

Formations

Tristan SALAÛN

Pour Kotlin, Android (Java et Kotlin)
premier niveau et avancé

Supports disponibles

- Android premier niveau.
- Android, perfectionnement (Réf. IOD).
- Kotlin.
- Kotlin, développer des applications pour Android (Réf. OTA).
- Kotlin mise en oeuvre (Réf. OTB).

Utilisation de cette présentation

Appuyons sur la touche ?

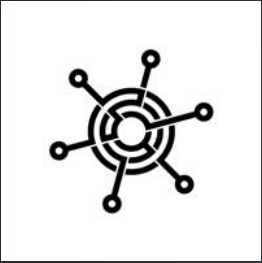
Présentations

- Qui suis-je.
- Qui êtes-vous ? Quelles sont vos attentes.
- Présentation du plan.

Qui suis-je

Présentations

- Développeur d'applications mobiles Android (depuis 2009).
- Formateur Android/Java et Kotlin, et Javacard (scolaires et pros).
- Co-fondateur (1/7) de [Startup Marseille](#).



- Fondateur de [Light4Events](#).



Présentations

Qui êtes-vous ?

Tour de table :

- Prénom.
- Expérience (Java, Kotlin, Mobile Android/iOS).
- Attentes.

Mise en place

Étapes de mise en place :

- [Android Studio](#).
- [ADB](#).
- Téléphone en mode développeur.
- Coloration de logcat.
- Mise en place des templates.
- Mise en place de scrcpy.

Mise en place

Android Studio

Pour commencer, nous allons vérifier que nous avons bien Android Studio fonctionnel avec la dernière version :



Mise en place

ADB

Vérifions que ADB est bien accessible en ligne de commande, cela nous sera utile par la suite.

- Ouvrons une invite de commande (Windows + R, cmd).
- Saisissons : `adb version`.

Si ce n'est pas le cas, ajoutons le au "PATH".

Mise en place

Téléphone en mode développeur

Nous vérifions que notre téléphone est en mode développeur :

- En ligne de commande tapons : `adb devices`
- Vérifions dans Android Studio que le device apparaît bien.

Dans le cas contraire, nous vérifions que le téléphone a bien le mode développeur activé.

Coloration de logcat

Mise en place

Nous allons colorer le logcat afin qu'il soit plus lisible :

- File / Settings
- Cherchons : "logcat"
- Editor / Color Scheme / Android Logcat
- Les couleurs que j'ai définies (à titre d'exemple) :

Assert	C00000
--------	--------

Debug	41BB2E
-------	--------

Error	FF6B68
-------	--------

Info	F5F550
------	--------

Verbose	BBBBBB
---------	--------

Warning	F4AC40
---------	--------

LiveTemplates

Mise en place

Les LiveTemplates, sont des raccourcis qui permettent d'écrire rapidement des bouts de code. Je vous fournis une série de LiveTemplates que nous utiliserons dans la suite de la formation. Procédons comme suit :

- Récupération des templates :
 - [Projet GitHub LiveTemplates](#)
 - Cliquons sur le bouton Code
 - Download ZIP
- Les copier dans :
 - Windows: `C:\Users<your_user>\AppData\Roaming\Google\AndroidStudio2022.3\templates` (par ex, selon la version installée)
 - Linux: `~/.AndroidStudio"version"/config/templates`
 - macOS: `~/Library/Preferences/AndroidStudio"version"/templates`
- File/Invalidate caches/Restart...
- Just Restart.
- File/Settings.

Mise en place

Utilisation

Pour utiliser les LiveTemplates, il suffit de taper le début de l'abréviation, lancer l'autocomplétion et le LiveTemplate sera exécuté.

- Par exemple tous les lives templates pour compose `and_compose_`.
- Par exemple tous les lives templates pour les tests UIAutomator commencent par `and_uia_`.
- Les autres templates commencent tous par `and_` et sont filtrés par le type de fichier dans lequel on se trouve (XML, Java, Kotlin).

Mise en place

Shell scripts

Afin d'automatiser certaines tâches, il est possible de les scripter. Pour cela je vous propose une série de scripts que vous pourrez utiliser après la formation :

Récupération des scripts :

- [Projet GitHub AndroidBashScripts](#)
- Cliquons sur le bouton Code
- Download ZIP

Pour les lancer sous Windows il faudra installer [Git](#).

Mise en place

scrcpy

Nous allons installer scrcpy qui nous permettra de partager l'affichage de notre téléphone.

- Téléchargeons la dernière version sur [GitHub](#).
- Décompressons le fichier ZIP dans un répertoire (`C:\tools` par exemple) ce qui donnera dans notre cas : `C:\tools\scrcpy-win64-v2.3.1`.
- Lançons l'utilitaire avec `scrcpy`.

Une solution si vous avez installé winget est d'utiliser la commande suivante :

```
1 winget install --silent --id=Genymobile.scrcpy -e
```

Copier

Réalisation d'une application Android simple en Kotlin

Nous allons commencer par poser les bases d'une application Android en Kotlin.

Nous allons suivre les étapes suivantes :

- Installer et lancer Android Studio.
- Créer un projet : IMC.
- Mettre en place le view binding.
- Définir notre interface graphique.
- Implémenter le code métier.

Réalisation d'une application Android simple en Kotlin

Installer et lancer Android Studio

Il suffit d'avoir une bonne connexion internet, et d'aller sur <https://developer.android.com/studio>.

Réalisation d'une application Android simple en Kotlin

Créer un projet : IMC

Réalisation d'une application Android simple en Kotlin

Mettre en place le view binding

Pour cela, nous allons suivre les indications de la page : [View Binding](#).

Réalisation d'une application Android simple en Kotlin

Modifions le build.gradle (Module :app)

```
1 android {  
2     ...  
3     buildFeatures {  
4         viewBinding true  
5     }  
6 }
```

Copier

Si l'on est en kotlin (build.gradle.kt) :

```
1 android {  
2     ...  
3     buildFeatures {  
4         viewBinding = true  
5     }  
6 }
```

Copier

Modifions la méthode onCreate de notre Activity

```
1 override fun onCreate(savedInstanceState: Bundle?) {  
2     super.onCreate(savedInstanceState)  
3     binding = ResultProfileBinding.inflate(layoutInflater)  
4     val view = binding.root  
5     setContentView(view)  
6 }
```

Copier

Réalisation d'une application Android simple en Kotlin

Usage

Nous pouvons maintenant utiliser nos éléments graphiques directement depuis la classe binding, par exemple :

```
1 binding.name.text = viewModel.name  
2 binding.button.setOnClickListener { viewModel.userClicked() }
```

[Copier](#)

Réalisation d'une application Android simple en Kotlin

Définir une première interface graphique



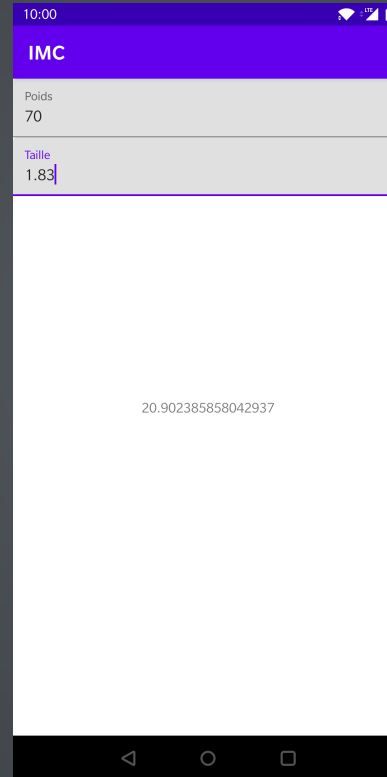
Réalisation d'une application Android simple en Kotlin

Implémenter les méthodes de base

- Ajouter un tag aux boutons avec des valeurs différentes.
- Définit une callback qui sera appelée par le `setOnClickListener`.
- Assigner la callback à nos 3 boutons en utilisant `with`.

Définir notre interface graphique

L'application IMC



L'application IMC

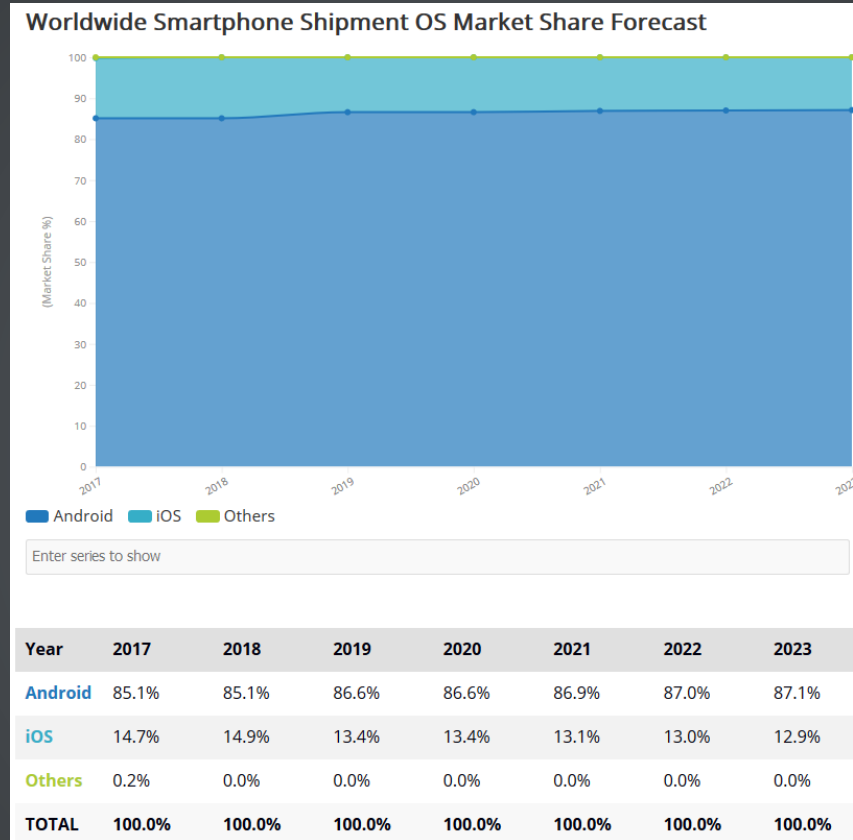
Implémenter le code métier

- Écouter les changements de valeur des champs textes.
- Calculer et afficher la valeur de l'IMC avec la formule : $\text{poids} / \text{taille (en m)}^2$.
- Afficher des indications supplémentaires.
- Enregistrer la taille.

La plate-forme Android

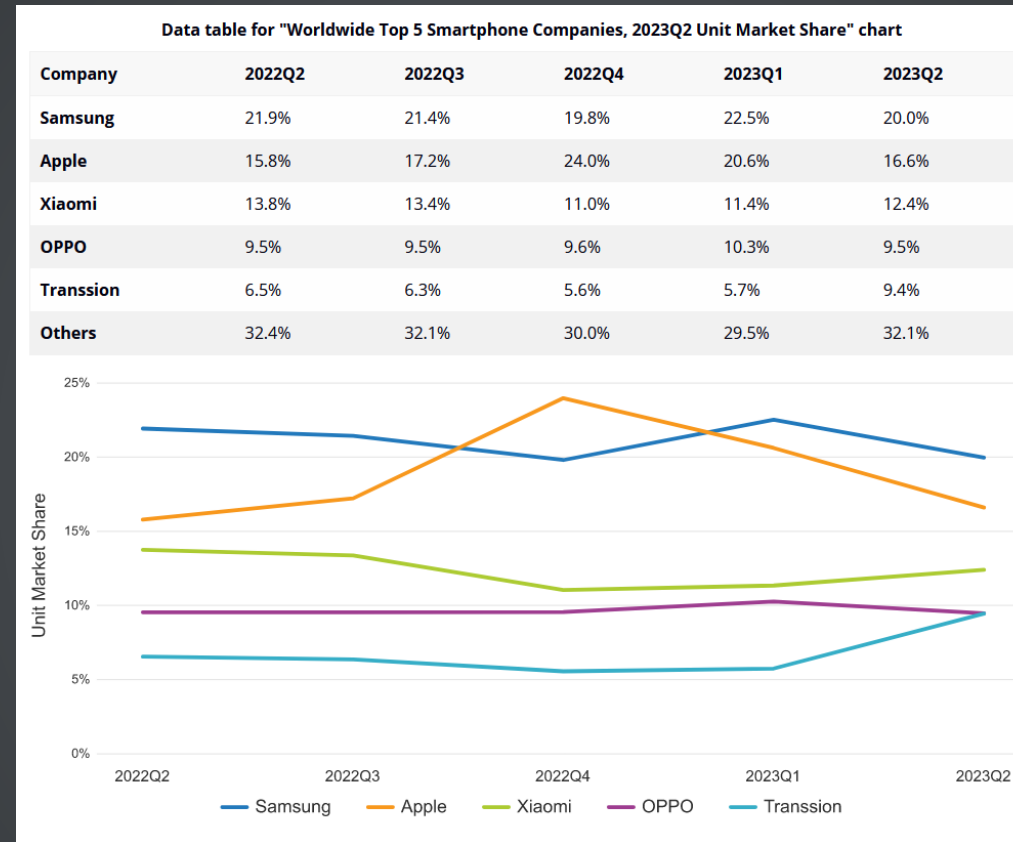
- L'architecture Android, Linux. Historiques et fonctionnalités.
- Les terminaux cibles.

Android dans le monde



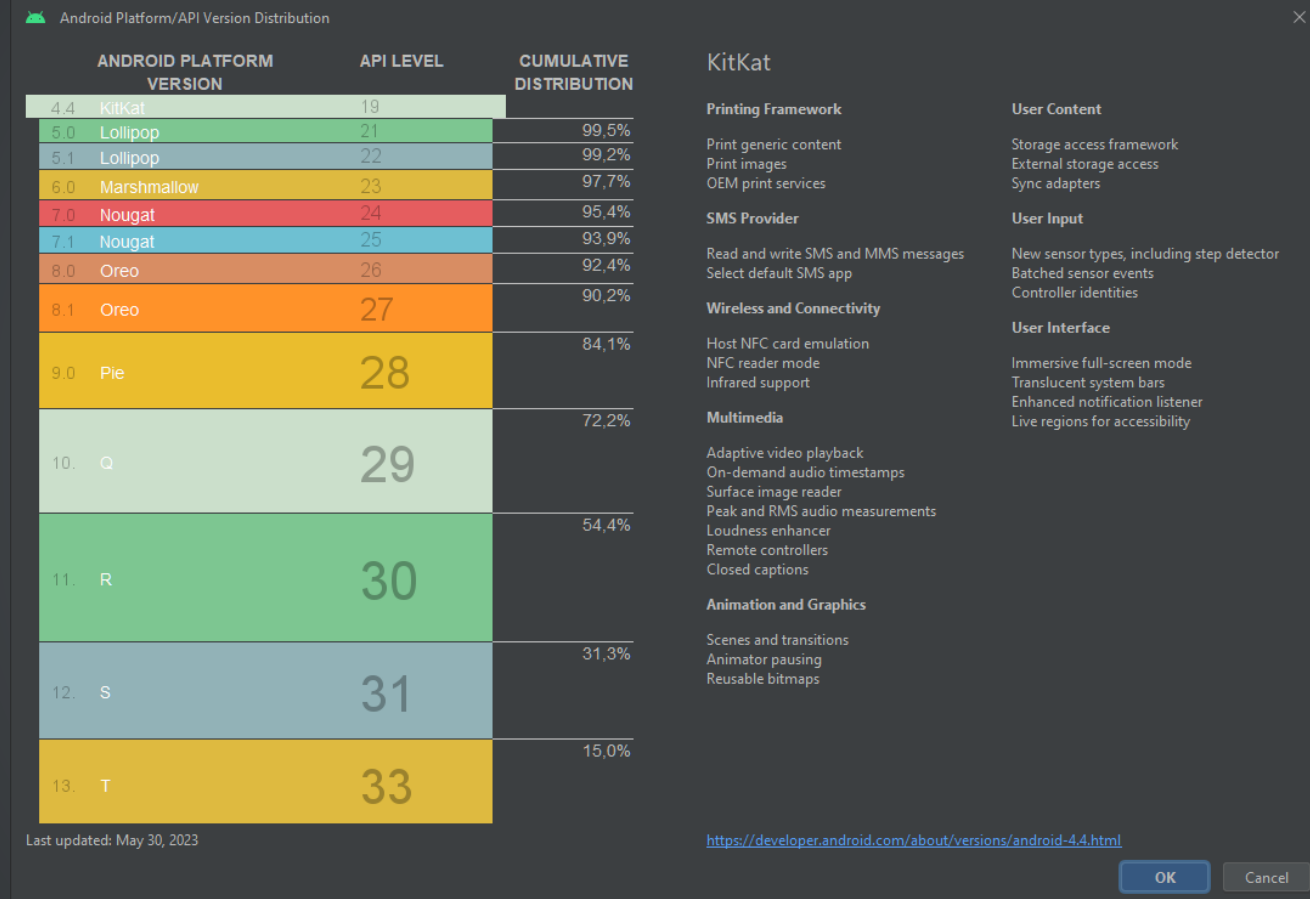
Source : <https://www.idc.com/promo/smartphone-market-share/os>

