Android Studio

Le XML

Qu’est ce qu’une ressource?

Les ressources sont des fichiers qui contiennent des informations qui ne sont :

- Pas en Java (ce n'est donc pas du code).
- Pas dynamique (le contenu d'une ressource restera inchangé entre le début de l'exécution de votre application et la fin de l'exécution).

Android est destiné à être utilisé sur un très grand nombre de supports différents, et il faut par conséquent s'adapter à ces supports. L'avantage des ressources, c'est qu'elles nous permettent de nous adapter facilement à toutes ces situations différentes.

Exemples

Votre application affiche une image. Cette image est constituée de pixels comme une photo par exemple, alors il faudra adapter sa taille à l'écran. Si la photo est petite, il faudra agrandir l'image et du coup elle va perdre en qualité. L'idéal sera donc de préparer à l'avance plusieurs images à afficher en fonction de la résolution de l'écran.

Vous avez fait votre application en français , elle a du succès, et on vous demande de la traduire en plusieurs langues. Aïe, tous vos textes se trouvent dans le code, comment allez-vous faire pour retrouver chaque texte et le traduire? Heureusement avec les ressources, vous pouvez organiser facilement les chaînes de caractères pour plus facilement travailler avec.

Pour déclarer des ressources, on passe très souvent par le format XML, c'est pourquoi un point sur ce langage est nécessaire.

XML: un langage de balises

Extensible Markup Language « langage de balisage extensible » en français

Le XML est un langage de balisage un peu comme le HTML — le XML est d'ailleurs indirectement un dérivé du HTML. Le principe d'un langage de programmation (Java, C++, etc.) est de donner des ordres à l'ordinateur pour qu'il puisse effectuer des calculs, puis éventuellement de mettre en forme le résultat de ces calculs dans une interface graphique.

À l'opposé, un langage de balisage (comme le XML) n'a pas pour objectif de donner des ordres à l'ordinateur, il se contente de définir une façon de mettre en forme l'information de façon à ce qu'on sache comment lire cette information. Ainsi, il est possible de développer un programme appelé **interpréteur** qui récupérera les données d'un fichier (structuré à l'aide d'un langage de balisage). Par exemple pour le HTML, c'est un navigateur qui interprète le code afin de donner un sens aux instructions. Si vous lisez un document HTML sans interpréteur, vous ne verrez que les sources, pas l'interprétation des balises.

Exemple pratique

Prenons l’exemple d’un fichier ou sont stockés des contacts:

```
1    Paul Pierre Jacques franck
```

Ce langage est très simple, les prénoms sont séparés par des espaces, on peut dire à l’ordinateur de lire la ligne et à chaque fois qu’il voit un espace c’est qu’on est arrivé à la fin d’un prénom.

```
1
2       Paul : 686878798
3       Pierre : 68675555
4       Jacques : 6868689
5       franck : 786755657
6
```

Là, pour l'interpréteur il est facile de comprendre que pour chaque ligne la suite de caractère est un prénom se terminant par deux-points, puis on trouve le numéro, et qu'après le retour à la ligne on passe à l'enregistrement suivant.

```
Paul : 686
    878798
Pier

    re : 68675555
Jacq

    ues : 6868689
franck : 786755657
```

Ici sans syntaxe particulière à respecter, mon interpréteur est incapable de lire mes données. Il est donc impossible de coder un interpréteur qui sache associer un prénom à un numéro de téléphone.

La syntaxe XML

Cette ligne permet d'indiquer que :

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <resources>
```

On utilise la version 1.0 de XML.

On utilise l'encodage des caractères qui s'appelle utf-8; c'est une façon de décrire les caractères que contiendra notre fichier.

```xml
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="AppTheme.NoActionBar">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
    </style>

    <style name="AppTheme.AppBarOverlay" parent="ThemeOverlay.AppCompat.Dark.ActionBar" />

    <style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />

</resources>
```

L'élément de base du format XML est la balise. Elle commence par un chevron ouvrant < et se termine par un chevron fermant >. Entre ces deux chevrons, on trouve un mot. Par exemple <resources>. Il faudra obligatoirement la fermer (la balise)

Il existe deux manières de fermer une balise ouvrante :

- Soit par une balise fermante (dans notre cas </resources>), auquel cas vous pourrez avoir du texte ou d'autres balises entre la balise ouvrante et la balise fermante. Étant donné que notre <resources> est destinée à contenir plusieurs caractéristiques, nous avons opté pour cette solution.
- Soit on ferme la balise directement dans son corps :<style/>. La seule différence est qu'on ne peut pas mettre de contenu entre deux balises… puisqu'il n'y en a qu'une. Dans notre exemple,

```
<style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />
```

Ce type d'informations, qu'il soit fermé par une balise fermante ou qu'il n'en n'ait pas besoin, s'appelle un nœud. Vous voyez donc que l'on a un nœud appelé resources, quatre nœuds appelés style, etc.

