

translated by Google

Cette page a été traduite par l'API Cloud Translation
([//cloud.google.com/translate/?hl=fr](https://cloud.google.com/translate/?hl=fr)).

 Automation Ready[Switch to English](#)

Prévisualiser votre UI avec des aperçus composables

Un composable est défini par une fonction et annoté avec `@Composable` :

```
@Composable
fun SimpleComposable() {
    Text("Hello World")
}
nippets/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L36-L39)
```

Hello World

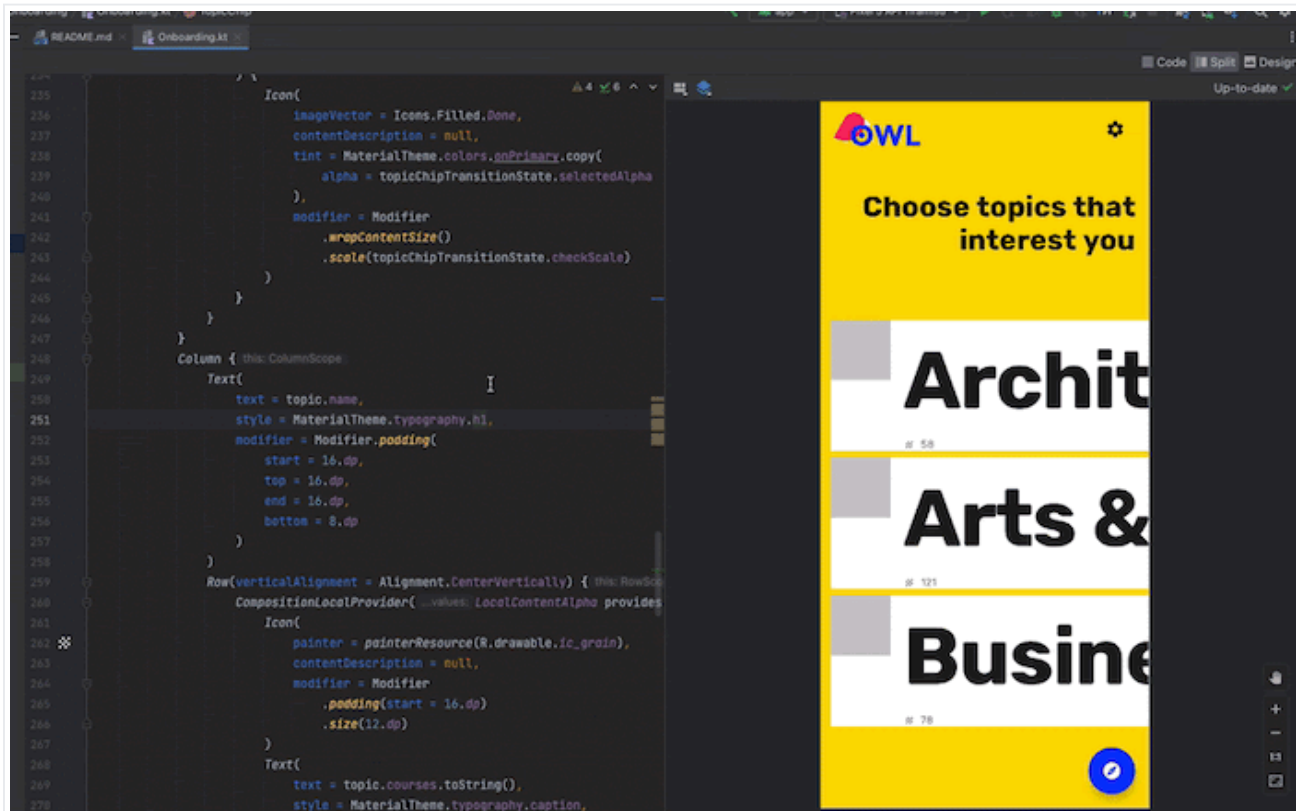
Pour afficher un aperçu de ce composable, créez un autre composable, annoté avec `@Composable` et `@Preview`. Ce nouveau composable annoté contient désormais le composable que vous avez créé initialement, `SimpleComposable` :

```
@Preview
@Composable
fun SimpleComposablePreview() {
    SimpleComposable()
}
nippets/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L43-L47)
```

L'annotation `@Preview`

(<https://developer.android.com/reference/kotlin/androidx/compose/ui/tooling/preview/Preview?hl=fr>) indique à Android Studio que ce composable doit être affiché dans la vue Conception de ce fichier. Vous pouvez voir les mises à jour en direct de l'aperçu de votre composable à mesure que vous effectuez vos modifications.


Remarque : Android Wear utilise différentes annotations pour les aperçus (`@WearPreviewDevices` et `@WearPreviewFontScales`). Pour en savoir plus, consultez [Tester les combinaisons de tailles d'écran et de police à l'aide des aperçus](https://developer.android.com/training/wearables/compose/screen-size?hl=fr#previews) (<https://developer.android.com/training/wearables/compose/screen-size?hl=fr#previews>).



Vous pouvez ajouter des paramètres manuellement dans votre code pour personnaliser la manière dont Android Studio affiche `@Preview`. Vous pouvez même ajouter l'annotation `@Preview` à la même fonction plusieurs fois pour prévisualiser un composable avec différentes propriétés.

Remarque : Si vous avez plusieurs aperçus composables, mais que vous souhaitez vous concentrer sur un seul à la fois, nous vous recommandons d'utiliser le mode Focus pour économiser des ressources de rendu. Passez au mode Grille ou Liste lorsque vous avez besoin d'afficher toutes les variantes d'UI en même temps. Vous pouvez basculer entre les modes à l'aide du menu en haut de l'onglet **Conception**.

