Utiliser des polices

Cette page explique comment définir des polices dans votre application Compose.

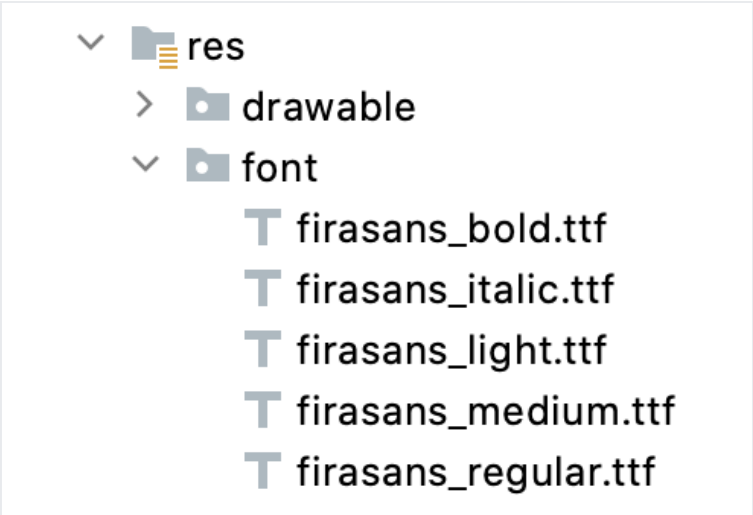
Définir la police

`Text` utilise un paramètre `fontFamily` pour définir la police utilisée dans le composable. Par défaut, les familles de polices serif, sans-serif, monospace et cursive sont incluses (https://developer.android.com/reference/kotlin/androidx/compose/ui/text/font/FontFamily?hl=fr#summary) :

```
@Composable
fun DifferentFonts() {
    Column {
        Text("Hello World", fontFamily = FontFamily.Serif)
        Text("Hello World", fontFamily = FontFamily.SansSerif)
    }
}
```

346ad18/compose/snippets/src/main/java/com/example/compose/snippets/text/TextSnippets.kt#L190-L196)

L'attribut `fontFamily` permet d'utiliser les polices personnalisées définies dans le dossier `res/font` :



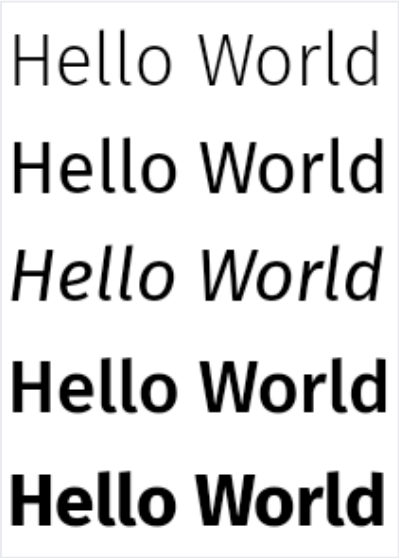
Cet exemple montre comment définir une propriété `fontFamily` en fonction de ces fichiers de polices et en utilisant la fonction `Font` (https://developer.android.com/reference/kotlin/androidx/compose/ui/text/font/package-summary?hl=fr#Font(kotlin.Int,androidx.compose.ui.text.font.FontWeight,androidx.compose.ui.text.font.FontStyle)) :

```
val firaSansFamily = FontFamily(
    Font(R.font.firasans_light, FontWeight.Light),
    Font(R.font.firasans_regular, FontWeight.Normal),
    Font(R.font.firasans_italic, FontWeight.Normal, FontStyle.Italic),
    Font(R.font.firasans_medium, FontWeight.Medium),
    Font(R.font.firasans_bold, FontWeight.Bold)
)
346ad18/compose/snippets/src/main/java/com/example/compose/snippets/text/TextSnippets.kt#L203-L209)
```

Vous pouvez transmettre cet attribut `fontFamily` au composable `Text`. Étant donné qu'un élément `fontFamily` peut inclure différentes épaisseurs de police, vous pouvez spécifier `fontWeight` manuellement afin de sélectionner celle qui convient au texte :

```
Column {
    Text(text = "text", fontFamily = firaSansFamily, fontWeight = FontWeight.Light
    Text(text = "text", fontFamily = firaSansFamily, fontWeight = FontWeight.Norma
    Text(
        text = "text",
        fontFamily = firaSansFamily,
        fontWeight = FontWeight.Normal,
        fontStyle = FontStyle.Italic
    )
    Text(text = "text", fontFamily = firaSansFamily, fontWeight = FontWeight.Mediu
```

```
    Text(text = "text", fontFamily = firaSansFamily, fontWeight = FontWeight.Bold)
}
```
346ad18/compose/snippets/src/main/java/com/example/compose/snippets/text/TextSnippets.kt#L216-L227)



Pour découvrir comment définir la typographie dans l'ensemble de votre application, consultez Systèmes de conception personnalisés dans Compose (https://developer.android.com/develop/ui/compose/designsystems/custom?hl=fr).

