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/)

# Writing Custom Task Classes

Contents

[Packaging a task class](#4d34og8)

[Writing a simple task class](#2s8eyo1)

[A standalone project](#17dp8vu)

[Incremental tasks](#3rdcrjn)

[Declaring and Using Command Line Options](#26in1rg)

[The Worker API](#lnxbz9)

[Re-using logic between task classes](#35nkun2)

Gradle supports two types of task. One such type is the simple task, where you define the task with an action closure. We have seen these in [Build Script Basics](http://docs.google.com/tutorial_using_tasks.html#tutorial_using_tasks). For this type of task, the action closure determines the behaviour of the task. This type of task is good for implementing one-off tasks in your build script.

The other type of task is the enhanced task, where the behaviour is built into the task, and the task provides some properties which you can use to configure the behaviour. We have seen these in [Authoring Tasks](http://docs.google.com/more_about_tasks.html#more_about_tasks). Most Gradle plugins use enhanced tasks. With enhanced tasks, you don’t need to implement the task behaviour as you do with simple tasks. You simply declare the task and configure the task using its properties. In this way, enhanced tasks let you reuse a piece of behaviour in many different places, possibly across different builds.

The behaviour and properties of an enhanced task is defined by the task’s class. When you declare an enhanced task, you specify the type, or class of the task.

Implementing your own custom task class in Gradle is easy. You can implement a custom task class in pretty much any language you like, provided it ends up compiled to bytecode. In our examples, we are going to use Groovy as the implementation language. Groovy, Java or Kotlin are all good choices as the language to use to implement a task class, as the Gradle API has been designed to work well with these languages. In general, a task implemented using Java or Kotlin, which are statically typed, will perform better than the same task implemented using Groovy.

[Packaging a task class](#4d34og8)

There are several places where you can put the source for the task class.

Build script

You can include the task class directly in the build script. This has the benefit that the task class is automatically compiled and included in the classpath of the build script without you having to do anything. However, the task class is not visible outside the build script, and so you cannot reuse the task class outside the build script it is defined in.

buildSrc project

You can put the source for the task class in the *rootProjectDir*/buildSrc/src/main/groovy directory. Gradle will take care of compiling and testing the task class and making it available on the classpath of the build script. The task class is visible to every build script used by the build. However, it is not visible outside the build, and so you cannot reuse the task class outside the build it is defined in. Using the buildSrc project approach separates the task declaration - that is, what the task should do - from the task implementation - that is, how the task does it.

See [Organizing Gradle Projects](http://docs.google.com/organizing_gradle_projects.html#organizing_gradle_projects) for more details about the buildSrc project.

Standalone project

You can create a separate project for your task class. This project produces and publishes a JAR which you can then use in multiple builds and share with others. Generally, this JAR might include some custom plugins, or bundle several related task classes into a single library. Or some combination of the two.

In our examples, we will start with the task class in the build script, to keep things simple. Then we will look at creating a standalone project.

[Writing a simple task class](#2s8eyo1)

To implement a custom task class, you extend [DefaultTask](http://docs.google.com/dsl/org.gradle.api.DefaultTask.html).

[Example: Defining a custom task](#1ksv4uv)

**build.gradle**

class GreetingTask extends DefaultTask {  
}

This task doesn’t do anything useful, so let’s add some behaviour. To do so, we add a method to the task and mark it with the [TaskAction](http://docs.google.com/javadoc/org/gradle/api/tasks/TaskAction.html) annotation. Gradle will call the method when the task executes. You don’t have to use a method to define the behaviour for the task. You could, for instance, call doFirst() or doLast() with a closure in the task constructor to add behaviour.

[Example: A hello world task](#44sinio)

**build.gradle**

class GreetingTask extends DefaultTask {  
 @TaskAction  
 def greet() {  
 println 'hello from GreetingTask'  
 }  
}  
  
// Create a task using the task type  
task hello(type: GreetingTask)

**Output of** gradle -q hello

> gradle -q hello  
hello from GreetingTask

Let’s add a property to the task, so we can customize it. Tasks are simply POGOs, and when you declare a task, you can set the properties or call methods on the task object. Here we add a greeting property, and set the value when we declare the greeting task.