L'un des principaux avantages de l'utilisation des composables `@Preview` est d'éviter de dépendre de l'émulateur dans Android Studio. Vous pouvez enregistrer le démarrage de l'émulateur, qui consomme beaucoup de mémoire, pour des modifications plus définitives de l'apparence, et utiliser la capacité de `@Preview` à effectuer et tester facilement de petites modifications de code.

Pour exploiter au mieux l'annotation `@Preview`, veuillez à définir vos écrans en termes d'état qu'ils reçoivent en entrée et d'événements qu'ils génèrent.  Automation Ready

Définissez votre `@Preview`

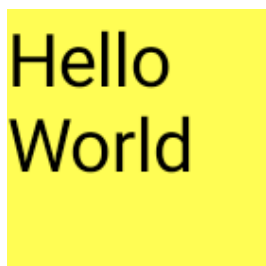
Android Studio propose plusieurs fonctionnalités pour étendre l'aperçu des composables. Vous pouvez modifier leur conception de conteneur, interagir avec ou les déployer directement sur un émulateur ou un appareil.

Dimensions

Par défaut, les dimensions `@Preview` sont choisies automatiquement pour encapsuler son contenu. Pour définir les dimensions manuellement, ajoutez les paramètres `heightDp` et `widthDp`. Ces valeurs sont déjà interprétées comme `dp` (<https://developer.android.com/reference/kotlin/androidx/compose/ui/unit/Dp?hl=fr>). Vous n'avez donc pas besoin d'ajouter `.dp` :

```
@Preview(widthDp = 50, heightDp = 50)
@Composable
fun SquareComposablePreview() {
    Box(Modifier.background(Color.Yellow)) {
        Text("Hello World")
    }
}
```

ppets/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L110-L116)

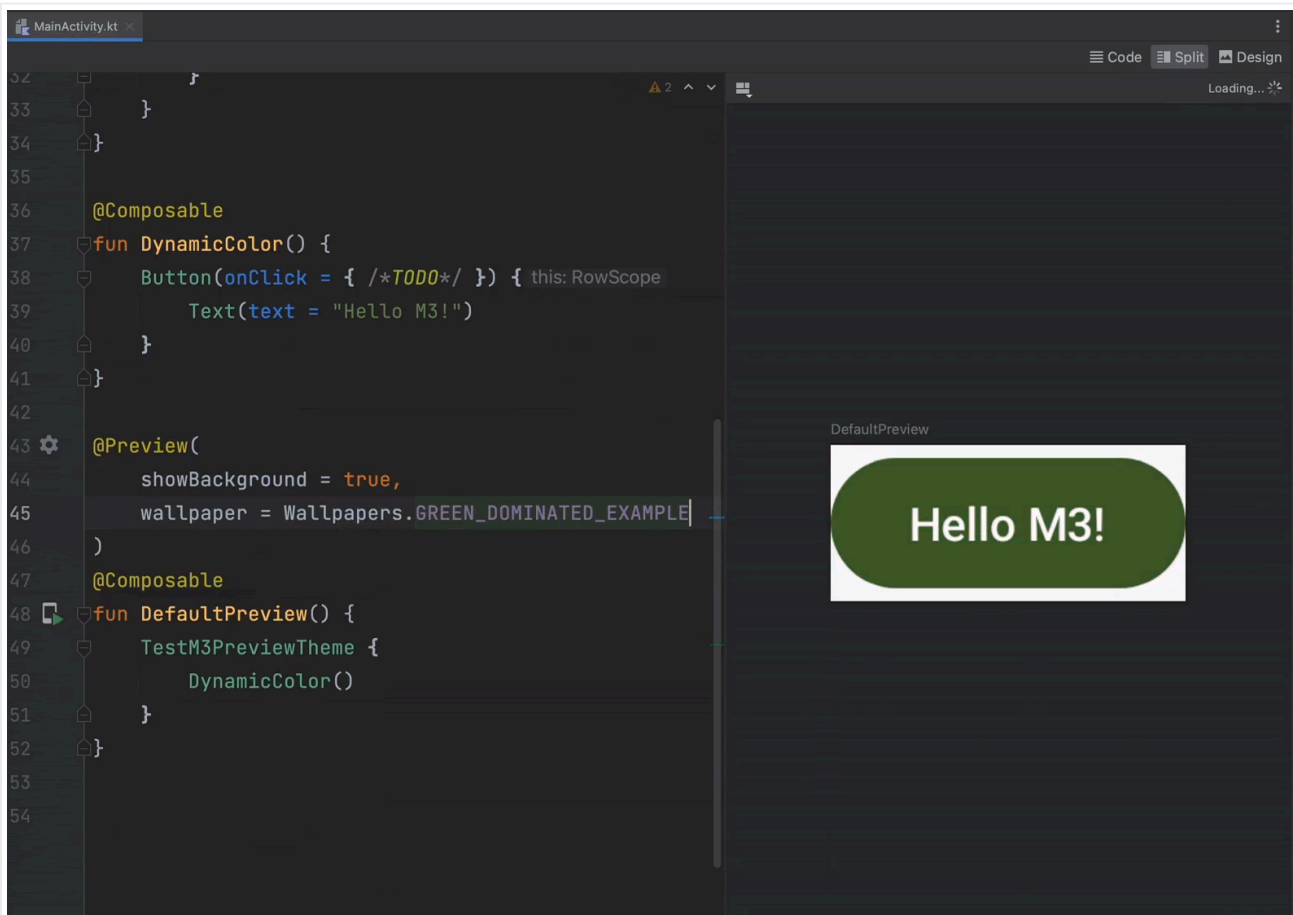


Prévisualisation de la couleur dynamique

Si vous avez activé la couleur dynamique (<https://m3.material.io/styles/color/dynamic-color/overview>) dans votre application, utilisez l'attribut `wallpaper` pour changer de fond d'écran et voir comment votre UI réagit au fond d'écran de différents utilisateurs. Sélectionnez l'un des thèmes de fond d'écran proposés par la classe `Wallpaper`

(<https://developer.android.com/reference/kotlin/androidx/compose/ui/tooling/preview/Wallpapers?hl=fr>).