Android dans le monde

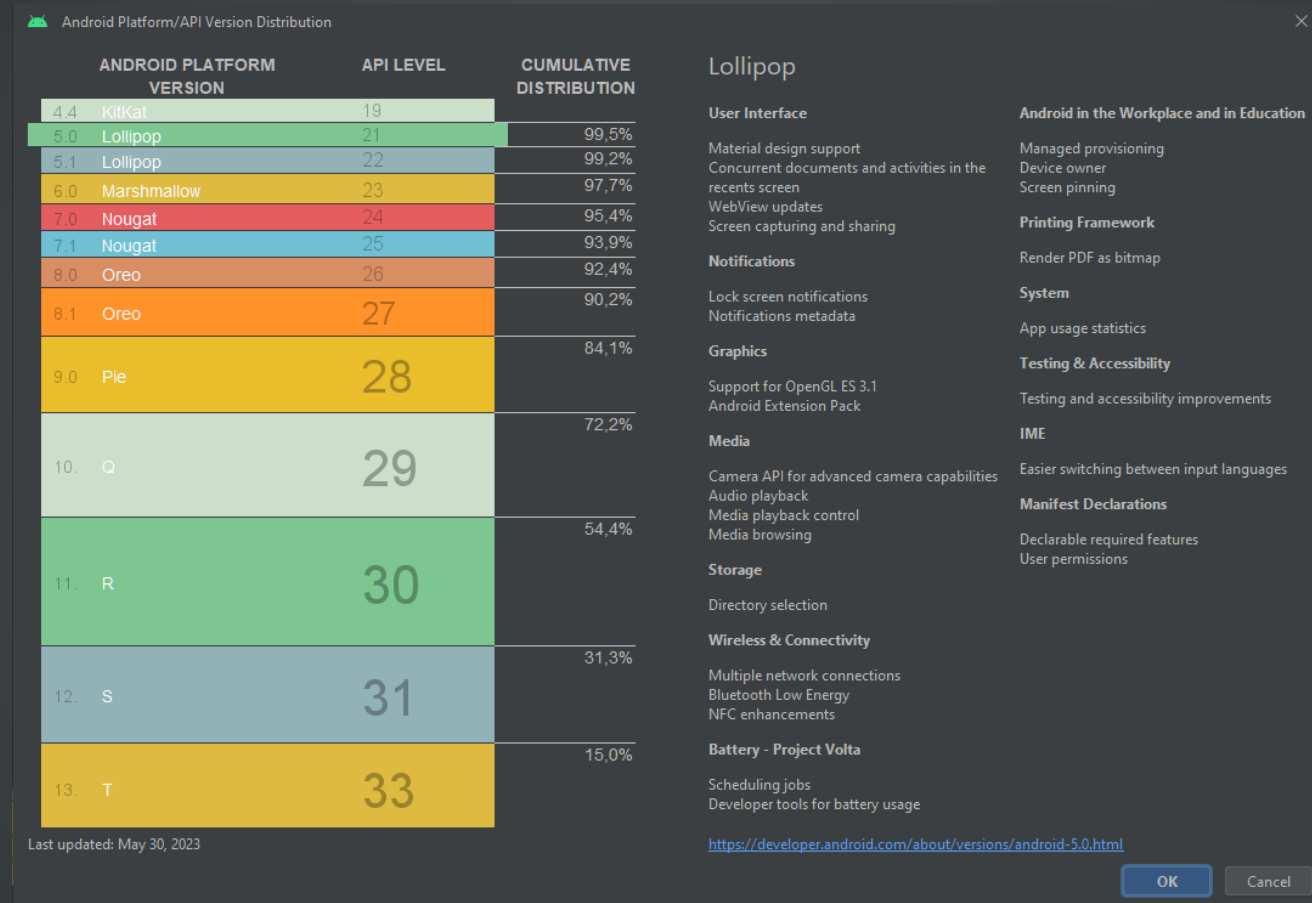


Source : <https://www.idc.com/promo/smartphone-market-share/vendor>

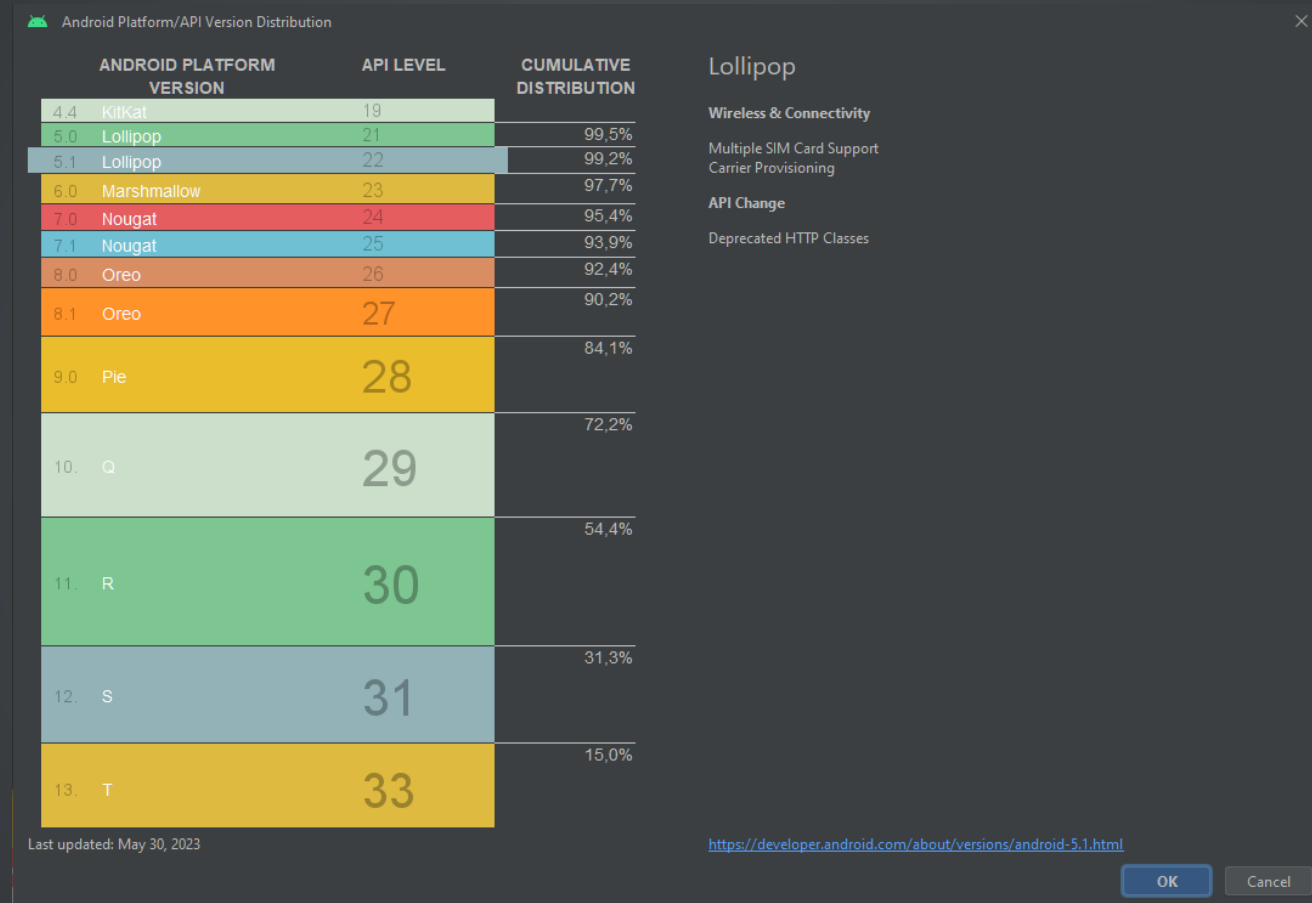
L'histoire d'Android



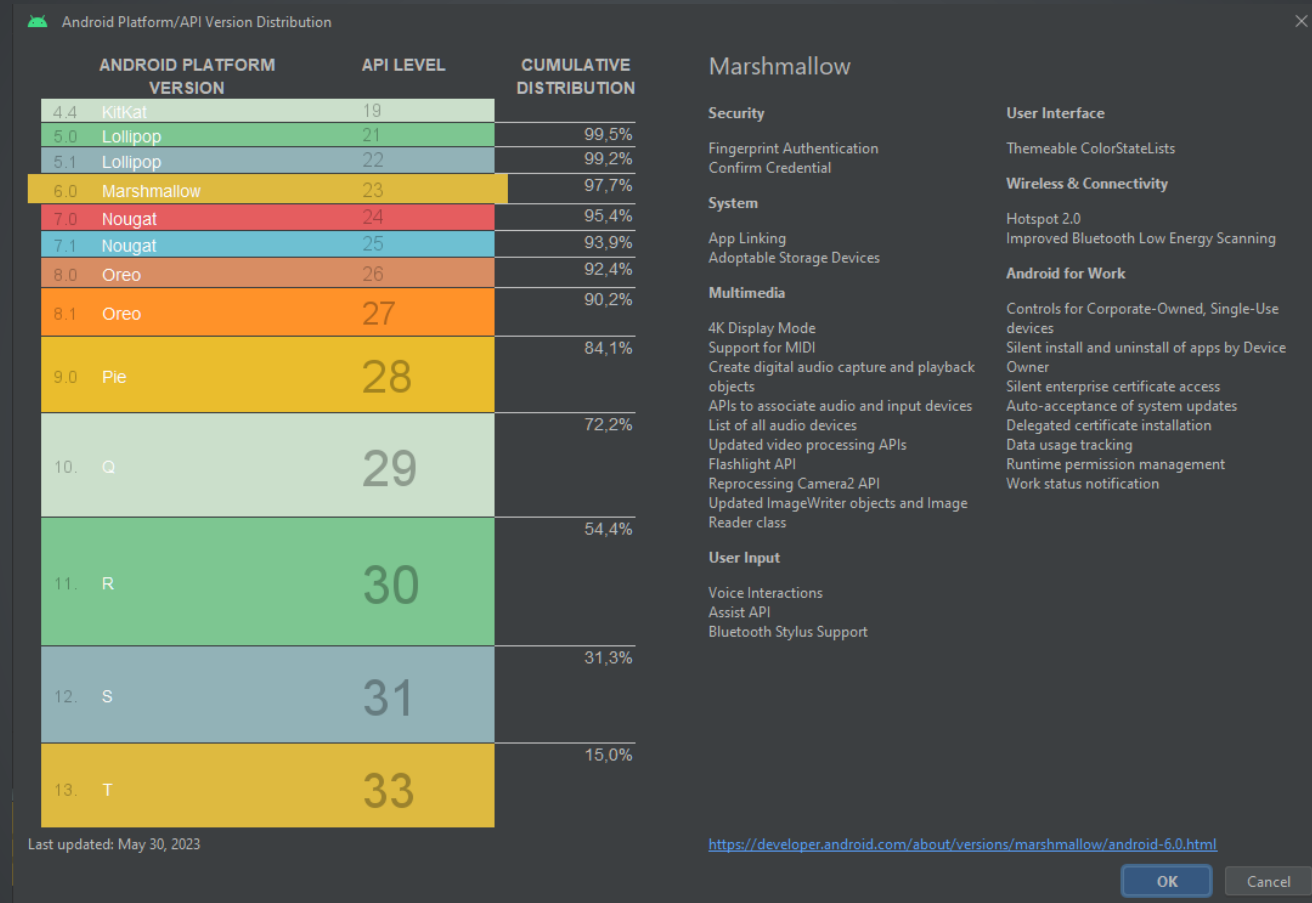
L'histoire d'Android



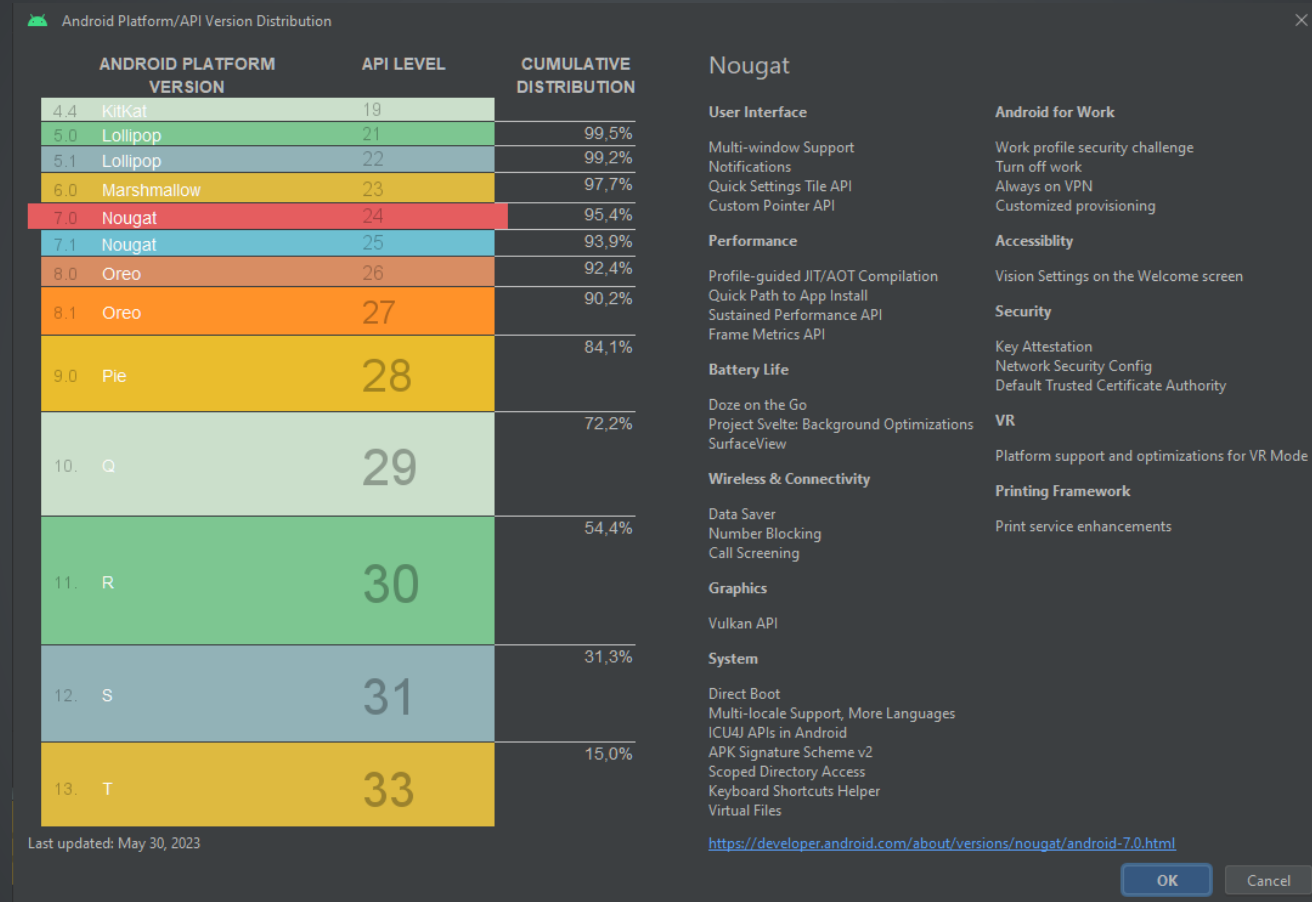
L'histoire d'Android



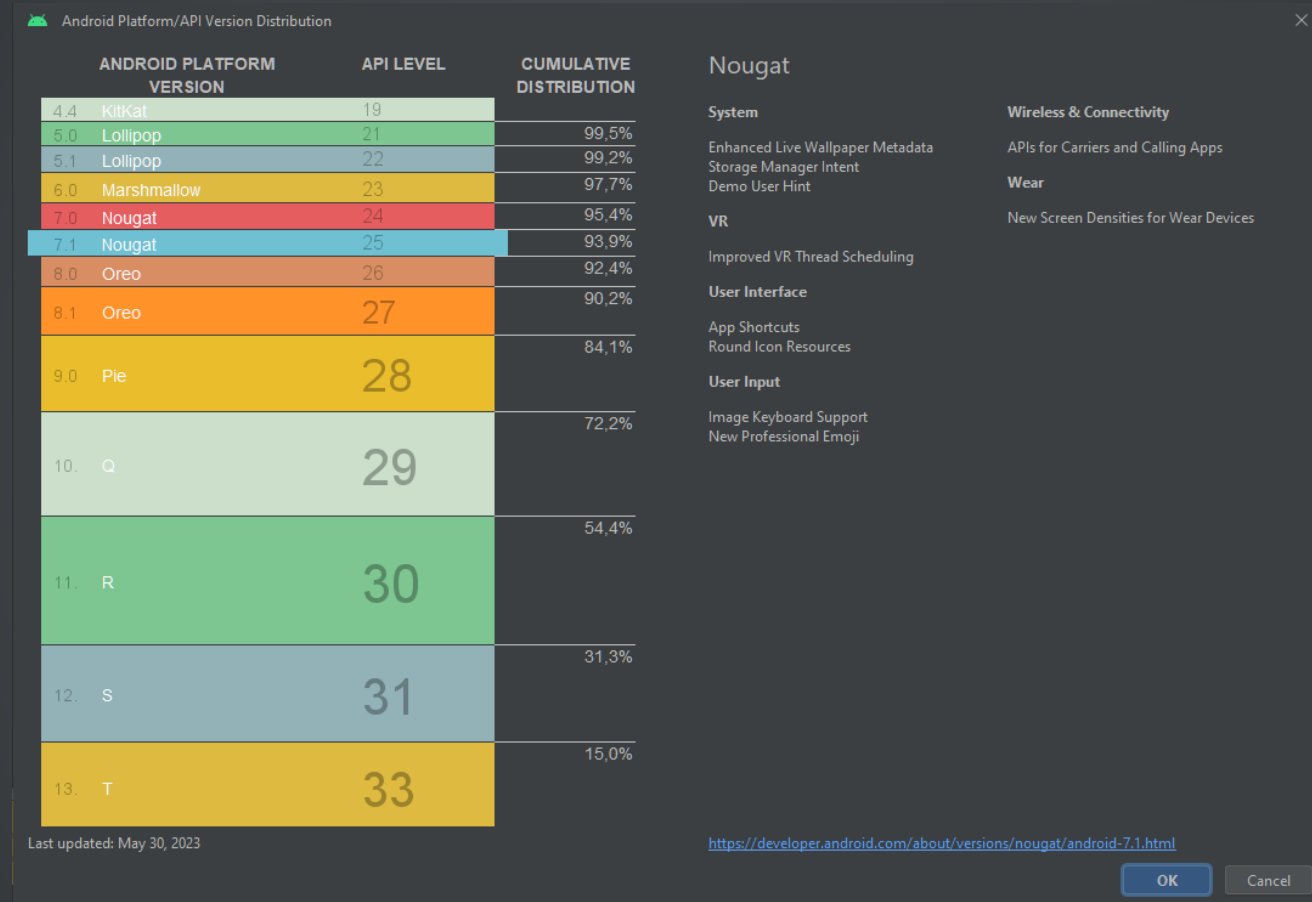
L'histoire d'Android



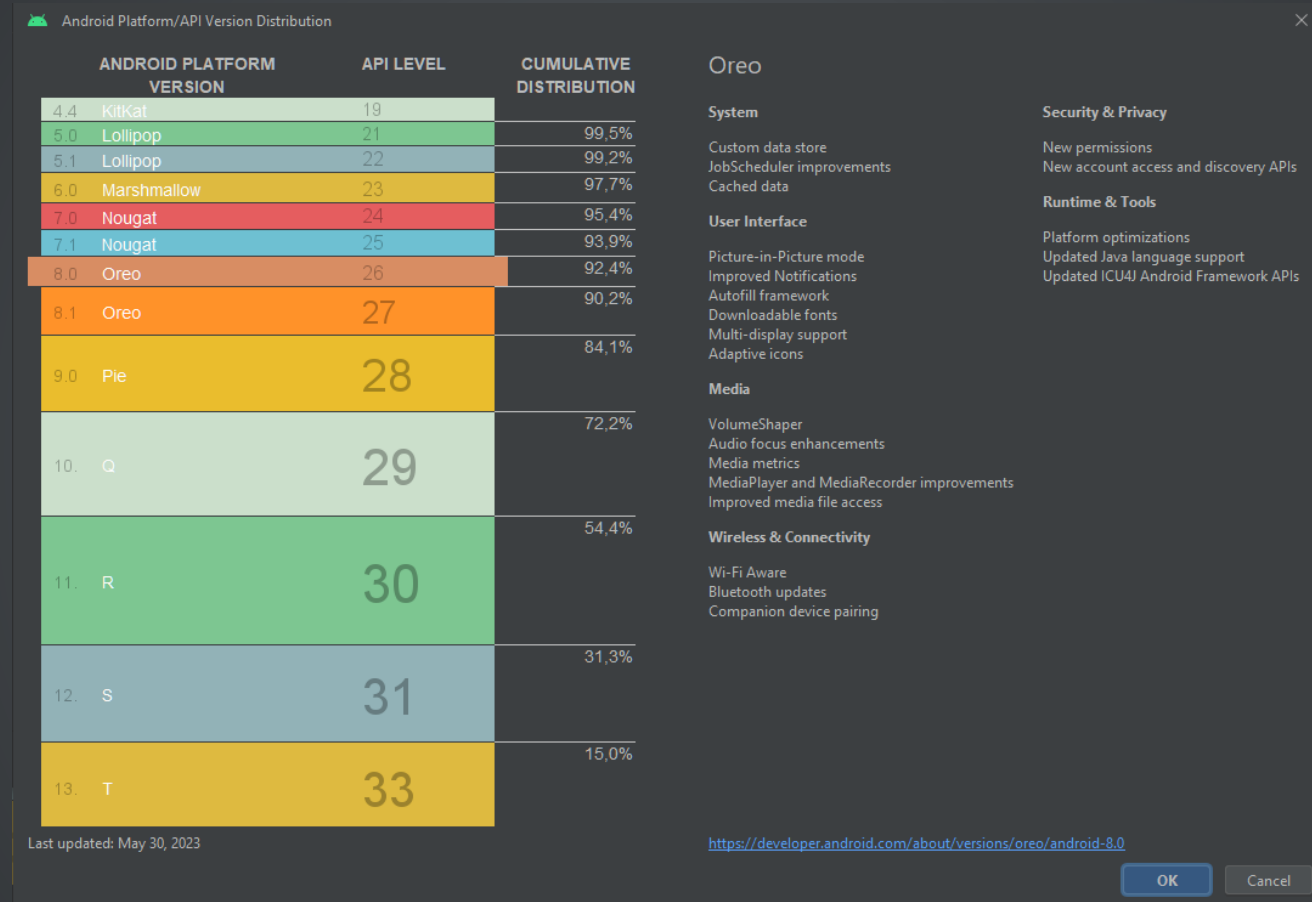
L'histoire d'Android



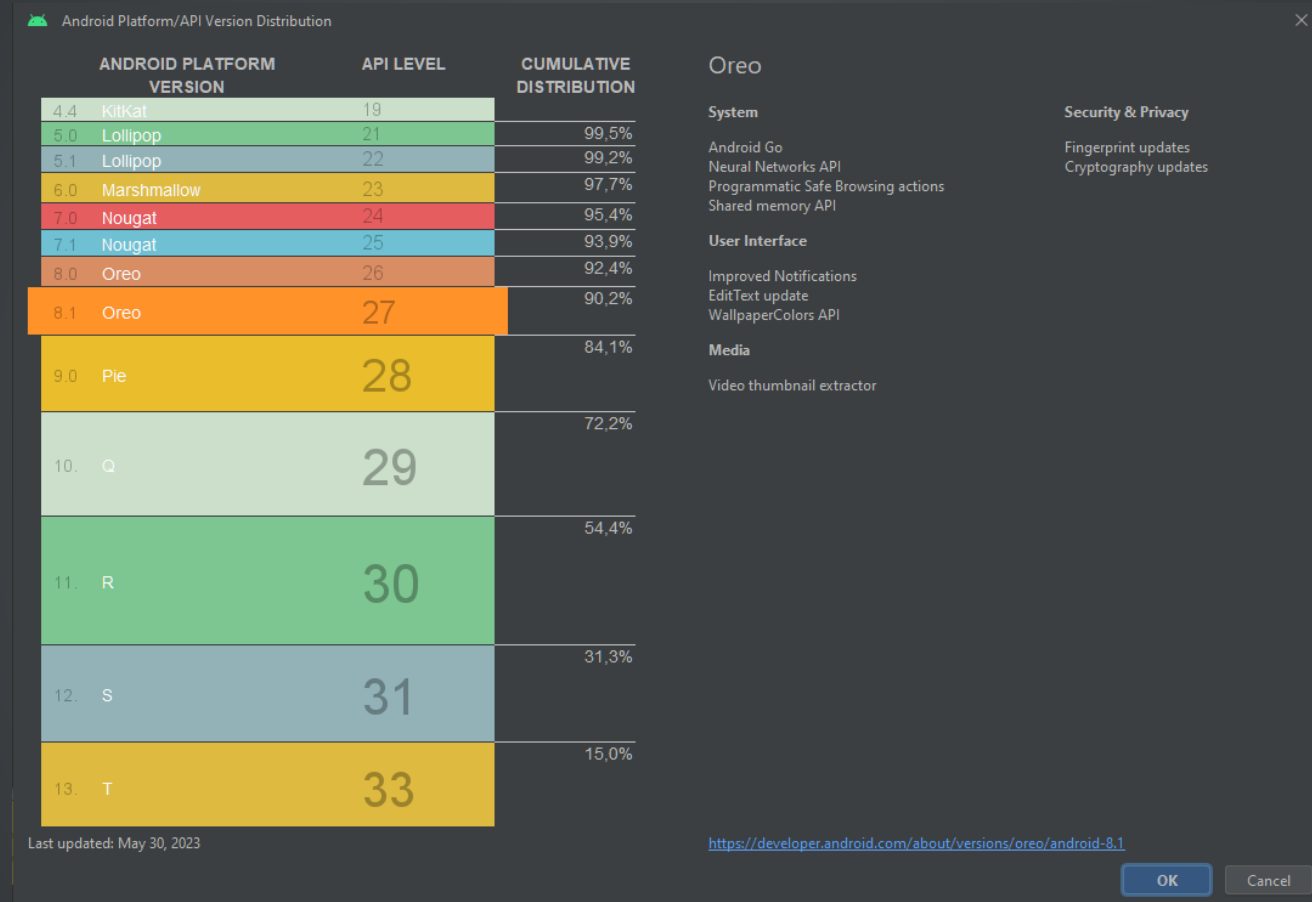
L'histoire d'Android



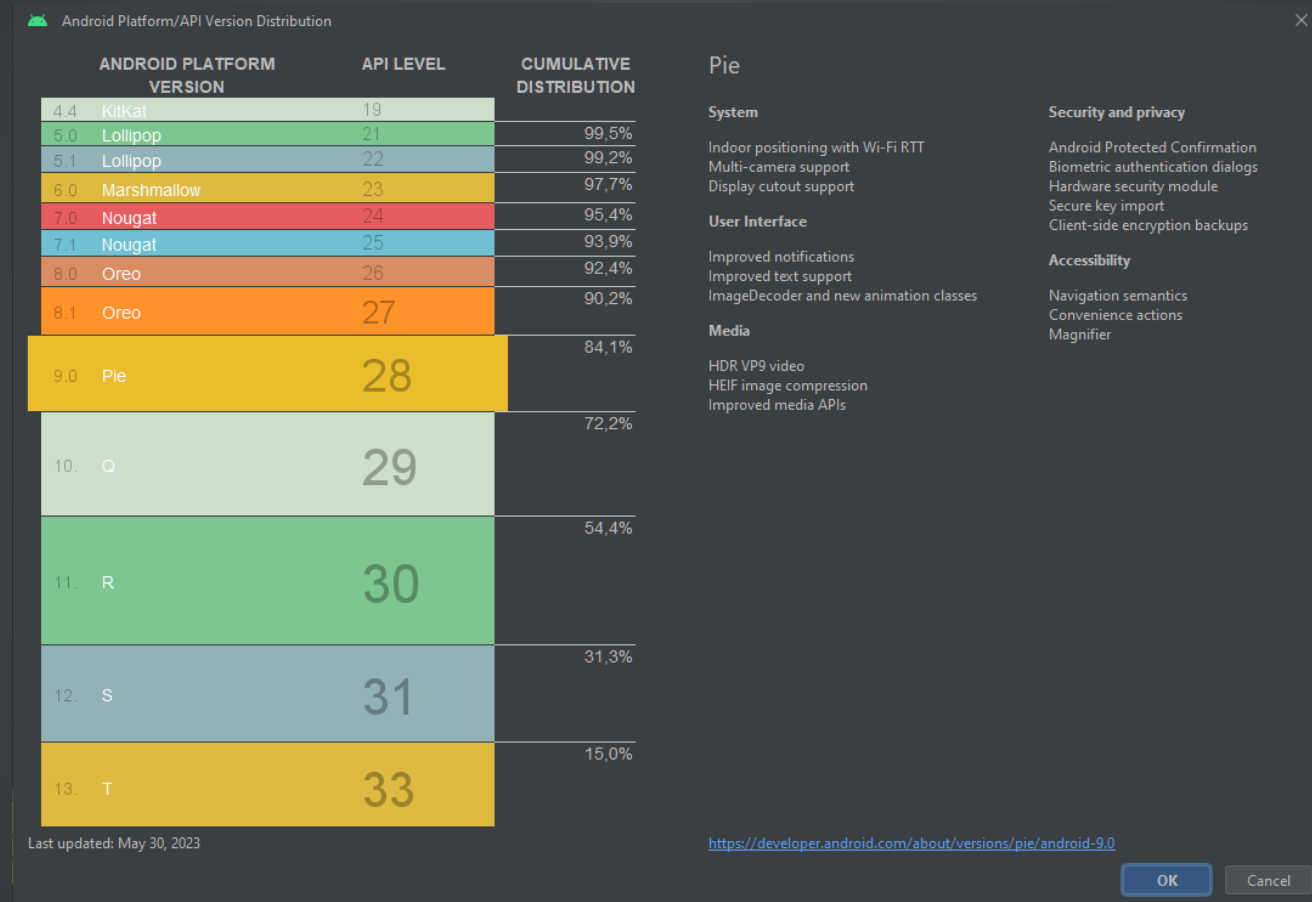
L'histoire d'Android



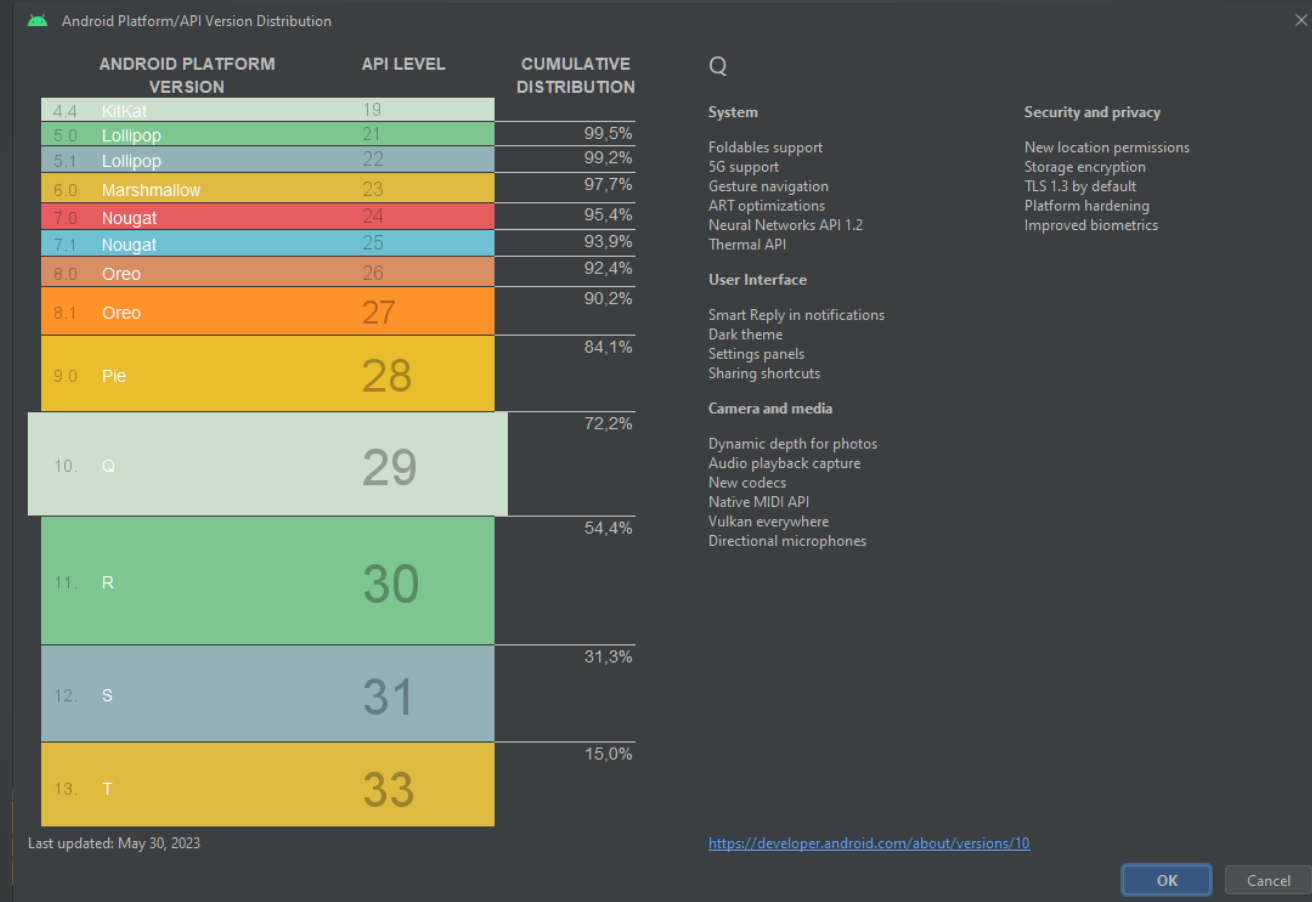
L'histoire d'Android



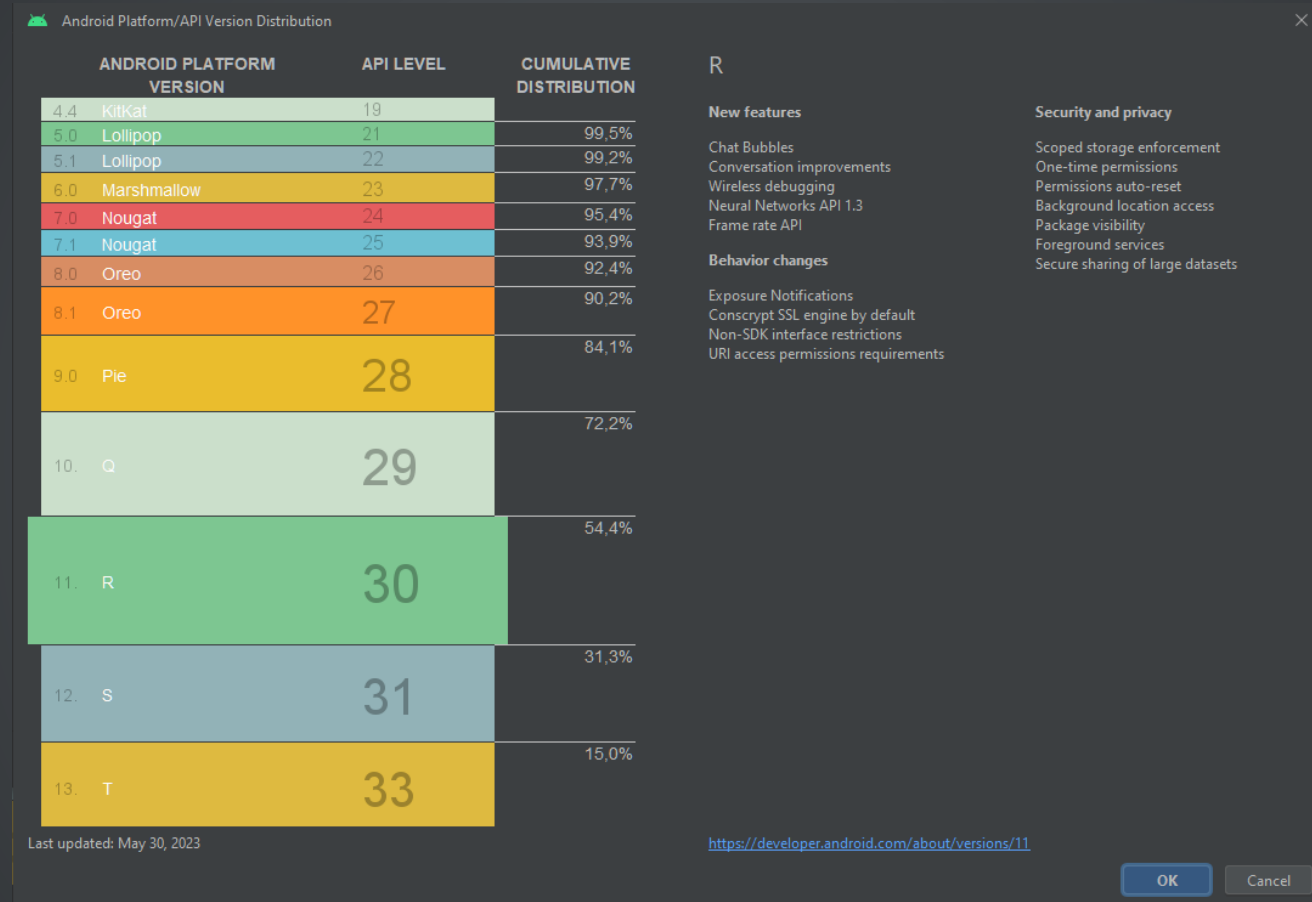
L'histoire d'Android



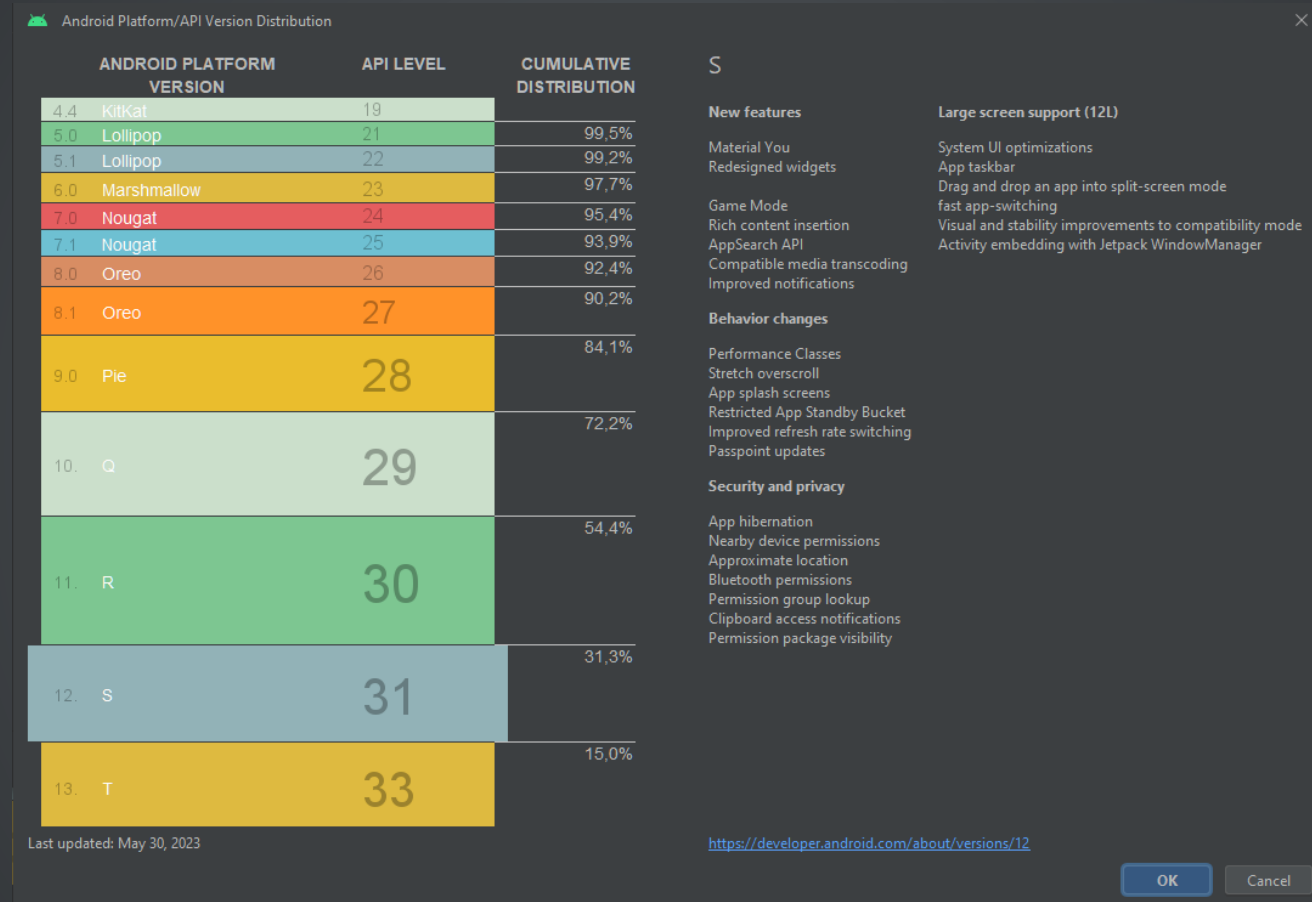
L'histoire d'Android



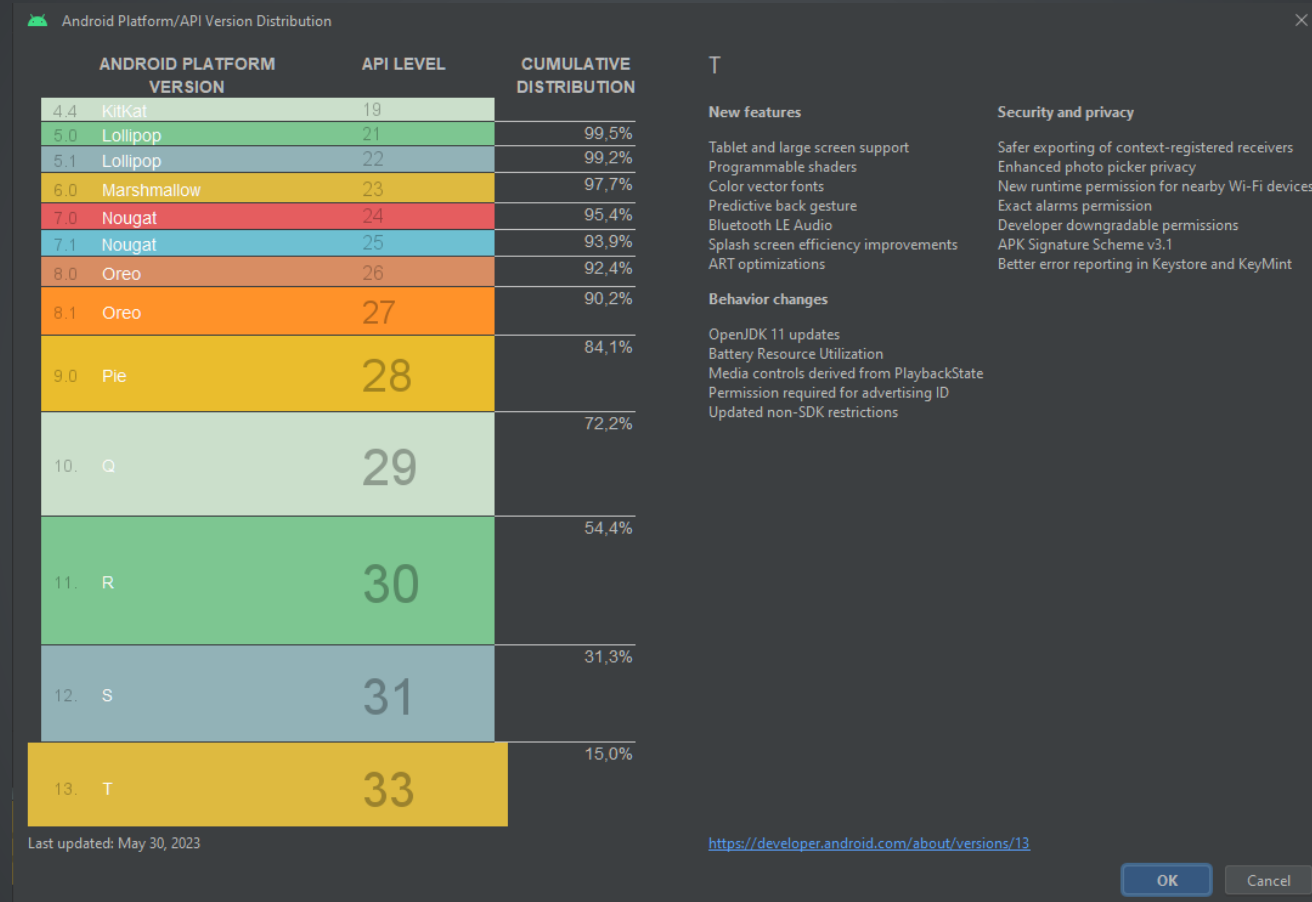
L'histoire d'Android



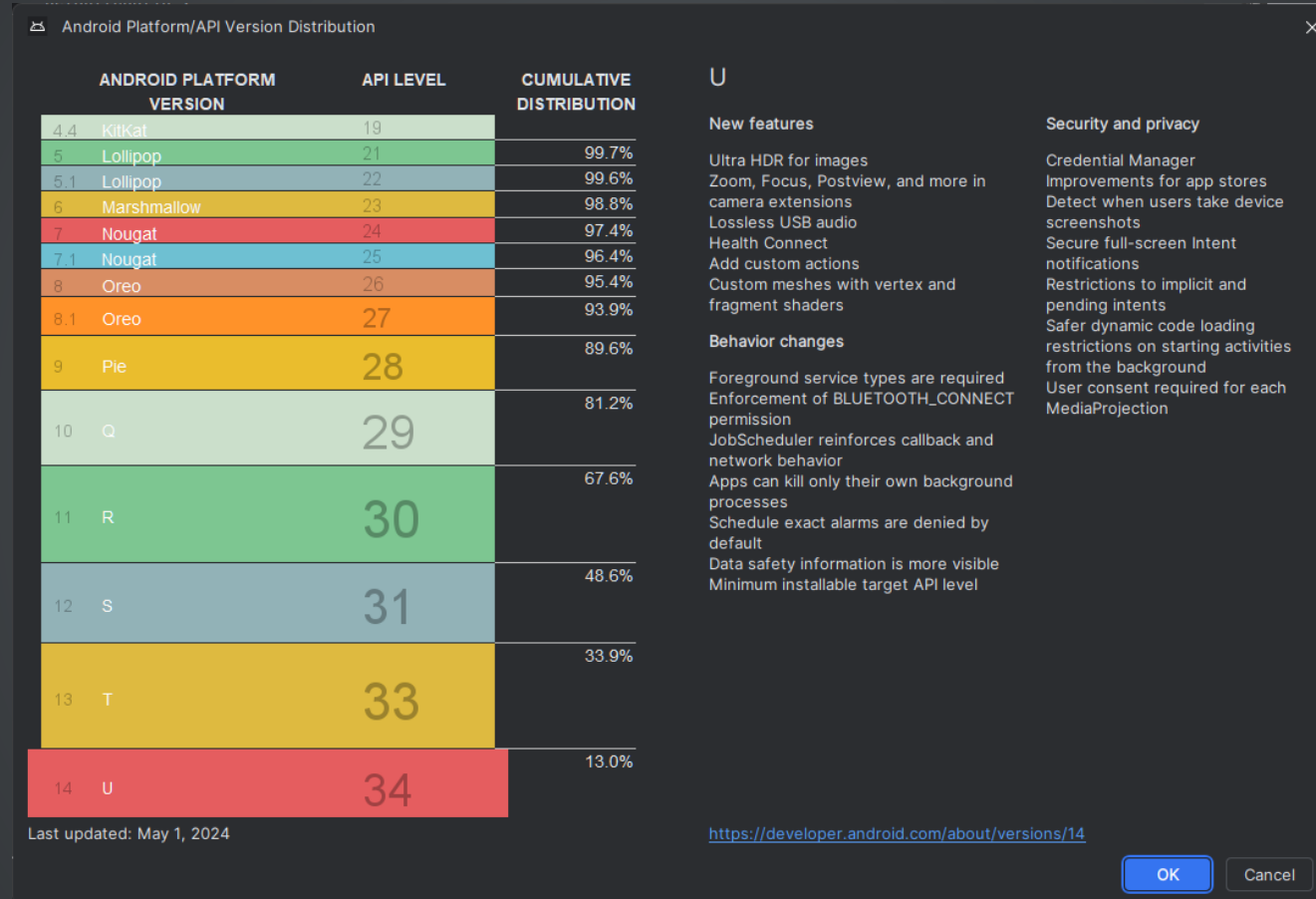
L'histoire d'Android



L'histoire d'Android



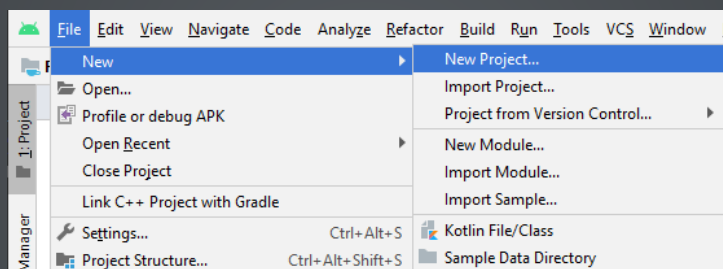
L'histoire d'Android



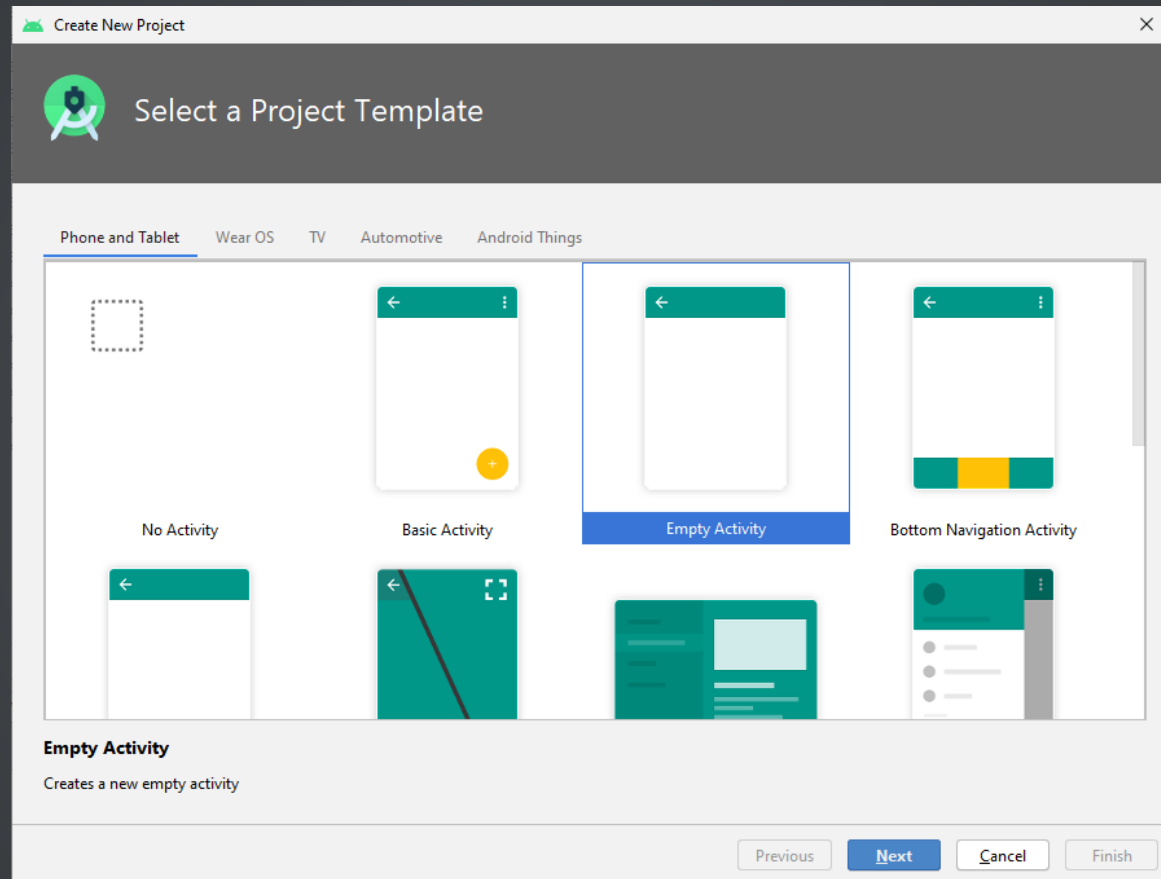
Les terminaux cibles

Pour obtenir la distribution des versions, vous devez créer un nouveau projet Android, puis sur l'écran "Configure Your Project" cliquons sur "Help me choose".

Les terminaux cibles



Les terminaux cibles



Les terminaux cibles

Create New Project

 Configure Your Project



Empty Activity

Creates a new empty activity

Name

My Application

Package name

com.example.myapplication

Save location

D:\MyApplication

Language

Kotlin

Minimum SDK

API 16: Android 4.1 (Jelly Bean)

 Your app will run on approximately 99.8% of devices.
[Help me choose](#)

☐ Use legacy android.support libraries 

Previous

Next

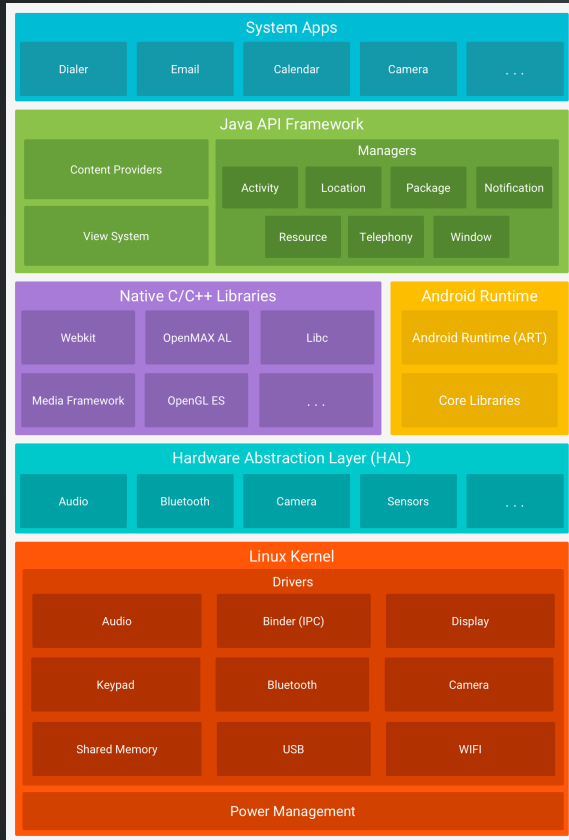
Cancel

Finish

Développement Android

- Les concepts de base. Le cycle développement.
- Les classes de base du framework.
- Le projet sous Android Studio.
- L'émulateur du SDK. Les outils du SDK, SDK manager, AVD manager.
- L'utilisation des outils sous Android Studio : debugger, profiler, etc.
- Les paramètres du manifest.
- La production de l'application, la publication.

Les concepts de base



Source : <https://developer.android.com/guide/platform>

Les concepts de base

Optimisation

Les applications sur smartphone doivent veiller à optimiser tous les points, tels quel :

- Mémoire.
- Processeur.
- Radio (WiFi, 3G, BT).
- Graphique.

En effet tous ces points vont avoir une répercussion sur la consommation d'énergie, et donc sur la batterie.

Les concepts de base

Le cycle développement

Le développement d'une application peut suivre plusieurs schémas, nous allons en détailler une possibilité :

1. Lancement : toutes les applications commencent par une idée. Cette idée est généralement affinée jusqu'à constituer une base solide pour une application.
2. Conception : la phase de conception consiste à définir l'expérience utilisateur de l'application, comme la présentation générale, son fonctionnement, etc., ainsi que la conversion de cette expérience utilisateur en une interface utilisateur appropriée, généralement avec l'aide d'un infographiste.
3. Développement : généralement la phase la plus consommatrice de ressources, c'est la phase de création réelle de l'application.
4. Stabilisation : quand le développement est suffisamment avancé, l'assurance qualité commence généralement à tester l'application et les bogues sont corrigés. Le plus souvent, une application passe par une phase bêta limitée, durant laquelle un public plus large a la possibilité de l'utiliser, de fournir des commentaires et d'obtenir des modifications.
5. Déploiement

Les concepts de base

Conception des interfaces utilisateur

Afin de définir notre interface utilisateur, Google fournit des guides pour bien réaliser cette tâche :

<https://developer.android.com/design/index.html>

Il nous appartiendra à bien qualifier notre cible (type de terminaux : smartphone, tablette, TV, montre, ordinateur de voiture, ...).

Les concepts de base

Phases de développement

Nous pouvons passer par plusieurs phases pour réaliser l'application finale :

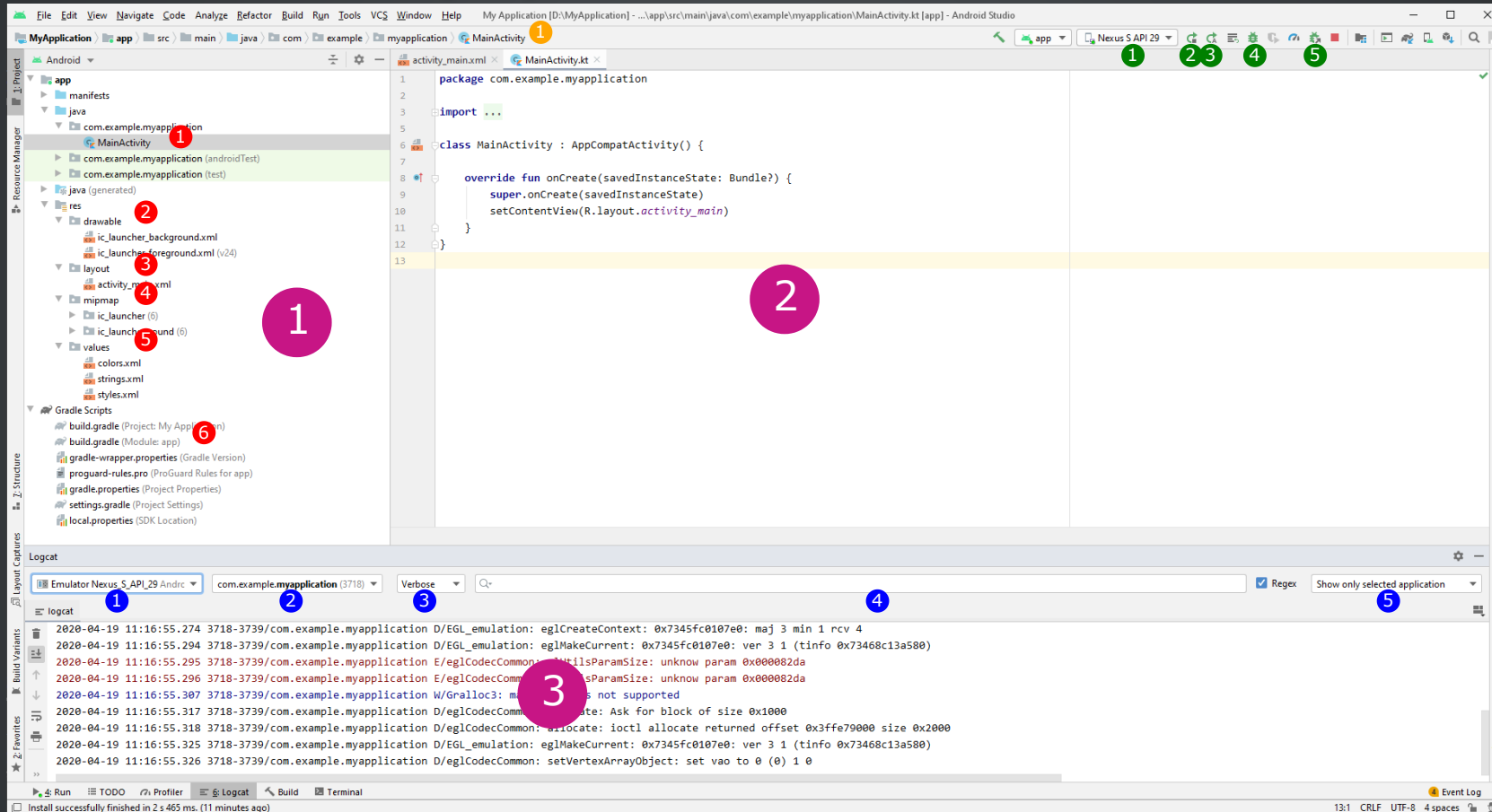
- **Prototype/MVP** : l'application est toujours en phase de preuve de concept, et seules les fonctionnalités principales ou des parties spécifiques de l'application fonctionnent. Des bogues majeurs y sont présents.
- **Alpha** : les fonctionnalités principales sont généralement entièrement présentes dans le code (qui est généré, mais pas entièrement testé). Des bogues majeurs sont encore présents, des fonctionnalités périphériques peuvent ne pas encore être présentes.
- **Bêta** : la plupart des fonctionnalités sont maintenant terminées, une partie des tests et de la correction des bogues a été effectuée. Des problèmes majeurs connus peuvent encore être présents.