[Example: A customizable hello world task](#2jxsxqh)

**build.gradle**

class GreetingTask extends DefaultTask {  
 String greeting = 'hello from GreetingTask'  
  
 @TaskAction  
 def greet() {  
 println greeting  
 }  
}  
  
// Use the default greeting  
task hello(type: GreetingTask)  
  
// Customize the greeting  
task greeting(type: GreetingTask) {  
 greeting = 'greetings from GreetingTask'  
}

**Output of** gradle -q hello greeting

> gradle -q hello greeting  
hello from GreetingTask  
greetings from GreetingTask

[A standalone project](#17dp8vu)

Now we will move our task to a standalone project, so we can publish it and share it with others. This project is simply a Groovy project that produces a JAR containing the task class. Here is a simple build script for the project. It applies the Groovy plugin, and adds the Gradle API as a compile-time dependency.

[Example: A build for a custom task](#z337ya)

**build.gradle**

plugins {  
 id 'groovy'  
}  
  
dependencies {  
 compile gradleApi()  
 compile localGroovy()  
}

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

We just follow the convention for where the source for the task class should go.

[Example: A custom task](#3j2qqm3)

**src/main/groovy/org/gradle/GreetingTask.groovy**

package org.gradle  
  
import org.gradle.api.DefaultTask  
import org.gradle.api.tasks.TaskAction  
  
class GreetingTask extends DefaultTask {  
 String greeting = 'hello from GreetingTask'  
  
 @TaskAction  
 def greet() {  
 println greeting  
 }  
}

[Using your task class in another project](#1y810tw)

To use a task class in a build script, you need to add the class to the build script’s classpath. To do this, you use a buildscript { } block, as described in [External dependencies for the build script](http://docs.google.com/tutorial_using_tasks.html#sec:build_script_external_dependencies). The following example shows how you might do this when the JAR containing the task class has been published to a local repository:

[Example: Using a custom task in another project](#4i7ojhp)

**build.gradle**

buildscript {  
 repositories {  
 maven {  
// END SNIPPET use-plugin  
// END SNIPPET use-task  
 def producerName = findProperty('producerName') ?: 'plugin'  
 def repoLocation = "../$producerName/build/repo"  
// START SNIPPET use-plugin  
// START SNIPPET use-task  
 url = uri(repoLocation)  
 }  
 }  
 dependencies {  
 classpath group: 'org.gradle', name: 'customPlugin',  
 version: '1.0-SNAPSHOT'  
 }  
}  
  
task greeting(type: org.gradle.GreetingTask) {  
 greeting = 'howdy!'  
}

[Writing tests for your task class](#2xcytpi)

You can use the [ProjectBuilder](http://docs.google.com/javadoc/org/gradle/testfixtures/ProjectBuilder.html) class to create [Project](http://docs.google.com/dsl/org.gradle.api.Project.html) instances to use when you test your task class.

[Example: Testing a custom task](#1ci93xb)

**src/test/groovy/org/gradle/GreetingTaskTest.groovy**

class GreetingTaskTest {  
 @Test  
 public void canAddTaskToProject() {  
 Project project = ProjectBuilder.builder().build()  
 def task = project.task('greeting', type: GreetingTask)  
 assertTrue(task instanceof GreetingTask)  
 }  
}

[Incremental tasks](#3rdcrjn)

| **✨** | Incremental tasks are an [incubating](http://docs.google.com/feature_lifecycle.html#feature_lifecycle) feature.  Since the introduction of the implementation described above (early in the Gradle 1.6 release cycle), discussions within the Gradle community have produced superior ideas for exposing the information about changes to task implementors to what is described below. As such, the API for this feature will almost certainly change in upcoming releases. However, please do experiment with the current implementation and share your experiences with the Gradle community.  The feature incubation process, which is part of the Gradle feature lifecycle (see [Feature Lifecycle](http://docs.google.com/feature_lifecycle.html#feature_lifecycle)), exists for this purpose of ensuring high quality final implementations through incorporation of early user feedback. |
| --- | --- |

With Gradle, it’s very simple to implement a task that is skipped when all of its inputs and outputs are up to date (see [Incremental Builds](http://docs.google.com/more_about_tasks.html#sec:up_to_date_checks)). However, there are times when only a few input files have changed since the last execution, and you’d like to avoid reprocessing all of the unchanged inputs. This can be particularly useful for a transformer task, that converts input files to output files on a 1:1 basis.

If you’d like to optimise your build so that only out-of-date inputs are processed, you can do so with an *incremental task*.

[Implementing an incremental task](#3whwml4)

For a task to process inputs incrementally, that task must contain an *incremental task action*. This is a task action method that contains a single [IncrementalTaskInputs](http://docs.google.com/javadoc/org/gradle/api/tasks/incremental/IncrementalTaskInputs.html) parameter, which indicates to Gradle that the action will process the changed inputs only.