Cette fonctionnalité nécessite Compose 1.4.0 ou version ultérieure.

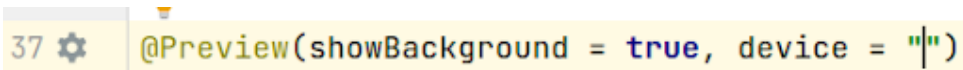


Utiliser avec différents appareils

Dans Android Studio Flamingo, vous pouvez modifier le paramètre `device` de l'annotation d'aperçu pour définir les configurations de vos composables sur différents appareils.

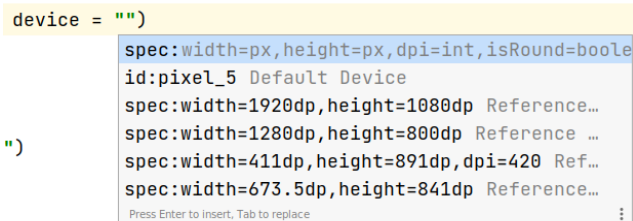


Lorsque le paramètre de l'appareil est une chaîne vide (`@Preview(device = "")`), vous pouvez appeler la saisie semi-automatique en appuyant sur `Ctrl` + `Space`. Vous pouvez ensuite définir les valeurs de chaque paramètre.



```
37 @Preview(showBackground = true, device = "")
```

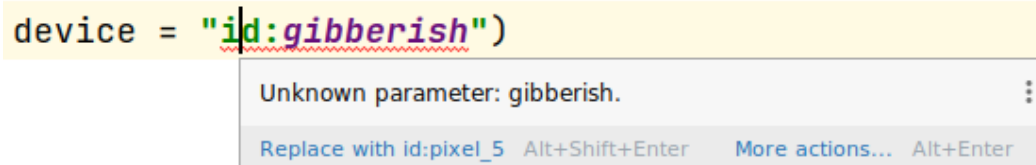
Dans la saisie semi-automatique, vous pouvez sélectionner n'importe quelle option d'appareil dans la liste, par exemple `@Preview(device = "id:pixel_4")`. Vous pouvez également saisir un appareil personnalisé en sélectionnant `spec:width=px,height=px,dpi=int...` pour définir les valeurs individuelles de chaque paramètre.



```
device = ""
spec:width=px,height=px,dpi=int,isRound=boolean
id:pixel_5 Default Device
spec:width=1920dp,height=1080dp Reference...
spec:width=1280dp,height=800dp Reference ...
spec:width=411dp,height=891dp,dpi=420 Ref...
spec:width=673.5dp,height=841dp Reference...
```

Pour appliquer la modification, appuyez sur `Enter`. Pour l'annuler, appuyez sur `Esc`.

Si vous définissez une valeur non valide, la déclaration est soulignée en rouge et une correction peut être disponible (`Alt` + `Enter` (`⌘` + `⌘` pour macOS) > **Remplacer par...**). L'inspection tente de fournir une correction qui ressemble le plus possible à votre saisie.



```
device = "id:gibberish"
```

Unknown parameter: gibberish.

Replace with id:pixel_5 Alt+Shift+Enter More actions... Alt+Enter

Paramètres régionaux

Pour tester différents paramètres régionaux utilisateurs, ajoutez le paramètre `locale` :

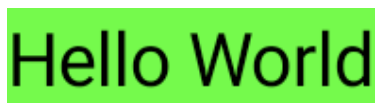
```
@Preview(locale = "fr-rFR")
@Composable
fun DifferentLocaleComposablePreview() {
    Text(text = stringResource(R.string.greeting))
}
// File: app/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L120-L124
```



Définir la couleur de l'arrière-plan

Par défaut, votre composable s'affiche avec un arrière-plan transparent. Pour ajouter un arrière-plan, ajoutez les paramètres `showBackground` et `backgroundColor`. N'oubliez pas que `backgroundColor` est un ARGB Long (<https://developer.android.com/reference/android/graphics/Color?hl=fr#color-longs>) et non une valeur `Color` :

```
@Preview(showBackground = true, backgroundColor = 0xFF00FF00)
@Composable
fun WithGreenBackground() {
    Text("Hello World")
}
// Snippets/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L102-L106
```



UI du système

Si vous devez afficher les barres d'état et d'action dans un aperçu, ajoutez le paramètre `showSystemUi` :