Polices téléchargeables

À partir de Compose 1.2.0 (https://developer.android.com/jetpack/androidx/releases/compose-ui?hl=fr#1.2.0), vous pouvez utiliser l'API de polices téléchargeables de l'application Compose pour télécharger des polices Google (https://fonts.google.com/?hl=fr) de manière asynchrone et les utiliser dans votre application.

Les polices téléchargeables basées sur des fournisseurs personnalisés ne sont pas disponibles pour le moment.

Utiliser des polices téléchargeables par programmation

Pour télécharger une police par programmation à partir de votre application, procédez comme suit :

1. Ajoutez la dépendance :

Groovy (#groovy)Kotlin (#kotlin)

```
dependencies {
    ...
    implementation("androidx.compose.ui:ui-text-google-fonts:1.9.0")
}
```

2. Initialisez `GoogleFont.Provider` (https://developer.android.com/reference/kotlin/androidx/compose/ui/text/googlefonts/GoogleFont.Provider?hl=fr) avec les identifiants pour Google Fonts :

```
val provider = GoogleFont.Provider(
    providerAuthority = "com.google.android.gms.fonts",
    providerPackage = "com.google.android.gms",
    certificates = R.array.com_google_android_gms_fonts_certs
)
```
ets/src/main/java/com/example/compose/snippets/text/TextDownloadableFontsSnippets.kt#L54-L58)

   Voici les paramètres reçus par le fournisseur :

   - Autorité du fournisseur de polices pour Google Fonts.
   - Package du fournisseur de polices permettant de vérifier l'identité du fournisseur.
   - Liste d'ensembles de hachages des certificats permettant de vérifier l'identité du fournisseur. Vous trouverez les hachages requis pour le fournisseur Google Fonts dans le fichier `font_certs.xml` (https://github.com/android/compose-samples/blob/main/Jetchat/app/src/main/res/values-v23/font_certs.xml) de l'application exemple Jetchat.

3. Définissez un `FontFamily` (https://developer.android.com/reference/kotlin/androidx/compose/ui/text/font/FontFamily?hl=fr) :

```
// ...
 import androidx.compose.ui.text.googlefonts.GoogleFont
 import androidx.compose.ui.text.font.FontFamily
 import androidx.compose.ui.text.googlefonts.Font
 // ...

val fontName = GoogleFont("Lobster Two")

val fontFamily = FontFamily(
```

```
    Font(googleFont = fontName, fontProvider = provider)
)
```
ets/src/main/java/com/example/compose/snippets/text/TextDownloadableFontsSnippets.kt#L64-L79)

Vous pouvez interroger d'autres paramètres de police tels que l'épaisseur et le style avec, respectivement, `FontWeight` (https://developer.android.com/reference/kotlin/androidx/compose/ui/text/font/FontWeight?hl=fr) et `FontStyle` (https://developer.android.com/reference/kotlin/androidx/compose/ui/text/font/FontStyle?hl=fr) :

```
// ...
 import androidx.compose.ui.text.googlefonts.GoogleFont
 import androidx.compose.ui.text.font.FontFamily
 import androidx.compose.ui.text.googlefonts.Font
 // ...

val fontName = GoogleFont("Lobster Two")

val fontFamily = FontFamily(
    Font(
        googleFont = fontName,
        fontProvider = provider,
        weight = FontWeight.Bold,
        style = FontStyle.Italic
    )
)
```
ts/src/main/java/com/example/compose/snippets/text/TextDownloadableFontsSnippets.kt#L85-L105)

4. Configurez la famille de polices (`FontFamily`) à utiliser dans la fonction composable Text :

```
Text(
    fontFamily = fontFamily, text = "Hello World!"
)
```
snippets/src/main/java/com/example/compose/snippets/text/TextDownloadableFontsSnippets.kt#L112-L114)