The incremental task action may supply an [IncrementalTaskInputs.outOfDate(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.tasks.incremental.IncrementalTaskInputs.html#org.gradle.api.tasks.incremental.IncrementalTaskInputs:outOfDate(org.gradle.api.Action)) action for processing any input file that is out-of-date, and a [IncrementalTaskInputs.removed(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.tasks.incremental.IncrementalTaskInputs.html#org.gradle.api.tasks.incremental.IncrementalTaskInputs:removed(org.gradle.api.Action)) action that executes for any input file that has been removed since the previous execution.

[Example: Defining an incremental task action](#2bn6wsx)

**build.gradle**

class IncrementalReverseTask extends DefaultTask {  
 @InputDirectory  
 def File inputDir  
  
 @OutputDirectory  
 def File outputDir  
  
 @Input  
 def inputProperty  
  
 @TaskAction  
 void execute(IncrementalTaskInputs inputs) {  
 println inputs.incremental ? 'CHANGED inputs considered out of date'  
 : 'ALL inputs considered out of date'  
 if (!inputs.incremental)  
 project.delete(outputDir.listFiles())  
  
 inputs.outOfDate { change ->  
 println "out of date: ${change.file.name}"  
 def targetFile = new File(outputDir, change.file.name)  
 targetFile.text = change.file.text.reverse()  
 }  
  
 inputs.removed { change ->  
 println "removed: ${change.file.name}"  
 def targetFile = new File(outputDir, change.file.name)  
 targetFile.delete()  
 }  
 }  
}

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

If for some reason the task is not run incremental, e.g. by running with --rerun-tasks, only the outOfDate action is executed, even if there were deleted input files. You should consider handling this case at the beginning, as is done in the example above.

For a simple transformer task like this, the task action simply needs to generate output files for any out-of-date inputs, and delete output files for any removed inputs.

A task may only contain a single incremental task action.

[Which inputs are considered out of date?](#qsh70q)

When Gradle has history of a previous task execution, and the only changes to the task execution context since that execution are to input files, then Gradle is able to determine which input files need to be reprocessed by the task. In this case, the [IncrementalTaskInputs.outOfDate(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.tasks.incremental.IncrementalTaskInputs.html#org.gradle.api.tasks.incremental.IncrementalTaskInputs:outOfDate(org.gradle.api.Action)) action will be executed for any input file that was *added* or *modified*, and the [IncrementalTaskInputs.removed(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.tasks.incremental.IncrementalTaskInputs.html#org.gradle.api.tasks.incremental.IncrementalTaskInputs:removed(org.gradle.api.Action)) action will be executed for any *removed* input file.

However, there are many cases where Gradle is unable to determine which input files need to be reprocessed. Examples include:

* There is no history available from a previous execution.
* You are building with a different version of Gradle. Currently, Gradle does not use task history from a different version.
* An upToDateWhen criteria added to the task returns false.
* An input property has changed since the previous execution.
* One or more output files have changed since the previous execution.

In any of these cases, Gradle will consider all of the input files to be outOfDate. The [IncrementalTaskInputs.outOfDate(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.tasks.incremental.IncrementalTaskInputs.html#org.gradle.api.tasks.incremental.IncrementalTaskInputs:outOfDate(org.gradle.api.Action)) action will be executed for every input file, and the [IncrementalTaskInputs.removed(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.tasks.incremental.IncrementalTaskInputs.html#org.gradle.api.tasks.incremental.IncrementalTaskInputs:removed(org.gradle.api.Action)) action will not be executed at all.

You can check if Gradle was able to determine the incremental changes to input files with [IncrementalTaskInputs.isIncremental()](http://docs.google.com/dsl/org.gradle.api.tasks.incremental.IncrementalTaskInputs.html#org.gradle.api.tasks.incremental.IncrementalTaskInputs:incremental).

[An incremental task in action](#3as4poj)

Given the incremental task implementation [above](#2bn6wsx), we can explore the various change scenarios by example. Note that the various mutation tasks ('updateInputs', 'removeInput', etc) are only present for demonstration purposes: these would not normally be part of your build script.