- **Version Release Candidate** : toutes les fonctionnalités sont terminées et testées. Sauf si de nouveaux bogues apparaissent, l'application est candidate à la publication.

Les classes de base du framework

Plusieurs classes de bases sont disponibles dans le Framework, nous les détaillerons par la suite :

- Activity
- Fragment (3.0+)
- Service
- Content providers
- Broadcast receivers
- Intent

Le projet sous Android Studio



Le projet sous Android Studio

Les grandes parties en violet :

1. La liste des fichiers.
2. La zone d'édition.
3. Les logs.

Le projet sous Android Studio

Détail des points en rouge :

1. Le code de notre application.
2. Le répertoire des ressources.
3. Le répertoire contenant les mises en forme des écrans (les layouts).
4. Le répertoire contenant les versions des icônes de l'application.
5. Le répertoire contenant les autres ressources.

Le projet sous Android Studio

Détail des points en orange :

1. Le chemin complet du fichier courant sélectionné.

Le projet sous Android Studio

Détail des points en vert :

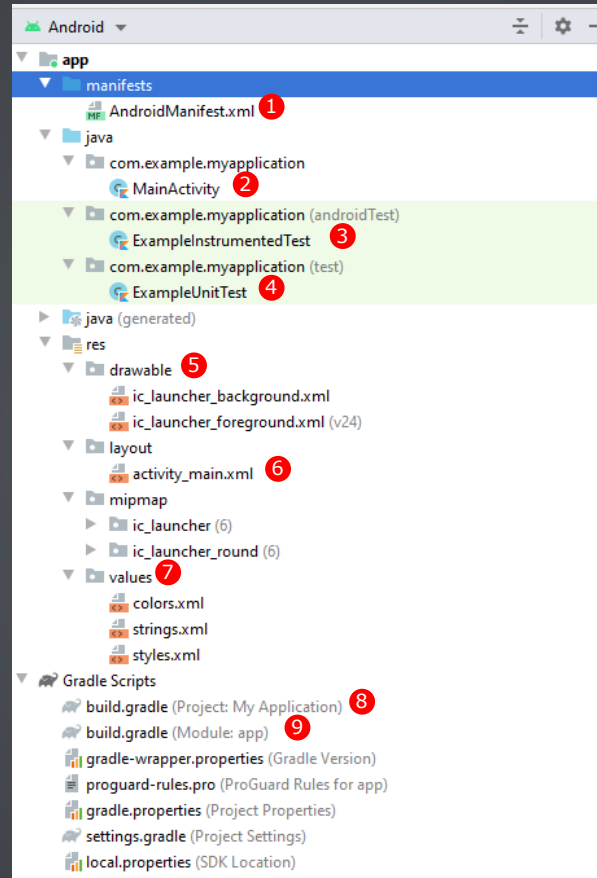
1. Le choix de la cible pour lancer le programme.
2. Relancer l'application.
3. Relancer l'activity.
4. Relancer en mode debug.
5. Attacher à la volée le debugger.

Le projet sous Android Studio

Détail des points en **bleu**, il s'agit du Logcat :

1. Nous pouvons choisir le device sur lequel se connecter.
2. Sélectionner le process pour filtrer les messages.
3. Sélectionner le type de log des messages.
4. Filtrer les messages.
5. Activer le filtre.

Le projet sous Android Studio



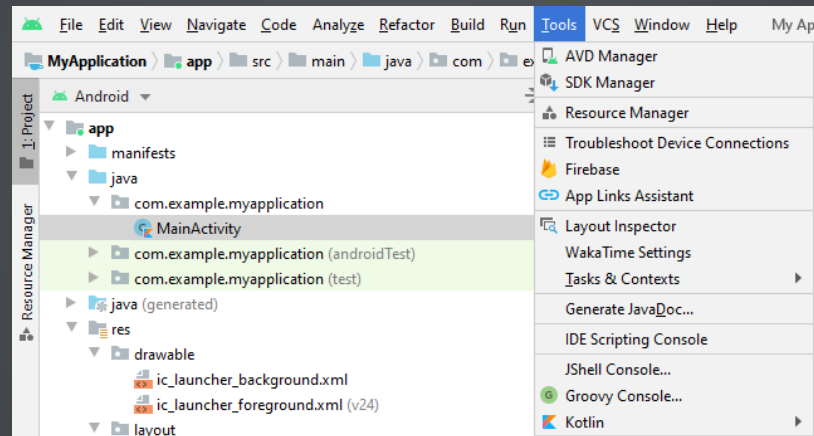
Le projet sous Android Studio

Les fichiers principaux **rouge** :

1. Le manifest (AndroidManifest.xml), qui contient la carte d'identité de notre application.
2. Le code de l'application.
3. Le code des tests graphiques.
4. Le code des tests unitaires.
5. Les ressources graphiques.
6. Les écrans (layouts) de notre application.
7. Les ressources (texte, couleurs, dimensions, styles, ...) de notre application.
8. Le fichier de configuration global du système de compilation (gradle).
9. Le fichier de configuration de notre projet (celui qui sera le plus souvent modifié).

Émulateur

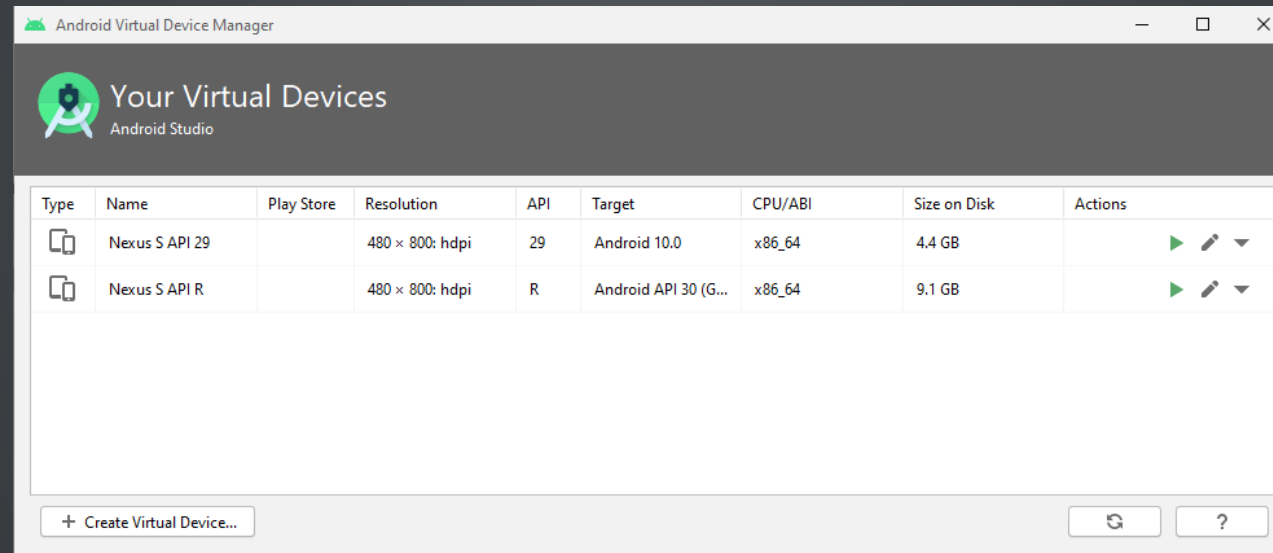
Lançons l'écran de gestion des machines virtuelles, avec le menu :
Tools/AVD Manager



Émulateur

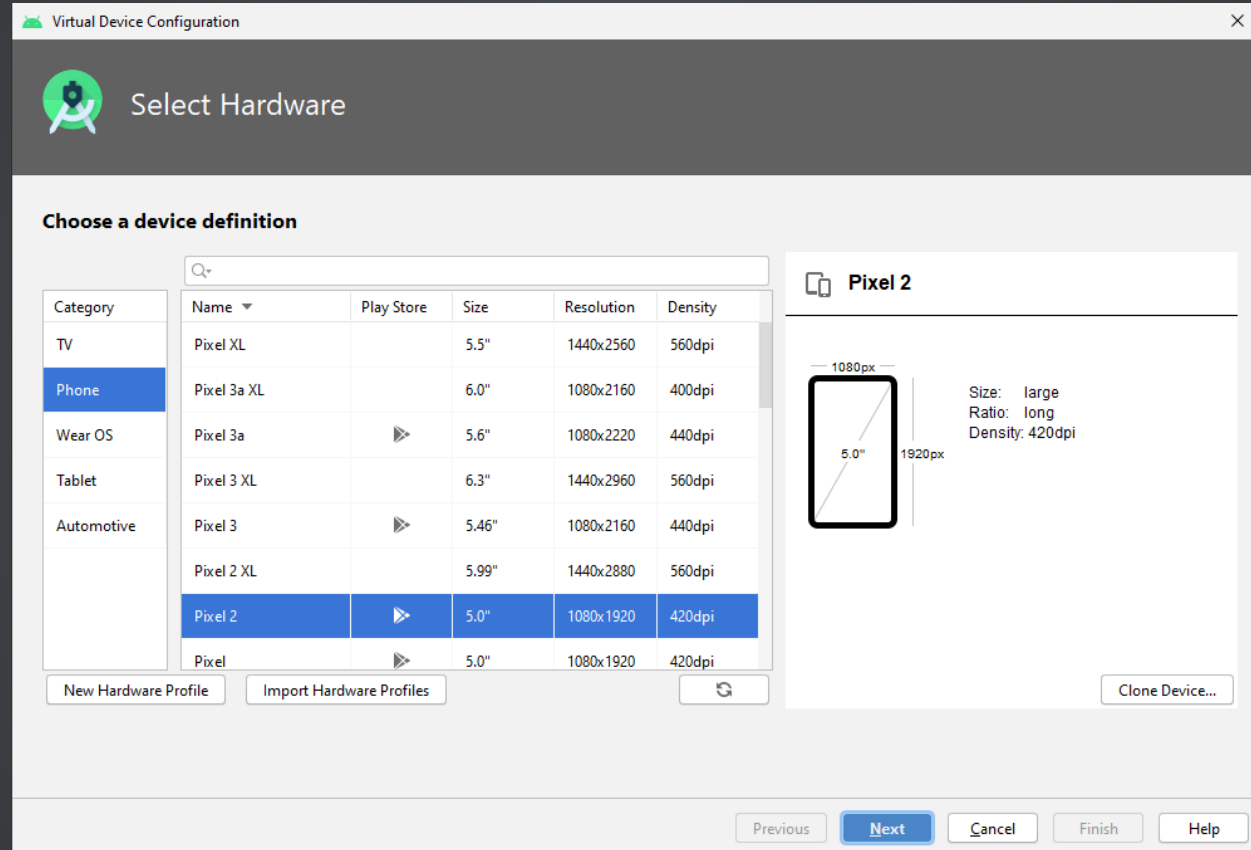
Choix de la machine virtuelle

Sur cet écran, nous pouvons choisir la machine virtuelle à lancer/configurer/supprimer. Nous pouvons aussi en créer une nouvelle, en cliquant sur le bouton "Create Virtual Device..."



Émulateur

Choisissons le modèle de device que nous voulons émuler :



Émulateur

Choisissons la version d'Android :

Virtual Device Configuration

 System Image

Select a system image

Recommended x86 Images Other Images

Release Name	API Level	ABI	Target
R Download	R	x86	Android API R (Google Play)
Q Download	29	x86	Android 10.0 (Google Play)
Pie Download	28	x86	Android 9.0 (Google Play)
Oreo Download	27	x86	Android 8.1 (Google Play)
Oreo Download	26	x86	Android 8.0 (Google Play)
Nougat Download	25	x86	Android 7.1.1 (Google Play)
Nougat Download	24	x86	Android 7.0 (Google Play)



 A system image must be selected to continue.

R



API Level

R

Android

Google Inc.

System Image

x86

We recommend these Google Play images because this device is compatible with Google Play.

Questions on API level?
[See the API level distribution chart](#)

Previous

Next

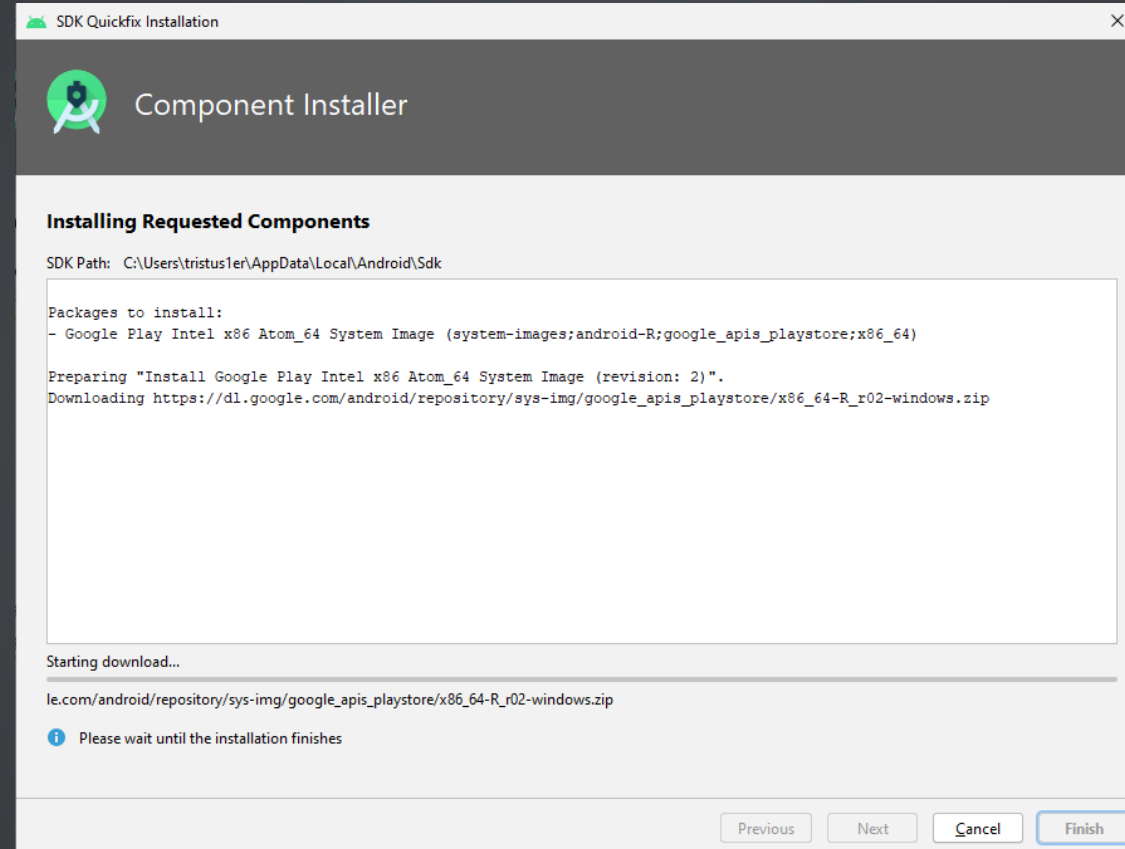
Cancel

Finish

Help

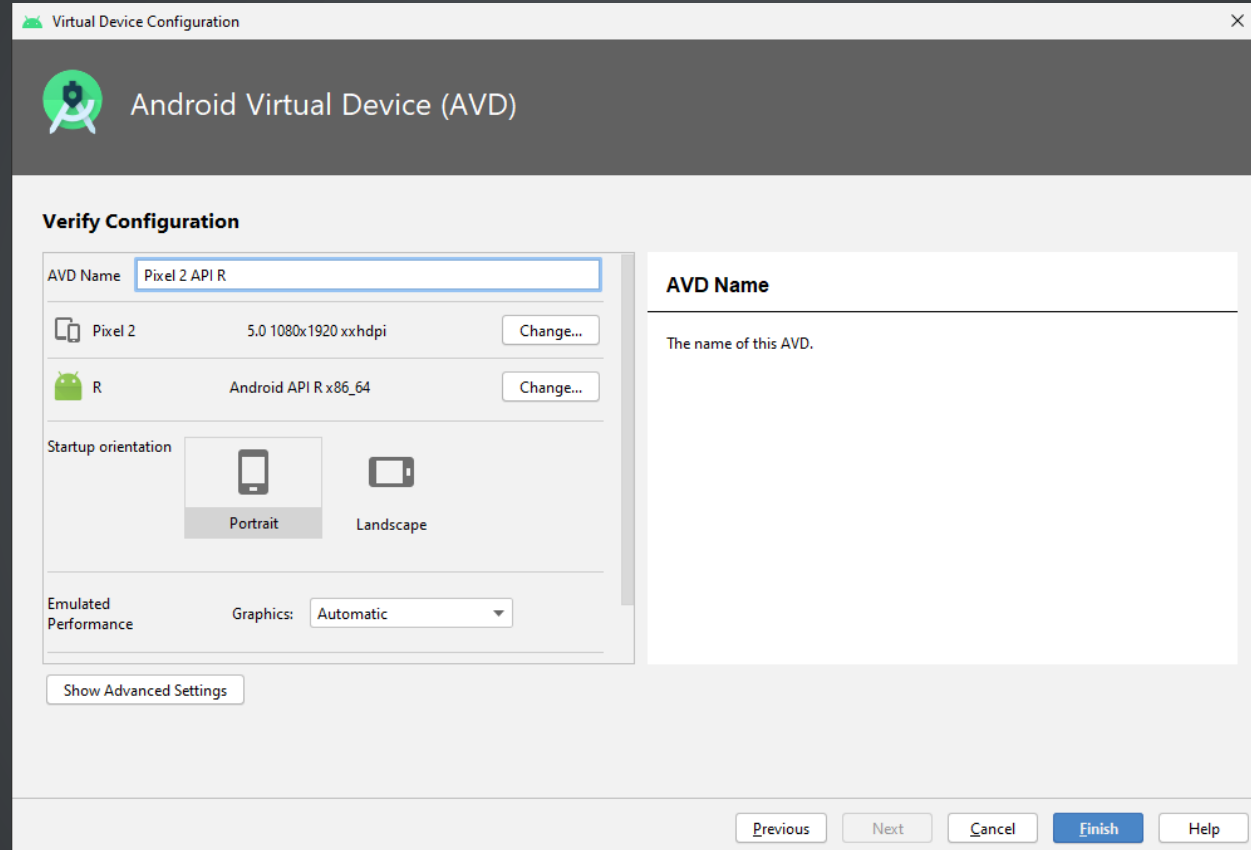
Émulateur

En cas de besoin, nous pouvons directement télécharger une image :



Émulateur

Donnons un nom à notre machine virtuelle, nous laissons les paramètres par défaut :



The screenshot shows the 'Virtual Device Configuration' window for an Android Virtual Device (AVD). The window has a title bar with the Android logo and the text 'Virtual Device Configuration'. Below the title bar is a header area with the Android logo and the text 'Android Virtual Device (AVD)'. The main content area is divided into two sections. The left section is titled 'Verify Configuration' and contains several settings: 'AVD Name' with a text field containing 'Pixel 2 API R', 'Pixel 2' with '5.0 1080x1920 xxhdpi' and a 'Change...' button, 'R' with 'Android API R x86_64' and a 'Change...' button, 'Startup orientation' with two buttons 'Portrait' and 'Landscape' (Portrait is selected), and 'Emulated Performance' with 'Graphics: Automatic' and a dropdown arrow. Below these settings is a 'Show Advanced Settings' button. The right section is titled 'AVD Name' and contains the text 'The name of this AVD.' Below the configuration settings is a row of five buttons: 'Previous', 'Next', 'Cancel', 'Finish' (highlighted in blue), and 'Help'.

