Android - Présentation générale

Pierre Nerzic

février-mars 2025

BY-NC-SA

Cette matière présente la programmation d'applications natives sur Android.

Cette semaine nous allons découvrir l'environnement de développement Android :

- Le SDK Android et Android Studio,
- Création d'une application simple,
- Communication avec une tablette.

Pierre Nerzic
BY-NC-SA

Introduction

Pierre Nerzic
BY-NC-SA

Programmation mobile ?

Actuellement, les applications sont :

- « **natives** », c'est à dire programmées en Java, C++, Kotlin, compilées et fournies avec leurs données sous la forme d'une archive Jar (fichier *APK*). C'est ce qu'on étudiera ici.
- « **progressive web apps** » (PWA), c'est une application pour navigateur internet, développée en HTML5, CSS3, JavaScript, dans un cadre logiciel (*framework*) tel que Node.js, Angular, React, Vue, Svelte... voir cours R6.A.05 Développement avancé l'an prochain.
- « **hybrides** », elles sont développées dans un framework comme Ionic, Flutter, React Native... Ces frameworks font abstraction des particularités du système : l'application est compilée sur différentes plateformes (Android, iOS, Windows, Linux...).

La charge d'apprentissage pour vous est la même.

BY-NC-SA

Programmation mobile ? (suite)

La programmation mobile tire profit du fait que le terminal d'affichage de l'application, un smartphone, peut être emmené partout, grâce à une connexion réseau omniprésente.

L'autre intérêt des smartphones est de posséder des capteurs : caméras, gyroscopes, boussole, GPS. Il y a quelques applications qui en tirent profit, par exemple le jeu Pokemon Go. Il fait partie d'une famille de logiciels de réalité augmentée (une couche d'images de synthèse superposée à une image de la réalité, en temps réel). Le problème est qu'il faut des connaissances en synthèse d'images temps réel, donc on ne peut pas présenter ça ici.

Le faible volume horaire disponible nous condamne à seulement étudier les bases d'une API de programmation mobile.

BY-NC-SA

Choix de l'API mobile

Le choix a été fait il y a longtemps de présenter Android dans ce cours, essentiellement parce que cette API utilisait le langage Java déjà connu des étudiants. D'autre part, elle est gratuite et bien documentée.

Pourquoi pas Flutter ? Cette API attire l'attention, mais :

- Les patrons MVC ou MVVM sont totalement ignorés par Flutter. Tout est aggluti né : interfaces, écouteurs, données.
- Le code source d'une grosse application est monstrueux.
- Des applications même petites prennent une place énorme dans la mémoire flash d'un smartphone.
- Flutter est inutile. Apprenez plutôt une API type PWA.
- Android Jetpack Compose qu'on étudiera ressemble beaucoup à Flutter, mais en mieux.

Si Flutter vous intéresse, apprenez-le par vous-même.

Pierre Nerzic
BY-NC-SA

Applications natives Android

Une application native Android est composée de :

- **Sources Kotlin** compilés pour une machine virtuelle appelée « *ART* » ($\neq$ `.class` Java)
- Fichiers appelés **ressources** :
  - format XML : interface (si appli « legacy »), textes...
  - format PNG : icônes, images...
- **Manifeste** = description du contenu du logiciel
  - version Android minimale nécessaire pour le smartphone,
  - fichiers présents dans l'archive avec leur signature,
  - autorisations nécessaires, durée de validité, etc.

Tout cet ensemble est géré à l'aide d'un IDE (environnement de développement) appelé *Android Studio* qui s'appuie sur un ensemble logiciel (bibliothèques, outils) appelé *SDK Android*.

BY-NC-SA

Kotlin

C'est un langage de programmation « symbiotique » de Java :

- une classe Kotlin est compilée dans le même code machine que Java,
- une classe Kotlin peut (en principe) utiliser les classes Java et réciproquement.
- On pouvait mélanger des sources Java et Kotlin dans une même application, mais Java ayant pris trop de retard, ce n'est plus possible.

Kotlin permet de développer des programmes plus sains qu'en Java. Par exemple, Kotlin oblige à vérifier chaque appel de méthode sur des variables objets pouvant valoir `null`, ce qui évite les `NullPointerException`.