First, consider the IncrementalReverseTask executed against a set of inputs for the first time. In this case, all inputs will be considered “out of date”:

[Example: Running the incremental task for the first time](#1pxezwc)

**build.gradle**

task incrementalReverse(type: IncrementalReverseTask) {  
 inputDir = file('inputs')  
 outputDir = file("$buildDir/outputs")  
 inputProperty = project.properties['taskInputProperty'] ?: 'original'  
}

**Build layout**

.  
├── build.gradle  
└── inputs  
 ├── 1.txt  
 ├── 2.txt  
 └── 3.txt

**Output of gradle -q incrementalReverse**

> gradle -q incrementalReverse  
ALL inputs considered out of date  
out of date: 1.txt  
out of date: 2.txt  
out of date: 3.txt

Naturally when the task is executed again with no changes, then the entire task is up to date and no files are reported to the task action:

[Example: Running the incremental task with unchanged inputs](#49x2ik5)

**Output of** gradle -q incrementalReverse

> gradle -q incrementalReverse

When an input file is modified in some way or a new input file is added, then re-executing the task results in those files being reported to [IncrementalTaskInputs.outOfDate(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.tasks.incremental.IncrementalTaskInputs.html#org.gradle.api.tasks.incremental.IncrementalTaskInputs:outOfDate(org.gradle.api.Action)):

[Example: Running the incremental task with updated input files](#2p2csry)

**build.gradle**

task updateInputs() {  
 doLast {  
 file('inputs/1.txt').text = 'Changed content for existing file 1.'  
 file('inputs/4.txt').text = 'Content for new file 4.'  
 }  
}

**Output of** gradle -q updateInputs incrementalReverse

> gradle -q updateInputs incrementalReverse  
CHANGED inputs considered out of date  
out of date: 1.txt  
out of date: 4.txt

When an existing input file is removed, then re-executing the task results in that file being reported to [IncrementalTaskInputs.removed(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.tasks.incremental.IncrementalTaskInputs.html#org.gradle.api.tasks.incremental.IncrementalTaskInputs:removed(org.gradle.api.Action)):

[Example: Running the incremental task with an input file removed](#147n2zr)

**build.gradle**

task removeInput() {  
 doLast {  
 file('inputs/3.txt').delete()  
 }  
}

**Output of** gradle -q removeInput incrementalReverse

> gradle -q removeInput incrementalReverse  
CHANGED inputs considered out of date  
removed: 3.txt

When an output file is deleted (or modified), then Gradle is unable to determine which input files are out of date. In this case, *all* input files are reported to the [IncrementalTaskInputs.outOfDate(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.tasks.incremental.IncrementalTaskInputs.html#org.gradle.api.tasks.incremental.IncrementalTaskInputs:outOfDate(org.gradle.api.Action)) action, and no input files are reported to the [IncrementalTaskInputs.removed(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.tasks.incremental.IncrementalTaskInputs.html#org.gradle.api.tasks.incremental.IncrementalTaskInputs:removed(org.gradle.api.Action)) action:

[Example: Running the incremental task with an output file removed](#3o7alnk)

**build.gradle**

task removeOutput() {  
 doLast {  
 file("$buildDir/outputs/1.txt").delete()  
 }  
}

**Output of** gradle -q removeOutput incrementalReverse

> gradle -q removeOutput incrementalReverse  
ALL inputs considered out of date  
out of date: 1.txt  
out of date: 2.txt  
out of date: 3.txt

When a task input property is modified, Gradle is unable to determine how this property impacted the task outputs, so all input files are assumed to be out of date. So similar to the changed output file example, *all* input files are reported to the [IncrementalTaskInputs.outOfDate(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.tasks.incremental.IncrementalTaskInputs.html#org.gradle.api.tasks.incremental.IncrementalTaskInputs:outOfDate(org.gradle.api.Action)) action, and no input files are reported to the [IncrementalTaskInputs.removed(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.tasks.incremental.IncrementalTaskInputs.html#org.gradle.api.tasks.incremental.IncrementalTaskInputs:removed(org.gradle.api.Action)) action:

[Example: Running the incremental task with an input property changed](#23ckvvd)

**Output of** gradle -q -PtaskInputProperty=changed incrementalReverse

> gradle -q -PtaskInputProperty=changed incrementalReverse  
ALL inputs considered out of date  
out of date: 1.txt  
out of date: 2.txt  
out of date: 3.txt

[Storing incremental state for cached tasks](#ihv636)

Using Gradle’s IncrementalTaskInputs is not the only way to create tasks that only works on changes since the last execution. Tools like the Kotlin compiler provide incrementality as a built-in feature. The way this is typically implemented is that the tool stores some analysis data about the state of the previous execution in some file. If such state files are [relocatable](http://docs.google.com/build_cache.html#sec:task_output_caching_inputs), then they can be declared as outputs of the task. This way when the task’s results are loaded from cache, the next execution can already use the analysis data loaded from cache, too.

However, if the state files are non-relocatable, then they can’t be shared via the build cache. Indeed, when the task is loaded from cache, any such state files must be cleaned up to prevent stale state to confuse the tool during the next execution. Gradle can ensure such stale files are removed if they are declared via task.localState.register() or a property is marked with the @LocalState annotation.