Virtual Device Configuration

Android Virtual Device (AVD)

Verify Configuration

AVD Name: Pixel 2 API R

Pixel 2 5.0 1080x1920 xxhdpi Change...

R Android API R x86_64 Change...

Startup orientation

Portrait Landscape

Emulated Performance Graphics: Automatic

Show Advanced Settings

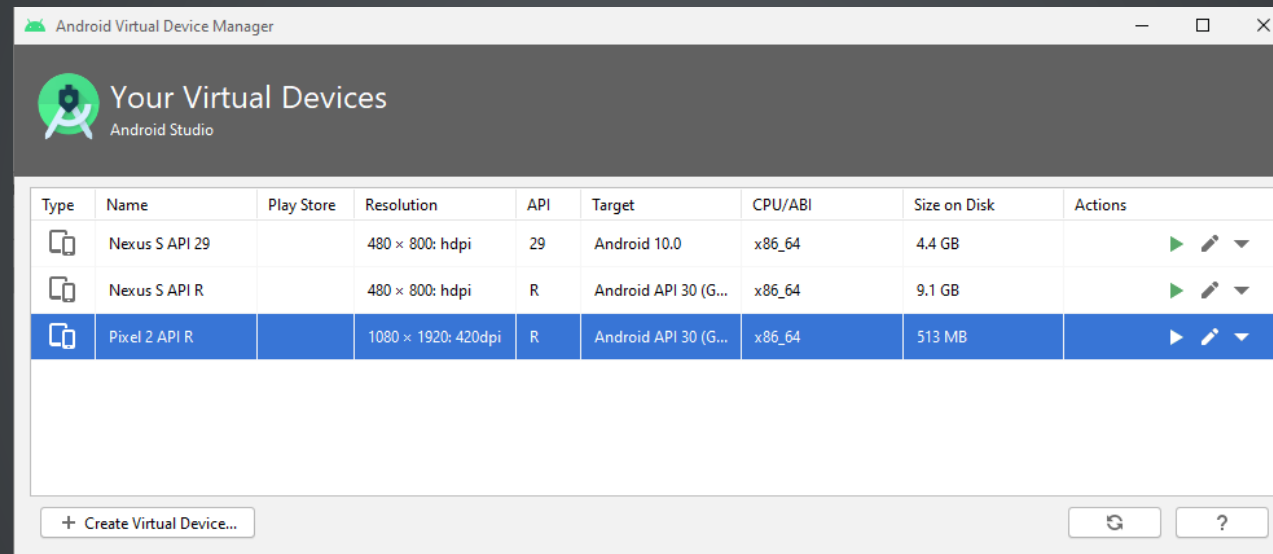
AVD Name

The name of this AVD.

Previous Next Cancel Finish Help

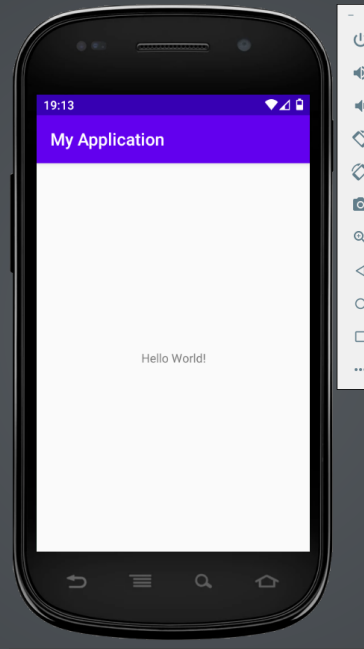
Émulateur

Notre machine virtuelle est maintenant disponible, lançons la :



Émulateur

L'émulateur permet de faire fonctionner nos applications sur des versions d'Android que nous n'avons pas forcément sur notre smartphone :

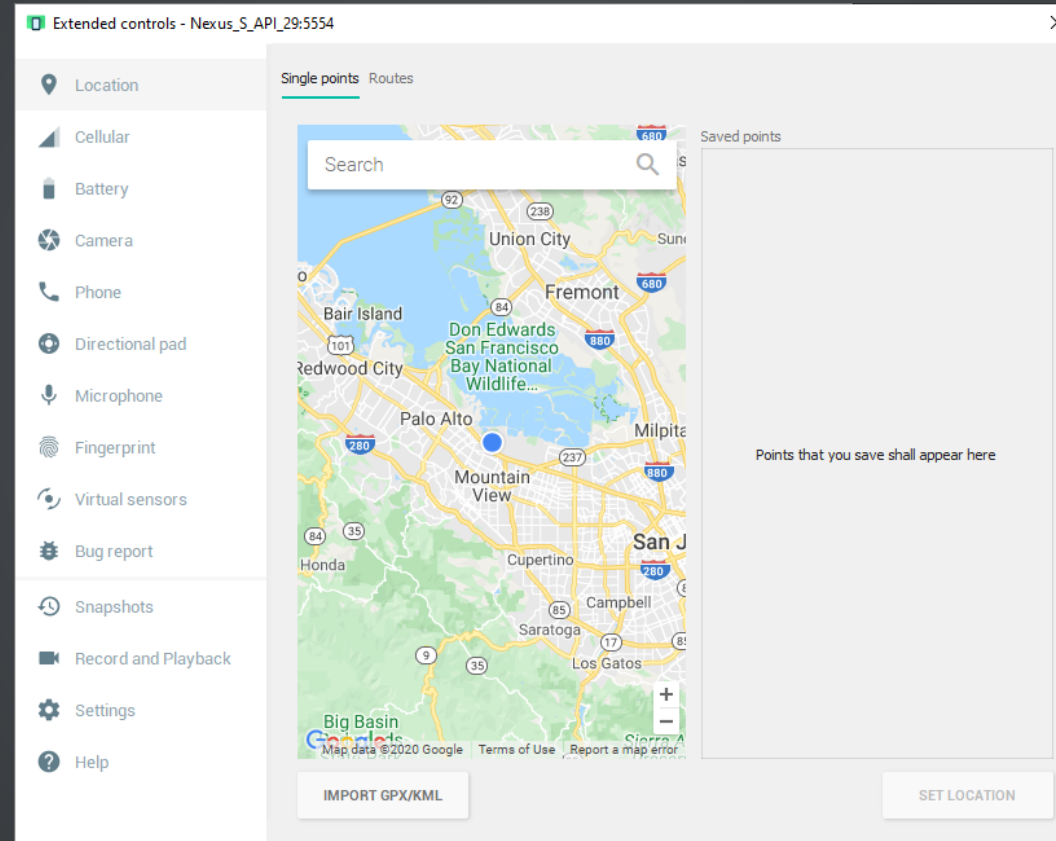


Cliquons sur les "..."

Émulateur

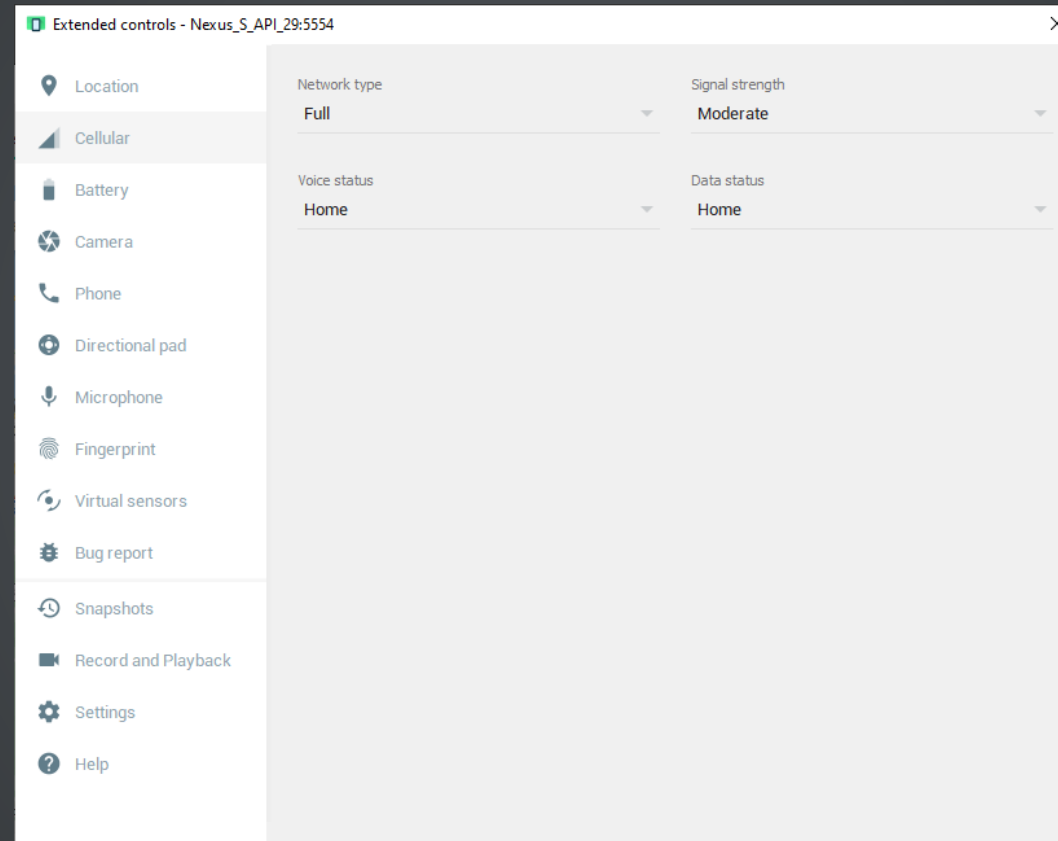
Nous pouvons :

Gérer la position du device directement sur une carte (nouvelle fonctionnalité Android Studio 3.6) :



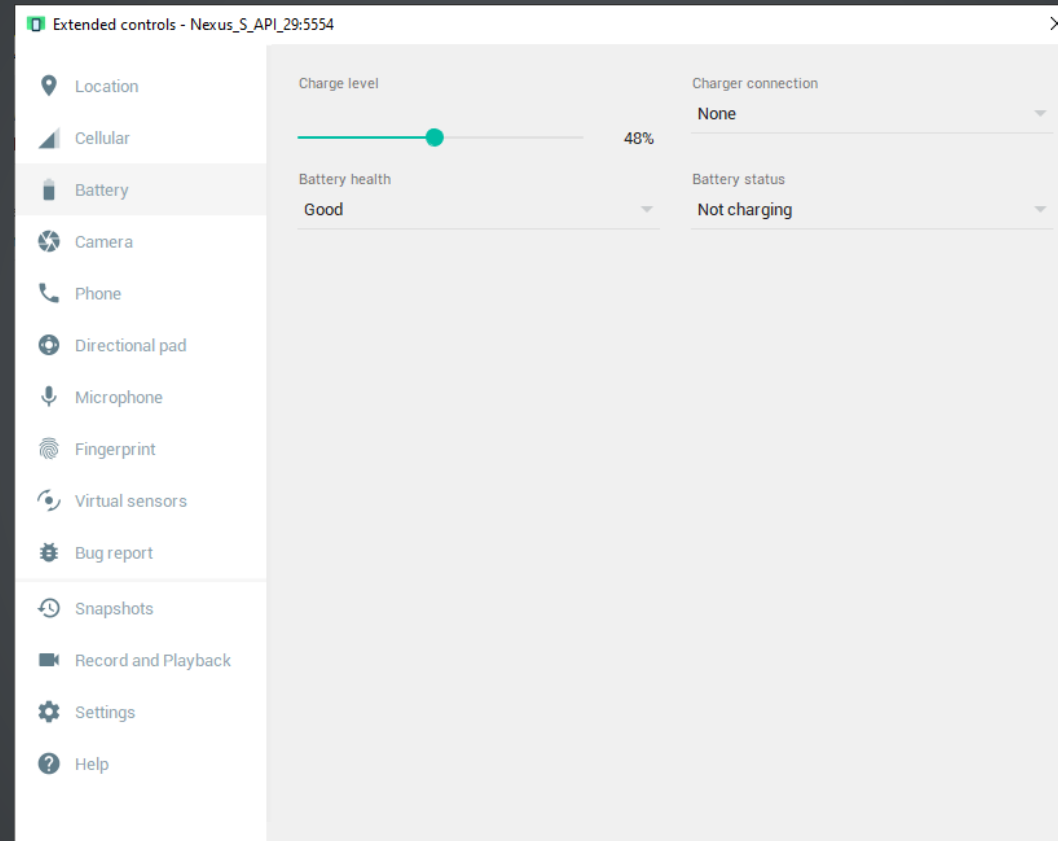
Émulateur

Gérer la réception réseau :



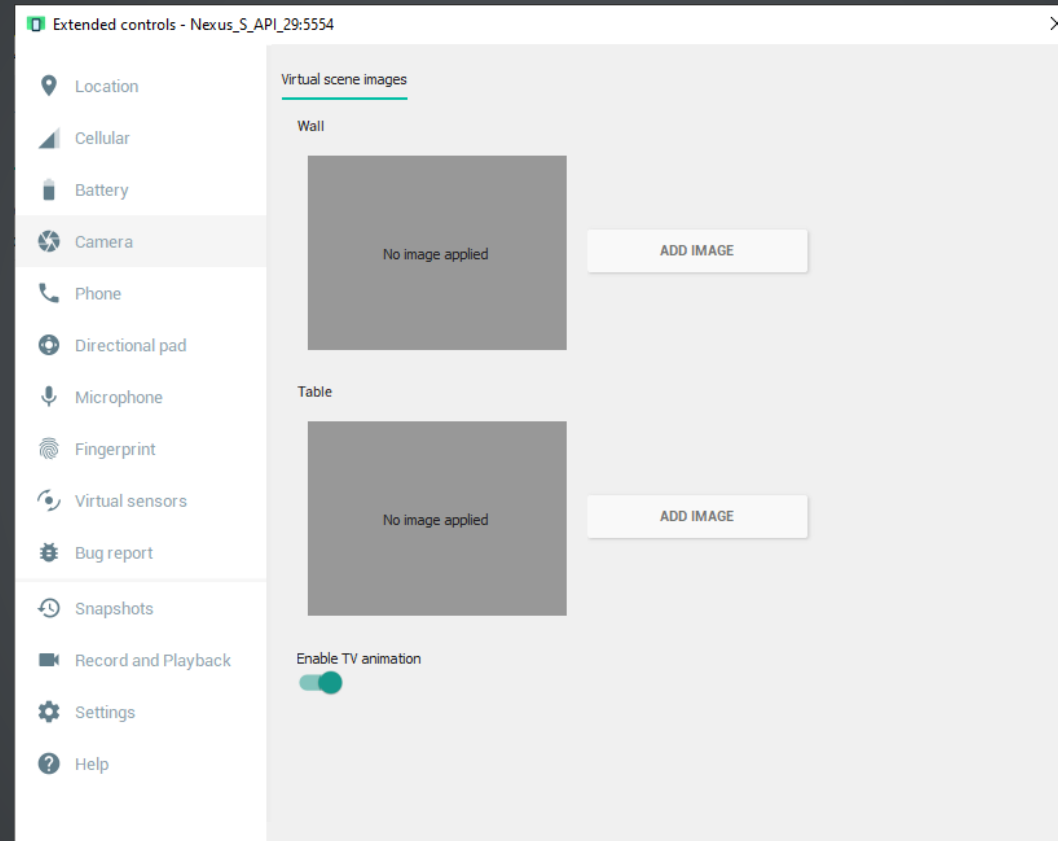
Émulateur

Définir le niveau de batterie :



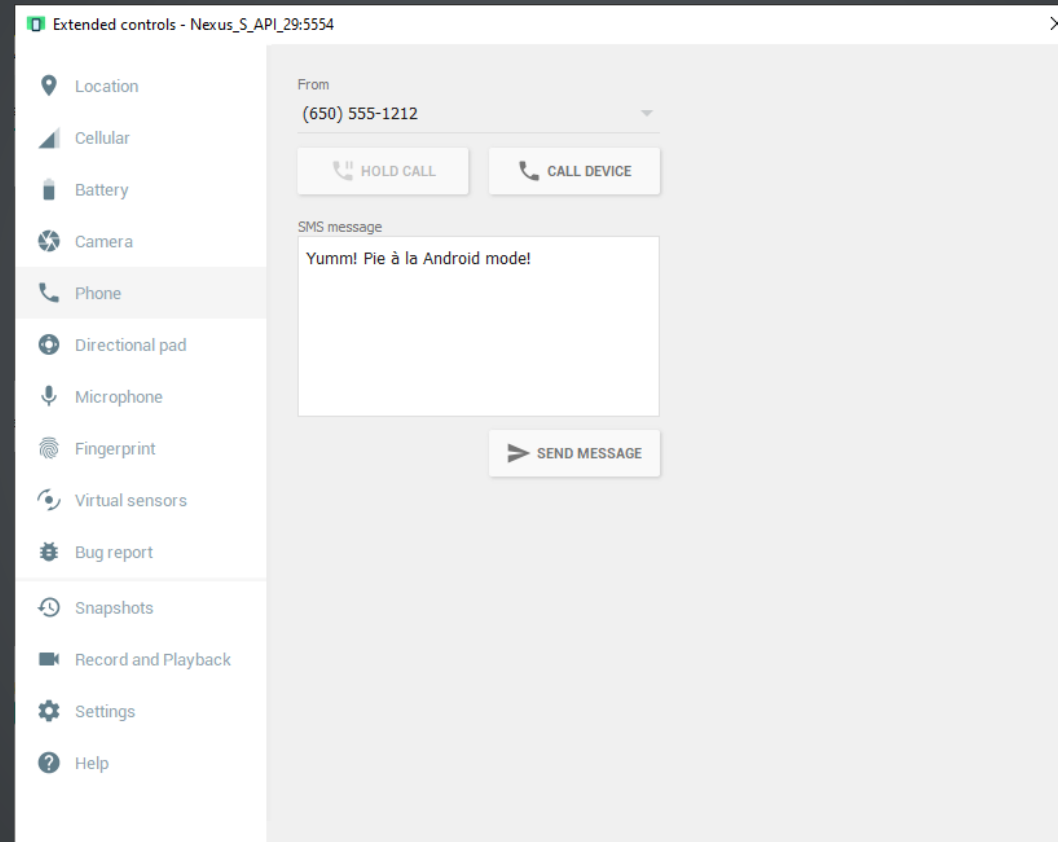
Émulateur

Définir ce que "voit" la/les caméra(s) :



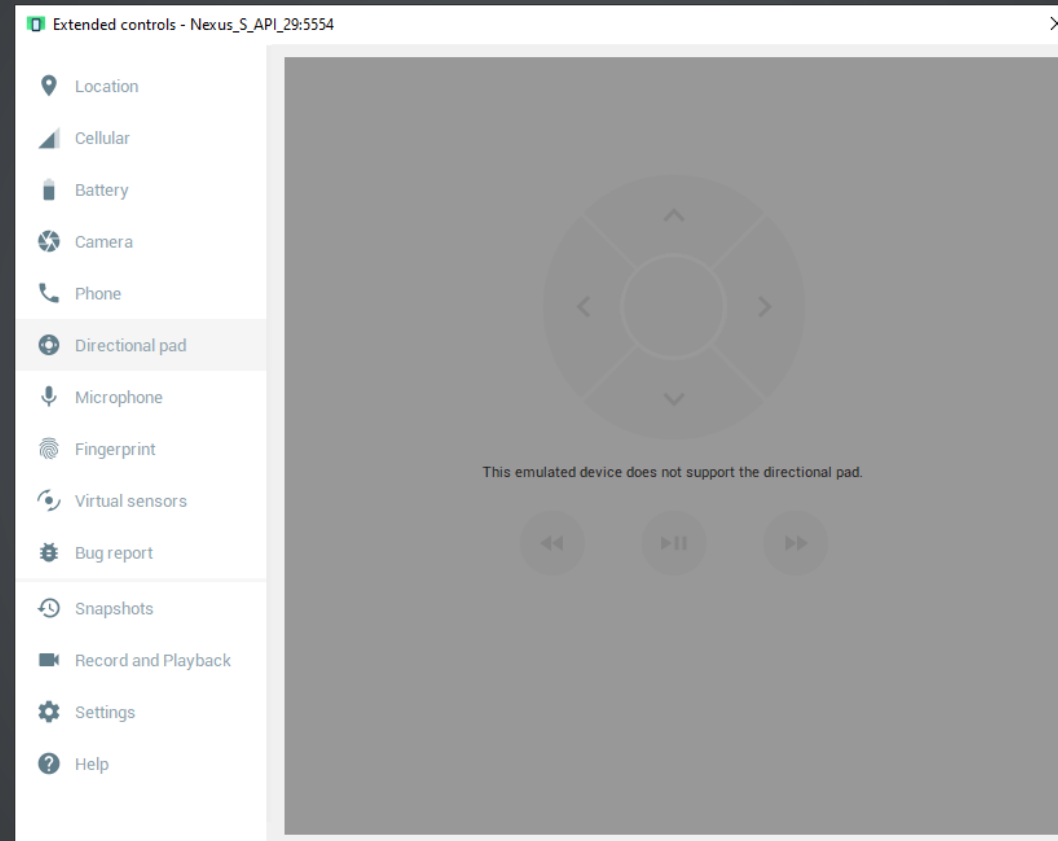
Émulateur

Lancer des appels téléphoniques
et envoyer des SMS :



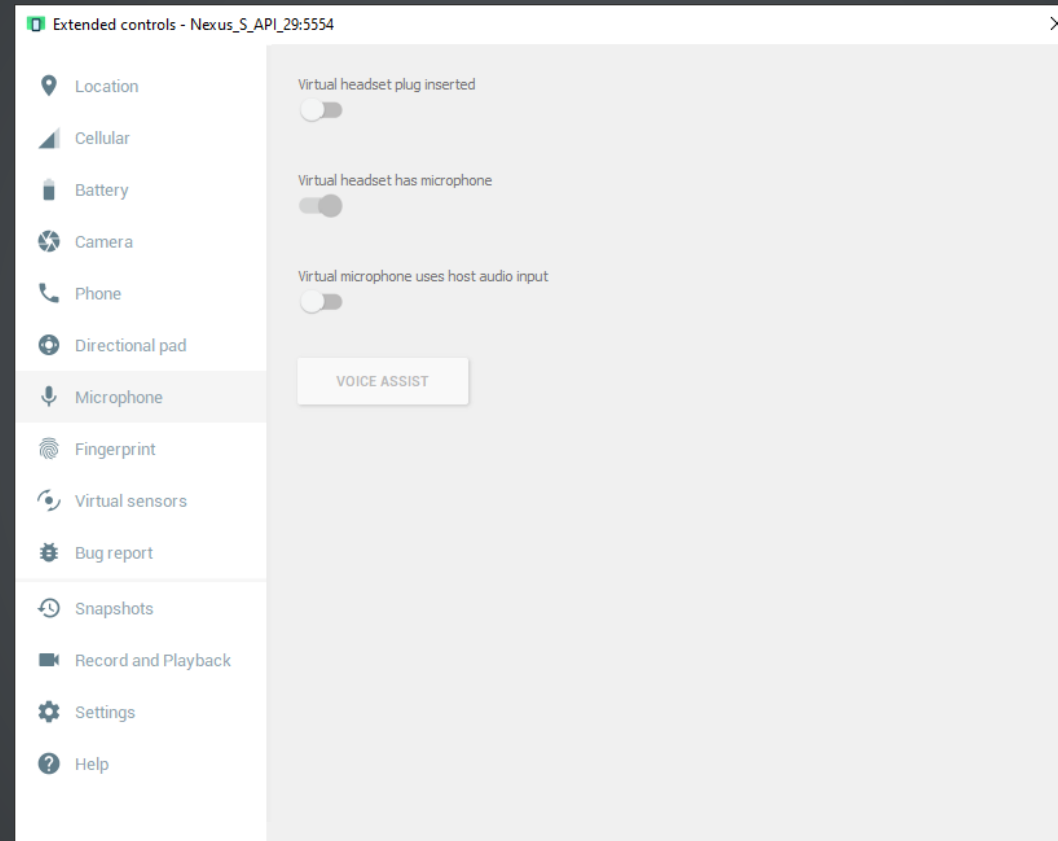
Émulateur

Émuler le pad :



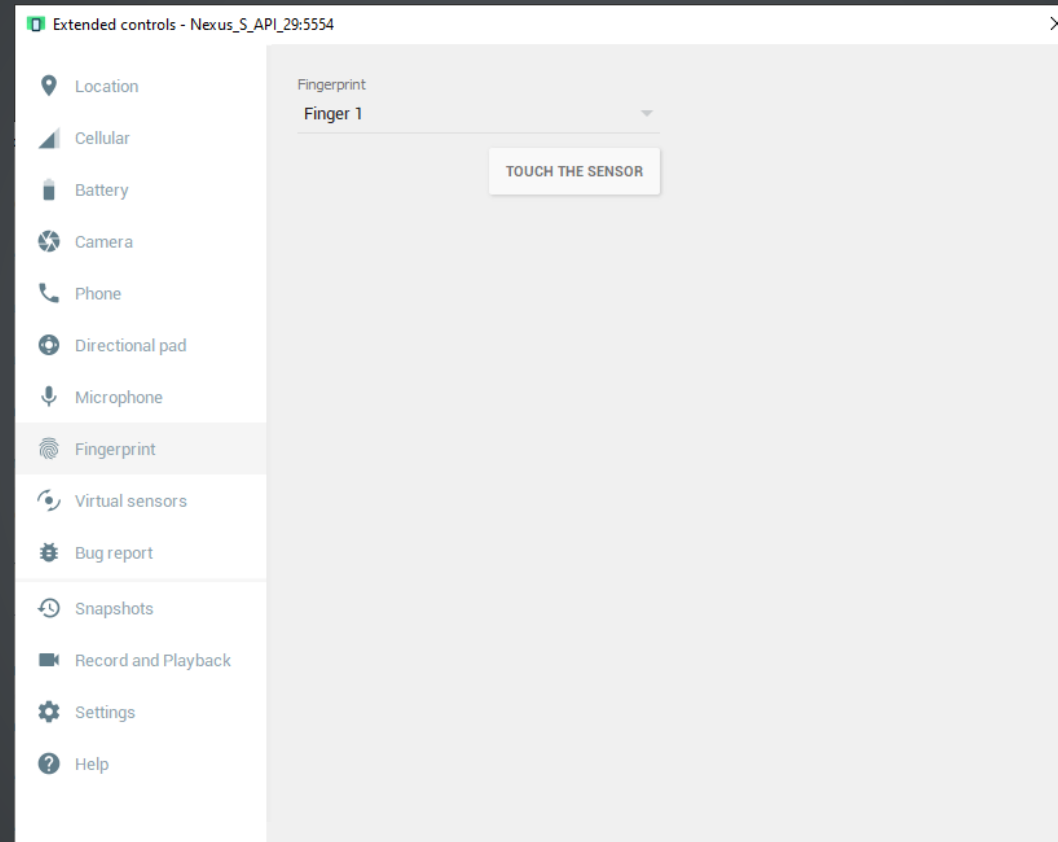
Émulateur

Émuler la gestion du micro :



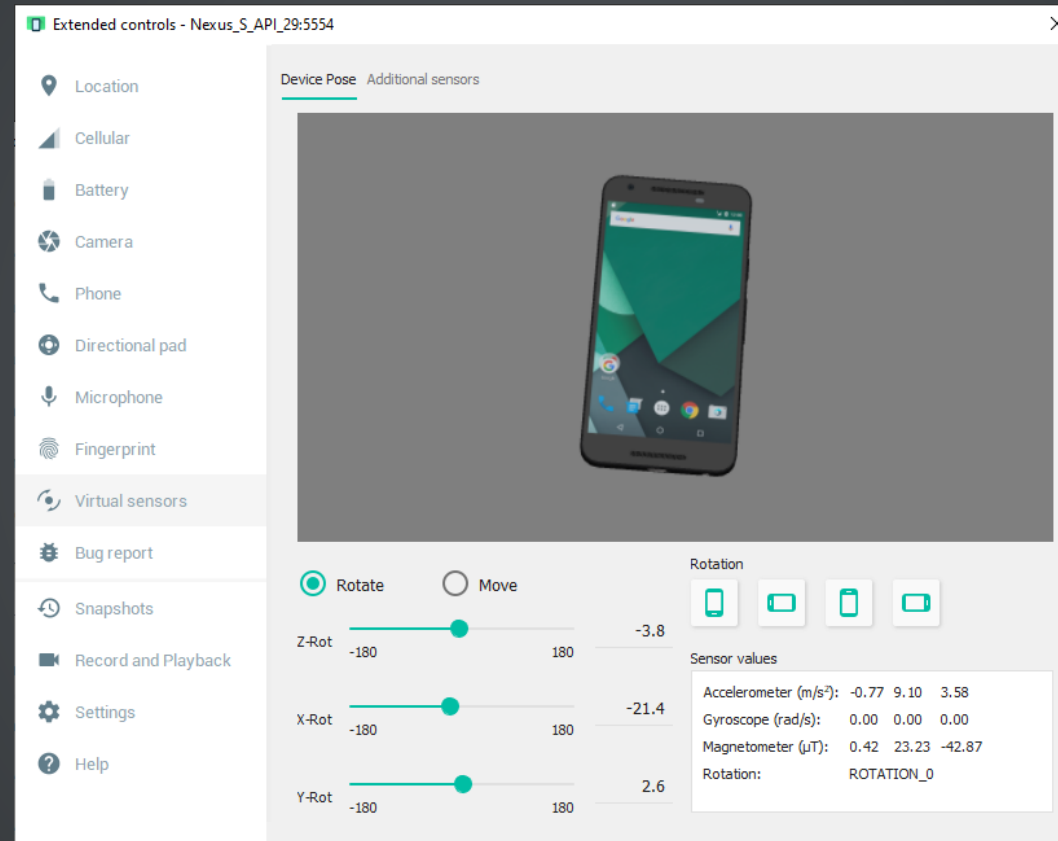
Émulateur

Gérer les empreintes tactiles :