BY-NC-SA

Exemple : objet pouvant être null

En Java, ça plante à l'exécution (ce n'est pas souhaitable) :

```
void afficherNomClient(Personne p) {
    System.out.println(p.getPrenom() + " " + p.getNom());
}

Personne p1 = null;
afficherNomClient(p1);          // NullPointerException
```

En Kotlin, le compilateur refuse le source équivalent :

```
fun afficherNomClient(p: Personne) { // p ne doit pas être null
    println(p.prenom + " " + p.nom)
}
var p1: Personne? = null             // p1 peut être null
afficherNomClient(p1)                // <-- refus de compiler
```


Pierre Nerzic
BY-NC-SA

Exemple : objet pouvant être null, suite

En Java amélioré avec des annotations :

```
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

void afficherNomClient(@NonNull Personne p) {
    System.out.println(p.getPrenom()+" "+p.getNom());
}

@Nullable Personne p1 = null;    // p1 peut être null
afficherNomClient(p1);           // <-- refus de compiler
```

En Java, il faut y penser, tandis que Kotlin vérifie systématiquement ça et de nombreuses autres choses (initialisations, etc.).

BY-NC-SA

Kotlin dans ce cours

Kotlin ne remplace pas une analyse sérieuse et une programmation rigoureuse. Kotlin permet seulement d'éviter de se faire piéger avec des bugs grossiers.

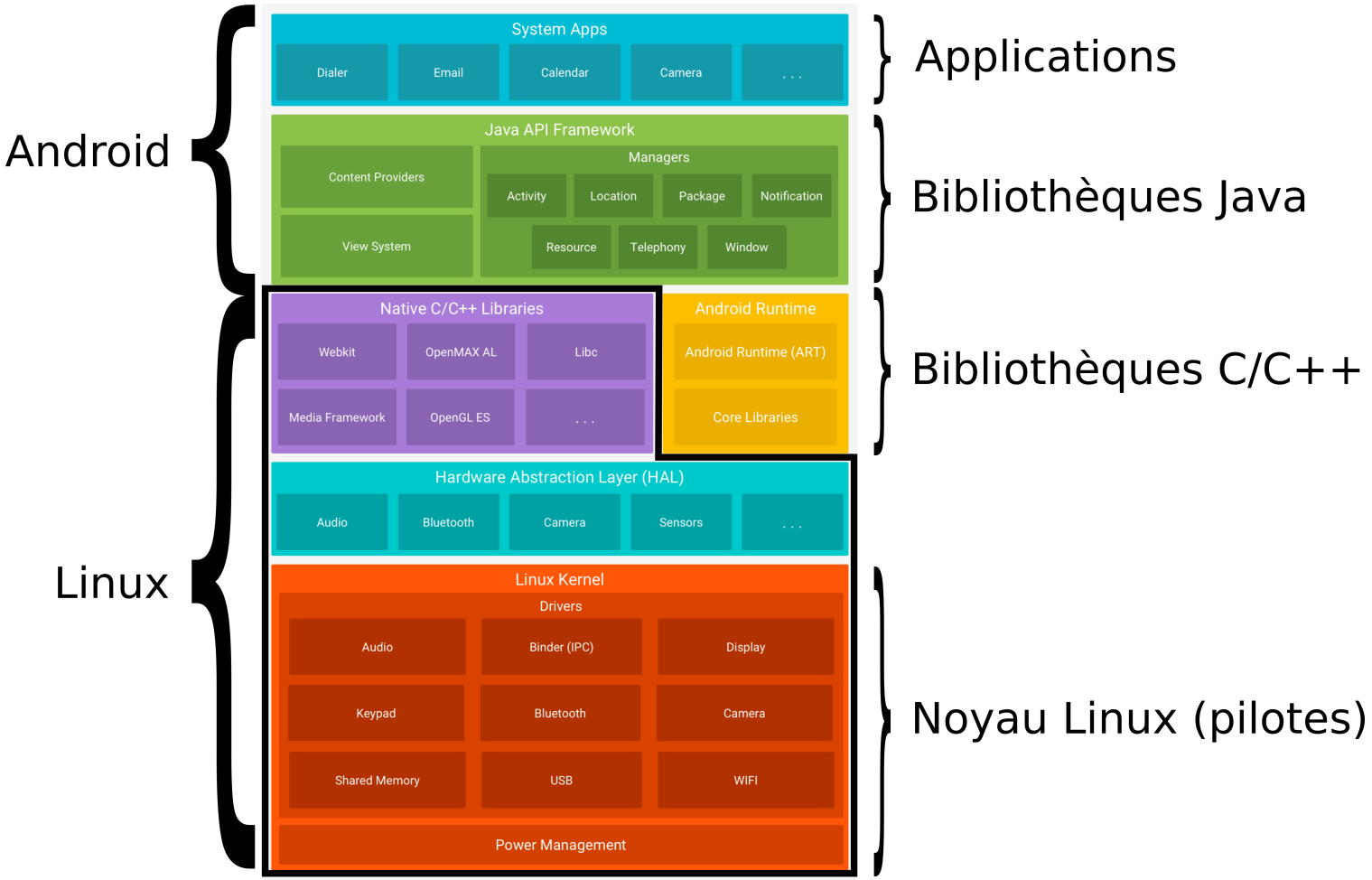
Il est relativement difficile d'apprendre Kotlin. Sa syntaxe est particulièrement abrégée, ex : définition implicite des variables membres à partir du constructeur, définition et appel implicites des setters/getters, liaison entre vues et variables membres d'une classe interface graphique, utilisation des *lambda*, etc. L'ensemble n'est pas toujours très lisible.