[Declaring and Using Command Line Options](#26in1rg)

| **✨** | The API for exposing command line options is an [incubating](http://docs.google.com/feature_lifecycle.html#feature_lifecycle) feature. |
| --- | --- |

Sometimes a user wants to declare the value of an exposed task property on the command line instead of the build script. Being able to pass in property values on the command line is particularly helpful if they change more frequently. The task API supports a mechanism for marking a property to automatically generate a corresponding command line parameter with a specific name at runtime.

[Declaring a command-line option](#32hioqz)

Exposing a new command line option for a task property is straightforward. You just have to annotate the corresponding setter method of a property with [Option](http://docs.google.com/javadoc/org/gradle/api/tasks/options/Option.html). An option requires a mandatory identifier. Additionally, you can provide an optional description. A task can expose as many command line options as properties available in the class.

Let’s have a look at an example to illustrate the functionality. The custom task UrlVerify verifies whether a given URL can be resolved by making a HTTP call and checking the response code. The URL to be verified is configurable through the property url. The setter method for the property is annotated with [Option](http://docs.google.com/javadoc/org/gradle/api/tasks/options/Option.html).

[Example: Declaring a command line option](#1hmsyys)

**UrlVerify.java**

import org.gradle.api.tasks.options.Option;  
  
public class UrlVerify extends DefaultTask {  
 private String url;  
  
 @Option(option = "url", description = "Configures the URL to be verified.")  
 public void setUrl(String url) {  
 this.url = url;  
 }  
  
 @Input  
 public String getUrl() {  
 return url;  
 }  
  
 @TaskAction  
 public void verify() {  
 getLogger().quiet("Verifying URL '{}'", url);  
  
 // verify URL by making a HTTP call  
 }  
}

All options declared for a task can be [rendered as console output](#41mghml) by running the help task and the --task option.

[Using an option on the command line](#2grqrue)

Using an option on the command line has to adhere to the following rules:

* The option uses a double-dash as prefix e.g. --url. A single dash does not qualify as valid syntax for a task option.
* The option argument follows directly after the task declaration e.g. verifyUrl --url=http://www.google.com/.
* Multiple options of a task can be declared in any order on the command line following the task name.

Getting back to the previous example, the build script creates a task instance of type UrlVerify and provides a value from the command line through the exposed option.

[Example: Using a command line option](#vx1227)

**build.gradle**

task verifyUrl(type: UrlVerify)

**Output of** gradle -q verifyUrl --url=http://www.google.com/

> gradle -q verifyUrl --url=http://www.google.com/  
Verifying URL 'http://www.google.com/'

[Supported data types for options](#3fwokq0)

Gradle limits the set of data types that can be used for declaring command line options. The use on the command line differ per type.

boolean, Boolean

Describes an option with the value true or false. Passing the option on the command line does not require assigning a value. For example --enabled equates to true. The absence of the option uses the default values assign to the property; that is false for boolean and null for the complex data type.

String

Describes an option with an arbitrary String value. Passing the option on the command line requires a key-value pair of option and value separated by an equals sign e.g. --containerId=2x94held.

enum

Describes an option as enum. The enum has to be passed on the command line as key-value pair similar to the String type e.g. --log-level=DEBUG. The provided value is not case sensitive.

List<String>, List<enum>

Describes an option that can takes multiple values of a given type. The values for the option have to be provided as distinct declarations e.g. --imageId=123 --imageId=456. Other notations like comma-separated lists or multiple values separated by a space character are currently not supported.

[Documenting available values for an option](#1v1yuxt)

In theory, an option for a property type String or List<String> can accept any arbitrary value. Expected values for such an option can be documented programmatically with the help of the annotation [OptionValues](http://docs.google.com/javadoc/org/gradle/api/tasks/options/OptionValues.html). This annotation may be assigned to any method that returns a List of one of the supported data types. In addition, you have to provide the option identifier to indicate the relationship between option and available values.

| **✨** | Passing a value on the command line that is not supported by the option does not fail the build or throw an exception. You’ll have to implement custom logic for such behavior in the task action. |
| --- | --- |

This example demonstrates the use of multiple options for a single task. The task implementation provides a list of available values for the option output-type.

