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/)

# Publishing

Contents

[Setting up basic publishing](#4d34og8)

[Adding custom artifacts to a publication](#2s8eyo1)

[Publishing a custom primary artifact (no component)](#17dp8vu)

[Signing artifacts](#3rdcrjn)

[Restricting publications to specific repositories](#26in1rg)

[Configuring publishing tasks](#lnxbz9)

[Terminology](#35nkun2)

The vast majority of software projects build something that aims to be consumed in some way. It could be a library that other software projects use or it could be an application for end users. *Publishing* is the process by which the thing being built is made available to consumers.

In Gradle, that process looks like this:

1. Define [what](#1ksv4uv) to publish
2. Define [where](#44sinio) to publish it to
3. [Do](#2jxsxqh) the publishing

Each of the these steps is dependent on the type of repository to which you want to publish artifacts. The two most common types are Maven-compatible and Ivy-compatible repositories, or Maven and Ivy repositories for short.

| **✨** | Looking for information on upload tasks and the archives configuration? See the [Legacy Publishing](http://docs.google.com/artifact_management.html#artifact_management) chapter. |
| --- | --- |

Gradle makes it easy to publish to these types of repository by providing some prepackaged infrastructure in the form of the [Maven Publish Plugin](http://docs.google.com/publishing_maven.html#publishing_maven) and the [Ivy Publish Plugin](http://docs.google.com/publishing_ivy.html#publishing_ivy). These plugins allow you to configure what to publish and perform the publishing with a minimum of effort.



*Figure 1. The publishing process*

Let’s take a look at those steps in more detail:

What to publish

Gradle needs to know what files and information to publish so that consumers can use your project. This is typically a combination of [artifacts](#z337ya) and metadata that Gradle calls a [publication](#3j2qqm3). Exactly what a publication contains depends on the type of repository it’s being published to.

For example, a publication destined for a Maven repository includes one or more artifacts — typically built by the project — plus a POM file describing the primary artifact and its dependencies. The primary artifact is typically the project’s production JAR and secondary artifacts might consist of "-sources" and "-javadoc" JARs.

Where to publish

Gradle needs to know where to publish artifacts so that consumers can get hold of them. This is done via [repositories](http://docs.google.com/dependency_management_terminology.html#sub:terminology_repository), which store and make available all sorts of artifact. Gradle also needs to interact with the repository, which is why you must provide the type of the repository and its location.

How to publish

Gradle automatically generates publishing tasks for all possible combinations of publication and repository, allowing you to publish any artifact to any repository. If you’re publishing to a Maven repository, the tasks are of type [PublishToMavenRepository](http://docs.google.com/dsl/org.gradle.api.publish.maven.tasks.PublishToMavenRepository.html), while for Ivy repositories the tasks are of type [PublishToIvyRepository](http://docs.google.com/dsl/org.gradle.api.publish.ivy.tasks.PublishToIvyRepository.html).

What follows is a practical example that demonstrates the entire publishing process.

[Setting up basic publishing](#4d34og8)

The first step in publishing, irrespective of your project type, is to apply the appropriate publishing plugin. As mentioned in the introduction, Gradle supports both Maven and Ivy repositories via the following plugins:

* [Maven Publish Plugin](http://docs.google.com/publishing_maven.html#publishing_maven)
* [Ivy Publish Plugin](http://docs.google.com/publishing_ivy.html#publishing_ivy)

These provide the specific publication and repository classes needed to configure publishing for the corresponding repository type. Since Maven repositories are the most commonly used ones, they will be the basis for this example and for the other samples in the chapter. Don’t worry, we will explain how to adjust individual samples for Ivy repositories.

Let’s assume we’re working with a simple Java library project, so only the following plugins are applied:

[Example: Applying the necessary plugins](#1y810tw)

**build.gradle**

plugins {  
 id 'java-library'  
 id 'maven-publish'  
}

Once the appropriate plugin has been applied, you can configure the publications and repositories. For this example, we want to publish the project’s production JAR file — the one produced by the jar task — to a custom, Maven repository. We do that with the following publishing {} block, which is backed by [PublishingExtension](http://docs.google.com/dsl/org.gradle.api.publish.PublishingExtension.html):

[Example: Configuring a Java library for publishing](#4i7ojhp)

**build.gradle**

group = 'org.example'  
version = '1.0'  
  
publishing {  
 publications {  
 myLibrary(MavenPublication) {  
 from components.java  
 }  
 }  
  
 repositories {  
 maven {  
 name = 'myRepo'  
 url = "file://${buildDir}/repo"  
 }  
 }  
}

This defines a publication called "myLibrary" that can be published to a Maven repository by virtue of its type: [MavenPublication](http://docs.google.com/dsl/org.gradle.api.publish.maven.MavenPublication.html). This publication consists of just the production JAR artifact and its metadata, which combined are represented by the java [component](#2xcytpi) of the project.