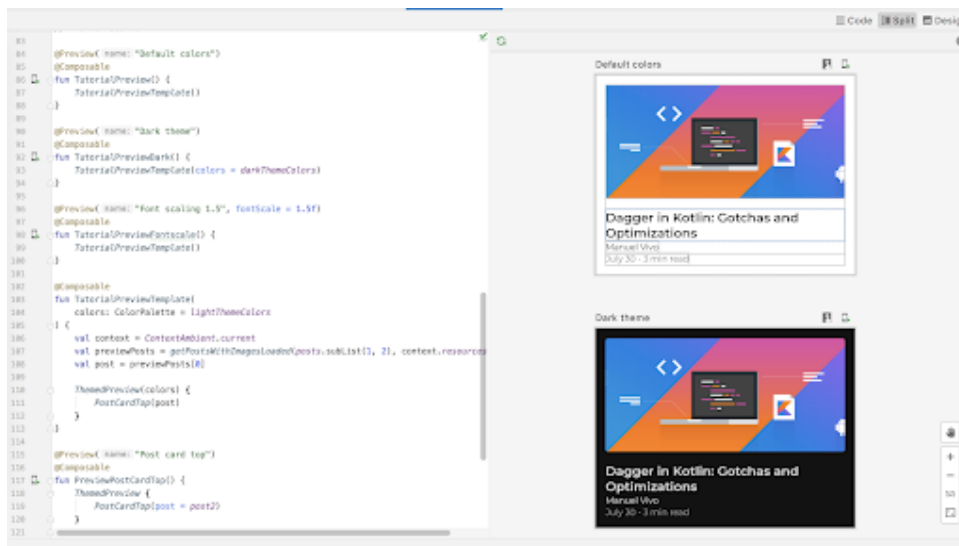
```
@Preview(showSystemUi = true)
@Composable
fun DecoratedComposablePreview() {
    Text("Hello World")
}
// Snippets/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L128-L132
```



Automation Ready

Mode UI

Le paramètre `uiMode` peut utiliser n'importe quelle constante `Configuration.UI_*` (https://developer.android.com/reference/android/content/res/Configuration?hl=fr#UI_MODE_NIGHT_MASK) et vous permet de modifier le comportement de l'aperçu en conséquence. Par exemple, vous pouvez passer l'aperçu en mode Nuit pour observer la réaction du thème.



Automation Ready

LocalInspectionMode

Vous pouvez lire le `CompositionLocal LocalInspectionMode`

(<https://developer.android.com/reference/kotlin/androidx/compose/ui/platform/package-summary?hl=fr#LocalInspectionMode>))

pour voir si le composable est affiché dans un aperçu (à l'intérieur d'un composant inspectable (<https://developer.android.com/reference/kotlin/androidx/compose/ui/platform/InspectableValue?hl=fr>)). Si la composition est affichée dans un aperçu, `LocalInspectionMode.current` prend la valeur `true`. Ces informations vous permettent de personnaliser votre aperçu. Par exemple, vous pouvez afficher une image d'espace réservé dans la fenêtre d'aperçu au lieu d'afficher des données réelles.

Vous pouvez ainsi contourner les limitations (#preview-limitations). Par exemple, en affichant des exemples de données au lieu d'appeler une requête réseau.

```
@Composable
fun GreetingScreen(name: String) {
    if (LocalInspectionMode.current) {
        // Show this text in a preview window:
        Text("Hello preview user!")
    } else {
        // Show this text in the app:
        Text("Hello $name!")
    }
}
```

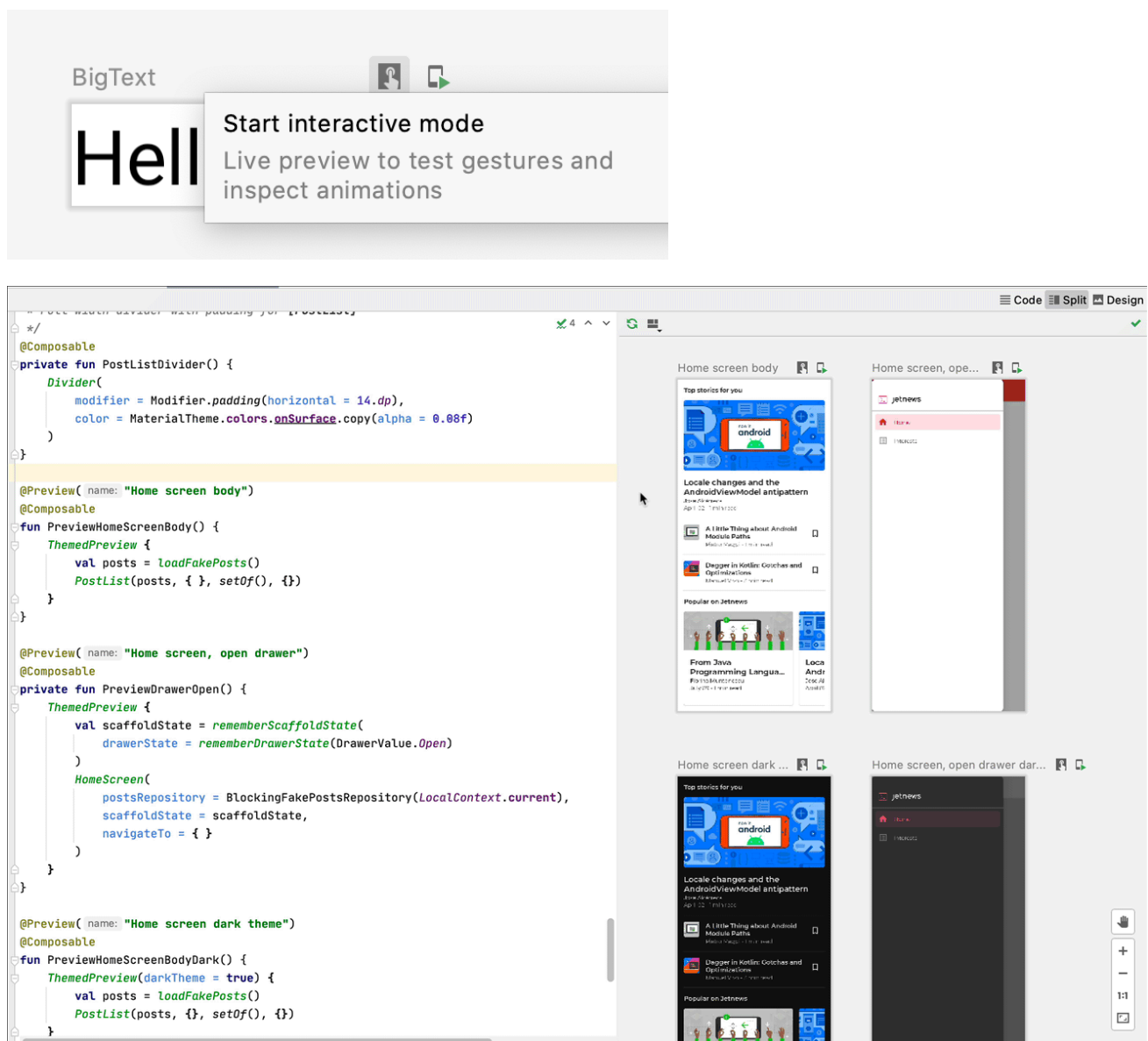
snippets/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L51-L60)

