html#getTasks--) and similar methods. |
| [Plugin classpath injection](#17dp8vu) | 2.8 | Injecting the code under test via[GradleRunner.withPluginClasspath(java.lang.Iterable)](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html#withPluginClasspath-java.lang.Iterable-). |
| [Inspecting build output in debug mode](#lnxbz9) | 2.9 | Inspecting the build’s text output when run in debug mode, using [BuildResult.getOutput()](http://docs.google.com/javadoc/org/gradle/testkit/runner/BuildResult.html#getOutput--). |
| [Automatic plugin classpath injection](#3j2qqm3) | 2.13 | Injecting the code under test automatically via [GradleRunner.withPluginClasspath()](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html#withPluginClasspath--) by applying the Java Gradle Plugin Development plugin. |

[Debugging build logic](#lnxbz9)

The runner uses the [Tooling API](http://docs.google.com/embedding.html#embedding) to execute builds. An implication of this is that the builds are executed in a separate process (i.e. not the same process executing the tests). Therefore, executing your *tests* in debug mode does not allow you to debug your build logic as you may expect. Any breakpoints set in your IDE will be not be tripped by the code being exercised by the test build.

The TestKit provides two different ways to enable the debug mode:

* Setting “org.gradle.testkit.debug” system property to true for the JVM *using* the GradleRunner (i.e. not the build being executed with the runner);
* Calling the [GradleRunner.withDebug(boolean)](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html#withDebug-boolean-) method.

The system property approach can be used when it is desirable to enable debugging support without making an adhoc change to the runner configuration. Most IDEs offer the capability to set JVM system properties for test execution, and such a feature can be used to set this system property.

[Testing with the Build Cache](#35nkun2)

To enable the [Build Cache](http://docs.google.com/build_cache.html#build_cache) in your tests, you can pass the --build-cache argument to [GradleRunner](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html) or use one of the other methods described in [Enable the build cache](http://docs.google.com/build_cache.html#sec:build_cache_enable). You can then check for the task outcome [TaskOutcome.FROM\_CACHE](http://docs.google.com/javadoc/org/gradle/testkit/runner/TaskOutcome.html#FROM_CACHE) when your plugin’s custom task is cached. This outcome is only valid for Gradle 3.5 and newer.

[Example: Testing cacheable tasks](#49x2ik5)

**BuildLogicFunctionalTest.groovy**

def "cacheableTask is loaded from cache"() {  
 given:  
 buildFile << """  
 plugins {  
 id 'org.gradle.sample.helloworld'  
 }  
 """  
  
 when:  
 def result = runner()  
 .withArguments( '--build-cache', 'cacheableTask')  
 .build()  
  
 then:  
 result.task(":cacheableTask").outcome == SUCCESS  
  
 when:  
 new File(testProjectDir.root, 'build').deleteDir()  
 result = runner()  
 .withArguments( '--build-cache', 'cacheableTask')  
 .build()  
  
 then:  
 result.task(":cacheableTask").outcome == FROM\_CACHE  
}

Note that TestKit re-uses a Gradle user home between tests (see [GradleRunner.withTestKitDir(java.io.File)](http://docs.google.com/javadoc/org/gradle/testkit/runner/GradleRunner.html#withTestKitDir-java.io.File-)) which contains the default location for the local build cache. For testing with the build cache, the build cache directory should be cleaned between tests. The easiest way to accomplish this is to configure the local build cache to use a temporary directory.

[Example: Clean build cache between tests](#2p2csry)

**BuildLogicFunctionalTest.groovy**

@Rule final TemporaryFolder testProjectDir = new TemporaryFolder()  
File buildFile  
File localBuildCacheDirectory  
  
def setup() {  
 localBuildCacheDirectory = testProjectDir.newFolder('local-cache')  
 testProjectDir.newFile('settings.gradle') << """  
 buildCache {  
 local {  
 directory '${localBuildCacheDirectory.toURI()}'  
 }  
 }  
 """  
 buildFile = testProjectDir.newFile('build.gradle')  
}

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/)