Vous pouvez également définir une typographie (https://developer.android.com/reference/kotlin/androidx/compose/material3/Typography?hl=fr) pour utiliser votre `FontFamily` :

```
val MyTypography = Typography(
    bodyMedium = TextStyle(
        fontFamily = fontFamily, fontWeight = FontWeight.Normal, fontSize = 12.sp/
    ),
    bodyLarge = TextStyle(
        fontFamily = fontFamily,
        fontWeight = FontWeight.Bold,
        letterSpacing = 2.sp,
        /*...*/
    ),
    headlineMedium = TextStyle(
        fontFamily = fontFamily, fontWeight = FontWeight.SemiBold/*...*/
    ),
    /*...*/
)
snippets/src/main/java/com/example/compose/snippets/text/TextDownloadableFontsSnippets.kt#L120-L134)
```

Définissez ensuite cette typographie sur le thème de votre application :

```
MyAppTheme(
    typography = MyTypography
)/*...*/
snippets/src/main/java/com/example/compose/snippets/text/TextDownloadableFontsSnippets.kt#L144-L146)
```

Pour obtenir un exemple d'application mettant en œuvre des polices téléchargeables dans Compose avec Material3 (https://m3.material.io/styles/typography/overview), consultez l'application exemple Jetchat (https://github.com/android/compose-samples/tree/main/Jetchat).

Ajouter des polices de remplacement

Vous pouvez déterminer une série de polices de remplacement au cas où celle que vous avez spécifiée ne se téléchargerait pas correctement. Par exemple, si votre police téléchargeable est définie comme suit :

```
// ...
 import androidx.compose.ui.text.googlefonts.Font
 // ...

val fontName = GoogleFont("Lobster Two")

val fontFamily = FontFamily(
    Font(googleFont = fontName, fontProvider = provider),
    Font(googleFont = fontName, fontProvider = provider, weight = FontWeight.Bold)
)
```
snippets/src/main/java/com/example/compose/snippets/text/TextDownloadableFontsSnippets.kt#L153-L167)

Vous pouvez définir les valeurs par défaut de votre police pour les deux épaisseurs, comme suit :

```
// ...
 import androidx.compose.ui.text.font.Font
 import androidx.compose.ui.text.googlefonts.Font
 // ...

val fontName = GoogleFont("Lobster Two")

val fontFamily = FontFamily(
    Font(googleFont = fontName, fontProvider = provider),
    Font(resId = R.font.my_font_regular),
    Font(googleFont = fontName, fontProvider = provider, weight = FontWeight.Bold)
    Font(resId = R.font.my_font_regular_bold, weight = FontWeight.Bold)
)
```
snippets/src/main/java/com/example/compose/snippets/text/TextDownloadableFontsSnippets.kt#L173-L190)

Veillez à ajouter les importations appropriées.

En définissant `FontFamily` de la sorte, vous créez une famille de polices (`FontFamily`) contenant deux chaînes, une pour chaque épaisseur de police. Le mécanisme de chargement tente d'abord de récupérer la police en ligne, puis la police située dans le dossier de ressources `R.font` local.

Déboguer votre implémentation

Pour vous aider à vérifier si la police est téléchargée correctement, vous pouvez définir un gestionnaire de coroutine de débogage. Le handle identifiera le comportement à suivre en cas

d'échec du chargement asynchrone de la police.

Commencez par créer un `CoroutineExceptionHandler` (https://kotlin.github.io/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines/-coroutine-exception-handler/index.html) :

```
val handler = CoroutineExceptionHandler { _, throwable ->
    // process the Throwable
    Log.e(TAG, "There has been an issue: ", throwable)
}
```

snippets/src/main/java/com/example/compose/snippets/text/TextDownloadableFontsSnippets.kt#L196-L199)

Transmettez-le à la méthode `createFontFamilyResolver` (https://developer.android.com/reference/kotlin/androidx/compose/ui/text/font/package-summary?hl=fr#createFontFamilyResolver(android.content.Context)) pour que le résolveur utilise le nouveau gestionnaire :

```
CompositionLocalProvider(
    LocalFontFamilyResolver provides createFontFamilyResolver(LocalContext.current
) {
    Column {
        Text(
            text = "Hello World!", style = MaterialTheme.typography.bodyMedium
        )
    }
}
```

nippets/src/main/java/com/example/compose/snippets/text/TextDownloadableFontsSnippets.kt#L207-L215)