Interagir avec votre @Preview

Android Studio fournit des fonctionnalités qui vous permettent d'interagir avec les aperçus que vous avez définis. Cette interaction vous aide à comprendre le comportement d'exécution de vos aperçus et vous permet de mieux parcourir votre UI avec les aperçus.

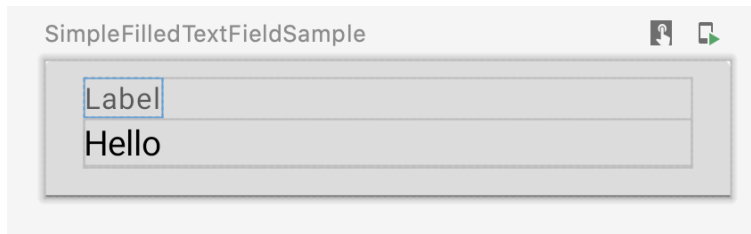
Mode interactif

Le mode interactif vous permet d'interagir avec un aperçu comme sur un appareil exécutant votre programme, tel qu'un téléphone ou une tablette. Il est isolé dans un environnement de bac à sable (c'est-à-dire des autres aperçus), où vous pouvez cliquer sur des éléments et effectuer des saisies dans l'aperçu. C'est un moyen rapide de tester différents états, gestes et même animations de votre composable.



Navigation dans le code et contours d'un composable

Vous pouvez pointer sur un aperçu pour afficher les contours des composables qu'il contient. Cliquez sur le contour d'un composable pour accéder à sa définition dans la vue de l'éditeur.



Automation Ready

Exécuter l'aperçu

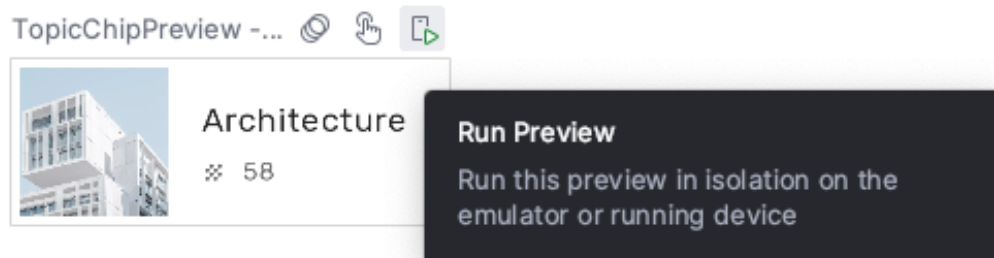
Vous pouvez exécuter un `@Preview` spécifique sur un émulateur ou un appareil physique. L'aperçu est déployé dans la même application de projet qu'un nouveau `Activity`. Il partage donc le même contexte et les mêmes autorisations. Il ne vous oblige pas à écrire du code passe-partout pour demander une autorisation si elle a déjà été accordée.