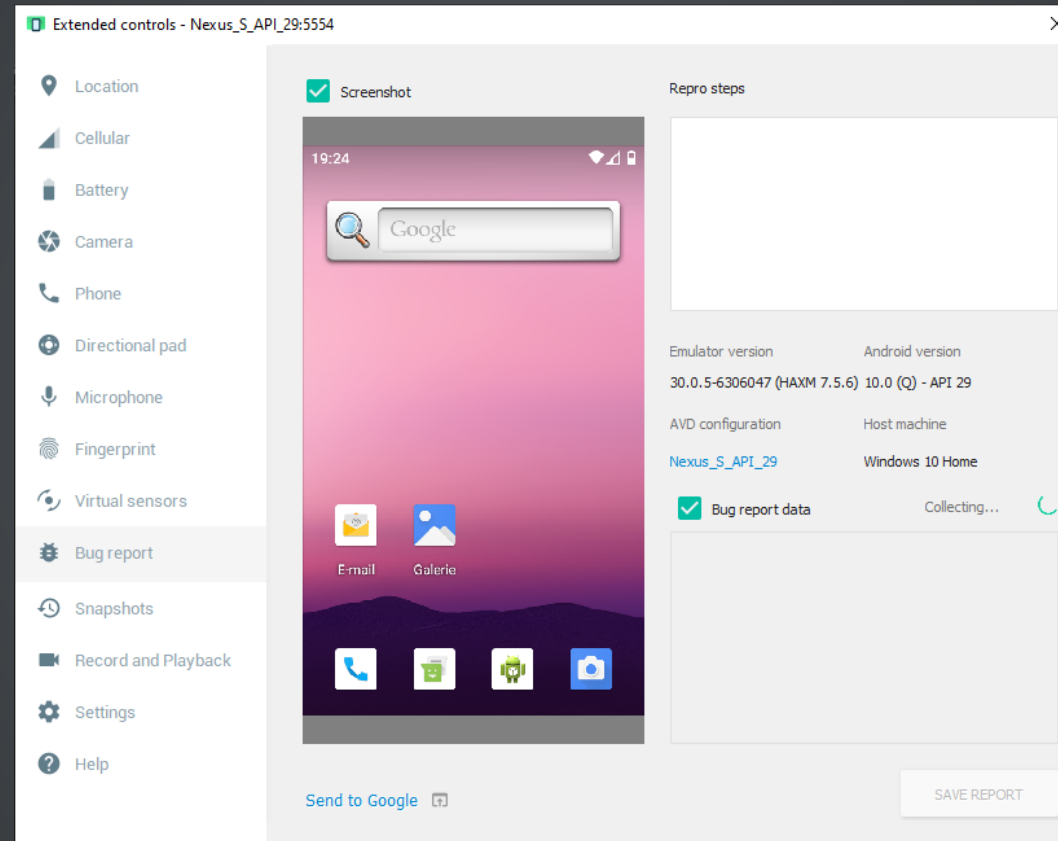
Émulateur

Émuler les capteurs (rotations, et accélération) :



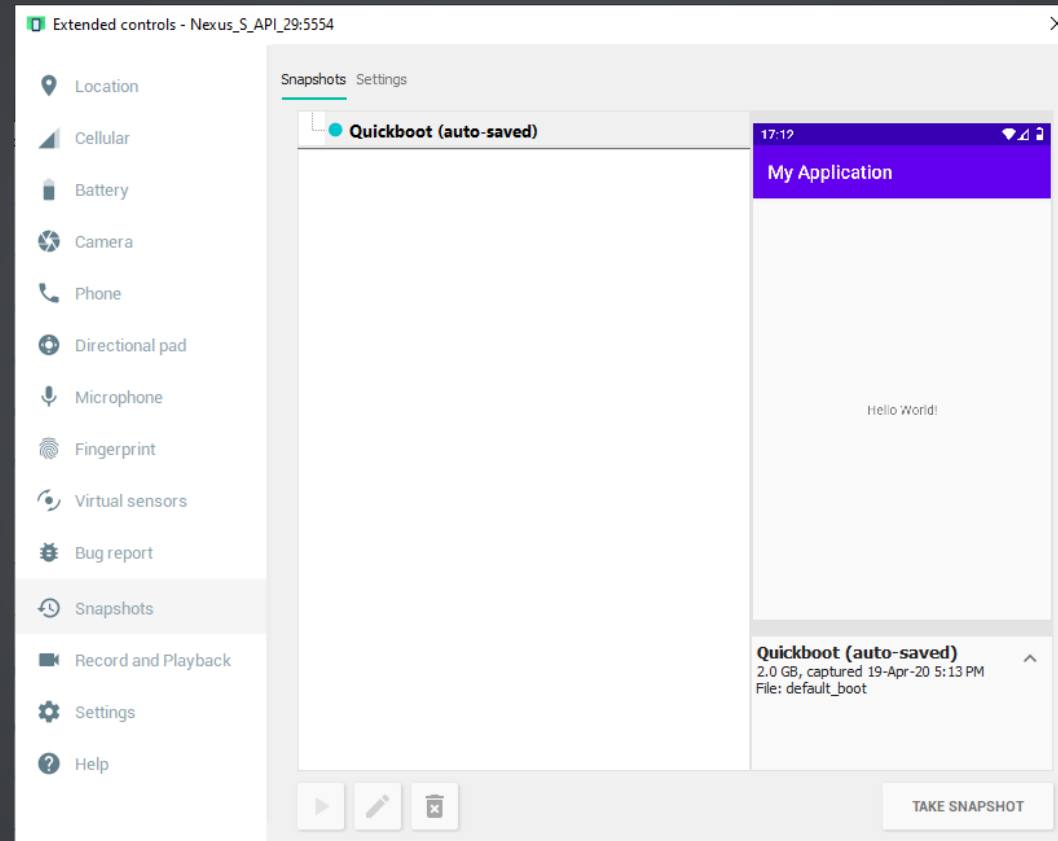
Émulateur

Envoyer un rapport de bug :



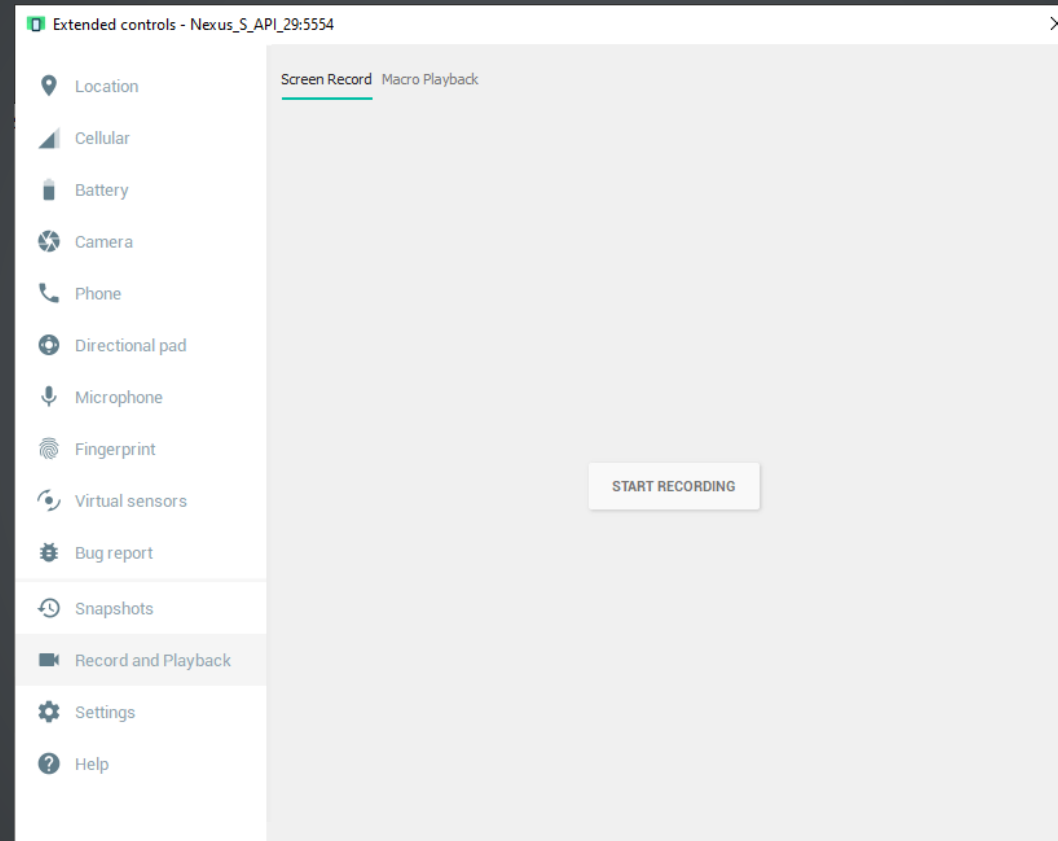
Émulateur

Définir des points de restauration :



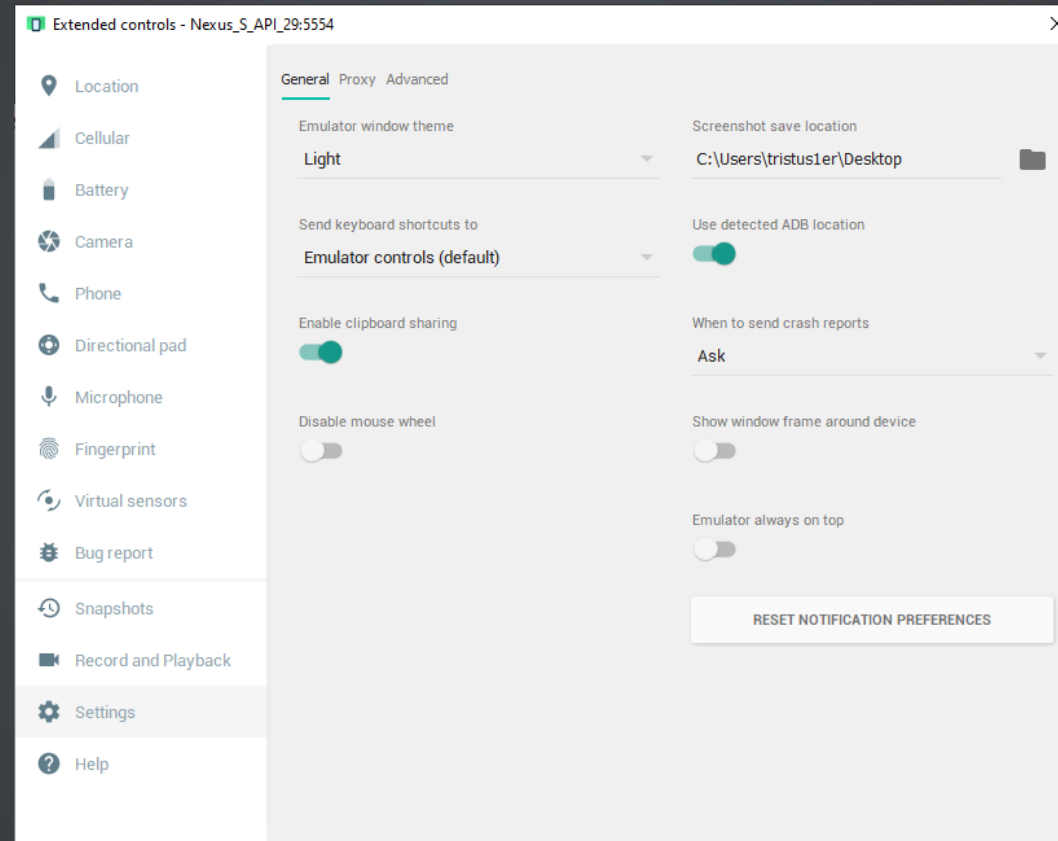
Émulateur

Enregistrer l'écran :



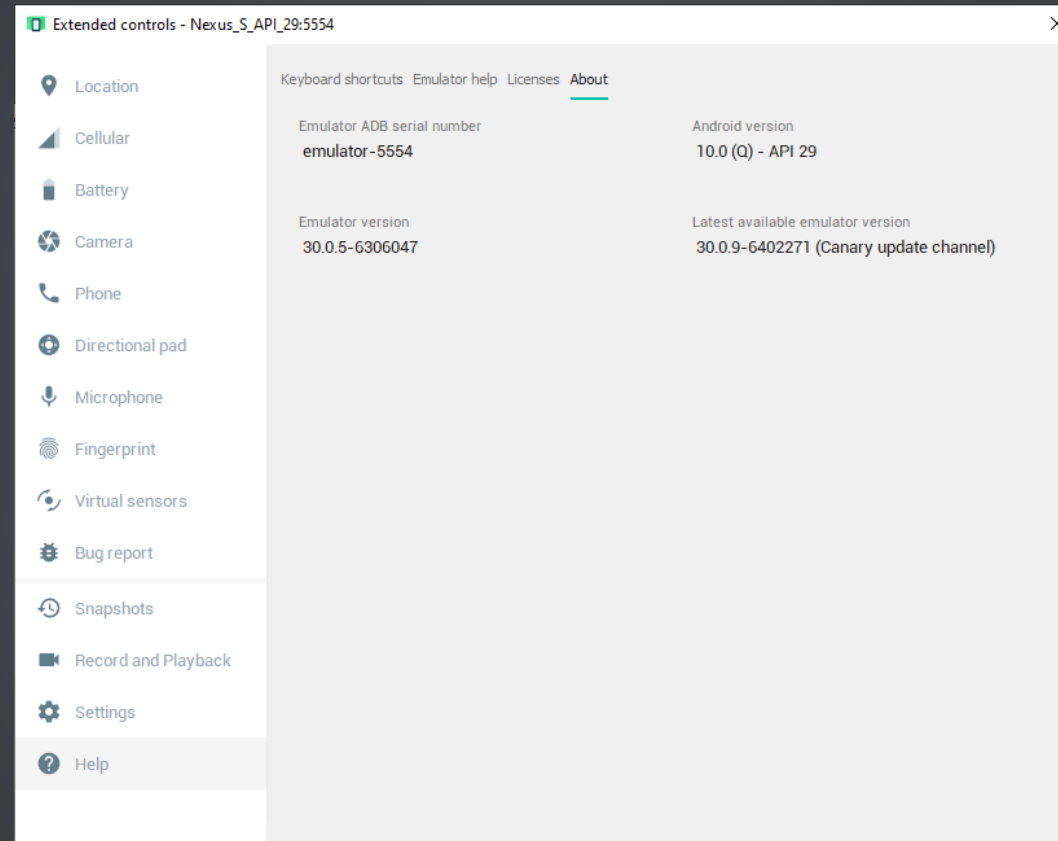
Émulateur

Définir certains paramètres :



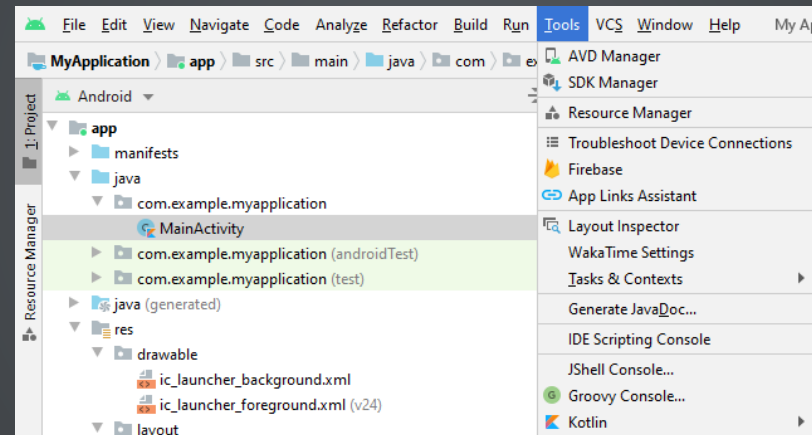
Émulateur

Obtenir des informations sur l'émulateur :



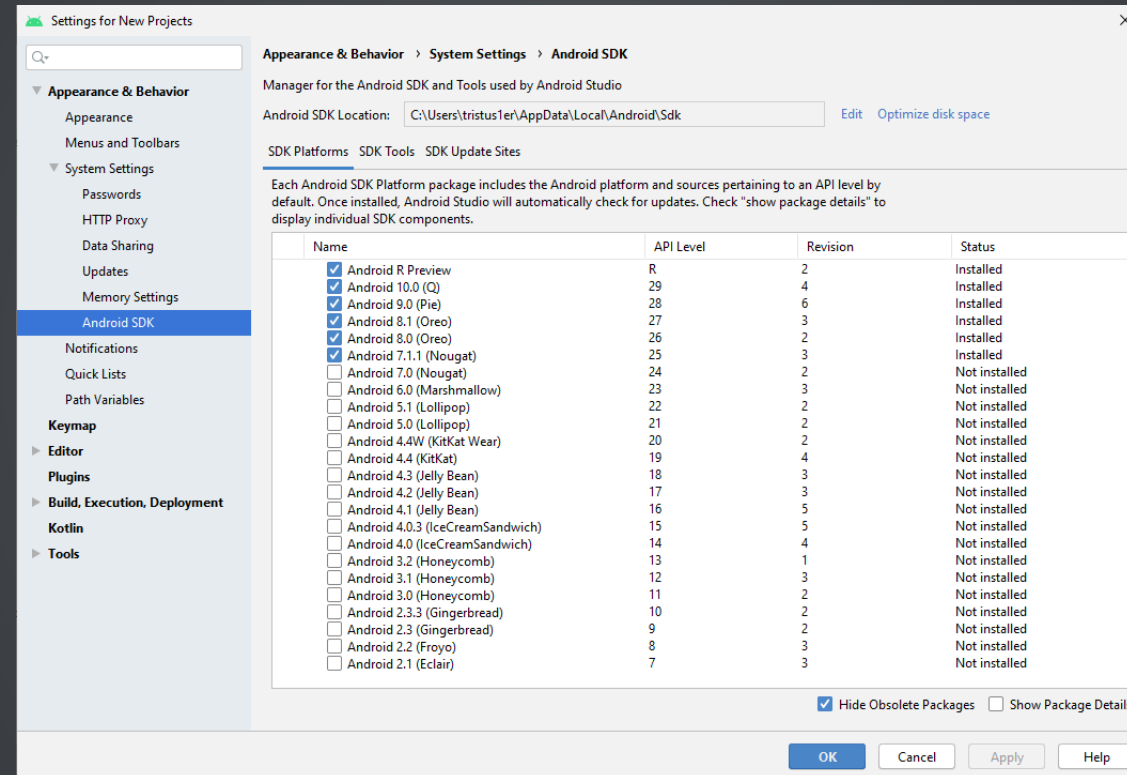
SDK

Lançons l'écran de gestion du SDK :
Tools/SDK Manager



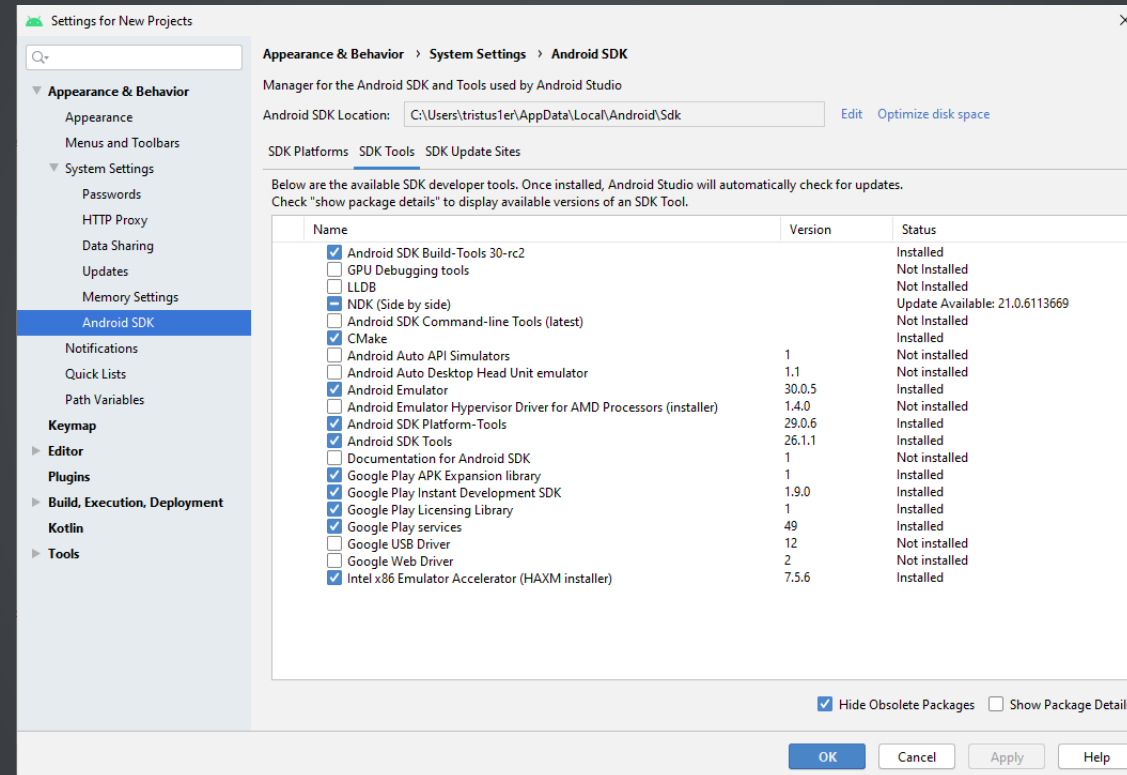
SDK

Nous pouvons choisir les versions de l'OS à télécharger :



SDK

Et les outils à installer :



Le Manifest

Le manifest est le fichier qui décrit l'application, pour les outils de compilation, pour le système d'exploitation et pour Google Play. On y retrouve principalement :

- Les interfaces avec l'extérieur : les activities, les services, ...
- Les permissions que nous allons utiliser.
- Le matériel nécessaire à notre application (NFC, Bluetooth, ...).

[Tous les détails ici.](#)

Le Manifest

Exemple

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.myapplication">
3     <application
4         android:allowBackup="true"
5         android:icon="@mipmap/ic_launcher"
6         android:label="@string/app_name"
7         android:roundIcon="@mipmap/ic_launcher_round"
8         android:supportsRtl="true"
9         android:theme="@style/AppTheme">
10     <activity android:name=".MainActivity">
11         <intent-filter>
12             <action android:name="android.intent.action.MAIN" />
13
14             <category android:name="android.intent.category.LAUNCHER" />
15         </intent-filter>
16     </activity>
17 </application>
```

[Copier](#)

La production de l'application, la publication

La production de l'application

Une fois développée, principalement via Android Studio (il existe d'autres moyens, par exemple via des outils de développement cross plate-forme), l'application est enregistrée dans un APK/App Bundle, qui contient :

- Le code de l'application.
- Les ressources.
- Les assets.
- Les certificats/signatures (pour l'App Bundle, la signature est effectuée par Google).
- Le Manifest.

Le fichier APK est un fichier ZIP qu'il est possible d'ouvrir avec n'importe quel utilitaire sachant traiter ce type de fichier (7Zip, Winzip, ...).

La production de l'application, la publication

La publication

Une fois packagée, l'application est le plus souvent publiée sur la boutique de Google : [Google Play](#). Il existe d'autres boutiques, des stores alternatifs, tels que :

- [F-Droid \(applications open source\)](#)
- [APK Miror](#)
- [App brain](#)
- [Amazon](#)
- [GETJAR](#)
- [Uptodown App Store](#)
- [Aptoide](#)
- [QooApp](#)

La production de l'application, la publication

Il peut aussi s'agir de stores de constructeurs, tels que :

- [Huawei AppGallery](#)
- [Samsung Galaxy App](#)
- [Boutique Mi](#)

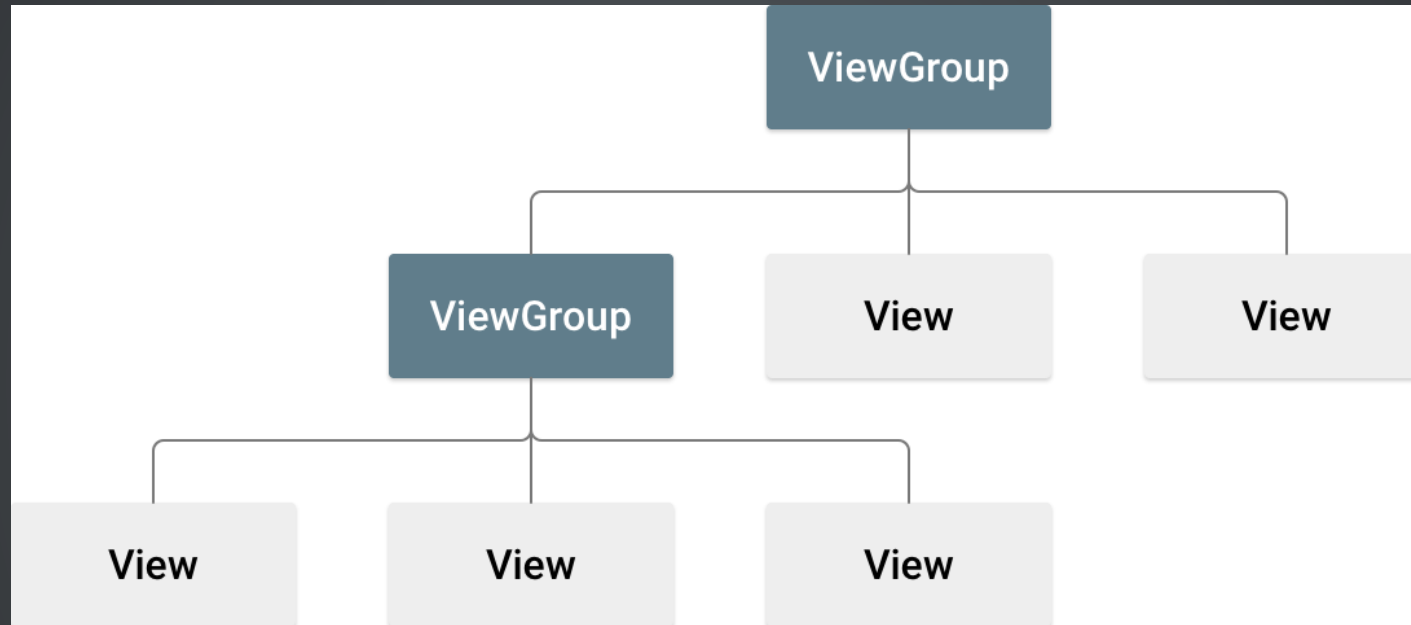
La production de l'application, la publication

La solution classique consiste quand même à publier sur [Google play store](#), via la [Google Play Console](#). Pour cela, il faudra un compte Google, et déboursier 25€ pour avoir la possibilité de publier une application sur le store, à vie.

Les interfaces utilisateurs

- Organisation générale du layout.
- Exemple de layouts : `LinearLayout`, `RelativeLayout`, `ConstraintLayout`.
- Les ressources : `drawables`, `string`, `styles`, ...
- La gestion événementielle.

Organisation générale du layout

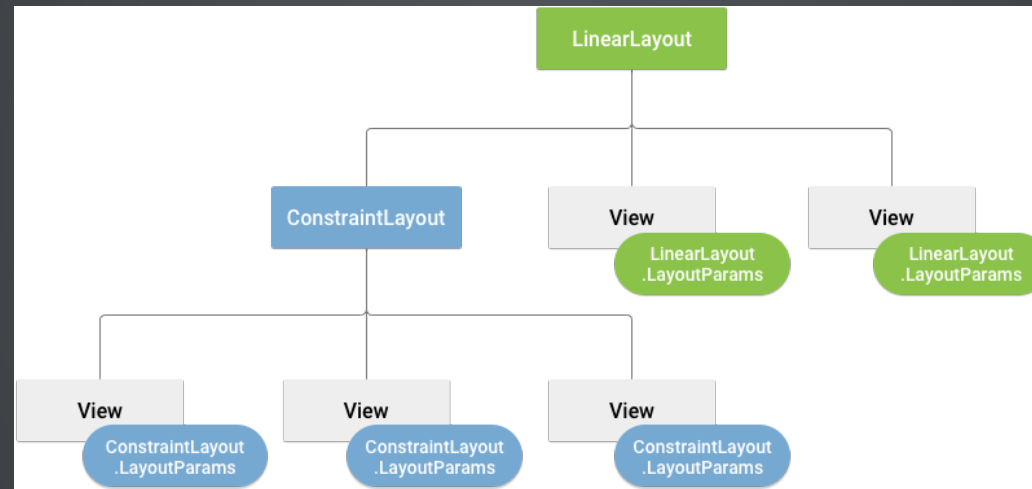


Source : <https://developer.android.com/guide/topics/ui/declaring-layout>

Organisation générale du layout

A chaque élément déclaré dans le layout, nous pouvons associer un id qui nous permettra de le retrouver/référencer dans le code (Java/Kotlin/XML).

Pour chaque layout, des contraintes particulières peuvent être appliquées (nous allons voir plusieurs layouts dans les slides suivantes).



Source : <https://developer.android.com/guide/topics/ui/declaring-layout>

Exemple de layouts

Exercices sur les Layouts

Dans tous les prochains exercices, vous n'utiliserez qu'un seul layout par écran.

Exemple de layouts

LinearLayout Exercice 1

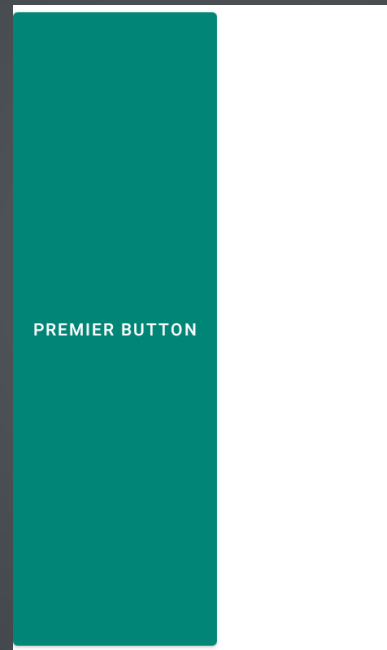
Les 2 boutons sont positionnés l'un au-dessous de l'autre, et prennent toute la largeur :



Exemple de layouts

LinearLayout Exercice 2

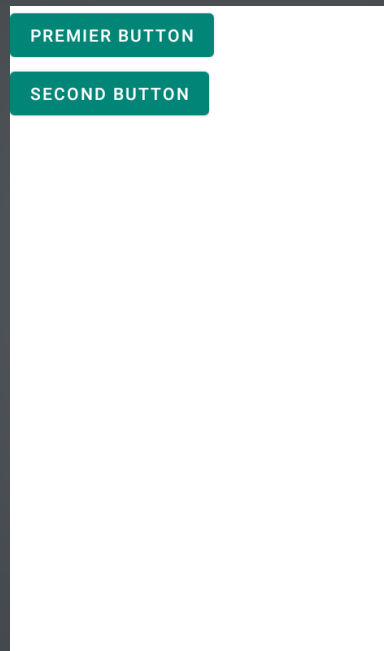
Le bouton prend toute la hauteur de l'écran :



Exemple de layouts

LinearLayout Exercice 3

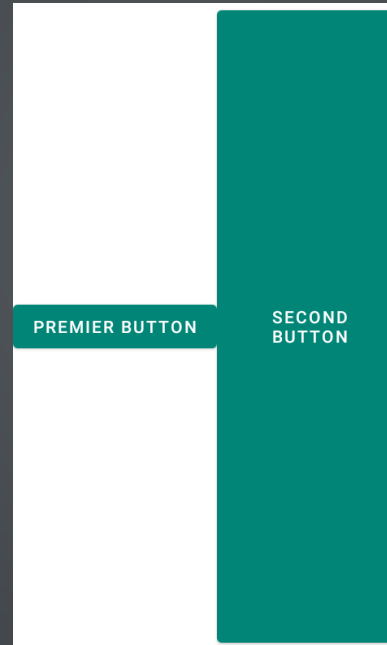
Les 2 boutons sont positionnés l'un au-dessous de l'autre, et prennent la largeur nécessaire à leur affichage :



LinearLayout Exercice 4

Exemple de layouts

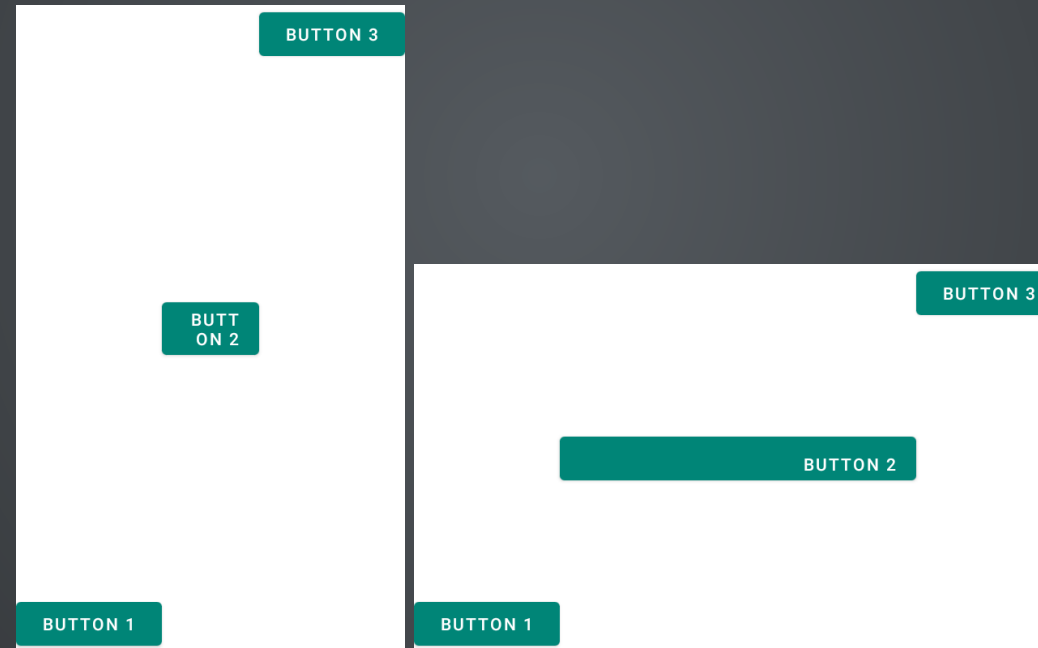
Les 2 boutons sont positionnés l'un à côté de l'autre, centré verticalement, le premier prend la hauteur nécessaire à son affichage, et le second prend toute la hauteur :



LinearLayout Exercice 5

Exemple de layouts

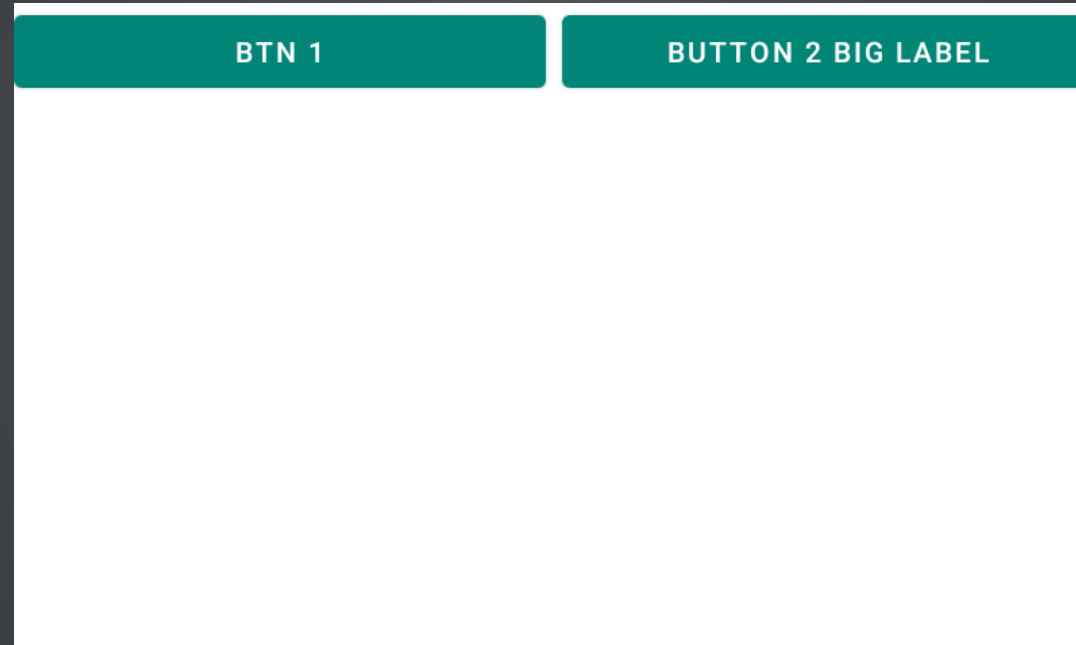
Les 3 boutons sont affichés les uns à côté des autres. Le bouton 1 est en bas, de largeur fixe : 120dp, celui du milieu prend la place disponible, et son texte est aligné en bas à droite. Le 3ième bouton est de largeur fixe : 120dp :



LinearLayout Exercice 6

Exemple de layouts

Les 2 boutons sont affichés l'un à côté de l'autre, et prennent exactement la moitié de la largeur de l'écran, avec 2 labels de taille très différente :



Exemple de layouts

Afficher un contenu très grand

Dans le cas de l'utilisation d'une listview, verticale, permettant d'ajouter beaucoup d'éléments sur notre écran, il est préférable d'englober notre LinearLayout dans une ScrollView, par exemple :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent">
5
6     <LinearLayout
7         android:layout_width="match_parent"
8         android:layout_height="wrap_content"
9         android:orientation="vertical">
10
11         <Button
12             android:layout_width="match_parent"
13             android:layout_height="wrap_content"
14             android:text="BTN" />
15
16         <Button
17             android:layout_width="match_parent"
```