[Example: Declaring available values for an option](#4f1mdlm)

**UrlProcess.java**

import org.gradle.api.tasks.options.Option;  
import org.gradle.api.tasks.options.OptionValues;  
  
public class UrlProcess extends DefaultTask {  
 private String url;  
 private OutputType outputType;  
  
 @Option(option = "url", description = "Configures the URL to be write to the output.")  
 public void setUrl(String url) {  
 this.url = url;  
 }  
  
 @Input  
 public String getUrl() {  
 return url;  
 }  
  
 @Option(option = "output-type", description = "Configures the output type.")  
 public void setOutputType(OutputType outputType) {  
 this.outputType = outputType;  
 }  
  
 @OptionValues("output-type")  
 public List<OutputType> getAvailableOutputTypes() {  
 return new ArrayList<OutputType>(Arrays.asList(OutputType.values()));  
 }  
  
 @Input  
 public OutputType getOutputType() {  
 return outputType;  
 }  
  
 @TaskAction  
 public void process() {  
 getLogger().quiet("Writing out the URL reponse from '{}' to '{}'", url, outputType);  
  
 // retrieve content from URL and write to output  
 }  
  
 private static enum OutputType {  
 CONSOLE, FILE  
 }  
}

[Listing command line options](#41mghml)

Command line options using the annotations [Option](http://docs.google.com/javadoc/org/gradle/api/tasks/options/Option.html) and [OptionValues](http://docs.google.com/javadoc/org/gradle/api/tasks/options/OptionValues.html) are self-documenting. You will see [declared options](#32hioqz) and their [available values](#1v1yuxt) reflected in the console output of the help task. The output renders options in alphabetical order.

[Example: Listing available values for option](#2u6wntf)

**Output of** gradle -q help --task processUrl

> gradle -q help --task processUrl  
Detailed task information for processUrl  
  
Path  
 :processUrl  
  
Type  
 UrlProcess (UrlProcess)  
  
Options  
 --output-type Configures the output type.  
 Available values are:  
 CONSOLE  
 FILE  
  
 --url Configures the URL to be write to the output.  
  
Description  
 -  
  
Group  
 -

[Limitations](#19c6y18)

Support for declaring command line options currently comes with a few limitations.

* Command line options can only be declared for custom tasks via annotation. There’s no programmatic equivalent for defining options.
* Options cannot be declared globally e.g. on a project-level or as part of a plugin.
* When assigning an option on the command line then the task exposing the option needs to be spelled out explicitly e.g. gradle check --tests abc does not work even though the check task depends on the test task.

[The Worker API](#lnxbz9)

| **✨** | The Worker API is an [incubating](http://docs.google.com/feature_lifecycle.html#feature_lifecycle) feature. |
| --- | --- |

As can be seen from the discussion of [incremental tasks](#3rdcrjn), the work that a task performs can be viewed as discrete units (i.e. a subset of inputs that are transformed to a certain subset of outputs). Many times, these units of work are highly independent of each other, meaning they can be performed in any order and simply aggregated together to form the overall action of the task. In a single threaded execution, these units of work would execute in sequence, however if we have multiple processors, it would be desirable to perform independent units of work concurrently. By doing so, we can fully utilize the available resources at build time and complete the activity of the task faster.

The Worker API provides a mechanism for doing exactly this. It allows for safe, concurrent execution of multiple items of work during a task action. But the benefits of the Worker API are not confined to parallelizing the work of a task. You can also configure a desired level of isolation such that work can be executed in an isolated classloader or even in an isolated process. Furthermore, the benefits extend beyond even the execution of a single task. Using the Worker API, Gradle can begin to execute tasks in parallel by default. In other words, once a task has submitted its work to be executed asynchronously, and has exited the task action, Gradle can then begin the execution of other independent tasks in parallel, even if those tasks are in the same project.

[Using the Worker API](#3tbugp1)

In order to submit work to the Worker API, two things must be provided: an implementation of the unit of work, and a configuration for the unit of work. The implementation is simply a class that extends java.lang.Runnable. This class should have a constructor that is annotated with javax.inject.Inject and accepts parameters that configure the class for a single unit of work. When a unit of work is submitted to the [WorkerExecutor](http://docs.google.com/javadoc/org/gradle/workers/WorkerExecutor.html), an instance of this class will be created and the parameters configured for the unit of work will be passed to the constructor.