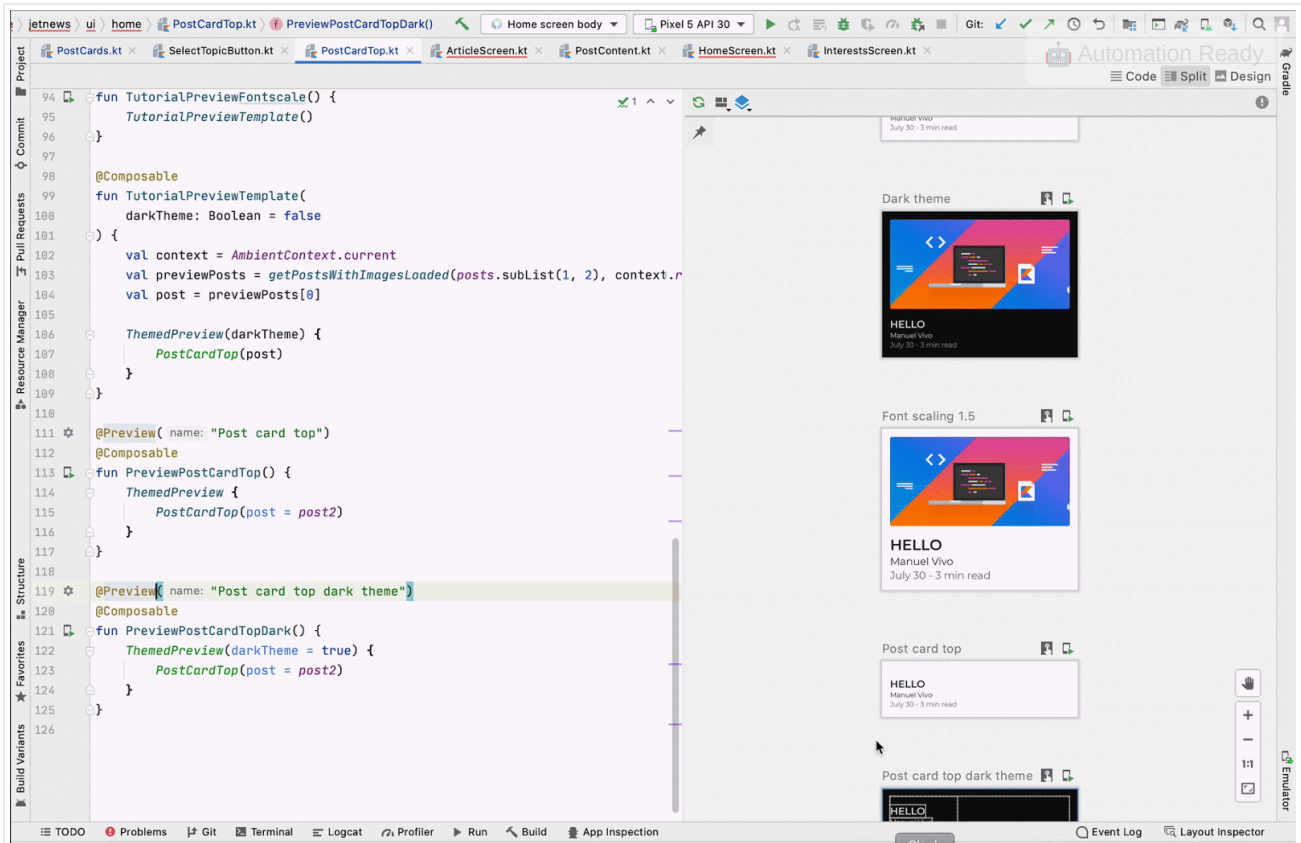
Remarque : Les arguments appliqués à l'annotation `@Preview` (par exemple, `widthDp`, `locale`, etc.) ne sont pas appliqués à l'aperçu exécuté et déployé.

Cliquez sur l'icône **Exécuter l'aperçu**



à côté de l'annotation `@Preview` ou en haut de l'aperçu. Android Studio déploie `@Preview` sur votre appareil ou votre émulateur connecté.





Copier le rendu @Preview

Chaque aperçu affiché peut être copié en tant qu'image en effectuant un clic droit dessus.