| **✨** | Components are the standard way of defining a publication. They are provided by plugins, usually of the language or platform variety. For example, the Java Plugin defines the components.java [SoftwareComponent](http://docs.google.com/javadoc/org/gradle/api/component/SoftwareComponent.html), while the War Plugin defines components.web. |
| --- | --- |

The example also defines a file-based Maven repository with the name "myRepo". Such a file-based repository is convenient for a sample, but real-world builds typically work with HTTPS-based repository servers, such as Maven Central or an internal company server.

| **✨** | You may define one, and only one, repository without a name. This translates to an implicit name of "Maven" for Maven repositories and "Ivy" for Ivy repositories. All other repository definitions must be given an explicit name. |
| --- | --- |

In combination with the project’s group and version, the publication and repository definitions provide everything that Gradle needs to publish the project’s production JAR. Gradle will then create a dedicated publishMyLibraryPublicationToMyRepoRepository task that does just that. Its name is based on the template publish*PubName*PublicationTo*RepoName*Repository. See the appropriate publishing plugin’s documentation for more details on the nature of this task and any other tasks that may be available to you.

You can either execute the individual publishing tasks directly, or you can execute publish, which will run all the available publishing tasks. In this example, publish will just run publishMyLibraryPublicationToMavenRepository.

| **✨** | Basic publishing to an Ivy repository is very similar: you simply use the Ivy Publish Plugin, replace MavenPublication with [IvyPublication](http://docs.google.com/dsl/org.gradle.api.publish.ivy.IvyPublication.html), and use ivy instead of maven in the repository definition.  There are differences between the two types of repository, particularly around the extra metadata that each support — for example, Maven repositories require a POM file while Ivy ones have their own metadata format — so see the plugin chapters for comprehensive information on how to configure both publications and repositories for whichever repository type you’re working with. |
| --- | --- |

That’s everything for the basic use case. However, many projects need more control over what gets published, so we look at several common scenarios in the following sections.

[Adding custom artifacts to a publication](#2s8eyo1)

Users often need to include additional artifacts with a publication, one of the most common examples being that of "-sources" and "-javadoc" JARs for JVM libraries. This is easy to do for both Maven- and Ivy-compatible repositories via the artifact configuration.

The following sample configures "-sources" and "-javadoc" JARs for a Java project and attaches them to the main (Maven) publication, i.e. the production JAR:

[Example: Adding an additional archive artifact to a MavenPublication](#1ci93xb)

**build.gradle**

task sourcesJar(type: Jar) {  
 classifier = 'sources'  
 from sourceSets.main.allJava  
}  
  
task javadocJar(type: Jar) {  
 classifier = 'javadoc'  
 from javadoc.destinationDir  
}  
  
publishing {  
 publications {  
 mavenJava(MavenPublication) {  
 from components.java  
  
 artifact sourcesJar  
 artifact javadocJar  
 }  
 }  
}

There are several important things to note about the sample:

* The artifact() method accepts archive tasks as an argument — like sourcesJar in the sample — as well as any type of argument accepted by [Project.file(java.lang.Object)](http://docs.google.com/dsl/org.gradle.api.Project.html#org.gradle.api.Project:file(java.lang.Object)), such as a File instance or string file path.
* Publishing plugins support different artifact configuration properties, so always check the plugin documentation for more details. The classifier and extension properties are supported by both the [Maven Publish Plugin](http://docs.google.com/publishing_maven.html#publishing_maven) and the [Ivy Publish Plugin](http://docs.google.com/publishing_ivy.html#publishing_ivy).
* Custom artifacts need to be distinct within a publication, typically via a unique combination of classifier and extension. See the documentation for the plugin you’re using for the precise requirements.
* If you use artifact() with an archive task, Gradle automatically populates the artifact’s metadata with the classifier and extension properties from that task. That’s why the above sample does not specify those properties in the artifact configurations.

When you’re attaching extra artifacts to a publication, remember that they are *secondary* artifacts that support a *primary* artifact. The metadata that a publication defines — such as dependency information — is associated with that primary artifact only. Thinking about publications in this way should help you determine whether you should be adding custom artifacts to an existing publication, or defining a new publication.

[Publishing a custom primary artifact (no component)](#17dp8vu)

If your build produces a primary artifact that isn’t supported by a predefined component, then you will need to configure a custom artifact. This isn’t much different to adding a custom artifact to an existing publication. There are just a couple of extra considerations:

* You may want to make the artifact available to other projects in the build
* You will need to manually construct the necessary metadata for publishing

Inter-project dependencies have nothing to do with publishing, but both features typically apply to the same set of artifacts in a Gradle project. So how do you tie them together?

