Table of Contents

[I. Create new projects and initial configuration 2](#_Toc19694656)

[1. Create new projects with Eclipse 2](#_Toc19694657)

[2. Structure and Gradle Wrapper 3](#_Toc19694658)

[a. Structure 3](#_Toc19694659)

[b. Gradle Wrapper 4](#_Toc19694660)

[3. Initial configuration 5](#_Toc19694661)

[a. Network configuration 5](#_Toc19694662)

[b. Gradle wrapper configuration 5](#_Toc19694663)

[II. Task and Dependencies 6](#_Toc19694664)

[1. Plugins 7](#_Toc19694665)

[2. Repositories and dependencies 7](#_Toc19694666)

[a. Repositories 7](#_Toc19694667)

[b. Dependencies 8](#_Toc19694668)

[3. Task 10](#_Toc19694669)

[a. Test 11](#_Toc19694670)

[b. Copy 11](#_Toc19694671)

[c. Parallel 12](#_Toc19694672)

[d. Flexible in creating tasks 12](#_Toc19694673)

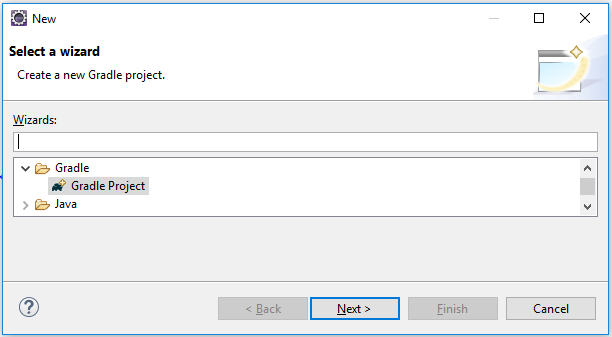
**USER GUIDE GRADLE**

# Create new projects and initial configuration

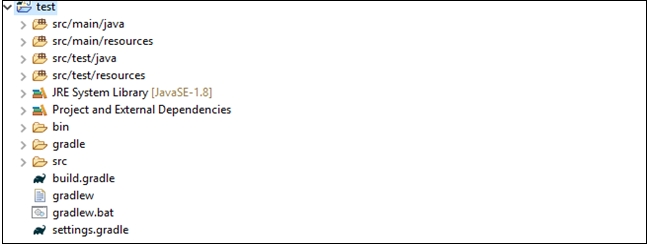
## Create new projects with Eclipse

Starting a project with Gradle in Eclipse perform the following steps:

From the Eclipse launch interface selected:

+ File > New > Other (Ctrl + N) --> New window will appear as shown below

+ Select: Gradle Project > Next > Next > “Name Project” > Finish --> Waiting for project Gradle complete. The project will appear as shown below.



## Structure and Gradle Wrapper

### Structure

Structure of Gradle project conclude:

+ src/test/java: Container all of the source material for building the project

+ src/test/resources:  The target classpath given the default resource definition

+ JRE System Library: all of java library

+ Project and External Dependencies: Show all dependencies for project

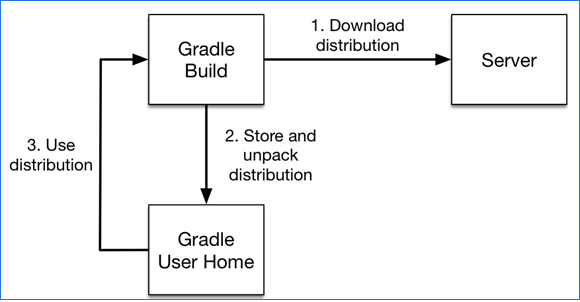
+ Gradle Wrapper: The Gradle Wrapper is an optional part of the Gradle build system. It consists of four files that you check into version control system.