[Example: Creating a unit of work implementation](#28h4qwu)

**build.gradle**

import org.gradle.workers.WorkerExecutor  
  
import javax.inject.Inject  
  
// The implementation of a single unit of work  
class ReverseFile implements Runnable {  
 File fileToReverse  
 File destinationFile  
  
 @Inject  
 public ReverseFile(File fileToReverse, File destinationFile) {  
 this.fileToReverse = fileToReverse  
 this.destinationFile = destinationFile  
 }  
  
 @Override  
 public void run() {  
 destinationFile.text = fileToReverse.text.reverse()  
 }  
}

The configuration of the worker is represented by a [WorkerConfiguration](http://docs.google.com/javadoc/org/gradle/workers/WorkerConfiguration.html) and is set by configuring an instance of this object at the time of submission. However, in order to submit the unit of work, it is necessary to first acquire the [WorkerExecutor](http://docs.google.com/javadoc/org/gradle/workers/WorkerExecutor.html). To do this, a constructor should be provided that is annotated with javax.inject.Inject and accepts a [WorkerExecutor](http://docs.google.com/javadoc/org/gradle/workers/WorkerExecutor.html) parameter. Gradle will inject the instance of [WorkerExecutor](http://docs.google.com/javadoc/org/gradle/workers/WorkerExecutor.html) at runtime when the task is created.

[Example: Submitting a unit of work for execution](#nmf14n)

**build.gradle**

class ReverseFiles extends SourceTask {  
 final WorkerExecutor workerExecutor  
  
 @OutputDirectory  
 File outputDir  
  
 // The WorkerExecutor will be injected by Gradle at runtime  
 @Inject  
 public ReverseFiles(WorkerExecutor workerExecutor) {  
 this.workerExecutor = workerExecutor  
 }  
  
 @TaskAction  
 void reverseFiles() {  
 // Create and submit a unit of work for each file  
 source.each { file ->  
 workerExecutor.submit(ReverseFile.class) { WorkerConfiguration config ->  
 // Use the minimum level of isolation  
 config.isolationMode = IsolationMode.NONE  
  
 // Constructor parameters for the unit of work implementation  
 config.params file, project.file("${outputDir}/${file.name}")  
 }  
 }  
 }  
}

Note that one element of the [WorkerConfiguration](http://docs.google.com/javadoc/org/gradle/workers/WorkerConfiguration.html) is the params property. These are the parameters passed to the constructor of the unit of work implementation for each item of work submitted. Any parameters provided to the unit of work *must* be java.io.Serializable.

Once all of the work for a task action has been submitted, it is safe to exit the task action. The work will be executed asynchronously and in parallel (up to the setting of max-workers). Of course, any tasks that are dependent on this task (and any subsequent task actions of this task) will not begin executing until all of the asynchronous work completes. However, other independent tasks that have no relationship to this task can begin executing immediately.

If any failures occur while executing the asynchronous work, the task will fail and a [WorkerExecutionException](http://docs.google.com/javadoc/org/gradle/workers/WorkerExecutionException.html) will be thrown detailing the failure for each failed work item. This will be treated like any failure during task execution and will prevent any dependent tasks from executing.

In some cases, however, it might be desirable to wait for work to complete before exiting the task action. This is possible using the [WorkerExecutor.await()](http://docs.google.com/javadoc/org/gradle/workers/WorkerExecutor.html#await--) method. As in the case of allowing the work to complete asynchronously, any failures that occur while executing an item of work will be surfaced as a [WorkerExecutionException](http://docs.google.com/javadoc/org/gradle/workers/WorkerExecutionException.html) thrown from the [WorkerExecutor.await()](http://docs.google.com/javadoc/org/gradle/workers/WorkerExecutor.html#await--) method.

| **✨** | Note that Gradle will only begin running other independent tasks in parallel when a task has exited a task action and returned control of execution to Gradle. When [WorkerExecutor.await()](http://docs.google.com/javadoc/org/gradle/workers/WorkerExecutor.html#await--) is used, execution does not leave the task action. This means that Gradle will not allow other tasks to begin executing and will wait for the task action to complete before doing so. |
| --- | --- |

[Example: Waiting for asynchronous work to complete](#37m2jsg)

**build.gradle**

// Create and submit a unit of work for each file  
source.each { file ->  
 workerExecutor.submit(ReverseFile.class) { config ->  
 config.isolationMode = IsolationMode.NONE  
 // Constructor parameters for the unit of work implementation  
 config.params file, project.file("${outputDir}/${file.name}")  
 }  
}  
  
// Wait for all asynchronous work to complete before continuing  
workerExecutor.await()  
logger.lifecycle("Created ${outputDir.listFiles().size()} reversed files in ${project.relativePath(outputDir)}")

[Isolation Modes](#1mrcu09)

Gradle provides three isolation modes that can be configured on a unit of work and are specified using the [IsolationMode](http://docs.google.com/javadoc/org/gradle/workers/IsolationMode.html) enum:

IsolationMode.NONE

This states that the work should be run in a thread with a minimum of isolation. For instance, it will share the same classloader that the task is loaded from. This is the fastest level of isolation.