[Copier](#)

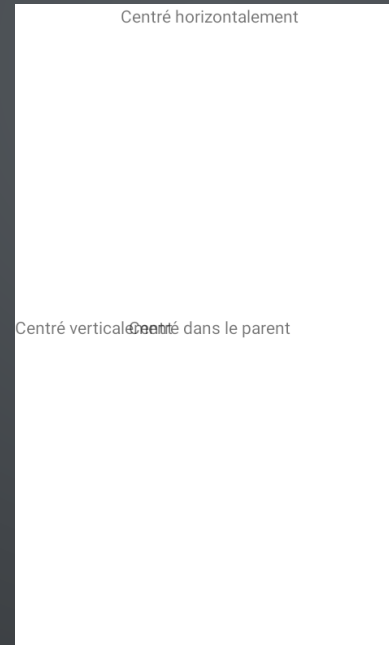
RelativeLayout Exercice 1

Exemple de layouts

Les 3 TextView sont alignés de la manière suivante :

- Centré horizontalement
- Centré verticalement
- Centré dans le parent

Que remarquez-vous d'inapproprié ?



RelativeLayout Exercice 2

Exemple de layouts

Nous allons positionner les TextView suivant les paroles de la chanson (dans le même ordre dans le XML) :

- En haut
- En bas
- A gauche
- A droite
- Ces soirées-là ! (centrée dans le parent)



RelativeLayout Exercice 3

Exemple de layouts

Nous allons maintenant positionner les TextView suivant les consignes suivantes :

- [I] En haut à gauche par défaut
- [II] En dessous de [I]
- [III] En dessous et à droite de [I]
- [IV] au-dessus de [V], bord aligné sur le bord gauche de [II]
- [V] En bas à droite



Exemple de layouts

TableLayout Exercice 1

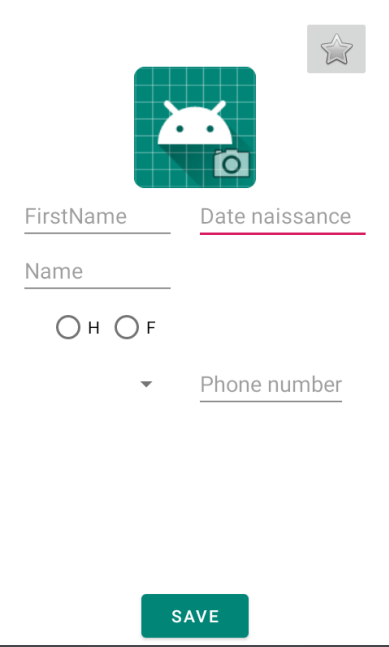
Le table layout nous permet de faire des alignements de type tableaux :

Les items précédés d'un V ouvrent un sous-menu	
N'ouvre pas un sous-menu	Non !
V Ouvre pas un sous-menu	Là oui !
V Ouvre pas un sous-menu	
Cet item s'étend sur 2 colonnes, il faut un très long texte.	

ConstraintLayout Exercise 1

Exemple de layouts

Le constraint layout est le dernier disponible. Il permet de réaliser des mises en forme complexes, tout en gardant une structure de layout plate (pas de layout imbriqué dans un autre layout) :

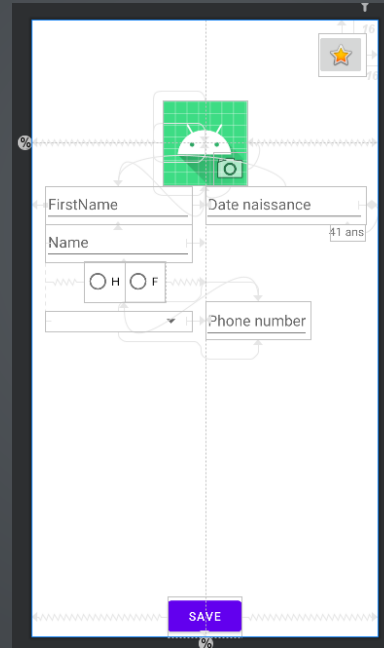


The image shows a mobile application form layout. At the top, there is a profile picture of an Android robot on a green grid background, with a camera icon in the bottom right corner. To the right of the profile picture is a star icon. Below the profile picture, there are two input fields: "FirstName" and "Date naissance". Below these, there is a "Name" input field. Under the "Name" field, there are two radio buttons labeled "H" and "F". Below the radio buttons, there is a dropdown menu with a downward arrow and the label "Phone number". At the bottom of the form, there is a green "SAVE" button.

Exemple de layouts

ConstraintLayout Exercise 1 (détails)

Le détail des contraintes mises en places pour réaliser cette mise en forme de l'écran :



Les ressources

Les différentes ressources

Les ressources peuvent être de plusieurs types :

- anim : des animations que l'on peut utiliser pour animer des éléments de notre interface graphique.
- drawable : des images, bitmap ou vectorielles.
- layout : l'organisation de nos écrans.
- menu : la description des menus (des activités, des fragments, ...)
- mipmap : les icônes de notre application.
- raw : des données brutes (mp3 par exemple).
- xml : des données en XML.

Les ressources

Les différentes ressources

- values
 - arrays : des tableaux (de chaînes de caractères ou d'entiers)
 - colors : la définition de nos couleurs.
 - bools : des booléens.
 - dimens : des dimensions (tailles d'images, de texte, de marges, ...).
 - integers : des valeurs numériques.
 - plurials : des chaînes de caractères prenant en compte le pluriel.
 - strings : nos chaînes de caractères.
 - styles : pour afficher nos éléments.

Les ressources

Accéder aux valeurs en Java/Kotlin

Nous allons voir plus en détail l'utilisation d'une ressource. Dans un premier temps, pour les récupérer, en Java/Kotlin, nous utiliserons la syntaxe suivante :

```
1 [package.]R.type.nom
2
3 // Par exemple
4 R.string.app_name
```

Copier

Les ressources

Accéder aux valeurs en XML

En XML la syntaxe est similaire :

```
1 @[package:]type/nom
2
3 // Par exemple :
4 @string/app_name
```

Copier

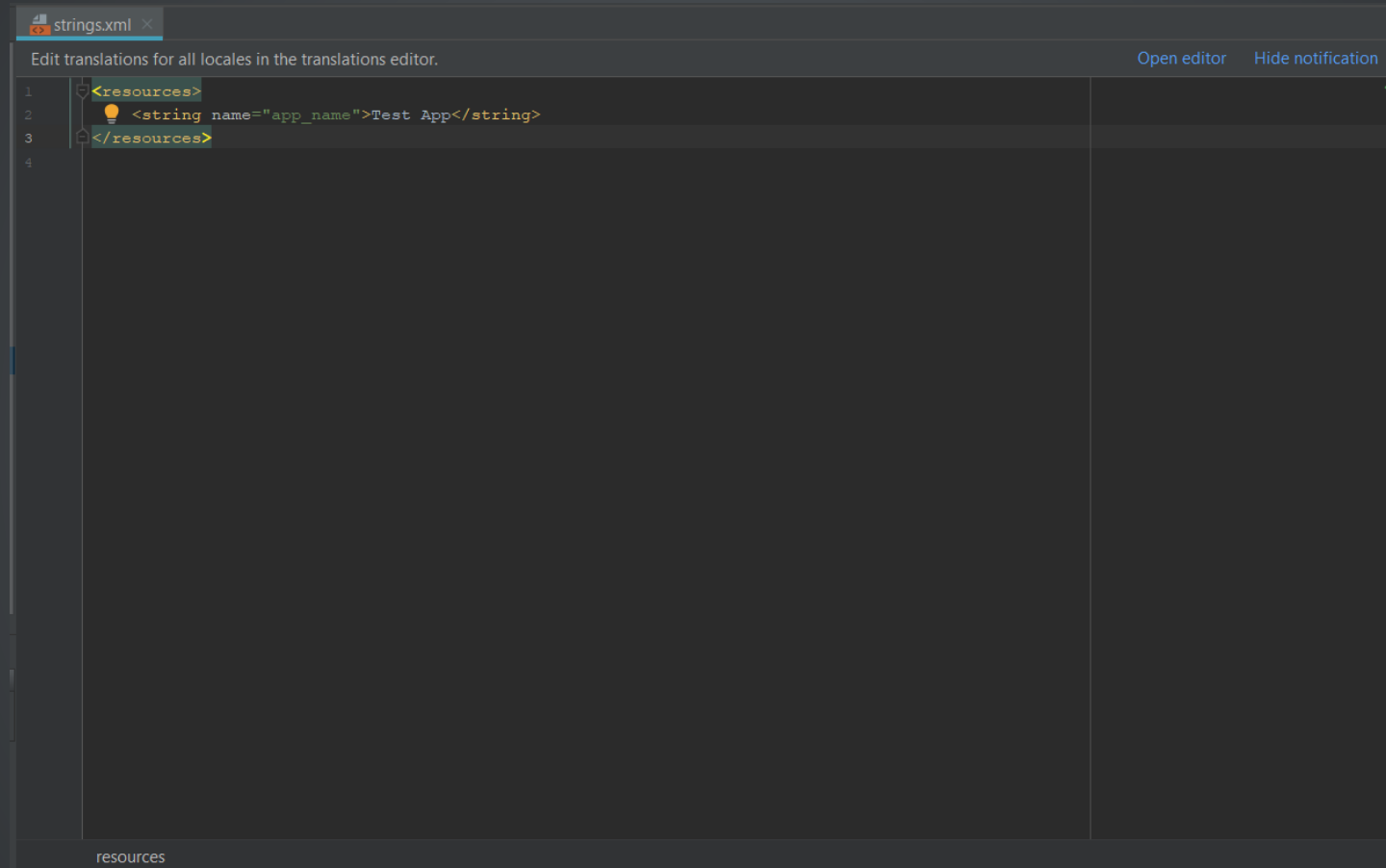
Les ressources

Internationalisation

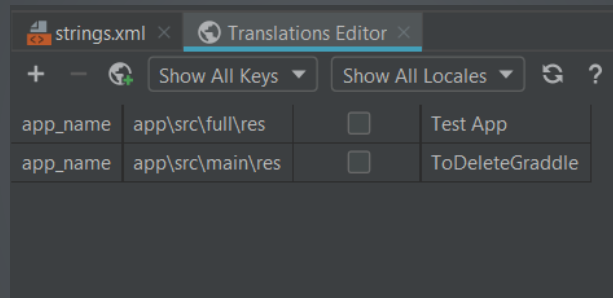
Voyons le fichier de ressource `strings.xml` et son utilisation pour internationaliser notre application.

- Ajouter une langue.
- Ajouter des clés.
- Tester dans l'éditeur.
- Tester sur l'émulateur.
- Tester sur le téléphone.

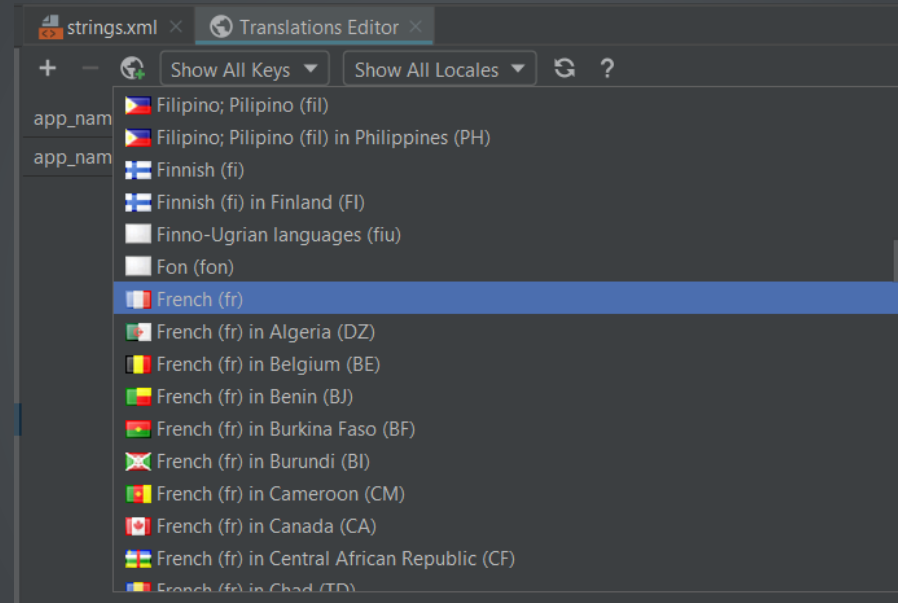
Les ressources



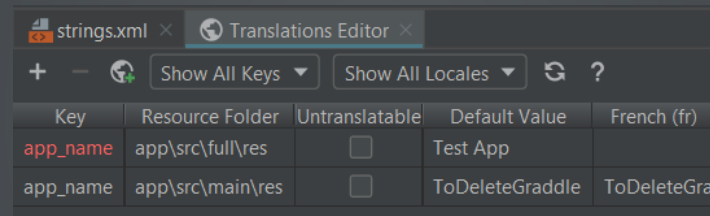
Les ressources



Les ressources

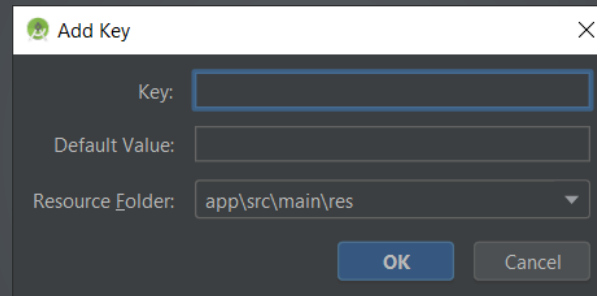


Les ressources



Key	Resource Folder	Untranslatable	Default Value	French (fr)
app_name	app\src\full\res	☐	Test App	
app_name	app\src\main\res	☐	ToDeleteGraddle	ToDeleteGra

Les ressources



A dialog box titled "Add Key" with a close button (X) in the top right corner. It contains three input fields: "Key:" (highlighted with a blue border), "Default Value:", and "Resource Folder:". The "Resource Folder:" field is a dropdown menu showing "app\src\main\res". At the bottom right are "OK" and "Cancel" buttons.

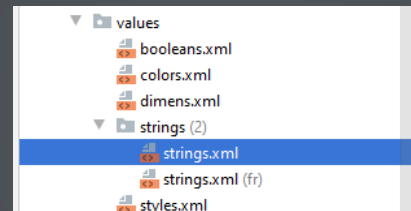
Key:

Default Value:

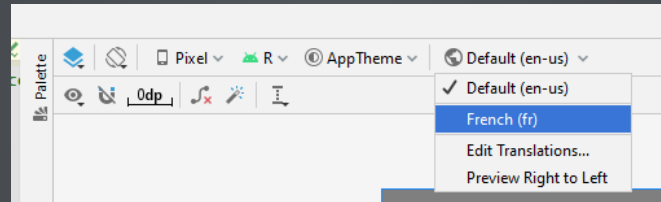
Resource Folder:

OK Cancel

Les ressources



Les ressources



Les ressources

Apostrophes et guillemets

Le format XML nous impose des contraintes par rapport aux caractères ' et ", regardons comment nous pouvons gérer ces cas :

```
1 <string name="good_example">"This'll work"</string>
2 <string name="good_example_2">This\'ll also work</string>
3 <string name="bad_example">This doesn't work</string>
4 <string name="bad_example_2">XML encodings don't work</string>
```

Copier

Les ressources

Formatage valeurs

Nous pouvons ajouter des variables dans notre texte, qui seront remplacées par les valeurs passées en paramètre, par exemple :

```
1 <string name="welcome_messages">Hello, %1$s! You have %2$d new messages.</string>
```

[Copier](#)

```
1 Resources res = getResources();  
2 String text = String.format(res.getString(R.string.welcome_messages), username, mailCount);
```

[Copier](#)

Les ressources

Le pluriel

Les ressources permettent aussi de gérer le pluriel :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <plurals name="numberOfSongsAvailable">
4         <!-- zero, one, two, few, many, other
5             As a developer, you should always supply "one" and "other"
6             strings. Your translators will know which strings are actually
7             needed for their language. Always include %d in "one" because
8             translators will need to use %d for languages where "one"
9             doesn't mean 1 (as explained above).
10            <item quantity="one">%d song found.</item>
11            <item quantity="other">%d songs found.</item>
12        </plurals>
13    </resources>
```

Copier

Les ressources

Exemple d'utilisation du pluriel

Il s'agit d'une valeur dynamique, nous utilisons donc du code pour définir la valeur de la chaîne de caractère, par exemple :

```
1 int count = getNumberOfSongsAvailable();  
2 Resources res = getResources();  
3 String songsFound = res.getQuantityString(R.plurals.numberOfSongsAvailable, count, count);
```

[Copier](#)

Les ressources

Mise en forme

Nous avons à disposition quelques outils de mise en forme de notre texte :

```
1 <b> for bold text.  
2 <i> for italic text.  
3 <u> for underline text.
```

Copier

Par exemple :

```
1 <string name="welcome">Welcome to <b>Android</b>!</string>
```

Copier

Les ressources

Tableaux

Nous pouvons aussi définir des tableaux, par exemple :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string-array name="planets_array">
4         <item>Mercury</item>
5         <item>Venus</item>
6         <item>Earth</item>
7         <item>Mars</item>
8     </string-array>
9 </resources>
```

[Copier](#)

Que l'on utilisera :

```
1 val planets = resources.getStringArray(R.array.planets_array)
```

[Copier](#)

Les ressources

Dimens

Le fichier `dimens.xml` contiendra des définitions de taille, par exemple la taille d'une image, d'une marge, d'un texte,

Nous avons plusieurs types de dimensions disponibles :

- `px` : mesure en pixel, à bannir
- `dp` : mesure qui s'adapte à la densité de pixel de l'écran.
- `sp` : mesure utilisée pour les textes, car elle prend en compte le paramétrage de l'utilisateur relatif à la taille de la police de caractère.

Les ressources

Colors

Nous pouvons définir toutes nos couleurs dans ce fichier. Par exemple :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <color name="colorPrimary">#6200EE</color>
4     <color name="colorPrimaryDark">#3700B3</color>
5     <color name="colorAccent">#03DAC5</color>
6 </resources>
```

Copier

Les ressources

Styles

Les styles nous permettent de regrouper des caractéristiques d'affichage d'éléments (des titres, des entêtes, ...). Par exemple :

```
1 <style name="Counter" parent="AppTheme">
2     <item name="android:textColor">@color/l4e_counter_text</item>
3     <item name="android:background">@drawable/rounded_corner</item>
4     <item name="android:padding">@dimen/l4e_counter_padding</item>
5 </style>
6
7 <style name="DialogTitle" parent="AppTheme">
8     <item name="android:padding">@dimen/l4e_dialog_title_text_padding</item>
9     <item name="android:textSize">@dimen/l4e_dialog_title_text_size</item>
10    <item name="android:textStyle">bold</item>
11 </style>
```

[Copier](#)

La gestion événementielle

Ajout d'un bouton

Nous allons ajouter un bouton à notre interface, via l'éditeur graphique, ou via l'éditeur texte.

La gestion événementielle

Afin de réagir aux click de n'utilisateur, nous allons brancher un `View.OnClickListener`, ou un `View.OnLongClickListener`.

Pour cela, nous avons 4 options que nous allons détailler :

- Par héritage
- Classe anonyme
- Attribut
- XML

Voyons tout cela en détail.

La gestion événementielle

Par héritage

Suivons les étapes suivantes :

- Faisons étendre notre MainActivity de View.OnClickListener.
- Implémentons la méthode en utilisant l'ampoule rouge proposée par l'éditeur.
- Remplissons la méthode onClick générée.
- Associons notre listener à un bouton

Contenu de la méthode onClick :

```
1 when(v?.id){  
2     R.id.activity_main_button_test -> Log.d(TAG, "onCreate click")  
3 }
```

Copier

Association du listener avec notre classe.

```
1 findViewById<Button>(R.id.activity_main_button_test).setOnClickListener(this@MainActivity)
```

Copier

La gestion événementielle

Par attribut

Déclarons un attribut de classe :

```
1 private val btnListener: View.OnClickListener = View.OnClickListener {  
2     when (it?.id) {  
3         R.id.activity_main_button_test -> Log.d(TAG, "onCreate click")  
4     }  
5 }
```

[Copier](#)

Attribuons notre listener à notre bouton :

```
1 findViewById<Button>(R.id.activity_main_button_test).setOnClickListener(btnListener)
```

[Copier](#)

La gestion événementielle

Par lambda

C'est la méthode la plus courante :

```
1 findViewById<Button>(R.id.activity_main_button_test).setOnClickListener(){ Log.d(TAG, "onCreate click") }
```

Copier

La gestion événementielle

Par XML

Dans le layout de notre activity, sur le bouton, ajoutons l'attribut onClick :

```
1 android:onClick="handleClick"
```

Copier

Créons la méthode en utilisant l'aide de l'éditeur, il nous reste plus qu'à implémenter la méthode.

Les éléments graphiques

Plusieurs éléments graphiques de base sont à notre disposition, nous allons en voir plusieurs, et la correspondance entre la partie graphique et la partie texte.

Les éléments graphiques

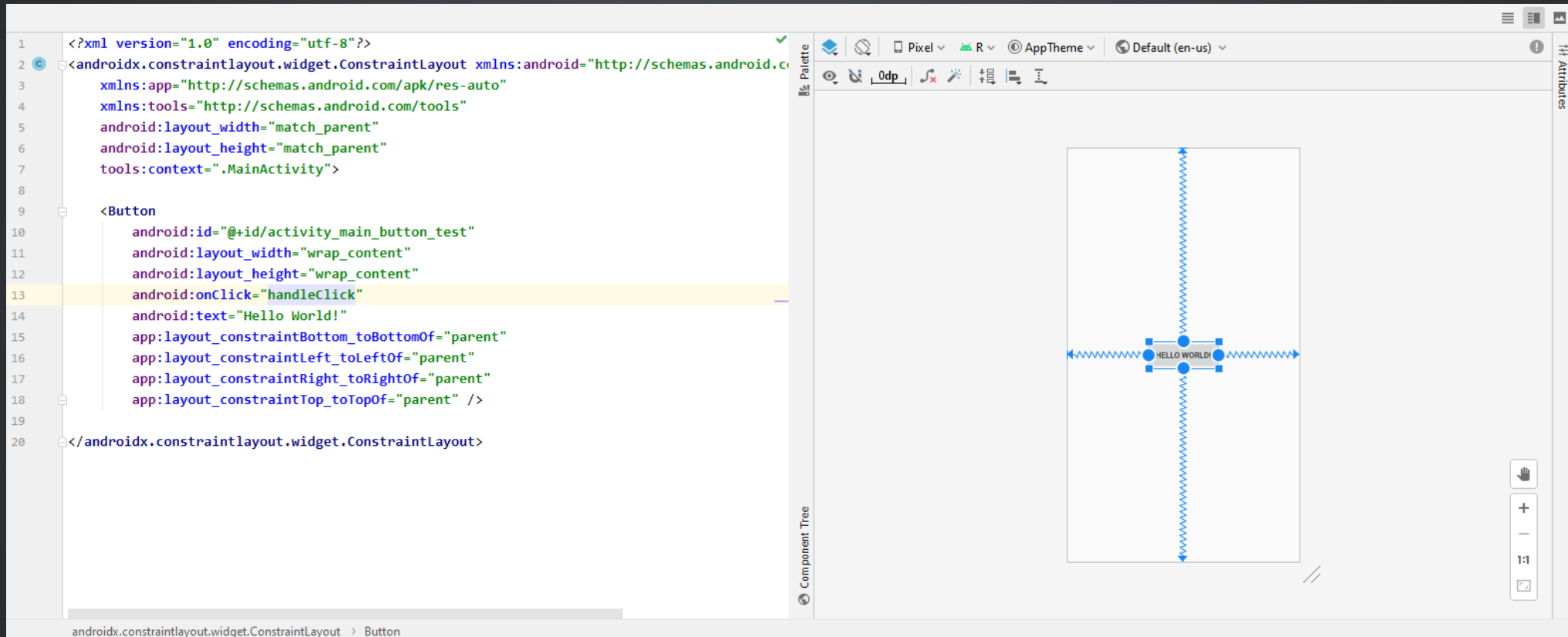
Choix du mode d'édition =>

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity">
8
9      <Button
10         android:id="@+id/activity_main_button_test"
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:onClick="handleClick"
14         android:text="Hello World!"
15         app:layout_constraintBottom_toBottomOf="parent"
16         app:layout_constraintLeft_toLeftOf="parent"
17         app:layout_constraintRight_toRightOf="parent"
18         app:layout_constraintTop_toTopOf="parent" />
19
20 </androidx.constraintlayout.widget.ConstraintLayout>
```

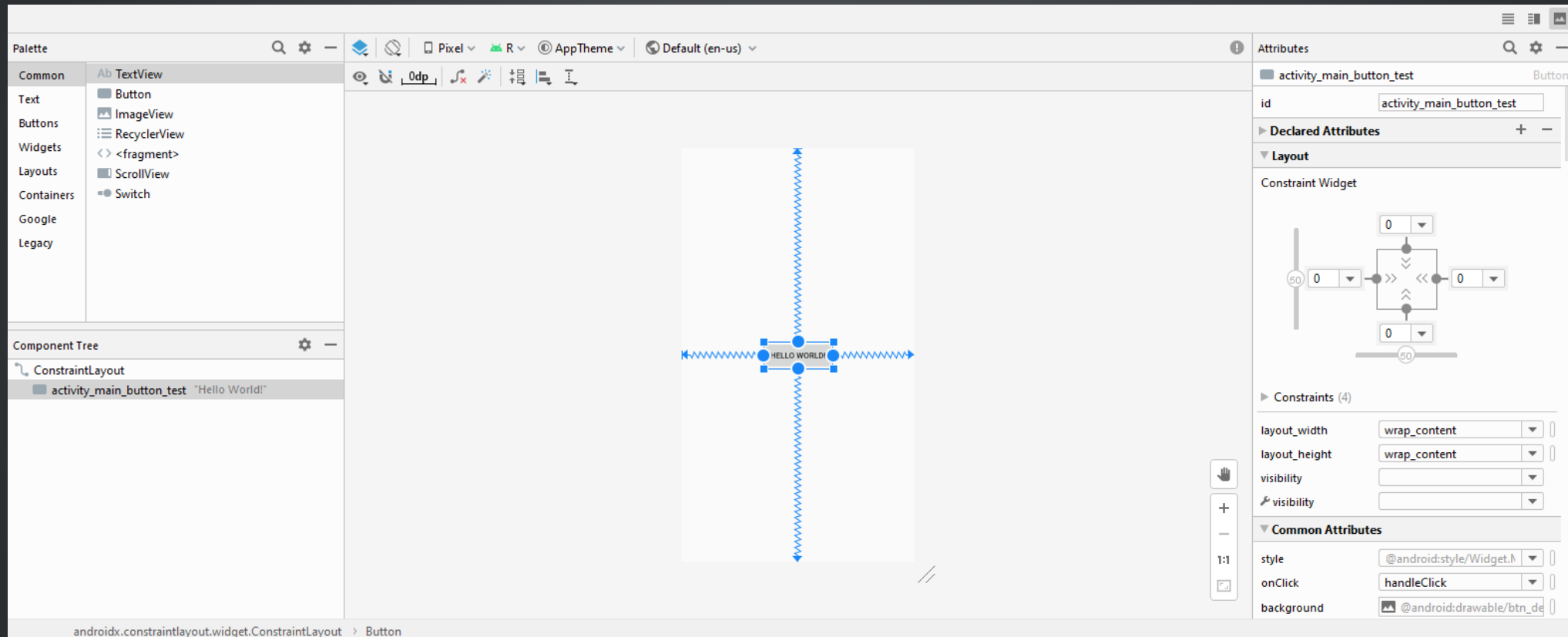
androidx.constraintlayout.widget.ConstraintLayout > Button

Éditeur : Split

Les éléments graphiques

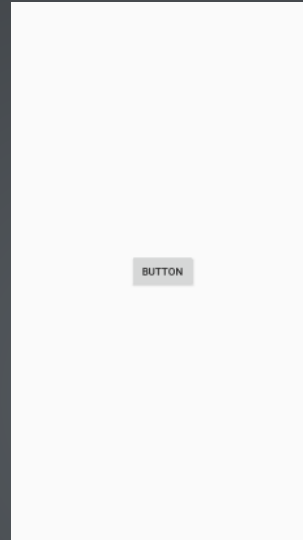


Les éléments graphiques



Button

Les éléments graphiques

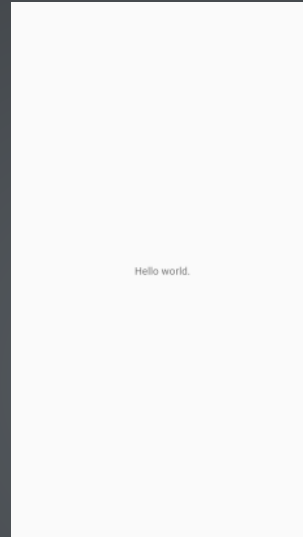


```
1 <Button
2     android:id="@+id/activity_main_button_test"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     android:onClick="handleClick"
6     android:text="Button"
7     app:layout_constraintBottom_toBottomOf="parent"
8     app:layout_constraintLeft_toLeftOf="parent"
9     app:layout_constraintRight_toRightOf="parent"
10    app:layout_constraintTop_toTopOf="parent" />
```

Copier

Texte affiché

Les éléments graphiques



```
1 <TextView
2     android:id="@+id/activity_main_textView_test"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     android:text="Hello world."
6     app:layout_constraintBottom_toBottomOf="parent"
7     app:layout_constraintLeft_toLeftOf="parent"
8     app:layout_constraintRight_toRightOf="parent"
9     app:layout_constraintTop_toTopOf="parent" />
```

Copier

Champ texte

Les éléments graphiques



```
1 <EditText
2     android:id="@+id/activity_main_editText_test"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     android:hint="Enter your first name"
6     android:inputType="textPersonName"
7     app:layout_constraintBottom_toBottomOf="parent"
8     app:layout_constraintLeft_toLeftOf="parent"
9     app:layout_constraintRight_toRightOf="parent"
10    app:layout_constraintTop_toTopOf="parent" />
```

Copier

Les éléments graphiques

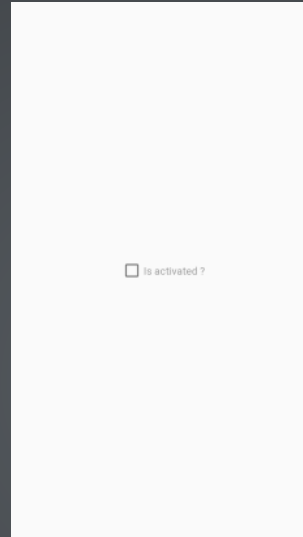
Champ texte (Material design)

Nous pouvons aussi utiliser la version moderne du champ texte, avec le `LiveTemplate` `and_material_design_edittext`.