* gradlew: Gradle start up script for UN\*X ([Unix-like](https://en.wikipedia.org/wiki/Unix-like) [operating system](https://en.wiktionary.org/wiki/operating_system) e.g., [Linux](https://en.wikipedia.org/wiki/Linux), [macOS](https://en.wikipedia.org/wiki/macOS" \o "wikipedia:macOS), [Solaris](https://en.wikipedia.org/wiki/Solaris), etc.)
* gradlew.bat: it’s a [batch file](https://en.wikipedia.org/wiki/Batch_file) used on Windows. You can even open it with a notepad to view the instructions in it. Batch files are like 'commands' written in a file to be executed. You use it (in case of Windows) to execute build commands. It also checks if gradle is installed or not. And in case it is not, it downloads and installs it.
* gralde/wrapper/gradle-wrapper.jar: The Wrapper JAR file containing code for downloading the Gradle distribution.
* gradle/wrapper/gradle-wrapper.properties: A properties file responsible for configuring the Wrapper runtime behavior e.g. the Gradle version compatible with this version.

+ Gradle configuration Tasks and Dependencies

* build.gradle: Task and dependencies configuration
* settings.gradle: Allow you to declare the projects which are to be included in the build.

### Gradle Wrapper

 Wrapper (in short just “Wrapper”). The Wrapper is a script that invokes a declared version of Gradle, downloading it beforehand if necessary. As a result, developers can get up and running with a Gradle project quickly.

## Initial configuration

### Network configuration

Gradle get library and dependencies from repo through internet. So we need set config http and https with proxy of PC Machine test.

Open file “gradlew.bat” (Window) or “gradlew” (UN\*X) and replace line:

set DEFAULT\_JVM\_OPTS="-Xmx64m" "-Xms64m"

replace:

set DEFAULT\_JVM\_OPTS=-Dhttp.proxyHost=<Your Host> -Dhttp.proxyPort=<Port> -Dhttps.proxyHost=<Your Host> -Dhttps.proxyPort=<Port>

ex:

set DEFAULT\_JVM\_OPTS=-Dhttp.proxyHost=10.10.10.10 -Dhttp.proxyPort=8080 -Dhttps.proxyHost=10.10.10.10 -Dhttps.proxyPort=8080

### Gradle wrapper configuration

Initial Gradle wrapper from gradle/wrapper/gradle-wrapper.properties has configuration as shown below:

distributionBase=GRADLE\_USER\_HOME

distributionPath=wrapper/dists

distributionUrl=https\://services.gradle.org/distributions/gradle-5.4-bin.zip

zipStoreBase=GRADLE\_USER\_HOME

zipStorePath=wrapper/dists

In case of impossible get .zip file from https address because of network issue. fix the error by: Check again Network configuration. If this is not resolved, the following method is available:

+ Manual download .zip from internet

+ Set new distributionUrl:

distributionUrl=file\:<link to local address of .zip file>

ex:

distributionUrl=file\:/d:/gradle/gradle-5.4-bin.zip

# Task and Dependencies

All Tasks and Dependencies of the project can be configure in build.gradle. Before configure Task and Dependencies need to consider the structure of initial build.gradle is include plugins, repositories, dependencies.

plugins {

// Apply the java-library plugin to add support for Java Library

id 'java-library'

}

repositories {

// Use jcenter for resolving your dependencies.

// You can declare any Maven/Ivy/file repository here.

jcenter()

}

dependencies {

// This dependency is exported to consumers, that is to say found on their compile classpath.

api 'org.apache.commons:commons-math3:3.6.1'

// This dependency is used internally, and not exposed to consumers on their own compile classpath.

implementation 'com.google.guava:guava:27.0.1-jre'

// Use JUnit test framework

testImplementation 'junit:junit:4.12'

}

## Plugins

Plugins mean the extensions that are installed into the project. Apply the library plugin support for project:

plugins {

// Apply the java-library plugin to add support for Java Library

id 'java'

id 'java-library'

id 'project-report'

id 'application'