You start by defining a custom artifact and attaching it to a Gradle [configuration](#3whwml4) of your choice. The following sample defines an RPM artifact that is produced by an rpm task (not shown) and attaches that artifact to the archives configuration:

[Example: Defining a custom artifact for a configuration](#2bn6wsx)

**build.gradle**

def rpmFile = file("$buildDir/rpms/my-package.rpm")  
def rpmArtifact = artifacts.add('archives', rpmFile) {  
 type 'rpm'  
 builtBy 'rpm'  
}

The artifacts.add() method — from [ArtifactHandler](http://docs.google.com/dsl/org.gradle.api.artifacts.dsl.ArtifactHandler.html) — returns an artifact object of type [PublishArtifact](http://docs.google.com/javadoc/org/gradle/api/artifacts/PublishArtifact.html) that can then be used in defining a publication, as shown in the following sample:

[Example: Attaching a custom PublishArtifact to a publication](#qsh70q)

**build.gradle**

publishing {  
 publications {  
 maven(MavenPublication) {  
 artifact rpmArtifact  
 }  
 }  
}

Now you can publish the RPM as well as depend on it from another project using the project(path: ':my-project', configuration: 'archives') syntax.

| **✨** | There is currently no easy way to define dependency information for a custom artifact. |
| --- | --- |

The groupId and artifactId properties are specific to Maven publications. See [IvyPublication](http://docs.google.com/dsl/org.gradle.api.publish.ivy.IvyPublication.html) for the relevant Ivy properties.

[Signing artifacts](#3rdcrjn)

The [Signing Plugin](http://docs.google.com/signing_plugin.html#signing_plugin) can be used to sign all artifacts and metadata files that make up a publication, including Maven POM files and Ivy module desciptors. In order to use it:

1. Apply the Signing Plugin
2. Configure the [signatory credentials](http://docs.google.com/signing_plugin.html#sec:signatory_credentials) — follow the link to see how
3. Specify the publications you want signed

Here’s an example that configures the plugin to sign the mavenJava publication:

[Example: Signing a publication](#3as4poj)

**build.gradle**

signing {  
 sign publishing.publications.mavenJava  
}

This will create a Sign task for each publication you specify and wire all publish*PubName*PublicationTo*RepoName*Repository tasks to depend on it. Thus, publishing any publication will automatically create and publish the signatures for its artifacts and metadata, as you can see from this output:

[Example: Sign and publish a project](#1pxezwc)

**Output of gradle publish**

> gradle publish  
> Task :generatePomFileForMavenJavaPublication  
> Task :compileJava  
> Task :processResources  
> Task :classes  
> Task :jar  
> Task :javadoc  
> Task :javadocJar  
> Task :sourcesJar  
> Task :signMavenJavaPublication  
> Task :publishMavenJavaPublicationToMavenRepository  
> Task :publish  
  
BUILD SUCCESSFUL in 0s  
9 actionable tasks: 9 executed

[Restricting publications to specific repositories](#26in1rg)

When you have defined multiple publications or repositories, you often want to control which publications are published to which repositories. For instance, consider the following sample that defines two publications — one that consists of just a binary and another that contains the binary and associated sources — and two repositories — one for internal use and one for external consumers:

[Example: Adding multiple publications and repositories](#49x2ik5)

**build.gradle**

publishing {  
 publications {  
 binary(MavenPublication) {  
 from components.java  
 }  
 binaryAndSources(MavenPublication) {  
 from components.java  
 artifact sourcesJar  
 }  
 }  
 repositories {  
 // change URLs to point to your repos, e.g. http://my.org/repo  
 maven {  
 name = 'external'  
 url = "$buildDir/repos/external"  
 }  
 maven {  
 name = 'internal'  
 url = "$buildDir/repos/internal"  
 }  
 }  
}

The publishing plugins will create tasks that allow you to publish either of the publications to either repository. They also attach those tasks to the publish aggregate task. But let’s say you want to restrict the binary-only publication to the external repository and the binary-with-sources publication to the internal one. To do that, you need to make the publishing *conditional*.

Gradle allows you to skip any task you want based on a condition via the [Task.onlyIf(org.gradle.api.specs.Spec)](http://docs.google.com/dsl/org.gradle.api.Task.html#org.gradle.api.Task:onlyIf(org.gradle.api.specs.Spec)) method. The following sample demonstrates how to implement the constraints we just mentioned:

[Example: Configuring which artifacts should be published to which repositories](#2p2csry)

**build.gradle**

tasks.withType(PublishToMavenRepository) {  
 onlyIf {  
 (repository == publishing.repositories.external &&  
 publication == publishing.publications.binary) ||  
 (repository == publishing.repositories.internal &&  
 publication == publishing.publications.binaryAndSources)  
 }  
}  
tasks.withType(PublishToMavenLocal) {  
 onlyIf {  
 publication == publishing.publications.binaryAndSources  
 }  
}