Quelles différences notez vous ?

Case à cocher

Les éléments graphiques

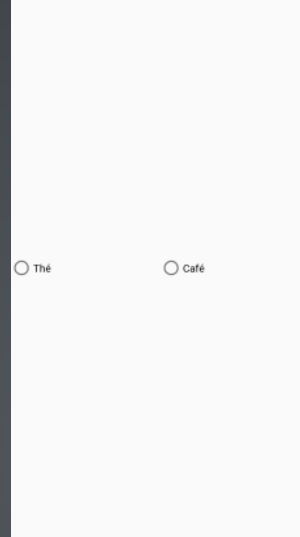


```
1 <CheckBox
2     android:id="@+id/activity_main_checkBox_test"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     android:hint="Is activated ?"
6     app:layout_constraintBottom_toBottomOf="parent"
7     app:layout_constraintLeft_toLeftOf="parent"
8     app:layout_constraintRight_toRightOf="parent"
9     app:layout_constraintTop_toTopOf="parent" />
```

Copier

Boutons de choix

Les éléments graphiques

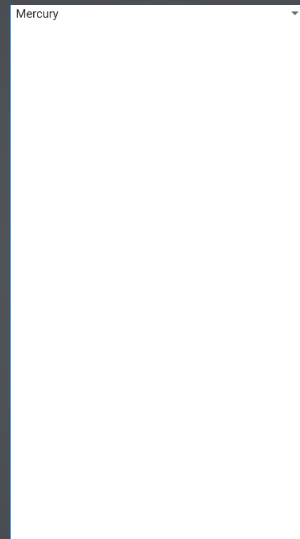


```
1 <RadioGroup
2     android:layout_width="match_parent"
3     android:layout_height="wrap_content"
4     android:orientation="horizontal"
5     app:layout_constraintBottom_toBottomOf="parent"
6     app:layout_constraintLeft_toLeftOf="parent"
7     app:layout_constraintRight_toRightOf="parent"
8     app:layout_constraintTop_toTopOf="parent">
9
10    <RadioButton
11        android:layout_width="wrap_content"
12        android:layout_height="wrap_content"
13        android:layout_weight="1"
14        android:text="Thé" />
15
16    <RadioButton
17        android:layout_width="wrap_content"
```

Copier

Les éléments graphiques

Liste de choix



Dans la méthode onCreate de notre Activity utilisons le LiveTemplate : [android_spinner_string_array](#).

Les éléments graphiques

Liste de choix encore plus rapide.

Gardons le string-array contenant la liste des planètes, retirons tout le code pour gérer le Spinner et ajoutons l'attribut `android:entries="@array/planets_array"`.

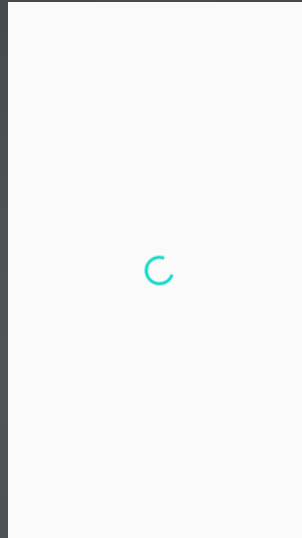
```
1 <Spinner
2     android:layout_width="match_parent"
3     android:layout_height="wrap_content"
4     android:entries="@array/planets_array" />
```

Copier

Régulièrement des facilités de développement sont disponibles dans le SDK ou l'IDE.

Barre de progression

Les éléments graphiques

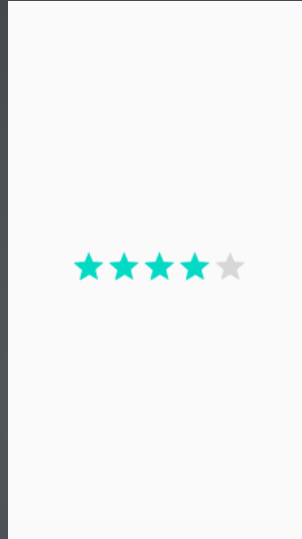


```
1 <ProgressBar
2     android:id="@+id/activity_main_progressBar_test"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     android:indeterminate="true"
6     app:layout_constraintBottom_toBottomOf="parent"
7     app:layout_constraintLeft_toLeftOf="parent"
8     app:layout_constraintRight_toRightOf="parent"
9     app:layout_constraintTop_toTopOf="parent" />
```

Copier

Notation (étoiles)

Les éléments graphiques

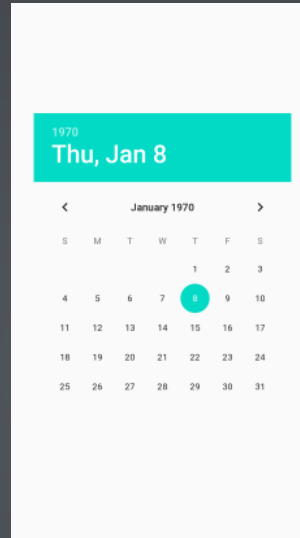


```
1 <RatingBar
2     android:id="@+id/activity_main_ratingBar_test"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     android:numStars="5"
6     android:rating="4"
7     app:layout_constraintBottom_toBottomOf="parent"
8     app:layout_constraintLeft_toLeftOf="parent"
9     app:layout_constraintRight_toRightOf="parent"
10    app:layout_constraintTop_toTopOf="parent" />
```

Copier

Date

Les éléments graphiques

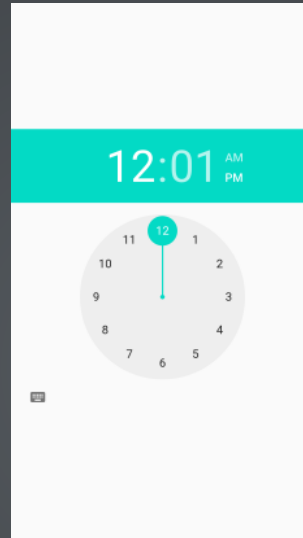


```
1 <DatePicker
2     android:id="@+id/activity_main_datePicker_test"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     app:layout_constraintBottom_toBottomOf="parent"
6     app:layout_constraintLeft_toLeftOf="parent"
7     app:layout_constraintRight_toRightOf="parent"
8     app:layout_constraintTop_toTopOf="parent" />
```

Copier

Time

Les éléments graphiques

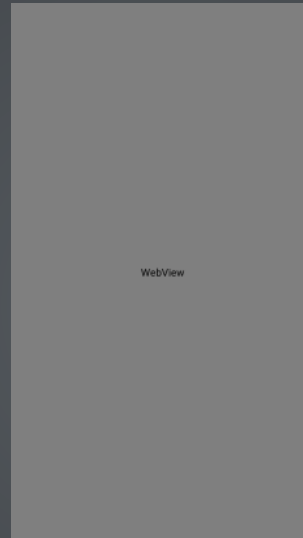


```
1 <TimePicker
2     android:id="@+id/activity_main_timePicker_test"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     app:layout_constraintBottom_toBottomOf="parent"
6     app:layout_constraintLeft_toLeftOf="parent"
7     app:layout_constraintRight_toRightOf="parent"
8     app:layout_constraintTop_toTopOf="parent" />
```

Copier

Les éléments graphiques

Vue Web



```
1 <WebView
2     android:id="@+id/activity_main_webView_test"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent" />
```

Copier

Les éléments graphiques

Vue Web (suite)

Pour fonctionner, nous allons préciser l'URL que nous souhaitons afficher dans la méthode onCreate de notre Activity :

```
1 var webView = findViewById<WebView>(R.id.activity_main_webView_test)
2 webView.loadUrl("http://light4events.fr")
```

Copier

Quel est le problème qui empêche la page de s'afficher ? Comment résoudre le problème ?
Que remarque-t-on quand on clique sur un lien ? Comment résoudre le problème ?

Les éléments graphiques

Material design

Nous avons utilisé une forme de l'EditText en Material Design, nous avons d'autres LiveTemplate à disposition, en voici la liste exhaustive :

- `and_material_design_divider`
- `and_material_design_edittext`
- `and_material_design_progress`
- `and_material_design_slider`
- `and_material_design_switch`

Le modèle de composants

- Les Intents et leur gestion par l'activité.
- La relation activité mère-fille.
- Les fragments, les services, les IntentServices.

Les Intents et leur gestion par l'activité

Les différents modules applicatifs ne sont pas directement instanciés par le développeur, un bus de messages permet au système de choisir le composant à monter en mémoire.

Les messages sont de type Intent.

Les valeurs envoyées dans l'Intent sont contenues dans un Bundle (optionnel).

Les intentions décrivent quelle est l'opération qui devra être effectuée.

Plusieurs composants applicatifs peuvent répondre à un même Intent, dans ce cas, l'utilisateur aura alors le choix.

Nous allons tester plusieurs Intent simple pour mettre en œuvre ces principes.

Les Intents et leur gestion par l'activité

LiveTemplates Intent

Nous avons à notre disposition plusieurs intents, entre autres :

- `and_intent_mail` : pour envoyer un email avec les champs préremplis.
- `and_intent_map` : pour ouvrir l'application Maps affichant une coordonnée.
- `and_intent_phone_call` : pour lancer le numéroteur téléphonique.
- `and_intent_sms` : pour envoyer un SMS (pour lancer l'application de SMS, prête à envoyer le SMS).
- `and_intent_web` : pour lancer un navigateur web pour afficher une URL.

Les Intents et leur gestion par l'activité

De la même manière, il est possible de déclarer dans le manifest des capacités de nos Activity à gérer des Intents. Par défaut l'Activity principale (MainActivity en général) gère l'appel par le Launcher. Dans le AndroidManifest.xml, nous voyons les lignes suivantes :

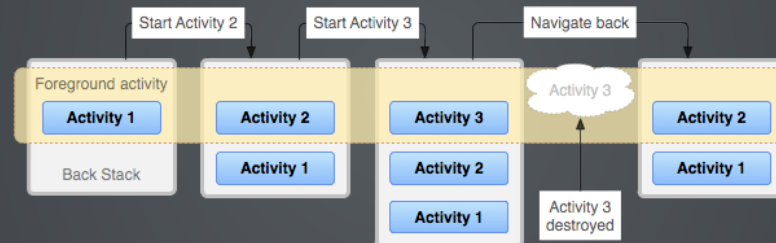
```
1 <intent-filter>
2     <action android:name="android.intent.action.MAIN" />
3     <category android:name="android.intent.category.LAUNCHER" />
4 </intent-filter>
```

[Copier](#)

La relation activité mère-fille

Une application contient souvent de multiples activités.

Lorsqu'on en lance une, celle précédemment ouverte se stoppe (pause). La nouvelle activité se retrouve au-dessus de la pile de type « last in, first out » c'est-à-dire « dernier arrivé, premier sorti ».



Référence.

La relation activité mère-fille

Mise en oeuvre

Pour visualiser les principales étapes du cycle de vie de notre activity, utilisons le LiveTemplate : `and_lifecycle_activity`, dans la classe, mais en dehors d'une méthode.

Au besoin, nous utilisons le LiveTemplate : `and_tag` pour définir la constante TAG au début de notre classe.

Démarrons notre application, regardons les logs et tournons l'écran.

Que remarquons-nous ?

La relation activité mère-fille

Communication inter Activity

Nous allons maintenant voir comment lancer une nouvelle Activity, et lui envoyer des informations. Pour cela, nous allons passer par plusieurs étapes :

- Ajoutons un champ texte à notre interface.
- Récupérons la valeur du texte saisie.
- Envoyons-la à notre Activity2.
- Affichons la dans l'Activity2.

La relation activité mère-fille

Nous déclarons le champ texte et le bouton dans notre layout :

```
1 <EditText
2     android:id="@+id/activity_main_editText_value"
3     android:layout_width="match_parent"
4     android:layout_height="wrap_content"
5     app:layout_constraintBottom_toBottomOf="parent"
6     app:layout_constraintEnd_toEndOf="parent"
7     app:layout_constraintStart_toStartOf="parent"
8     app:layout_constraintTop_toTopOf="parent" />
9
10 <Button
11     android:id="@+id/activity_main_button_send"
12     android:layout_width="match_parent"
13     android:layout_height="wrap_content"
14     android:text="Envoyer"
15     app:layout_constraintTop_toBottomOf="@id/activity_main_editText_value" />
```

Copier

La relation activité mère-fille

Créons notre Activity2 via le click droit/New/Activity/Empty Activity.

Lançons cette activité lors du click bouton, avec en paramètre le texte de l'EditText : Nous déclarons une variable d'instance :

```
1 lateinit var etValue: EditText
```

Copier

Puis ajoutons le code suivant à notre méthode onCreate :

```
1 etValue = findViewById<EditText>(R.id.activity_main_editText_value)
2 findViewById<Button>(R.id.activity_main_button_send).setOnClickListener() {
3     var intent = Intent(this@MainActivity, Main2Activity::class.java)
4     intent.putExtra(INTENT_PARAMS_EXTRA_VALUE, etValue.text.toString())
5     startActivity(intent)
6 }
```

Copier

Je m'aide du LiveTemplate [and_intent_activity](#) pour écrire le code.

La relation activité mère-fille

Du côté de Main2Activity, je récupère la valeur envoyée, dans la méthode onCreate, avec le code :

```
1 var value = savedInstanceState?.getString(INTENT_PARAMS_EXTRA_VALUE) ?: intent.getStringExtra(INTENT_PARAMS_EXTRA_VALUE)
```

Copier

Je m'aide du LiveTemplate [and_intent_onCreate](#) pour écrire le code.

J'ajoute le même champ et bouton que la première activité, et j'affecte la valeur reçue à l'EditText :

```
1 findViewById<EditText>(R.id.activity_main2_editText_value).setText(value)
```

Copier

La relation activité mère-fille

Une activity peut aussi retourner des valeurs, par exemple demandons au système de sélectionner un contact de notre répertoire téléphonique.

Pour cela, sur un click de bouton, par exemple, je vais lancer le code généré par le LiveTemplate [and_intent_select_contact](#) et suivre les instructions.

La relation activité mère-fille

Retour de valeur

Voyons ensemble comment ce mécanisme fonctionne, je souhaite demander une information à une Activity, qui se lancera et me renverra une valeur (dans notre cas, une chaîne de caractère, mais cela peut être n'importe quel type d'information : un contact de notre répertoire, une image, ...).

Pour cela 3 étapes sont nécessaires :

- Activity1 : lancer l'Activity2 avec la méthode : `startActivityForResult`
- Activity2 : effectuer notre traitement (dans notre cas, laisser l'utilisateur saisir un texte et valider avec un bouton).
- Activity1 : récupérer la valeur envoyée par l'Activity2.

La relation activité mère-fille

Dans l'Activity, déclarons la constante :

```
1 const val REQUEST_CODE_GET_TEXT = 4567 // Arbitrary value
```

Copier

Dans l'Activity, toujours, lançons l'appel à l'Activity2, en attendant une réponse :

```
1 var intent = Intent(this@MainActivity, Main2Activity::class.java)
2 startActivityForResult(intent, REQUEST_CODE_GET_TEXT)
```

Copier

La relation activité mère-fille

Dans l'Activity2, je récupère la valeur saisie par l'utilisateur et je la renvoie. Pour cela, je m'aide du LiveTemplate : `and_intent_returnValue`. Nous pouvons maintenant récupérer la valeur renvoyée par l'Activity2, dans l'Activity1.

Pour cela, nous utilisons le LiveTemplate `and_onActivityResult` en dehors d'une méthode dans la classe MainActivity.

La relation activité mère-fille

Pour améliorer l'expérience utilisateur, nous pouvons préciser quel type d'informations l'utilisateur doit saisir afin de présenter un clavier adapté.

Les détails : [Specify the input method type](#).

Par exemple pour un numéro de téléphone :

```
1 <EditText
2     android:id="@+id/phone"
3     android:layout_width="fill_parent"
4     android:layout_height="wrap_content"
5     android:hint="@string/phone_hint"
6     android:inputType="phone" />
```

Copier

La relation activité mère-fille

Gestion du bouton OK

Nous pouvons aussi modifier le bouton OK/Valider du clavier pour refléter l'action à réaliser et déclencher le code directement depuis l'action sur le clavier, au lieu de passer par un bouton. Par exemple pour notre champ téléphone :

```
1 android:hint="@string/phone_hint"
2 android:inputType="phone"
3 android:imeOptions="actionSend"
```

[Copier](#)

Dans notre Activity, nous gérons le click sur le bouton "Envoyer" du clavier avec :

```
1 etValue.setOnEditorActionListener(){v, actionId, event ->
2     if(actionId == EditorInfo.IME_ACTION_SEND){
3         Log.d(TAG, "Action send to be handled.")
4         true
5     } else {
6         false
7     }
8 }
```

[Copier](#)

Les fragments, les services, les IntentServices

Fragments

Étudions les templates :

- click droit / New / Activity / Primary/Detail Flow
- click droit / New / Activity / Tabbed Activity

Les fragments, les services, les IntentServices

Intent service

Créons un IntentService en utilisant le template :
File / New / Service / Service (IntentService)

Les fragments, les services, les IntentServices

Service

Créons un Service en utilisant le template :
File / New / Service / Service

Les fragments, les services, les IntentServices

Création d'une implémentation d'exemple (en Kotlin) : un service de génération de random.
Pour cela dans notre service, utilisons dans la classe de service un LiveTemplate pour écrire l'implémentation du service.

```
1 and_service_implementation
```

[Copier](#)

Pour appeler notre service (en Kotlin), nous utiliserons dans notre activity, un LiveTemplate, et nous suivrons à chaque fois les indications en TODO.

```
1 and_service_calling
```

[Copier](#)

La gestion des données

- Les préférences.
- Les fichiers, le stockage interne et externe.
- SQLite / Room.
- Les Content Provider.

Les préférences

Stocker une préférence

Le framework nous met à disposition une possibilité de stocker des informations dans l'application : les `SharedPreferences`. Sous forme de clé/valeur, nous pouvons stocker de l'information, dans un fichier, lié à l'application. Pour cela, nous utilisons le code suivant :

```
1 val sharedPref = PreferenceManager.getDefaultSharedPreferences(applicationContext)
2 val editor = sharedPref.edit()
3 editor.putString(SHARED_PREFS_VALUE, "Value")
4 editor.commit()
```

Copier

Ou plus rapidement le LiveTemplate : [and_SharedPreferences_save](#).

Nous n'oublions pas d'ajouter au gradle, la dépendance :

```
implementation 'androidx.preference:preference-ktx:1.2.1'
```

Les préférences

Charger une préférence

Pour récupérer la valeur stockée, nous utilisons :

```
1 val sharedPref = PreferenceManager.getDefaultSharedPreferences(applicationContext)
2 val defaultValue = resources.getInteger(R.string.value_default)
3 sharedPref.getInt(SHARED_PREFS_VALUE, defaultValue)
```

Copier

Ou plus rapidement le LiveTemplate : [and_SharedPreferences_load](#).

Nous pouvons bien entendu mutualiser l'objet sharedPref.

La version plus moderne consiste à utiliser un [datastore](#).

Les fichiers

```
1 @file:JvmName("FileTools")
2
3 package com.example.myapplication
4
5 import android.content.Context
6 import java.io.File
7
8 fun readFile(context: Context, isExternal: Boolean, path: String, fileName: String): String {
9     val currentFile = getFile(context, isExternal, path, fileName)
10
11     // READ
12     // val inputAsString = FileInputStream(currentFile).bufferedReader().use { it.readText() }
13
14     var sb = StringBuilder()
15     currentFile?.forEachLine { sb.append( "$it\n" ) }
16
17     return sb.toString()
18 }
```

[Copier](#)

Les fichiers

Utilisation

Utilisons les méthodes que nous venons d'écrire :

```
1 var isExternal = false
2 var path = "trs"
3 var fileName = "test.txt"
4 var append = true
5
6 writeFile(this@MainActivity, isExternal, path, fileName, "AAAAA\n", append)
7
8 val stringContent = readFile(this@MainActivity, isExternal, path, fileName)
9 if (BuildConfig.DEBUG) {
10     Log.d(TAG, "onCreate $stringContent")
11 }
```

Copier

SQLite / Room

Android propose un système de base de donnée : SQLite, celui-ci étant un peu complexe à utiliser, plusieurs outils ont été mis en place pour gérer une base de donnée plus facilement (des ORM : Mapping Objet-Relationnel). L'ORM officiel sur Android est maintenant [ROOM](#).

Voyons un peu plus en détail la mise en œuvre de cette base de donnée. Nous allons étudier 2 méthodologies différentes (utilisant 2 outils différents) :

1. En utilisant la documentation officielle : [ROOM](#).
2. En utilisant des LivesTemplates (and_db_room_00_documentation) (En JAVA uniquement).

SQLite / Room

Aller plus loin

Les bases de données ne pouvant pas être appelées dans le thread graphique et pouvant évoluer dans le temps, il est recommandé d'utiliser les LiveData pour gérer l'affichage des données. Nous pouvons suivre le tutoriel suivant : [Android Room with a View - Kotlin](#).

Les Content Provider

Définition

Le content provider permet de partager avec d'autres applications des données, de manière standardisée.
[Plus de détails, ici.](#)

Les Content Provider

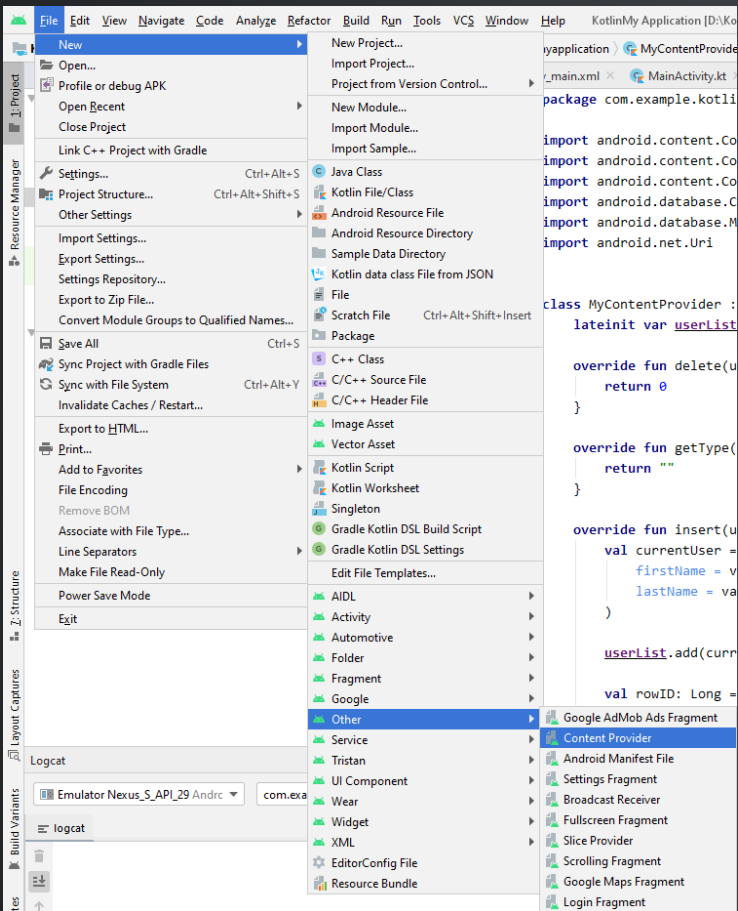
Utilisation d'un content provider

Nous allons utiliser un content provider, par exemple le MediaStore. Créons un nouveau projet [ContentProviderClient](#), et listons les sons disponibles sur notre téléphone :

```
1 Uri media = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
2 String[] projection = { MediaStore.Audio.Media._ID,           // 0
3   MediaStore.Audio.Media.ARTIST,           // 1
4   MediaStore.Audio.Media.TITLE,           // 2
5   MediaStore.Audio.Media.ALBUM_ID,        // 3
6   MediaStore.Audio.Media.ALBUM,          // 4
7   MediaStore.Audio.Media.DATA,           // 5
8   MediaStore.Audio.Media.DISPLAY_NAME,    // 6
9   MediaStore.Audio.Media.DURATION };      // 7
10
11 String selection = MediaStore.Audio.Media.IS_MUSIC + " != 0";
12
13 Cursor cursor = getContentResolver().query(media,projection,selection,null,null);
14 while(cursor.moveToNext()){
15     if(BuildConfig.DEBUG){
16         Log.d(TAG, "onCreate " +
17             cursor.getString(0) + " " +
```

[Copier](#)

Les Content Provider



Les Content Provider

New Android Component

 Configure Component

Creates a new content provider component and adds it to your Android manifest.

Class Name:

URI Authorities:

☒ Exported

☒ Enabled

Source Language:

A semicolon separated list of one or more URI authorities that identify data under the purview of the content provider.

Previous

Next

Cancel

Finish

Les Content Provider

POJO : La classe User

Déclarons une nouvelle classe qui va contenir nos données, la classe User :

```
1 data class User(var firstName: String, var lastName: String)
```

[Copier](#)

Les Content Provider

Query

Pour commencer, nous allons simplement retourner une liste en dur via le code :

```
1  override fun query(  
2      uri: Uri, projection: Array<String>?, selection: String?,  
3      selectionArgs: Array<String>?, sortOrder: String?  
4  ): Cursor? {  
5  
6      val cursor = MatrixCursor(arrayOf("name", "firstname"))  
7  
8      cursor.newRow()  
9          .add("name", "SALAUN")  
10         .add("firstname", "Tristan")  
11  
12      cursor.newRow()  
13          .add("name", "SNOW")  
14          .add("firstname", "John")  
15  
16      return cursor  
17  }
```

[Copier](#)

ArrayList

Les Content Provider

Afin d'avoir une certaine persistance et aller plus rapidement, dans l'implémentation du ContentProvider, nous allons utiliser une ArrayList.

Déclarons une variable de classe :

```
1 lateinit var userList: ArrayList<User>
```

[Copier](#)

Nous allons initialiser la liste dans la méthode onCreate :

```
1 override fun onCreate(): Boolean {  
2     // Init some values at the create  
3     userList = arrayListOf(User("Tristan", "SALAUN"))  
4     return true  
5 }
```

[Copier](#)

Notre méthode query va donc prendre la forme :

```
1 override fun query(  
2     uri: Uri, projection: Array<String>?, selection: String?,  
3     selectionArgs: Array<String>?, sortOrder: String?  
4 ): Cursor? {  
5  
6     val cursor = MatrixCursor(arrayOf("name", "firstname"))  
7  
8     for (curentUser in this.userList) {  
9         cursor.addRow()  
10         .add("name", curentUser.lastName)  
11         .add("firstname", curentUser.firstName)  
12     }  
13  
14     // cursor.addRow()  
15     // .add("name", "SALAUN")  
16     // .add("firstname", "Tristan")
```

[Copier](#)

Les Content Provider

La méthode insert prendra quant à elle la forme :

```
1 override fun insert(uri: Uri, values: ContentValues?): Uri? {  
2     val currentUser = User(  
3         firstName = values?.getString(UserContract.MyDatas.KEY_COL_FIRSTNAME),  
4         lastName = values?.getString(UserContract.MyDatas.KEY_COL_NAME)  
5     )  
6  
7     userList.add(currentUser)  
8  
9     val rowID: Long = userList.size.toLong()  
10  
11     val _uri =  
12     ContentUris.withAppendedId(UserContract.MyDatas.CONTENT_URI, rowID)  
13     context!!.contentResolver.notifyChange(_uri, null)  
14     return _uri  
15 }
```

Copier

Les Content Provider

Partager notre Content Provider

L'objectif étant que notre Content provider soit utilisable par d'autres applications, il est d'usage de définir une classe Contrat, dans notre cas la classe `UserContract` :

```
1 import android.net.Uri
2 import android.provider.BaseColumns
3
4 class UserContract {
5
6     companion object {
7         // The Authority
8         private const val AUTHORITY = "com.exemple.contentprovider.provider"
9
10        // The path to the data... and explain
11        private const val PATH_TO_DATA = "users" //Vous pouvez déclarer plusieurs paths (les paths utilisent les /)
12    }
13
14    interface MyDatas : BaseColumns {
15        companion object {
16
17            // The URI and explain, with example if you want
```

Copier

Les Content Provider

Création de ContentProviderClient

C'est la classe UserContract que nous utilisons aussi dans le projet client de notre ContentProvider : ContentProviderClient.

Créons le projet ContentProviderClient.

Les Content Provider

Lecture dans ContentProviderClient

Ouvrons le projet ContentProviderClient, et ajoutons le code pour interroger notre nouveau ContentProvider :

```
1 Cursor cursor = getContentResolver().query(Uri.parse("content://com.exemple.contentprovider.provider"), null, null, null, null);
2 if(cursor.moveToFirst()) {
3     StringBuilder strBuild=new StringBuilder();
4     while (!cursor.isAfterLast()) {
5         strBuild.append("\n"+cursor.getString(0)+ "-" + cursor.getString(1));
6         cursor.moveToNext();
7     }
8     if(BuildConfig.DEBUG){
9         Log.d(TAG, "onCreate " + strBuild);
10    }
11 }
```

[Copier](#)

Les Content Provider

Insertion dans ContentProviderClient

Ajoutons maintenant un appel pour insérer une donnée dans le ContentProvider :

```
1 // Defines an object to contain the new values to insert
2 val newValues = ContentValues().apply {
3     /*
4      * Sets the values of each column and inserts the data. The arguments to the "put"
5      * method are "column name" and "value".
6      */
7     put(UserContract.MyDatas.KEY_COL_FIRSTNAME, "John")
8     put(UserContract.MyDatas.KEY_COL_NAME, "Doe")
9 }
10
11 val newUri = contentResolver.insert(
12     UserContract.MyDatas.CONTENT_URI, // The UserDictionary content URI
13     newValues                          // The values to insert
14 )
```

[Copier](#)

Les Content Provider

Pour aller un peu plus loin : [Content provider basics](#).