}

Above are some essential plugins. In addition, many plugins can be added depending on the use case. Visit the link address to view the complete gradle plugin: <https://plugins.gradle.org/>

## Repositories and dependencies

Setup dependencies for project following two step: add repositories (Direct link to the library sources) and add dependencies

### Repositories

Can set Repo from local or from internet as shown below:

repositories {

mavenCentral() <https://mvnrepository.com/>

jcenter() <<https://bintray.com/bintray/jcenter>>

maven {url "link to repo"} <dependencies from repo>

flatDir { dirs 'libs'} <Local dependencies from folder libs>

}

Ex internet repo:

repositories{ maven{ url 'https://dl.google.com/dl/android/maven2' } }

### Dependencies

There are three methods of adding Dependencies: compile, api and implementation. Compile configuration was deprecated and should be replaced by implementation or api. In this article api and implementation is recommended. The difference of these three methods can be found here:

<https://docs.gradle.org/current/userguide/java_library_plugin.html#sec:java_library_separation>

<https://stackoverflow.com/questions/44493378/whats-the-difference-between-implementation-and-compile-in-gradle>

Structure of dependencies:

dependencies {

implementation 'com.google.guava:guava:27.0.1-jre' <compile or api>

testImplementation 'junit:junit:4.12'

implementation fileTree(include: ['\*.jar'], dir: 'libs')

}

Can use replace “implementation” = “api” or “compile”

“implementation” is the group of dependencies you need to build your application while “testImplementation” is a group of dependencies that you need only for testing. The better approach is to replace all compile dependencies with implementation dependencies. And only where you leak a module’s interface, you should use api. That should cause a lot less recompilation. The dependencies can found in website repo or more simply search with Google. Also to understand which one to use in which case, please refer here:

<https://docs.gradle.org/current/userguide/java_library_plugin.html>

+ Declaring a dependency with a dynamic version:

dependencies { implementation 'org.springframework:spring-web:5.+' }

+ [Declaring a file dependency](https://docs.gradle.org/current/userguide/declaring_dependencies.html#sec:declaring_file_dependency):

configurations {

antContrib

externalLibs

deploymentTools

}

dependencies {

antContrib files('ant/antcontrib.jar')

externalLibs files('libs/commons-lang.jar', 'libs/log4j.jar')

deploymentTools(fileTree('tools') { include '\*.exe' })

}

+ [Rich version declaration](https://docs.gradle.org/current/userguide/declaring_dependencies.html#sub:declaring_dependency_rich_version):

Gradle supports a rich model for declaring versions, which allows to combine different level of version information. The terms and their meaning are explained below, from the strongest to the weakest:

dependencies {

implementation('org.slf4j:slf4j-api') {

version {

strictly '[1.7, 1.8[' < 1.7 – 1.8 >

prefer '1.7.25' < select if exists >

}

}

constraints {

implementation('org.springframework:spring-core') {

version {

require '4.2.9.RELEASE' < not be lower >

reject '4.3.16.RELEASE' < not accepted >

}

}

}

}

## Task

Tasks are the cornerstone of getting things done in Gradle. They represent single atomic pieces of work within a build such as creating a JAR or linking an executable. This guide will walk you through the process of customising your build using small, tailored tasks. Because Gradle is an open programming language that depends on idea of the developer, there are many ways to deploy a task. Here are some sample tasks used for the tester.

**\*Before writing a new Tasks:**

+ Move all test scripts to src/test/java:



+ Configure settings.gradle:

rootProject.name = 'gradle'

include 'Gradle\_Automation', 'scripts.browsers'

or: You can leave it as default without mind\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

With rootProject.name: set name for root project and include: set package contain classes.

