Gradle groovy version

Gradle kotlin version

The productFlavors represent different versions of your application that you might want to build and distribute (like free, paid, different regions, etc.).

In Android development, **product flavors** and **build types** are used together to create multiple versions of the single app (also known as variants) for different purposes, such as testing or releasing to different markets. Each combination of a **product flavor** and a **build type** generates a unique build variant.

**Product Flavors** are used to define different versions of the app. For example, you might have a free version with ads and a paid version without ads.

**Build Types** define different configurations for building your app, like enabling/disabling debugging tools or optimizing the code for release.

When combined, each flavor and build type creates a unique **build variant**. So, if you have two flavors (paid, free) and two build types (alpha, beta), it will generate **four build variants**:

1. freeAlpha
2. freeBeta
3. paidAlpha
4. paidBeta

Each variant has unique properties, such as different app IDs, version name suffixes, or even different resources.

flavorDimensions.addAll(listOf("version"))

is used in Android's Kotlin DSL to specify the **dimensions** for the product flavors in your project. A **flavor dimension** is a way of grouping different product flavors, and it helps you organize your flavors into meaningful categories when you have multiple variations of your app.

**Why Use flavorDimensions?**

The flavorDimensions property allows you to define **multiple dimensions** when you have different types of variations for your app. You might need multiple dimensions if you want to organize your product flavors into different categories, like:

1. **Version-based variations**: Free vs. Paid.
2. **Environment-based variations**: Development, Staging, Production.
3. **Localization-based variations**: Different regions like US, EU, Asia.

If you want to have **version-based** and **environment-based** variations, you would add more dimensions:

flavorDimensions.addAll(listOf("version", "environment"))

To restrict a dependency like debugImplementation(libs.androidx.ui.test.manifest) to only certain build variants, such as excluding it from the **alpha** variant, you need to control which build types or flavors the dependency is applied to. In your case, you can achieve this by using Gradle's ability to customize dependencies based on specific build types and product flavors.

Gradle generates build variants based on the **product flavors** and **build types** combinations,

There are 2 versions of gradle now  
groovy and dsl

To change groovy to DSl

First step should be change to file extension. App level gradle and project level gradle file.

Kotlin dsl will take each strings in “” quates

We will wrap plugins in id(“ ”)

And kotlin specific plugin will wrap into kotlin()

And next will each fields we will consider is as like variable so make change

minSDKVersion = 24

buildTypes will take in create(“”) or getByName(“”)

progaurd will go in function body.

Compile options will take =

Build features take =

If you are using cannary and eguana version then there android studio gives you default version categlog system

From hedge hog it was not using version catalog defaults.

For benefits using version catalog some times librraies version are similar to each others, and there we can grouping of libraries.

When we are creating multi module based app then each module has keep their own gradle dependencies, and version catalog allows here two you can place all single dependencies in single place.

Bom stands for build of material.

To create version catalog.

There is specific need three types of blogs

Version

Libraries

Plugins

// Exclude from all release variants (any flavor + release build type)  
*configurations*.all **{** if (*name*.*contains*("release", ignoreCase = true)) {  
 *exclude*(group = "androidx.ui", module = "ui-test-manifest")  
 }  
**}**

android-compose-bom = { module = "androidx.compose:compose-bom", version.ref = "compose-bom" }

android-compose-ui = { group = "androidx.compose.ui" , name = "ui"}

android-compose-ui-graphics = { group = "androidx.compose.ui" , name = "graphics"}

android-compose-bom = { module = "androidx.compose:compose-bom", version.ref = "compose-bom" }

android-compose-ui = { group = "androidx.compose.ui" , name = "ui"}

android-compose-ui-graphics = { group = "androidx.compose.ui" , name = "graphics"}

kapt vs ksp

kapt is much faster gradle build.

Kotlin assets processing tool here java annotation process works with kotlin code.

This tools is very slow when process execute, room dagger both heavy annotation libraries, which generate code behind the scene,

Ksp = kotlin symbol processing

If you want to use ksp at least your kotlin version should be 1.9.0

Dagger hilt plugin should be 2.48

// id("com.google.devtools.ksp") version "1.9.0-1.0-13" apply false

Needs for ksp