La gestion réseau

- Les infos de connectivité. Utiliser HTTP.
- Parser du JSON.
- Les accès aux Web Services : Volley, Retrofit.

Connectivité et HTTP

Commençons par mettre en place une classe utilitaire : NetworkTool :

```
1 import android.content.Context
2 import android.net.ConnectivityManager
3 import android.net.NetworkCapabilities
4
5 import android.net.NetworkInfo
6 import android.os.Build
7
8 class NetworkTool {
9
10     companion object {
11
12         // NEED : <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
13         private fun isNetworkAvailable(context: Context): Boolean {
14             val connectivityManager = context.getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
15             if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
16                 val nw = connectivityManager.activeNetwork ?: return false
17                 val actNw = connectivityManager.getNetworkCapabilities(nw) ?: return false
```

Copier

Connectivité et HTTP

Puis notre receiver : NetworkStateReceiver

```
1 import android.content.BroadcastReceiver
2 import android.content.Context
3 import android.content.Intent
4 import android.net.ConnectivityManager
5 import android.net.NetworkInfo
6 import android.util.Log
7 import java.lang.Boolean
8
9 class NetworkStateReceiver: BroadcastReceiver() {
10
11     companion object {
12         private const val TAG = "NetworkStateReceiver"
13     }
14
15     // post event if there is no Internet connection
16     override fun onReceive(context: Context?, intent: Intent) {
17         Log.d(TAG, "onReceive $intent")
18     }
19 }
```

Copier

Connectivité et HTTP

Il ne nous reste plus qu'à enregistrer le listener : nous commençons par déclarer l'attribut de classe :

```
1 private val networkStateReceiver = NetworkStateReceiver()
```

[Copier](#)

Puis dans la méthode onCreate on enregistre le listener :

```
1 registerReceiver(networkStateReceiver, IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION))
```

[Copier](#)

Et on le désenregistre dans le onStop :

```
1 unregisterReceiver(networkStateReceiver)
```

[Copier](#)

Pour tester, on peut couper la connexion WiFi par exemple, et la réactiver.

Parser du JSON

Nous allons utiliser la librairie [GSON](#).

Nous allons suivre les étapes suivantes :

- Ajouter une référence à la librairie dans nos dépendances.
- Générer du JSON à partir d'objets.
- Parser du JSON pour instancier des objets.

Cet exemple est largement inspiré de : [Android : Utiliser Gson pour faciliter l'utilisation du Json](#)

Parser du JSON

Ajout de la dépendance.

Ajoutons une dépendance dans notre fichier build.gradle (Module: app) :

```
1 dependencies {  
2     ...  
3     // JSon parser  
4     implementation 'com.google.code.gson:gson:2.10.1'  
5 }
```

Copier

Parser du JSON

Générer du JSON à partir d'un objet.

Créons l'objet Book :

```
1 data class Book(var id: String, var name: String, var author: String, var genre: String, var numpages: Int, var releaseDate: String, var cover: String)
```

Copier

Et remplissons une liste de Book :

```
1 var list = arrayListOf(  
2     Book(  
3         id = "01fsEF", name = "1984",  
4         author = "George Orwell",  
5         genre = "Fiction dystopique",  
6         numpages = 376,  
7         releaseDate = "1949",  
8         cover = "1984.png"  
9     ),  
10    Book(  
11        id = "3H1J0n",  
12        name = "Le Meilleur des mondes",  
13        author = "Aldous Huxley",  
14        genre = "Science-fiction",  
15        numpages = 285,  
16        releaseDate = "1932",  
17        cover = "meilleur-des-mondes.jpg"
```

Copier

Parser du JSON

Nous pouvons maintenant transformer notre liste d'objets en JSON.

```
1 val gson = Gson()
2 val listType = object : TypeToken<ArrayList<Book?>>>() {}.type
3 val jsonResult: String = gson.toJson(list, listType)
4 Log.d(TAG, "onCreate jsonResult = $jsonResult")
```

Copier

Parser du JSON

Bonus

Une fonction pour faire un formatage "pretty print" :

```
1 private fun jsonToPrettyFormat(jsonString: String?): String? {  
2     val json = Gson().fromJson(jsonString, JsonElement::class.java)  
3     val gson = GsonBuilder()  
4         .serializeNulls()  
5         .disableHtmlEscaping()  
6         .setPrettyPrinting()  
7         .create()  
8     return gson.toJson(json)  
9 }
```

Copier

Que l'on appellera par exemple :

```
1 Log.d(TAG, "onCreate jsonResult = ${jsonToPrettyFormat(jsonResult)}")
```

Copier

Parser du JSON

JSON vers objet

Nous allons maintenant réaliser l'opération inverse, consistant à passer du JSON (format texte) vers des objets :

```
1 // Deserialization
2 val bookList2: ArrayList<Book> = Gson().fromJson(jsonResult, listType)
3 Log.d(TAG, "onCreate bookList2 = $bookList2")
```

[Copier](#)

Parser du JSON

Autres librairies

Nous avons vu l'usage de GSON, mais il existe d'autres librairies (liste non exhaustive) :

- [Jackson](#)
- [klaxon](#)
- [Moshi](#)
- [kotlinx.serialization](#)

[Un article comparant les solutions \(2019\)](#)

[Un article comparant les solutions \(2020\)](#)

[Un article comparant les solutions \(2022\)](#)

Les accès aux Web Services

Il existe plusieurs façons de faire des appels à des web services, par exemple :

- [URLConnection](#) (sans librairie).
- [Volley](#) (sur GitHub).
- [Android Asynchronous Http Client](#).
- [Retrofit](#).

La solution la plus populaire, semble être Retrofit, c'est celle-là que nous allons utiliser. Il est à noter que [Ktor](#) gagne en popularité.

Les accès aux Web Services

Application de news

Nous allons écrire une application nous permettant d'afficher des nouvelles, en provenance d'un site proposant une API gratuitement (pour les développeurs). Les étapes que nous allons suivre sont :

- Création du compte sur le site.
- Ajout de la librairie à notre projet.
- Écriture des fichiers POJO.
- Écriture de l'interface avec l'API.
- Instantiation du client.
- Lancement d'une requête.

La source du sujet est inspiré de : [Live data, ViewModel, Retrofit Android Architecture Component](#).

Les accès aux Web Services

Points techniques

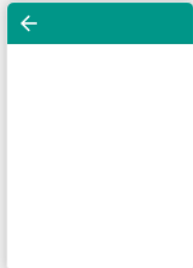
Dans cette application, nous verrons les points suivants :

- LiveData.
- La librairie [GSON](#).
- La librairie [Retrofit](#).
- Le service en ligne [JSON => POJO](#).
- Le service de news par API : [News API](#).

Les accès aux Web Services

 Create New Project

 Configure Your Project



Empty Activity

Creates a new empty activity

Name

Package name

Save location

Language

Minimum SDK

 Your app will run on approximately 99.8% of devices.
[Help me choose](#)

☐ Use legacy android.support libraries 

Previous

Next

Cancel

Finish

Les accès aux Web Services

Ajout des librairies à notre projet

```
1 // JSon parsing
2 implementation 'com.google.code.gson:gson:2.10.1'
3
4 // Network calls
5 implementation 'com.squareup.retrofit2:retrofit:2.8.1'
6 implementation 'com.squareup.retrofit2:converter-gson:2.8.1'
7
8 // handle recyclerview
9 implementation "androidx.recyclerview:recyclerview:1.3.2"
10
11 // handle livedata life cycle in fragments
12 implementation "androidx.fragment:fragment-ktx:1.8.0"
13
14 // Handle images loading (choose one)
15 // implementation 'com.squareup.picasso:picasso:2.71828'
16 implementation 'com.github.bumptech.glide:glide:4.15.1'
```

Copier

Les accès aux Web Services

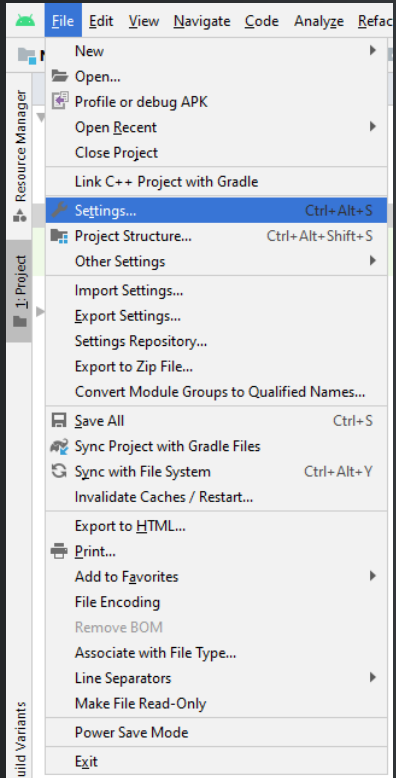
Écriture des fichiers POJO

Pour écrire les fichiers qui correspondent à notre API de news, nous allons suivre les étapes suivantes :

- Nous allons installer le plugin JSON To Kotlin.
- Nous allons utiliser le JSON renvoyé par le site web.
- Nous allons le copier/coller dans le plugin JSON To Kotlin.
- Renseigner le champ Class Name: avec : NewsResponse.
- Il ne reste qu'à cliquer sur Generate.

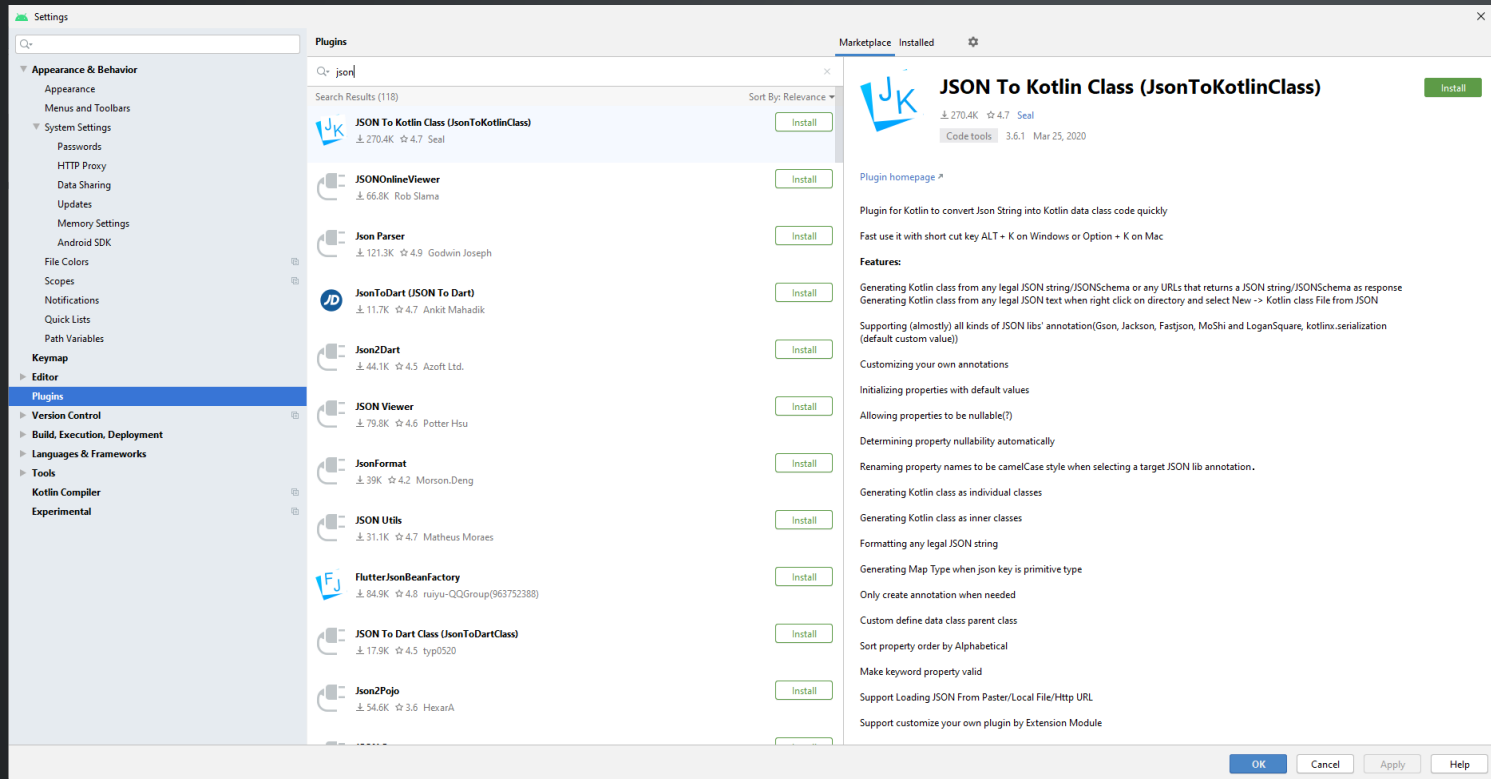
Les accès aux Web Services

Installons le plugin JSON to Kotlin



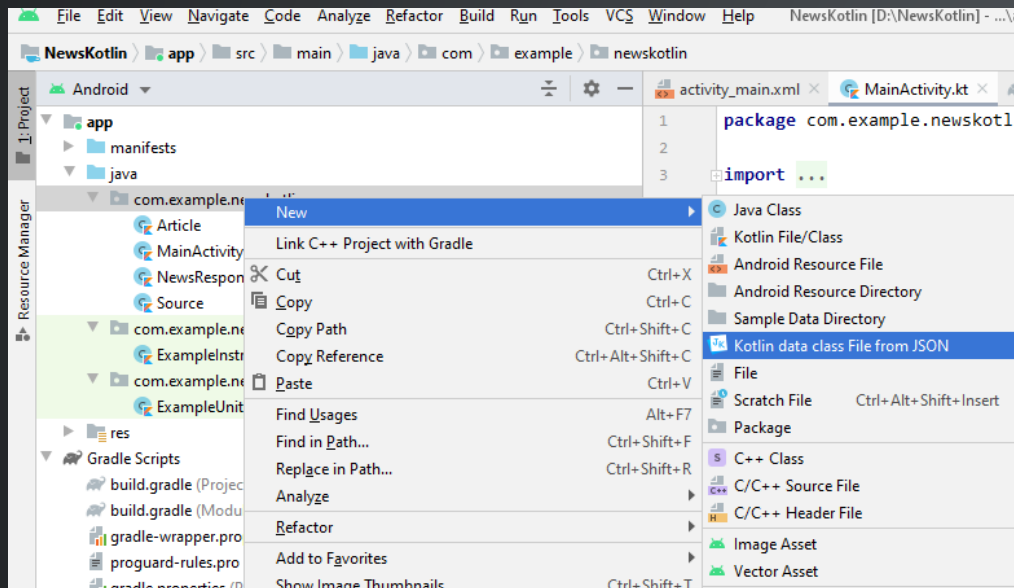
Les accès aux Web Services

Installons le plugin JSON to Kotlin



Les accès aux Web Services

Utilisons JSON to Kotlin



Les accès aux Web Services

Création du compte sur le site de news

Afin d'accéder à l'API de News, nous allons suivre les étapes suivantes :

- Créer un compte (gratuit) sur le site **News API** : <https://newsapi.org/>.
- Aller dans la section : "News sources" (tout en bas), puis France.
- Copier l'URL de l'API (par exemple : `http://newsapi.org/v2/top-headlines?country=fr&apiKey=API_KEY`)
- Récupérer le contenu du message (click droit, code source de la page => view-source:`http://newsapi.org/v2/top-headlines?country=fr&apiKey=API_KEY` (sur chrome).
- Générer les POJO (détail page suivante).

Les accès aux Web Services

 Generate Kotlin Data Class Code

 Please input the JSON String and class name to generate Kotlin data class

JSON Text: Tips: you can use JSON string , http urls or local file just right click on text area Format

```
1 {
2   "status": "ok",
3   "totalResults": 34,
4   "articles": [
5     {
6       "source": {
7         "id": null,
8         "name": "20minutes.fr"
9       },
10      "author": "20 Minutes avec AFP",
11      "title": "Policiers percutés à Colombes : L'enquête antiterroriste",
12      "description": "Le suspect devrait être mis en examen",
13      "url": "https://www.20minutes.fr/societe/2771383-20200501-policiers",
14      "urlToImage": "https://img.20mn.fr/MPsoxAvnR8CfZpFjY1Kivg/648x360_c",
15      "publishedAt": ,
16      "content": "Deux motards de la police ont été grièvement blessés. I
17    },
18    {
19      "source": {
20        "id": null,
21        "name": "Jeuxvideo.com"
22      },
23      "author": "daFrans",
```

Class Name:

Advanced Like this version? Please star here: <https://github.com/wuseal/JsonToKotlinClass>

Generate Cancel

Les accès aux Web Services

Paramétrage avancé

Il est possible de configurer le générateur avec le bouton "Advanced". Nous allons par exemple activer l'Annotation Gson.

Nous avons donc maintenant 3 nouvelles classes :

- NewsResponse.
- Article.
- Source.

Vérifions que les types des attributs sont bien des `String`.

Les accès aux Web Services

Écriture de l'interface avec l'API

Créons une nouvelle interface : NewsApi contenant le code suivant :

```
1 import retrofit2.Call
2 import retrofit2.http.GET
3 import retrofit2.http.Query
4
5 interface NewsApi {
6     @GET("top-headlines")
7     fun getNewsList(
8         @Query("country") newsSource: String?,
9         @Query("apiKey") apiKey: String?
10    ): Call<NewsResponse?>?
11 }
```

Copier

Les accès aux Web Services

Creation de la classe de service

```
1 import retrofit2.Retrofit
2 import retrofit2.converter.gson.GsonConverterFactory
3
4 class RetrofitService {
5     private val retrofit = Retrofit.Builder()
6         .baseUrl("https://newsapi.org/v2/")
7         .addConverterFactory(GsonConverterFactory.create())
8         .build()
9
10    fun <S> createService(serviceClass: Class<S>): S {
11        return retrofit.create(serviceClass)
12    }
13 }
```

Copier

Les accès aux Web Services

NewsRepository

```
1 import android.util.Log
2 import androidx.lifecycle.MutableLiveData
3 import retrofit2.Call
4 import retrofit2.Callback
5 import retrofit2.Response
6
7 object NewsRepository {
8     private const val TAG = "NewsRepository"
9
10    private var newsApi: NewsApi? = null
11
12    init {
13        newsApi = RetrofitService().createService(NewsApi::class.java)
14    }
15
16    fun getNews(country: String?, key: String?): MutableLiveData<NewsResponse?> {
17        val newsData = MutableLiveData<NewsResponse?>()
```

Copier

Les accès aux Web Services

NewsViewModel

```
1 import androidx.lifecycle.LiveData
2 import androidx.lifecycle.MutableLiveData
3 import androidx.lifecycle.ViewModel
4
5 class NewsViewModel : ViewModel() {
6     private var mutableLiveData: MutableLiveData<NewsResponse?>? = null
7     fun init() {
8         if (mutableLiveData != null) {
9             return
10        }
11        mutableLiveData = NewsRepository.getNews("fr", "96c6792fdad6434fbc3a893daba40e0f")
12    }
13
14    fun getNewsRepository(): LiveData<NewsResponse?>? {
15        return mutableLiveData
16    }
17 }
```

[Copier](#)

Les accès aux Web Services

Ajout d'un fragment

Nous allons ajouter un fragment NewsFragment à notre projet, et dans le layout du fragment, ajouter une RecyclerView :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     tools:context=".NewsFragment">
7     <androidx.recyclerview.widget.RecyclerView
8         android:id="@+id/rvNews"
9         android:layout_width="match_parent"
10        android:layout_height="match_parent" />
11 </FrameLayout>
```

Copier

Les accès aux Web Services

Modification du layout

Nous allons ajouter ce fragment au layout de notre `activity_main.xml` :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     tools:context=".MainActivity">
7     <fragment android:id="@+id/news_fragment"
8         android:name="com.example.todeletenetwork.NewsFragment"
9         android:layout_width="match_parent"
10        android:layout_height="match_parent" />
11 </FrameLayout>
```

Copier

Les accès aux Web Services

Création du layout pour afficher une news

Nous allons par exemple écrire un layout : `item_news.xml` :

[illegible]

Copier

Les accès aux Web Services

Création de l'adapter : NewsAdapter

```
1 import android.content.Context
2 import android.view.LayoutInflater
3 import android.view.View
4 import android.view.ViewGroup
5 import android.widget.ImageView
6 import android.widget.TextView
7 import androidx.recyclerview.widget.RecyclerView
8 import androidx.recyclerview.widget.RecyclerView.ViewHolder
9 import com.bumptech.glide.Glide
10
11 class NewsAdapter(val context: Context, val myDataset: List<Article>):
12     RecyclerView.Adapter<NewsAdapter.MyViewHolder>() {
13         private var mDataset: List<Article>? = null
14         private var mInflater: LayoutInflater? = null
15
16         init {
17             mInflater = LayoutInflater.from(context)
```

Copier

Les accès aux Web Services

Appel dans notre fragment

Déclarons les variables de classes suivantes :

```
1 private var newsAdapter: NewsAdapter? = null
2 private val newsViewModel: NewsViewModel by viewModels()
3 private var rvHeadline: RecyclerView? = null
4 private var articleArrayList = arrayListOf<Article>()
```

Copier

Les accès aux Web Services

Appel dans notre fragment

Déclarons la fonction suivante :

```
1 private fun setupRecyclerView(context: Context) {  
2     if (newsAdapter == null) {  
3         newsAdapter = NewsAdapter(context, articleArrayList)  
4         rvHeadline?.layoutManager = LinearLayoutManager(context)  
5         rvHeadline?.adapter = newsAdapter  
6         //rvHeadline?.setItemAnimator(DefaultItemAnimator())  
7         //rvHeadline?.setNestedScrollingEnabled(true)  
8     } else {  
9         newsAdapter?.notifyDataSetChanged()  
10    }  
11 }
```

Copier

Les accès aux Web Services

Appel dans notre fragment

Remplaçons le corps de notre méthode onCreateView :

```
1 // Inflate the layout for this fragment
2 val rootLayout = inflater.inflate(R.layout.fragment_news, container, false)
3
4 context?.let { safeContext ->
5     rvHeadline = rootLayout.findViewById(R.id.rvNews)
6     newsViewModel.init()
7     newsViewModel.getNewsRepository()?.observe(viewLifecycleOwner) {
8         val newsArticles = it?.articles
9         if (newsArticles != null) {
10             articleArrayList.addAll(newsArticles)
11             newsAdapter?.notifyDataSetChanged()
12         }
13     }
14     setupRecyclerView(safeContext)
15 }
16 return rootLayout
```

Copier

Les accès aux Web Services

Erreurs possibles

Si vous avez l'erreur :

```
1 java.lang.NoSuchMethodError: No static method metafactory(Ljava/lang/invoke/MethodHandles
```

Copier

Il faut activer la compatibilité JVM 1.8 en ajoutant dans le build.gradle, le code suivant :

```
1  android {  
2  ...  
3      compileOptions {  
4          sourceCompatibility JavaVersion.VERSION_1_8  
5          targetCompatibility JavaVersion.VERSION_1_8  
6      }  
7      kotlinOptions {  
8          jvmTarget = "1.8"  
9      }  
10 }
```

Copier

Les accès aux Web Services

Erreurs possibles

Si vous avez l'erreur, sur l'émulateur :

```
1 java.net.SocketException: socket failed: EPERM (Operation not permitted)
```

Copier

Il suffit que nous désinstallions l'application, pour la réinstaller.

Les accès aux Web Services

Erreurs possibles

Si nous avons une `SocketTimeoutException`, il suffit d'allonger la durée du Timeout.
Dans la classe : `RetrofitService` remplaçons

```
1 private val retrofit = Retrofit.Builder()
2   .baseUrl("https://newsapi.org/v2/")
3   .addConverterFactory(GsonConverterFactory.create())
4   .build()
```

[Copier](#)

Par :

```
1 var client = OkHttpClient.Builder()
2   .connectTimeout(100, TimeUnit.SECONDS)
3   .readTimeout(100, TimeUnit.SECONDS).build()
4 private val retrofit = Retrofit.Builder()
5   .baseUrl("https://newsapi.org/v2/").client(client)
6   .addConverterFactory(GsonConverterFactory.create(Gson()))
7   .build()
```

[Copier](#)

Cela nous permet d'attendre un peu plus de temps avant d'avoir une erreur. Il faut dans ce cas-là, afficher une indication à l'utilisateur pour qu'il comprenne qu'il doit attendre.

Les accès aux Web Services

Erreurs possibles

Si nous avons l'erreur suivante :

```
1 Caused by: java.lang.IllegalArgumentException: Binary XML file line #11: Must specify unique android:id, android:tag, or have a parent with an id for  
com.example.todeletenetwork.NewsFragment
```

Copier

Il suffit de préciser un id au tag fragment dans le layout de notre activity.

Compléments

- Les capteurs. Les API Google de localisation.

Utilisation des capteurs

Mise en œuvre de capteurs.

Nous allons développer une application qui va éteindre l'écran quand on est proche de l'écran, et le rallumer quand on est à distance. Pour cela plusieurs étapes :

- Nous allons lister les capteurs.
- Récupérer le capteur de proximité.
- S'enregistrer sur les changements.
- Réagir aux changements.

Utilisation des capteurs

Liste des capteurs (Java).

Pour lister tous les capteurs disponibles sur notre smartphone, utilisons le code suivant :

```
1 SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
2 List<Sensor> liste = sensorManager.getSensorList(Sensor.TYPE_ALL);
3 if (BuildConfig.DEBUG) {
4     for (Sensor currentSensor : liste) {
5         Log.d(TAG, "onCreate sensor " + currentSensor);
6     }
7 }
```

[Copier](#)

Utilisation des capteurs

Liste des capteurs (Kotlin).

Pour lister tous les capteurs disponibles sur notre smartphone, utilisons le code suivant :

```
1 val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
2 var liste = sensorManager.getSensorList(Sensor.TYPE_ALL)
3 if (BuildConfig.DEBUG) {
4     for (currentSensor in liste) {
5         Log.d(TAG, "onCreate sensor $currentSensor")
6     }
7 }
```

[Copier](#)

Utilisation des capteurs

Récupérons le capteur de proximité (Java).

```
1 private Sensor mProximitySensor = null; // En attribut de classe.  
2 mProximitySensor = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
```

[Copier](#)

Utilisation des capteurs

Récupérons le capteur de proximité (Kotlin)

```
1 var mProximitySensor:Sensor? = null // En attribut de classe.  
2 mProximitySensor = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

[Copier](#)

Utilisation des capteurs

Enregistrons-nous, sur les changements (Java).

Dans un premier temps, nous récupérons le manager des capteurs :

```
1 private SensorManager mSensorManager = null; // En attribut de classe.  
2 mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

[Copier](#)

Puis nous nous enregistrons sur les changements uniquement quand l'application est active :

```
1 @Override  
2 protected void onResume() {  
3     super.onResume();  
4     mSensorManager.registerListener(mSensorEventListener, mProximitySensor, SensorManager.SENSOR_DELAY_NORMAL);  
5 }  
6  
7 @Override  
8 protected void onPause() {  
9     super.onPause();  
10    mSensorManager.unregisterListener(mSensorEventListener, mProximitySensor);  
11 }
```

[Copier](#)

Utilisation des capteurs

Le listener (Java).

Le listener qui va réagir aux changements est définis par :

```
1 final SensorEventListener mSensorEventListener = new SensorEventListener() {  
2     public void onAccuracyChanged(Sensor sensor, int accuracy) {  
3         // Que faire en cas de changement de précision ?  
4     }  
5  
6     public void onSensorChanged(SensorEvent sensorEvent) {  
7         if(BuildConfig.DEBUG){  
8             Log.d(TAG, "onSensorChanged " + sensorEvent.values.length);  
9             Log.d(TAG, "onSensorChanged " + sensorEvent.values[0]);  
10        }  
11    }  
12 };
```

Copier

Il ne nous reste plus qu'à implémenter notre code fonctionnel.

Exemple plus poussé d'utilisation du capteur de proximité pour **éteindre l'écran**.

Utilisation des capteurs

Enregistrons-nous, sur les changements (Kotlin)

Dans un premier temps, nous récupérons le manager des capteurs :

```
1 var mSensorManager: SensorManager? = null // En attribut de classe.  
2 mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
```

[Copier](#)

Puis nous nous enregistrons sur les changements uniquement quand l'application est active :

```
1 override fun onResume() {  
2     super.onResume()  
3     mSensorManager?.registerListener(  
4         mSensorEventListener,  
5         mProximitySensor,  
6         SensorManager.SENSOR_DELAY_NORMAL  
7     )  
8 }  
9  
10 override fun onPause() {  
11     super.onPause()  
12     mSensorManager?.unregisterListener(mSensorEventListener, mProximitySensor)  
13 }
```

[Copier](#)

Utilisation des capteurs

Le listener (Kotlin)

Le listener qui va réagir aux changements est définis par :

```
1 val mSensorEventListener: SensorEventListener = object : SensorEventListener {  
2     override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {  
3         // Que faire en cas de changement de précision ?  
4     }  
5  
6     override fun onSensorChanged(sensorEvent: SensorEvent) {  
7         if (BuildConfig.DEBUG) {  
8             Log.d(TAG, "onSensorChanged ${sensorEvent.values.size}")  
9             Log.d(TAG, "onSensorChanged ${sensorEvent.values[0]}")  
10        }  
11    }  
12 }
```

Copier

Il ne nous reste plus qu'à implémenter notre code fonctionnel.

Exemple plus poussé d'utilisation du capteur de proximité pour [éteindre l'écran](#).

Annexe

Mode développeur

Pour mettre son téléphone Android en mode développeur :

- Allons dans les paramètres.
- À propos du téléphone.
- Cliquons plusieurs fois sur le numéro de build.
- Une fois débloqué, nous pouvons activer le mode debugger, et la connexion USB.

Si au lieu de cliquer sur la version du build, nous cliquons sur la version d'Android, une surprise nous attend :-)

Annexe

Erreur classique

Une erreur : `Unsupported major.minor version 52.0` nous indique, que nous avons compilé le code avec une version de JDK 1.8 et que nous essayons de le lancer avec une version antérieure :

Références

Les références

- La référence : <https://kotlinlang.org/docs/reference>
- La source : <https://github.com/JetBrains/kotlin>
- Autre ressource très complète en FR : <https://riptutorial.com/Download/kotlin-fr.pdf>
- Rappels des bases : <https://kotlinlang.org/docs/reference/basic-syntax.html>

Références

Cours en ligne

- Sur Openclassrooms : <https://openclassrooms.com/fr/courses/5353106-initiez-vous-a-kotlin>
- Sur CodinGame : <https://www.codingame.com/playgrounds/28826/formation-kotlin/les-bases-de-kotlin>
- Kotlin Koans : <https://play.kotlinlang.org/koans/overview>

Références

Tests/Challenges

- <https://github.com/Kotlin/kotlin-koans-edu>
- <https://tech.io/explore>
- <https://github.com/SK1dev/KotlinChallenges/blob/master/README.md>
- <https://github.com/SK1dev/KotlinChallengesPart2>
- <https://github.com/igorwojda/kotlin-coding-puzzle>

Références

Articles sur des points précis

- <https://typealias.com/>
- <http://zetcode.com/all/>
- Pourquoi adopter Kotlin : <https://medium.com/videdressing-engineering/pourquoi-je-suis-pass%C3%A9-%C3%A0-kotlin-7d40d79054a4>

Références

Livres

- <https://kotlinlang.org/docs/books.html>

Références

Liste des mots clés

- <https://kotlinlang.org/docs/reference/keyword-reference.html>

Références

Kotlin pour les applications Backend

- <https://soat.developpez.com/tutoriels/kotlin-application-backend/>

Références

Kotlin pour Android

- <https://www.udacity.com/course/ud9012>
- <https://blog.ippon.fr/2017/12/11/introduction-a-kotlin-pour-android/>
- <https://kotlinlang.org/docs/reference/android-overview.html>
- <https://developer.android.com/kotlin>
- <https://codelabs.developers.google.com/codelabs/android-room-with-a-view-kotlin/#0>
- <https://codelabs.developers.google.com/codelabs/kotlin-coroutines/#0>