### Test

A sample Task gradle execute “CS\_1” classes in rootProject set on settings.gradle:

task CS\_1(type: Test, dependsOn: testClasses){

include "\*\*/\*CS\_1\*"

//exclude '\*\*/\*CS\_2\*' <ignore class CS\_2>

}

Configure all Task with type “Test” include create reports xml and html and set destination for each one. Configure testLogging show on command line and set continue run all Task if there is any error task.

tasks.withType(Test) {

ignoreFailures = true

testLogging {

events "started", "passed", "skipped", "failed"

}

reports {

junitXml.enabled = true

reports.junitXml.destination = file("$buildDir/test\_results/x")

html.enabled = true

reports.html.destination = file("$buildDir/htmlreport")

}

}

### Copy

Another Tasks with type “Copy”: Copies files into a destination directory. This task can also rename and filter files as it copies. Example in below perform Copy xml report from result Junit test.

task Copy(type: Copy){

from "$buildDir/test\_results/x"

include "\*.xml"

into "$buildDir/reportJunit"

}

### Parallel

Suppose there are three Classes that need to execute in parallel, to write a task to do that we perform the following:

task parallel(type: Test, dependsOn: testClasses){

include "\*\*/\*IE1\*"

include "\*\*/\*FF1\*"

include "\*\*/\*CH1\*"

maxParallelForks = Runtime.runtime.availableProcessors()

//maxParallelForks = Runtime.runtime.availableProcessors().intdiv(2) ?: 1

}

### Flexible in creating tasks

Declaring tasks that execute classes will be lengthy and time consuming. Assume we have five classes: IE\_01, IE\_02, IE\_03, IE\_04, IE\_05. To write the task to execute them as we normally do as follows:

task IE\_01(type: Test, dependsOn: testClasses){ include "\*\*/\*IE\_01\*" }

task IE\_02(type: Test, dependsOn: testClasses){ include "\*\*/\*IE\_02\*" }

task IE\_03(type: Test, dependsOn: testClasses){ include "\*\*/\*IE\_03\*" }

task IE\_04(type: Test, dependsOn: testClasses){ include "\*\*/\*IE\_04\*" }

task IE\_05(type: Test, dependsOn: testClasses){ include "\*\*/\*IE\_05\*" }

Another way of writing all these tasks:

6.times { counter ->

task "IE\_0$counter"(type: Test, dependsOn: testClasses){

include "\*\*/\*IE\_0$counter"+"\*"

}

}

Similar to the following case: We have classes: IE1, IE2, IE3, CH1, CH2, CH3, FF1, FF2, FF3. To write the task parallel to execute with: “Para1” execute IE1, CH1, FF1; “Para2” execute IE2, CH2, FF2; and “Para3” execute IE3, CH3, FF3. We do it as follows:

4.times { counter ->

task "Para$counter"(type: Test, dependsOn: testClasses){

include "\*\*/\*IE$counter"+"\*"

include "\*\*/\*CH$counter"+"\*"

include "\*\*/\*FF$counter"+"\*"

maxParallelForks = Runtime.runtime.availableProcessors()

//maxParallelForks = 3;

}

}

Another case we need to perform after each Task type: Test is a Task type: Copy. We do it as follows:

3.times { counter ->

task "P$counter"(type: Test, dependsOn: testClasses){

include "\*\*/\*Browser\_00$counter"+"\_IE\*"

include "\*\*/\*Browsers\_Firefox\_00$counter"+"\*"

include "\*\*/\*Browsers\_Chrome\_00$counter"+"\*"

maxParallelForks = Runtime.runtime.availableProcessors()

}

}

3.times{ counter ->

task "parallel$counter"(type: Copy){

dependsOn ":P$counter"

from "$buildDir/test\_results/x"

include "\*.xml"

into "$buildDir/reportJunit"

}

}

So when we execute parallel1 (2,3) with set **dependOn**: P1(2,3). The result is always do **dependOn** task before execute parallel.

\*The above are some examples of how to deploy tasks in Gradle. To be able to better use the Gradle, please refer to the information page of the manufacturer:

<https://docs.gradle.org/>