```xml
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="AppTheme.NoActionBar">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
    </style>

    <style name="AppTheme.AppBarOverlay" parent="ThemeOverlay.AppCompat.Dark.ActionBar" />

    <style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />

</resources>
```

Un langage de balisage n'a pas de sens en lui-même. Dans notre exemple, notre nœud s'appelle Resources, on en déduit, nous humains qu'il représente des ressources, mais si on avait décidé de l'appeler "fkldjsdfljsdfkls", il aurait autant de sens au niveau informatique. C'est à vous d'attribuer un sens à votre fichier XML au moment de l'interprétation.

Le nœud <resources>, qui est le nœud qui englobe tous les autres nœuds, s'appelle la racine. Il y a dans un fichier XML au moins une racine et au plus une racine. Oui ça veut dire qu'il y a exactement une racine par fichier.

On peut établir toute une hiérarchie dans un fichier XML. En effet, entre la balise ouvrante et la balise fermante d'un nœud, il est possible de mettre d'autres nœuds. Les nœuds qui se trouvent dans un autre nœud s'appellent des **enfants**, et le nœud encapsulant s'appelle le **parent**.

Les nœuds peuvent avoir des attributs pour indiquer des informations.
Dans notre exemple:

```
<item name="windowActionBar">false</item>
<item name="windowNoTitle">true</item>
```

le nœud <item> a l'attribut name afin de préciser quel nom il a:

