Library Version - Material vs Material3

Using different versions or types of libraries (like **Material** vs **Material3**) can **affect your Compose project**, especially with APIs like TextFieldDefaults.colors() vs textFieldColors().

Understanding the Impact of Using Different Library Versions (Material vs Material 3)

In Jetpack Compose, there are **two main UI libraries** for styling and theming:

Library	Package	Style System
Material	androidx.compose.material	**Material Design 2**
Material3	androidx.compose.material3	**Material Design 3** (also called M3 or MD3)

These two libraries may look similar, but their APIs, theming, and component structures **can be different**, even for the same UI component like TextField.

Real Example: TextField Styling

Material 3 Style

```
colors = TextFieldDefaults.colors(
    focusedContainerColor = MaterialTheme.colorScheme.primary,
    unfocusedContainerColor = MaterialTheme.colorScheme.primary,
    focusedTextColor = Color.White,
    unfocusedTextColor = Color.White,
    cursorColor = Color.White,
    focusedIndicatorColor = Color.Transparent,
    unfocusedIndicatorColor = Color.Transparent
)
```

- Uses TextFieldDefaults.colors() → available in **Material 3**
- Works with **MaterialTheme.colorScheme** (the new theme system)
- Supports container-based customization (e.g., focusedContainerColor)
- This works only if you're using androidx.compose.material3

Old Material Style (Material 2)

```
colors = TextFieldDefaults.textFieldColors(
    focusedIndicatorColor = Color.Transparent,
    unfocusedIndicatorColor = Color.Transparent,
    focusedLabelColor = Color.Green,
    unfocusedLabelColor = Color.White,
    containerColor = MaterialTheme.colorScheme.primary,
    textColor = Color.White,
    cursorColor = Color.White
)
```

- Uses TextFieldDefaults.textFieldColors() → used in **Material 2**
- Expects older theming system (MaterialTheme.colors)
- Not compatible with Material 3 APIs or structure
- This will **not compile** in Material3 — these parameters won't exist

What Happens If You Mix or Use the Wrong Version?

Issue	What It Looks Like
Unresolved Reference	You get errors like: Unresolved reference: textFieldColors or textColor is not a valid parameter
Unexpected Styling	Even if it compiles, the theme may look off, or not match your design
Inconsistent UI	Combining material and material3 components can break consistent design behavior and responsiveness
Confusion	Tutorials online may use different versions than your project → leads to confusion for students when copying code

Best Practices

Tip	Why It Matters
Stick to either material OR material3	Mixing both can cause conflicts and styling issues
Use code completion (Ctrl + Space)	Helps you see what methods are available in the version you're using
Check documentation for version-specific APIs	Different versions = different function names & parameters
Use MaterialTheme.colorScheme only with Material3	Material2 uses MaterialTheme.colors instead

How to Check the Version of Material3 in Your Project

Your project uses **version catalogs** (libs.versions.toml) **together with** build.gradle.kts, so here's how to understand what version you're using:

Step 1: Check build.gradle.kts

Open your module-level **build.gradle.kts** and find this:

```
implementation(libs.androidx.material3)
```

This tells us:
Material3 is being pulled from the version catalog (libs.versions.toml)
But the actual version is defined **in the TOML file**.

Step 2: Check libs.versions.toml

Now open the file libs.versions.toml and look for this:

```
[libraries]

androidx-material3 = { group = "androidx.compose.material3",
name = "material3" }
```

To know the version, first we need to know the concept of BOM.

What is a BOM (Bill of Materials) in Gradle?

A **BOM (Bill of Materials)** is a Gradle feature that helps manage **consistent versions across related libraries**.

Instead of specifying the version of each library individually, you define a single BOM version, and all related libraries (like material3, ui, foundation, etc.) will use **compatible versions** automatically.

This is especially helpful in Jetpack Compose, where many libraries (like material3, ui, runtime, tooling, etc.) must match versions to work properly.

How BOM Works in Your Project

In your **build.gradle.kts**, you likely have:

```
implementation(platform(libs.androidx.compose.bom))
```

```
implementation(libs.androidx.material3)
```

In this case, you're **not hardcoding the Material3 version** — you're letting the BOM manage it.

There Are 2 Cases Here:

Case 1: No version specified directly

In your **libs.versions.toml**:

```
[libraries]
```

```
androidx-material3 = { group = "androidx.compose.material3",
name = "material3" }
```

If there's **no version specified**, it means the version is **inherited from the Compose BOM**.

Your BOM version is set here:

```
[versions]
```

```
composeBom = "2025.03.01"
```

So your material3, ui, foundation, and other Compose libraries are all using versions **defined by** this Compose BOM (2025.03.01).

To see what exact library versions are used by this BOM version, check:

https://developer.android.com/jetpack/compose/bom/bom-mapping

Case 2: Version is specified directly

If you explicitly add the version like this:

```
androidx-material3 = { group = "androidx.compose.material3",
name = "material3", version = "1.2.0" }
```

Then you are **not relying on the BOM**, and Material3 will use exactly version 1.2.0.

But in this case, you need to make sure that **other Compose libraries** (like ui, runtime, etc.) are compatible with that version.
Using mismatched versions can lead to **runtime crashes or compilation issues**.