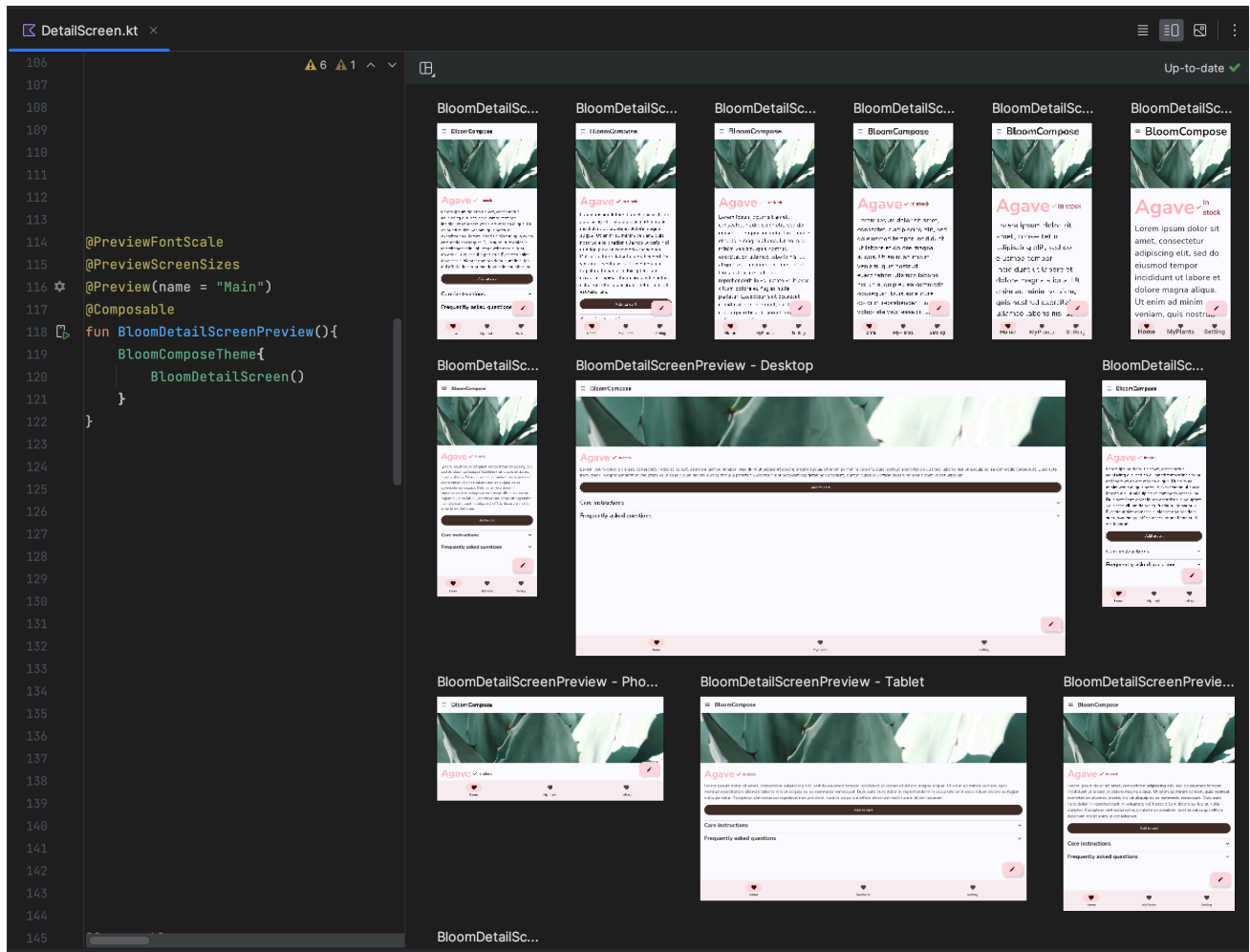


Plusieurs aperçus de la même annotation @Preview

Vous pouvez présenter plusieurs versions du même composable @Preview avec des spécifications différentes ou des paramètres différents transmis au composable. Vous pouvez ainsi réduire le code récurrent que vous devriez écrire autrement.

Modèles d'aperçus multiples

`androidx.compose.ui:ui-tooling-preview` 1.6.0-alpha01+ introduit des modèles d'API Multipreview : `@PreviewScreenSizes`, `@PreviewFontScales`, `@PreviewLightDark` et `@PreviewDynamicColors`. Ainsi, avec une seule annotation, vous pouvez prévisualiser votre UI Compose dans des scénarios courants.



Créer des annotations multiprévues personnalisées

L'aperçu multiple vous permet de définir une classe d'annotation comportant elle-même plusieurs annotations `@Preview` avec des configurations différentes. Ajouter cette annotation à une fonction composable affiche automatiquement tous les aperçus en une seule fois. Par exemple, vous pouvez utiliser cette annotation pour prévisualiser plusieurs appareils, tailles de police ou thèmes en même temps sans répéter ces définitions pour chaque composable.

Commencez par créer votre propre classe d'annotation personnalisée :

```
@Preview(
    name = "small font",
    group = "font scales",
    fontScale = 0.5f
)
```

```
@Preview(
    name = "large font",
    group = "font scales",
    fontScale = 1.5f
)
annotation class FontScalePreviews
nippets/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L64-L74
```



Vous pouvez utiliser cette annotation personnalisée pour vos composables d'aperçu :

```
@FontScalePreviews
@Composable
fun HelloWorldPreview() {
    Text("Hello World")
}
nippets/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L78-L82
```



Vous pouvez combiner plusieurs annotations d'aperçus multiples et d'aperçus normaux pour créer un ensemble d'aperçus plus complet. Combiner des annotations d'aperçus multiples ne signifie pas que toutes les différentes combinaisons sont affichées. Chaque annotation d'aperçus multiples agit indépendamment et n'affiche que ses propres variantes.

```
@Preview(
    name = "Spanish",
    group = "locale",
    locale = "es"
)
@FontScalePreviews
annotation class CombinedPreviews
```

```
@CombinedPreviews
@Composable
fun HelloWorldPreview2() {
    MaterialTheme { Surface { Text(stringResource(R.string.hello_world)) } }
}
```

snippets/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L86-L98)



La nature combinée des aperçus multiples et des aperçus normaux vous permet de tester plus complètement de nombreuses propriétés de projets à plus grande échelle.

@Preview et les grands ensembles de données

Très souvent, vous devez transmettre un grand ensemble de données à votre aperçu composable. Pour ce faire, il vous suffit de transmettre des exemples de données à une fonction Aperçu du composable en ajoutant un paramètre avec l'annotation

@PreviewParameter

(<https://developer.android.com/reference/kotlin/androidx/compose/ui/tooling/preview/PreviewParameter?hl=fr>)

.

```
@Preview
@Composable
fun UserProfilePreview(
    @PreviewParameter(UserPreviewParameterProvider::class) user: User
) {
```

```

        UserProfile(user)
    }
}

```

Automation Ready

snippets/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L136-L148

Pour fournir les exemples de données, créez une classe qui implémente

PreviewParameterProvider

(<https://developer.android.com/reference/kotlin/androidx/compose/ui/tooling/preview/PreviewParameterProvider?hl=fr>)

et renvoie les exemples de données sous forme de séquence.

```

class UserPreviewParameterProvider : PreviewParameterProvider<User> {
    override val values = sequenceOf(
        User("Elise"),
        User("Frank"),
        User("Julia")
    )
}

```

snippets/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L152-L158

Un aperçu s'affiche pour chaque élément de données de la séquence :



Vous pouvez utiliser la même classe de fournisseur pour plusieurs aperçus. Si nécessaire, limitez le nombre d'aperçus en définissant le paramètre "limit".

```

@Preview
@Composable
fun UserProfilePreview2(
    @PreviewParameter(UserPreviewParameterProvider::class, limit = 2) user: User
) {
    UserProfile(user)
}

```

snippets/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L162-L168

Limites et bonnes pratiques



Android Studio exécute le code d'aperçu directement dans la zone d'aperçu. Il ne nécessite pas l'exécution d'un émulateur ou d'un appareil physique, car il utilise une partie transférée du framework Android appelée `Layoutlib`. `Layoutlib` est une version personnalisée du framework Android conçue pour s'exécuter en dehors des appareils Android. L'objectif de la bibliothèque est de fournir un aperçu d'une mise en page dans Android Studio qui est très proche de son rendu sur les appareils.