Vous pouvez également utiliser l'API `isAvailableOnDevice` (https://developer.android.com/reference/kotlin/androidx/compose/ui/text/googlefonts/package-summary?hl=fr#(androidx.compose.ui.text.googlefonts.GoogleFont.Provider).isAvailableOnDevice(android.content.Context)) du fournisseur pour vérifier s'il est disponible et si les certificats sont configurés correctement. Pour ce faire, vous pouvez appeler la méthode `isAvailableOnDevice` qui renvoie la valeur "false" si le fournisseur n'est pas configuré correctement.

```
val context = LocalContext.current
LaunchedEffect(Unit) {
    if (provider.isAvailableOnDevice(context)) {
```

```
        Log.d(TAG, "Success!")
    }
}
```
nippets/src/main/java/com/example/compose/snippets/text/TextDownloadableFontsSnippets.kt#L224-L229)

Mises en garde

Plusieurs mois sont nécessaires pour que les nouvelles polices Google Fonts soient disponibles sur Android. Il existe un intervalle de temps entre le moment où une police est ajoutée sur fonts.google.com (https://fonts.google.com/?hl=fr) et sa disponibilité via l'API de polices téléchargeables (dans le système de vue ou dans Compose). Les polices qui viennent d'être ajoutées risquent de ne pas se charger dans votre application et de générer une erreur `IllegalStateException` (https://docs.oracle.com/javase/7/docs/api/java/lang/IllegalStateException.html). Pour aider les développeurs à identifier cette erreur par rapport aux autres types d'erreurs de chargement de police, nous avons ajouté un message descriptif pour cette exception dans Compose en précisant les modifications ici (https://android-review.googlesource.com/c/platform/frameworks/support/+/2098457/). Si vous rencontrez des problèmes, signalez-les à l'aide de l'outil de suivi des problèmes (https://issuetracker.google.com/issues/new?component=779818&hl=fr).

Utiliser des polices variables

Une police variable est un format de police qui permet à un fichier de police de contenir différents styles. Avec les polices variables, vous pouvez modifier les axes (ou paramètres) pour générer le style de votre choix. Ces axes peuvent être standards (épaisseur, largeur, inclinaison et italique, par exemple) ou personnalisés (ils varient selon les polices variables).

Figure 1. *Texte utilisant la même police variable, personnalisée avec différentes valeurs d'axe.*

L'utilisation de polices variables au lieu de fichiers de police classiques vous permet de n'avoir qu'un seul fichier de police au lieu de plusieurs.

Avertissement : Les polices variables ne sont compatibles qu'avec Android O et les versions ultérieures.

Pour en savoir plus sur les polices variables, consultez Google Fonts Knowledge (https://fonts.google.com/knowledge/topics/variable_fonts?hl=fr), le catalogue complet (https://fonts.google.com/?vfonly=true&hl=fr) des polices variables disponibles et un tableau (https://fonts.google.com/variablefonts?hl=fr) des axes compatibles pour chaque police.

Ce document vous explique comment implémenter une police variable dans votre application Compose.

Charger une police variable

1. Téléchargez la police variable que vous souhaitez utiliser (par exemple, Roboto Flex (https://fonts.google.com/specimen/Roboto+Flex)%7B:.external%7D?hl=fr)) et placez-la dans le dossier `app/res/font` de votre application. Assurez-vous que le fichier .`ttf` que vous ajoutez est la *version à variation* de la police. Assurez-vous également que le nom de votre fichier de police est entièrement en minuscules et ne contient aucun caractère spécial.

★ **Remarque** : Les polices variables ne sont actuellement pas compatibles avec les polices téléchargeables. Pour connaître les dernières informations, consultez ce bug

(https://issuetracker.google.com/issues/223262013?hl=fr).