```
<item name="windowActionBar">false</item>
```

avec une valeur false

```
<item name="windowNoTitle">true</item>
```

avec une valeur true

Attention le format XML en lui-même ne peut pas détecter si l'absence de l'attribut name est une anomalie, cela retirerait toute la liberté que permet le format.

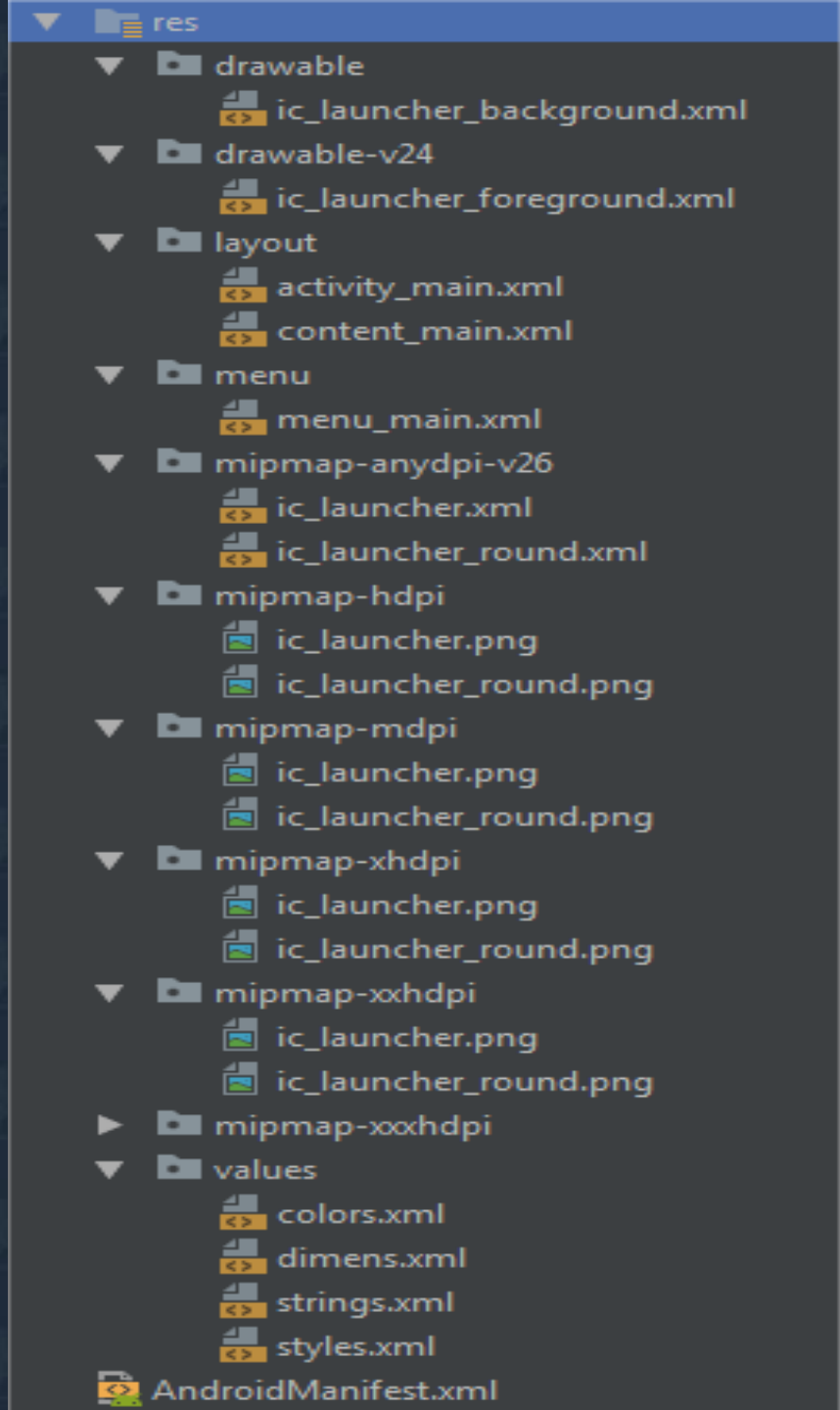
En revanche, le XML est intransigeant sur la syntaxe. Si vous ouvrez une balise, n'oubliez pas de la fermer.

Revenons sur les ressources

Les ressources sont des éléments capitaux dans une application Android. Comme Android est destiné à être utilisé sur une grande variété de supports, il fallait trouver une solution pour permettre à une application de s'afficher de la même manière sur un écran 7" que sur un écran 10", ou faire en sorte que les textes s'adaptent à la langue de l'utilisateur. C'est pourquoi les différents éléments qui doivent s'adapter de manière très précise sont organisés de manière tout aussi précise, de façon à ce qu'Android sache quels éléments utiliser pour quels types de terminaux.

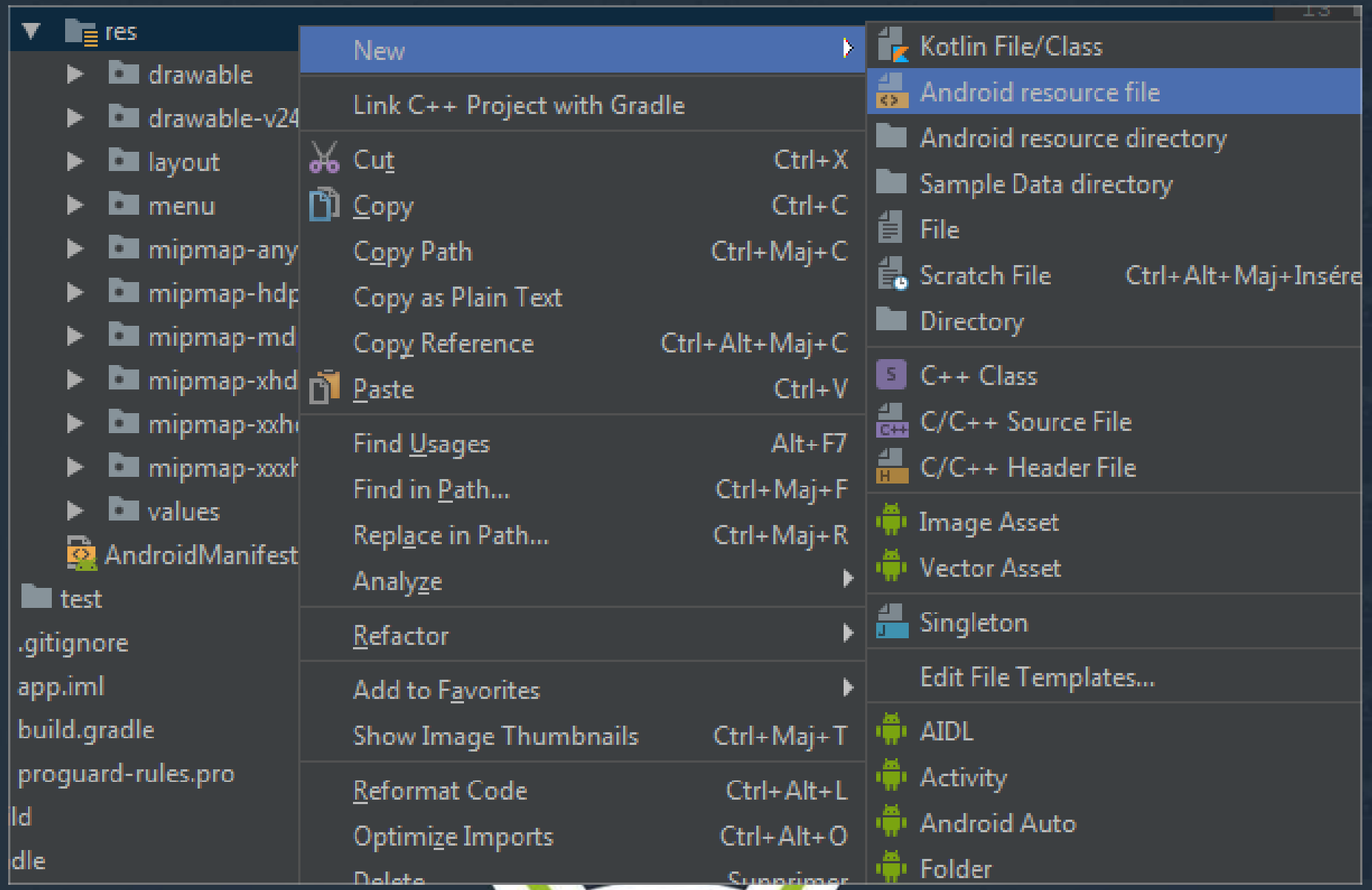
Comme vu dans la présentation de l'interface, à la création d'un nouveau projet, Android Studio crée certains répertoires par défaut, comme le montre la figure

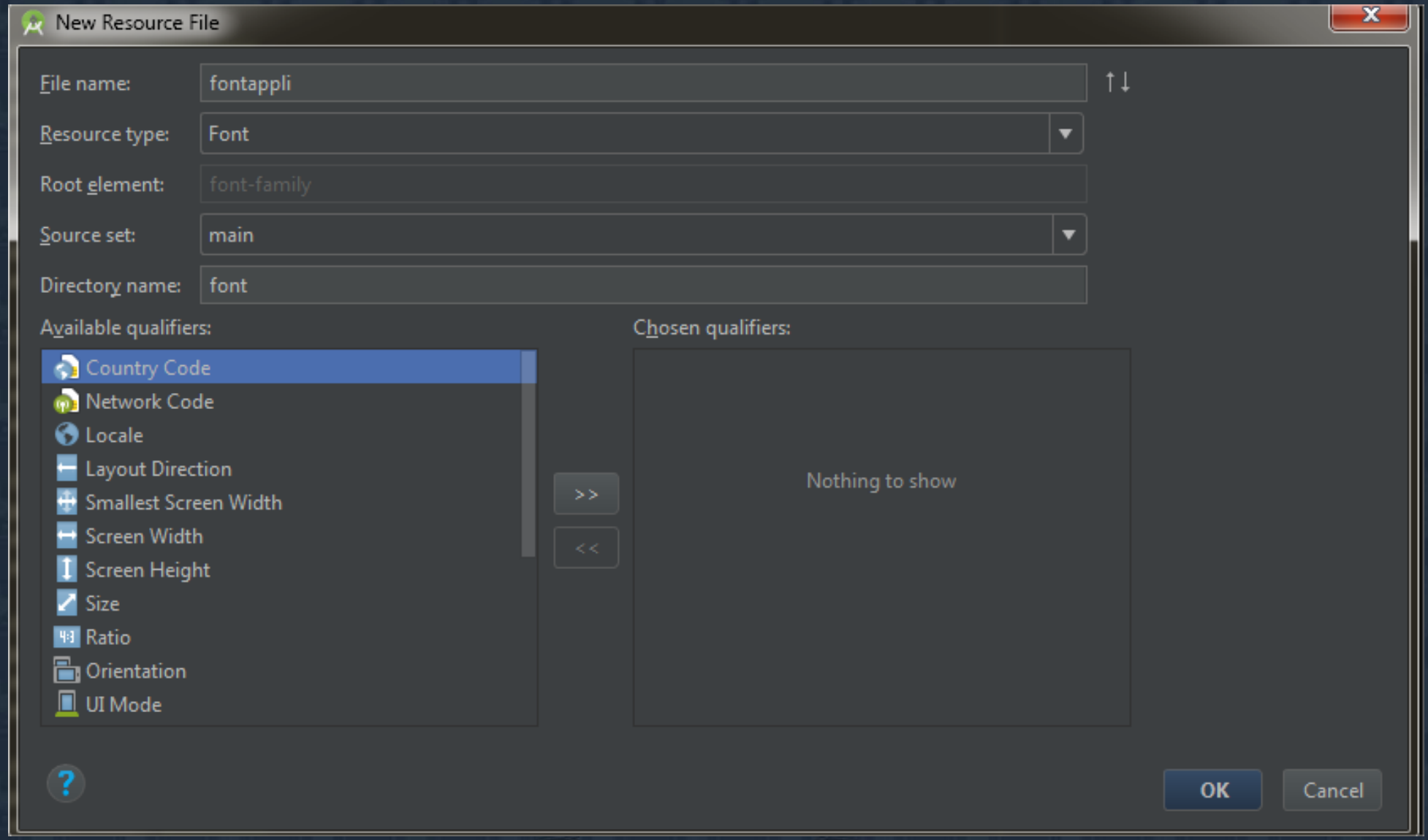