**Output of gradle publish**

> gradle publish  
> Task :generatePomFileForBinaryAndSourcesPublication  
> Task :compileJava  
> Task :processResources  
> Task :classes  
> Task :jar  
> Task :sourcesJar  
> Task :publishBinaryAndSourcesPublicationToExternalRepository SKIPPED  
> Task :publishBinaryAndSourcesPublicationToInternalRepository  
> Task :generatePomFileForBinaryPublication  
> Task :publishBinaryPublicationToExternalRepository  
> Task :publishBinaryPublicationToInternalRepository SKIPPED  
> Task :publish  
  
BUILD SUCCESSFUL in 0s  
8 actionable tasks: 8 executed

You may also want to define your own aggregate tasks to help with your workflow. For example, imagine that you have several publications that should be published to the external repository. It could be very useful to publish all of them in one go without publishing the internal ones.

The following sample demonstrates how you can do this by defining an aggregate task — publishToExternalRepository — that depends on all the relevant publish tasks:

[Example: Defining your own shorthand tasks for publishing](#147n2zr)

**build.gradle**

task publishToExternalRepository {  
 group = 'publishing'  
 description = 'Publishes all Maven publications to the external Maven repository.'  
 dependsOn tasks.withType(PublishToMavenRepository).matching {  
 it.repository == publishing.repositories.external  
 }  
}

This particular sample automatically handles the introduction or removal of the relevant publishing tasks by using [TaskCollection.withType(java.lang.Class)](http://docs.google.com/javadoc/org/gradle/api/tasks/TaskCollection.html#withType-java.lang.Class-) with the [PublishToMavenRepository](http://docs.google.com/dsl/org.gradle.api.publish.maven.tasks.PublishToMavenRepository.html) task type. You can do the same with [PublishToIvyRepository](http://docs.google.com/dsl/org.gradle.api.publish.ivy.tasks.PublishToIvyRepository.html) if you’re publishing to Ivy-compatible repositories.

[Configuring publishing tasks](#lnxbz9)

The publishing plugins create their non-aggregate tasks after the project has been evaluated, which means you cannot directly reference them from your build script. If you would like to configure any of these tasks, you should use deferred task configuration. This can be done in a number of ways via the project’s tasks collection.

For example, imagine you want to change where the generatePomFileFor*PubName*Publication tasks write their POM files. You can do this by using the [TaskCollection.withType(java.lang.Class)](http://docs.google.com/javadoc/org/gradle/api/tasks/TaskCollection.html#withType-java.lang.Class-) method, as demonstrated by this sample:

[Example: Configuring a dynamically named task created by the publishing plugins](#3o7alnk)

**build.gradle**

tasks.withType(GenerateMavenPom).all {  
 def matcher = name =~ /generatePomFileFor(\w+)Publication/  
 def publicationName = matcher[0][1]  
 destination = "$buildDir/poms/${publicationName}-pom.xml"  
}

The above sample uses a regular expression to extract the name of the publication from the name of the task. This is so that there is no conflict between the file paths of all the POM files that might be generated. If you only have one publication, then you don’t have to worry about such conflicts since there will only be one POM file.

[Terminology](#35nkun2)

Artifact

A file or directory produced by a build, such as a JAR, a ZIP distribution, or a native executable.

Artifacts are typically designed to be used or consumed by users or other projects, or deployed to hosting systems. In such cases, the artifact is a single file. Directories are common in the case of inter-project dependencies to avoid the cost of producing the publishable artifact.

Component

Any single version of a [module](http://docs.google.com/dependency_management_terminology.html#sub:terminology_module).

Components are defined by plugins and provide a simple way to define a publication for publishing. They comprise one or more [artifacts](#z337ya) as well as the appropriate metadata. For example, the java component consists of the production JAR — produced by the jar task — and its dependency information.

Configuration

A named collection of [dependencies](http://docs.google.com/dependency_management_terminology.html#sub:terminology_configuration) or [artifacts](#z337ya).

Gradle’s configurations can be somewhat confusing because they apply to both dependencies and artifacts. The main difference is that dependencies are consumed by the project, while artifacts are produced by it. Even then, the artifacts produced by a project are often consumed as dependencies by other projects.

Configurations allow different aspects of the build to work with known subsets of a project’s dependencies or artifacts, e.g. the dependencies required for compilation, or the artifacts related to a project’s API.

Publication

A description of the files and metadata that should be published to a repository as a single entity for use by consumers.

A publication has a name and consists of one or more artifacts plus information about those artifacts. The nature of that information depends on what type of repository you publish the publication to. In the case of Maven, the information takes the form of a POM.

One thing to bear in mind is that Maven repositories only allow a single *primary* artifact, i.e. one with metadata, but they do allow *secondary* artifacts such as packages of the associated source files and documentation ("-sources" and "-javadoc" JARs in the Java world).

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/)