2. Pour charger une police variable, définissez un `FontFamily` à l'aide de la police placée dans le répertoire `res/font/` :

```
// In Typography.kt
@OptIn(ExperimentalTextApi::class)
val displayLargeFontFamily =
    FontFamily(
        Font(
            R.font.robotoflex_variable,
            variationSettings = FontVariation.Settings(
                FontVariation.weight(950),
                FontVariation.width(30f),
                FontVariation.slant(-6f),
            )
        )
    )
```

pose/snippets/src/main/java/com/example/compose/snippets/text/VariableFontsSnippets.kt#L45-L57)

L'API `FontVariation` vous permet de configurer des axes de typographie standards tels que weight (https://developer.android.com/reference/kotlin/androidx/compose/ui/text/font/FontVariation?hl=fr#weight(kotlin.Int)), width (https://developer.android.com/reference/kotlin/androidx/compose/ui/text/font/FontVariation?hl=fr#width(kotlin.Float)) et slant (https://developer.android.com/reference/kotlin/androidx/compose/ui/text/font/FontVariation?hl=fr#slant(kotlin.Float)). Il s'agit d'axes standards disponibles avec n'importe quelle police variable. Vous pouvez créer différentes configurations de la police en fonction de l'endroit où elle sera utilisée.

3. Les polices variables ne sont disponibles que pour les versions O et ultérieures d'Android. Ajoutez donc un garde-fou et configurez une solution de remplacement appropriée :

```
// In Typography.kt
val default = FontFamily(
    /*
    * This can be any font that makes sense
    */
    Font(
        R.font.robotoflex_static_regular
```

```
    )
)
@OptIn(ExperimentalTextApi::class)
val displayLargeFontFamily = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES
    FontFamily(
        Font(
            R.font.robotoflex_variable,
            variationSettings = FontVariation.Settings(
                FontVariation.weight(950),
                FontVariation.width(30f),
                FontVariation.slant(-6f),
            )
        )
    )
} else {
    default
}
```

pose/snippets/src/main/java/com/example/compose/snippets/text/VariableFontsSnippets.kt#L62-L85)

4. Extrayez les paramètres dans un ensemble de constantes pour faciliter la réutilisation et remplacez les paramètres de police par ces constantes :

```
// VariableFontDimension.kt
object DisplayLargeVFConfig {
    const val WEIGHT = 950
    const val WIDTH = 30f
    const val SLANT = -6f
    const val ASCENDER_HEIGHT = 800f
    const val COUNTER_WIDTH = 500
}

@OptIn(ExperimentalTextApi::class)
val displayLargeFontFamily = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES
    FontFamily(
        Font(
            R.font.robotoflex_variable,
            variationSettings = FontVariation.Settings(
                FontVariation.weight(DisplayLargeVFConfig.WEIGHT),
                FontVariation.width(DisplayLargeVFConfig.WIDTH),
                FontVariation.slant(DisplayLargeVFConfig.SLANT),
            )
        )
    )
} else {
    default
}
```

se/snippets/src/main/java/com/example/compose/snippets/text/VariableFontsSnippets.kt#L100-L123)

5. Configurez la typographie (https://developer.android.com/develop/ui/compose/designsystems/material3?hl=fr#typography) Material Design 3 pour utiliser `FontFamily` :

```
// Type.kt
val Typography = Typography(
    displayLarge = TextStyle(
        fontFamily = displayLargeFontFamily,
        fontSize = 50.sp,
        lineHeight = 64.sp,
        letterSpacing = 0.sp,
        /***/
    )
)
```

ose/snippets/src/main/java/com/example/compose/snippets/text/VariableFontsSnippets.kt#L152-L161)

Cet exemple utilise la typographie Material 3 (https://developer.android.com/develop/ui/compose/designsystems/material3?hl=fr#typography) de `displayLarge`, qui présente des paramètres de police par défaut et des utilisations recommandées différents. Par exemple, vous devez utiliser `displayLarge` pour les textes courts et essentiels, car il s'agit du texte le plus grand à l'écran.

Avec Material 3, vous pouvez modifier les valeurs par défaut de `TextStyle` et `fontFamily` pour personnaliser votre typographie. Dans l'extrait ci-dessus, vous configurez des instances de `TextStyle` pour personnaliser les paramètres de police de chaque famille de polices.