Type	Description	Présence de fichiers XML
Dessin et image (`res/drawable`)	On y trouve les images matricielles (les images de type PNG, JPEG ou encore GIF) ainsi que des fichiers XML qui permettent de décrire des dessins (ce qui donne des images vectorielles qui ne se dégradent pas quand on les agrandit).	Oui
Mise en page ou interface graphique (`res/layout`)	Les fichiers XML qui représentent la disposition des vues (on abordera cet aspect, qui est très vaste, dans la prochaine partie).	Exclusivement
Menu (`res/menu`)	Les fichiers XML pour pouvoir constituer des menus.	Exclusivement
Donnée brute (`res/raw`)	Données diverses au format brut. Ces données ne sont pas des fichiers de ressources standards, on pourrait y mettre de la musique ou des fichiers HTML par exemple.	Le moins possible
Différentes variables (`res/values`)	Il est plus difficile de cibler les ressources qui appartiennent à cette catégorie tant elles sont nombreuses. On y trouve entre autre des variables standards, comme des chaînes de caractères, des dimensions, des couleurs, etc.	Exclusivement



Ajouter une ressource avec Android Studio

Heureusement, les développeurs d'Android Studio ont pensé à nous en créant un petit menu qui vous aidera à créer des répertoires de manière simple, sans avoir à retenir de syntaxe. En revanche, il vous faudra parler un peu anglais, je le crains. Faites un clic droit sur res . Vous aurez un menu un peu similaire à celui représenté à l'image suivante, qui s'affichera.