On apprendra les concepts peu à peu, sur des exemples.

BY-NC-SA

Quelques détails supplémentaires sur Android

Android est une sur-couche au dessus d'un système Linux :



Pierre Nerzic
BY-NC-SA

Historique

- Né en 2004, racheté par Google en 2005, version 1.5 publiée en 2007
- De nombreuses versions depuis. On en est à la version 16 (déc. 2024) et l'API 36. La version 16 est le numéro pour le grand public, et les versions d'API sont pour les développeurs. Exemples :
  - Android 7.0 Nougat = API 24,
  - Android 7.1 Nougat = API 25,
  - Android 13 Tiramisu = API 33
  - Android 15 Vanilla Ice Cream = API 35

Une API (*Application Programming Interface*) est un ensemble de bibliothèques de classes pour programmer des applications. Son numéro de version est lié à ses possibilités.

BY-NC-SA

Remarque sur les versions d'API

Chaque API apporte des fonctionnalités supplémentaires. Il y a compatibilité ascendante. Certaines fonctionnalités deviennent *dépréciées* au fil du temps, mais restent généralement disponibles.

On souhaite toujours programmer avec la dernière API (fonctions plus complètes et modernes), mais les utilisateurs ont souvent des smartphones plus anciens, qui n'ont pas cette API.