6. Maintenant que vous avez défini votre typographie, transmettez-la au `MaterialTheme` M3 :

```
MaterialTheme(
    colorScheme = MaterialTheme.colorScheme,
    typography = Typography,
    content = content
)
```

se/snippets/src/main/java/com/example/compose/snippets/text/VariableFontsSnippets.kt#L278-L282)

7. Enfin, utilisez un composable `Text` et spécifiez le style pour l'un des styles typographiques définis, `MaterialTheme.typography.displayLarge` :

```
@Composable
@Preview
fun CardDetails() {
    MyCustomTheme {
        Card(
            shape = RoundedCornerShape(8.dp),
            elevation = CardDefaults.cardElevation(defaultElevation = 4.dp),
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp)
        ) {
            Column(
                modifier = Modifier.padding(16.dp)
            ) {
                Text(
                    text = "Compose",
                    style = MaterialTheme.typography.displayLarge,
                    modifier = Modifier.padding(bottom = 8.dp),
                    maxLines = 1
                )
                Text(
                    text = "Beautiful UIs on Android",
                    style = MaterialTheme.typography.headlineMedium,
                    modifier = Modifier.padding(bottom = 8.dp),
                    maxLines = 2
                )
                Text(
                    text = "Jetpack Compose is Android’s recommended modern t
                    style = MaterialTheme.typography.bodyLarge,
                    modifier = Modifier.padding(bottom = 8.dp),
                    maxLines = 3
                )
            }
        }
    }
}
se/snippets/src/main/java/com/example/compose/snippets/text/VariableFontsSnippets.kt#L287-L322)
```

Chaque composable `Text` est configuré via le style de son thème Material et contient une configuration de police variable différente. Vous pouvez utiliser `MaterialTheme.typography` pour récupérer la typographie fournie au composable `MaterialTheme` M3.

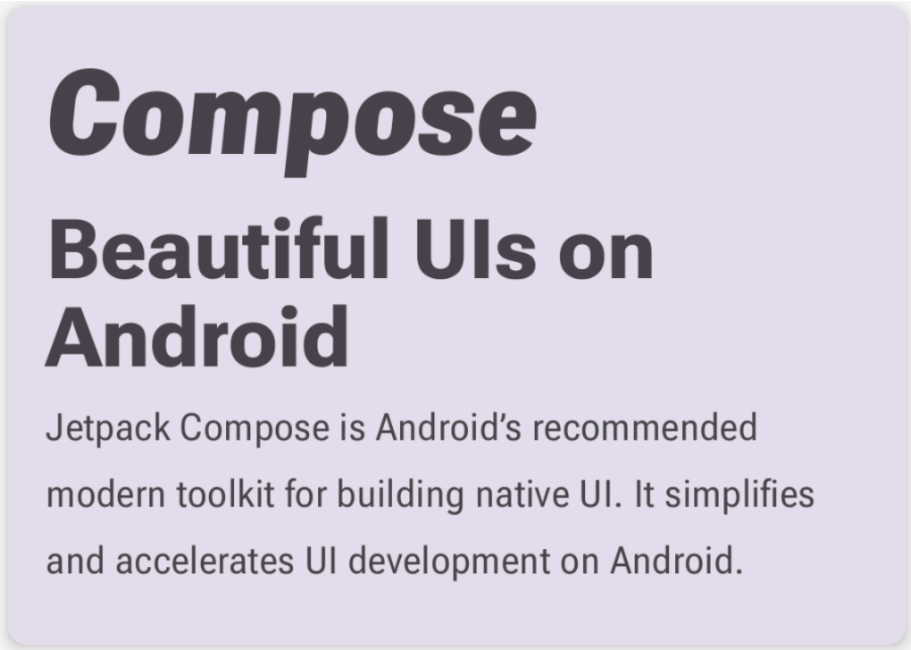


***Figure 2**. Police variable appliquée dans trois configurations différentes.*

Utiliser des axes personnalisés

Les polices peuvent également avoir des axes personnalisés. Elles sont définies dans le fichier de police lui-même. Par exemple, la police Roboto Flex possède l'axe de hauteur d'ascendante (`"YTAS"`), qui ajuste la hauteur des ascendantes en minuscules, tandis que la largeur de contre-forme (`"XTRA"`) ajuste la largeur de chaque lettre.

Vous pouvez modifier la valeur de ces axes avec les paramètres `FontVariation`.

Pour en savoir plus sur les axes personnalisés que vous pouvez configurer pour une police, consultez le tableau des axes compatibles (https://fonts.google.com/variablefonts?hl=fr) pour chaque police.