Limites des aperçus

En raison de la façon dont les aperçus sont rendus dans Android Studio, ils sont légers et ne nécessitent pas l'ensemble du framework Android pour être rendus. Toutefois, cette approche présente les limites suivantes :

- Aucun accès au réseau
- Aucun accès aux fichiers
- Certaines API `Context` peuvent ne pas être entièrement disponibles.

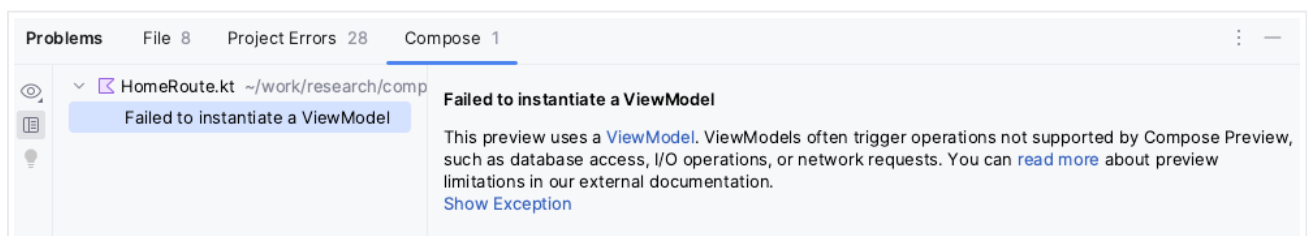
Aperçus et `ViewModels`

Les aperçus sont limités lorsque vous utilisez `ViewModel`

(<https://developer.android.com/viewmodel?hl=fr>) dans un composable. Le système d'aperçu n'est pas en mesure de construire tous les paramètres transmis à un `ViewModel`, tels que les dépôts, les cas d'utilisation, les gestionnaires ou autres. De plus, si votre `ViewModel` participe à l'injection de dépendances (comme avec `Hilt`

(<https://developer.android.com/training/dependency-injection/hilt-android?hl=fr>)), le système d'aperçu ne peut pas créer l'intégralité du graphique de dépendances pour construire le `ViewModel`.

Lorsque vous essayez de prévisualiser un composable avec `ViewModel`, Android Studio affiche une erreur lors du rendu du composable en question :



Avertissement : Vous ne devez pas transmettre les instances `ViewModel` à d'autres fonctions composables. Dans ce cas, vous associez les composables au type `ViewModel`, ce qui les rend moins réutilisables, plus difficiles à tester et impossibles à prévisualiser.

Si vous souhaitez prévisualiser un composable qui utilise un `ViewModel`, vous devez créer un autre composable avec les paramètres de `ViewModel` transmis en tant qu'arguments du composable. Ainsi, vous n'avez pas besoin de prévisualiser le composable qui utilise `ViewModel`.

```
@Composable
fun AuthorScreen(viewModel: AuthorViewModel = viewModel()) {
    AuthorScreen(
        name = viewModel.authorName,
        // ViewModel sends the network requests and makes posts available as a state
        posts = viewModel.posts
    )
}

@Composable
fun AuthorScreen(
    name: NameLabel,
    posts: PostsList
) {
    // ...
}

@Preview
@Composable
fun AuthorScreenPreview(
    // You can use some sample data to preview your composable without the need to c
    name: String = sampleAuthor.name,
    posts: List<Post> = samplePosts[sampleAuthor]
) {
    AuthorScreen(
        name = NameLabel(name),
        posts = PostsList(posts)
    )
}
```

Classe d'annotation `@Preview`

Vous pouvez toujours effectuer un Ctrl+clic ou un ⌘+clic sur l'annotation `@Preview` dans Android Studio pour obtenir la liste complète des paramètres que vous pouvez ajuster lorsque vous personnalisez votre aperçu.

```
annotation class Preview(
    val name: String = "",
```

```
val group: String = "",
@IntRange(from = 1) val apiLevel: Int = -1,
val widthDp: Int = -1,
val heightDp: Int = -1,
val locale: String = "",
@FloatRange(from = 0.01) val fontScale: Float = 1f,
val showSystemUi: Boolean = false,
val showBackground: Boolean = false,
val backgroundColor: Long = 0,
@UiMode val uiMode: Int = 0,
@Device val device: String = Devices.DEFAULT,
@Wallpaper val wallpaper: Int = Wallpapers.NONE,
)
ppets/src/main/java/com/example/compose/snippets/tooling/AndroidStudioComposeSnippets.kt#L173-L187
```



Ressources supplémentaires

Pour en savoir plus sur la façon dont Android Studio favorise la facilité d'utilisation de [@Preview](#) et découvrir d'autres conseils sur les outils, consultez le blog [Compose Tooling](#) (<https://medium.com/androiddevelopers/compose-tooling-42621bd8719b>).

[Suivant](#)

[Prévisualiser et déboguer des animations](#) →

(<https://developer.android.com/develop/ui/compose/tooling/animation-preview?hl=fr>)

Le contenu et les exemples de code de cette page sont soumis aux licences décrites dans la [Licence de contenu](#) (<https://developer.android.com/license?hl=fr>). Java et OpenJDK sont des marques ou des marques déposées d'Oracle et/ou de ses sociétés affiliées.

Dernière mise à jour le 2025/09/09 (UTC).