IsolationMode.CLASSLOADER

This states that the work should be run in a thread with an isolated classloader. The classloader will have the classpath from the classloader that the unit of work implementation class was loaded from as well as any additional classpath entries added through [WorkerConfiguration.classpath(java.lang.Iterable)](http://docs.google.com/javadoc/org/gradle/workers/WorkerConfiguration.html#classpath-java.lang.Iterable-).

IsolationMode.PROCESS

This states that the work should be run with a maximum level of isolation by executing the work in a separate process. The classloader of the process will use the classpath from the classloader that the unit of work was loaded from as well as any additional classpath entries added through [WorkerConfiguration.classpath(java.lang.Iterable)](http://docs.google.com/javadoc/org/gradle/workers/WorkerConfiguration.html#classpath-java.lang.Iterable-). Furthermore, the process will be a *Worker Daemon* which will stay alive and can be reused for future work items that may have the same requirements. This process can be configured with different settings than the Gradle JVM using [WorkerConfiguration.forkOptions(org.gradle.api.Action)](http://docs.google.com/javadoc/org/gradle/workers/WorkerConfiguration.html#forkOptions-org.gradle.api.Action-).

[Worker Daemons](#46r0co2)

When using IsolationMode.PROCESS, gradle will start a long-lived *Worker Daemon* process that can be reused for future work items.

[Example: Submitting an item of work to run in a worker daemon](#2lwamvv)

**build.gradle**

workerExecutor.submit(ReverseFile.class) { WorkerConfiguration config ->  
 // Run this work in an isolated process  
 config.isolationMode = IsolationMode.PROCESS  
  
 // Configure the options for the forked process  
 config.forkOptions { JavaForkOptions options ->  
 options.maxHeapSize = "512m"  
 options.systemProperty "org.gradle.sample.showFileSize", "true"  
 }  
  
 // Constructor parameters for the unit of work implementation  
 config.params file, project.file("${outputDir}/${file.name}")  
}

When a unit of work for a Worker Daemon is submitted, Gradle will first look to see if a compatible, idle daemon already exists. If so, it will send the unit of work to the idle daemon, marking it as busy. If not, it will start a new daemon. When evaluating compatibility, Gradle looks at a number of criteria, all of which can be controlled through [WorkerConfiguration.forkOptions(org.gradle.api.Action)](http://docs.google.com/javadoc/org/gradle/workers/WorkerConfiguration.html#forkOptions-org.gradle.api.Action-).

executable

A daemon is considered compatible only if it uses the same java executable.

classpath

A daemon is considered compatible if its classpath contains all of the classpath entries requested. Note that a daemon is considered compatible if it has more classpath entries in addition to those requested.

heap settings

A daemon is considered compatible if it has at least the same heap size settings as requested. In other words, a daemon that has higher heap settings than requested would be considered compatible.

jvm arguments

A daemon is considered compatible if it has set all of the jvm arguments requested. Note that a daemon is considered compatible if it has additional jvm arguments beyond those requested (except for arguments treated specially such as heap settings, assertions, debug, etc).

system properties

A daemon is considered compatible if it has set all of the system properties requested with the same values. Note that a daemon is considered compatible if it has additional system properties beyond those requested.

environment variables

A daemon is considered compatible if it has set all of the environment variables requested with the same values. Note that a daemon is considered compatible if it has more environment variables in addition to those requested.

bootstrap classpath

A daemon is considered compatible if it contains all of the bootstrap classpath entries requested. Note that a daemon is considered compatible if it has more bootstrap classpath entries in addition to those requested.

debug

A daemon is considered compatible only if debug is set to the same value as requested (true or false).

enable assertions

A daemon is considered compatible only if enable assertions is set to the same value as requested (true or false).

default character encoding

A daemon is considered compatible only if the default character encoding is set to the same value as requested.

Worker daemons will remain running until either the build daemon that started them is stopped, or system memory becomes scarce. When available system memory is low, Gradle will begin stopping worker daemons in an attempt to minimize memory consumption.

[Re-using logic between task classes](#35nkun2)

There are different ways to re-use logic between task classes. The easiest case is when you can extract the logic you want to share in a separate method or class and then use the extracted piece of code in your tasks. For example, the [Copy](http://docs.google.com/dsl/org.gradle.api.tasks.Copy.html) task re-uses the logic of the [Project.copy(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.Project.html#org.gradle.api.Project:copy(org.gradle.api.Action)) method. Another option is to add a task dependency on the task which outputs you want to re-use. Other options include using [task rules](http://docs.google.com/more_about_tasks.html#sec:task_rules) or the [worker API](#lnxbz9).

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/)