1. Pour utiliser des axes personnalisés, définissez des fonctions pour les axes `ascenderHeight` et `counterWidth` personnalisés :

```
fun ascenderHeight(ascenderHeight: Float): FontVariation.Setting {
    require(ascenderHeight in 649f..854f) { "'Ascender Height' must be in 649
    return FontVariation.Setting("YTAS", ascenderHeight)
}

fun counterWidth(counterWidth: Int): FontVariation.Setting {
    require(counterWidth in 323..603) { "'Counter width' must be in 323..603"
    return FontVariation.Setting("XTRA", counterWidth.toFloat())
}
se/snippets/src/main/java/com/example/compose/snippets/text/VariableFontsSnippets.kt#L328-L336)
```

Ces fonctions effectuent les opérations suivantes :

- Définissez des garde-fous pour les valeurs qu'ils peuvent accepter. Comme vous pouvez le voir dans le catalogue de polices variables (https://fonts.google.com/variablefonts?vfquery=roboto+flex&hl=fr), `ascenderHeight (YTAS)` a une valeur minimale de `649f` et une valeur maximale de `854f`.
- Renvoie le paramètre de police afin que la configuration soit prête à être ajoutée à la police. Dans la méthode `FontVariation.Setting()`, le nom de l'axe (`YTAS, XTRA`) est codé en dur et prend la valeur comme paramètre.

2. À l'aide des axes avec la configuration de la police, transmettez des paramètres supplémentaires à chaque `Font` chargé :

```
@OptIn(ExperimentalTextApi::class)
val displayLargeFontFamily = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES
    FontFamily(
        Font(
            R.font.robotoflex_variable,
            variationSettings = FontVariation.Settings(
                FontVariation.weight(DisplayLargeVFConfig.WEIGHT),
                FontVariation.width(DisplayLargeVFConfig.WIDTH),
                FontVariation.slant(DisplayLargeVFConfig.SLANT),
                ascenderHeight(DisplayLargeVFConfig.ASCENDER_HEIGHT),
                counterWidth(DisplayLargeVFConfig.COUNTER_WIDTH)
            )
        )
    )
} else {
    default
}
```


Notez que la hauteur des ascendantes en minuscules a été augmentée et que le reste du texte est plus large :

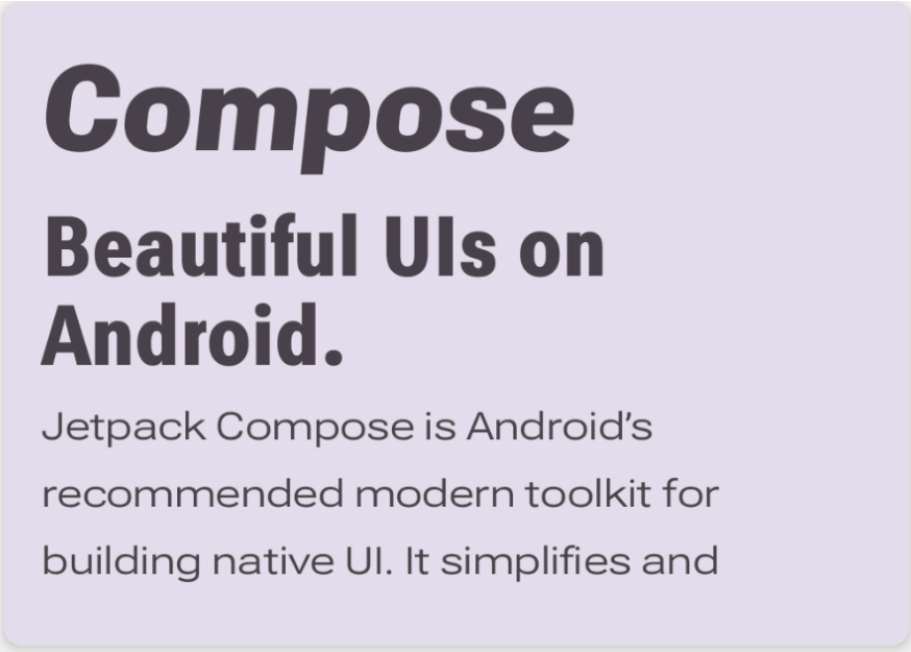


Figure 3 *: Texte affichant des axes personnalisés définis sur des polices variables.*

Ressources supplémentaires

Pour en savoir plus, consultez l'article de blog suivant sur les polices variables :

- Just your type: Variable fonts in Compose (https://medium.com/androiddevelopers/just-your-type-variable-fonts-in-compose-5bf63b357994)

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