res
drawable
drawable-v24
layout
menu
mipmap-any
mipmap-hdp
mipmap-md
mipmap-xhd
mipmap-xxh
mipmap-xxxh
values
AndroidManifest
test
.gitignore
app.iml
build.gradle
proguard-rules.pro
ld
dle
New
Link C++ Project with Gradle
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Maj+C
Copy as Plain Text
Copy Reference Ctrl+Alt+Maj+C
Paste Ctrl+V
Find Usages Alt+F7
Find in Path... Ctrl+Maj+F
Replace in Path... Ctrl+Maj+R
Analyze
Refactor
Add to Favorites
Show Image Thumbnails Ctrl+Maj+T
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Kotlin File/Class
Android resource file
Android resource directory
Sample Data directory
File
Scratch File Ctrl+Alt+Maj+Insére
Directory
C++ Class
C/C++ Source File
C/C++ Header File
Image Asset
Vector Asset
Singleton
Edit File Templates...
AIDL
Activity
Android Auto
Folder


New Resource File
File name:
fontappli
Resource type:
Font
Root element:
font-family
Source set:
main
Directory name:
font
Available qualifiers:
Country Code
Network Code
Locale
Layout Direction
Smallest Screen Width
Screen Width
Screen Height
Size
Ratio
Orientation
UI Mode
>>
<<
Chosen qualifiers:
Nothing to show
OK
Cancel