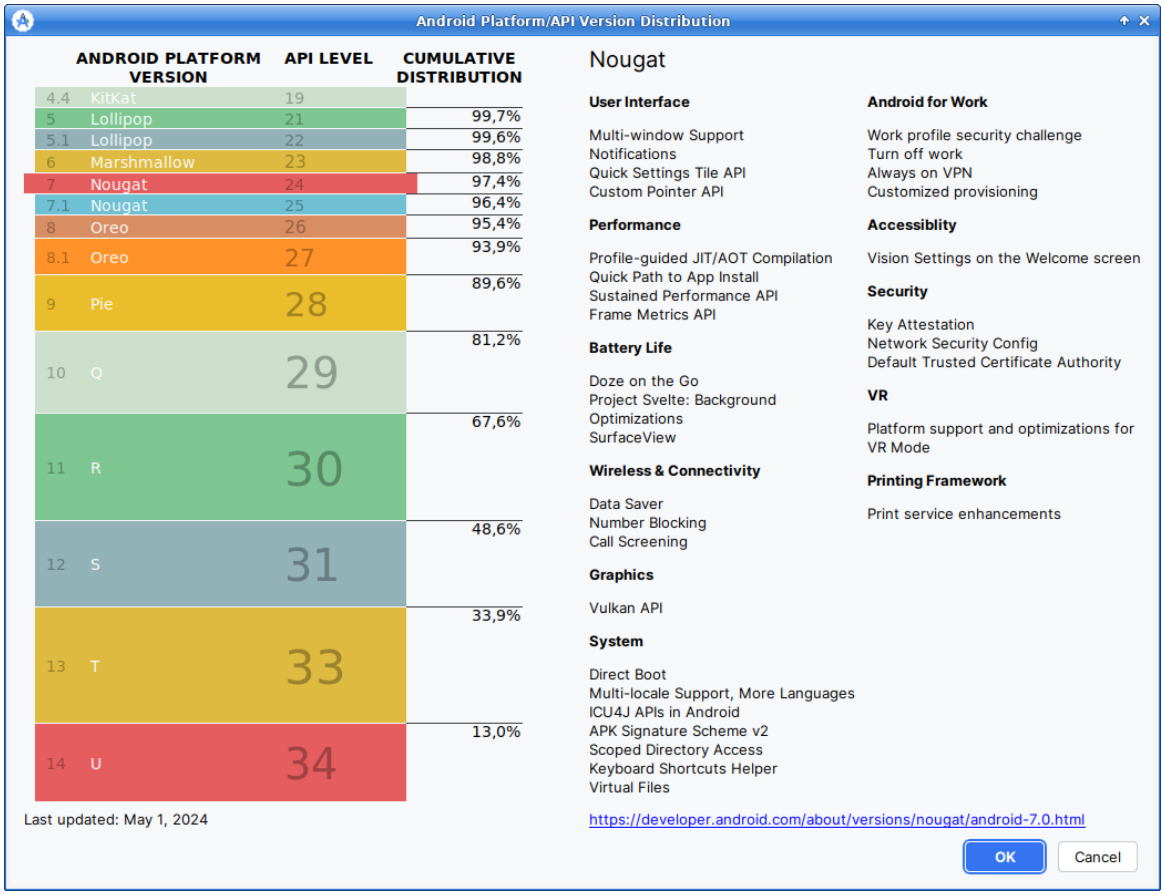
Or les constructeurs ne proposent **aucune** mise à jour majeure. Les smartphones restent toute leur vie avec l'API qu'ils ont à la naissance, sauf ceux avec Android One . La raison est purement commerciale.

Les développeurs doivent donc choisir une API qui correspond à la majorité des smartphones existant sur le marché.

BY-NC-SA

Distribution des versions

Voici la proportion des API en janvier 2025 :



Pierre Nerzic
BY-NC-SA

Remarques diverses

Évolution et obsolescence voulues et très rapides

- Suivre les modes et envies du marché, réaliser des profits
- Ce que vous allez apprendre sera rapidement dépassé (1 an)
  - syntaxiquement (méthodes, paramètres, classes, ressources...)
  - Exemple : Jetpack Compose dans Android...
  - mais pas les grands concepts (principes, organisation...) qu'on retrouve aussi sur iOS
- Vous êtes condamné(e) à une autoformation permanente, mais c'est comme ça dans ce métier.

BY-NC-SA

SDK Android et Android Studio

Pierre Nerzic
BY-NC-SA

SDK et Android Studio

Le *Software Development Kit* (SDK) contient :

- les librairies de classes et fonctions pour créer des logiciels
- les outils de fabrication des logiciels (compilateur. . .)
- *AVD* : un émulateur de tablettes pour tester les applications
- *ADB* : un outil de communication avec les vraies tablettes

Le logiciel Android Studio offre :

- un éditeur de sources et de ressources
- des outils de compilation : *gradle*
- des outils de test et de mise au point

BY-NC-SA

Android Studio

Pour commencer, il faut installer Android Studio selon la procédure expliquée sur cette page.

Pour le SDK, vous avez le choix, soit de l'installer automatiquement avec Studio, soit de faire une installation personnalisée. En général, vous pouvez choisir ce que vous voulez ajouter au SDK (version des librairies, versions des émulateurs de smarphones), à l'aide du *SDK Manager*.

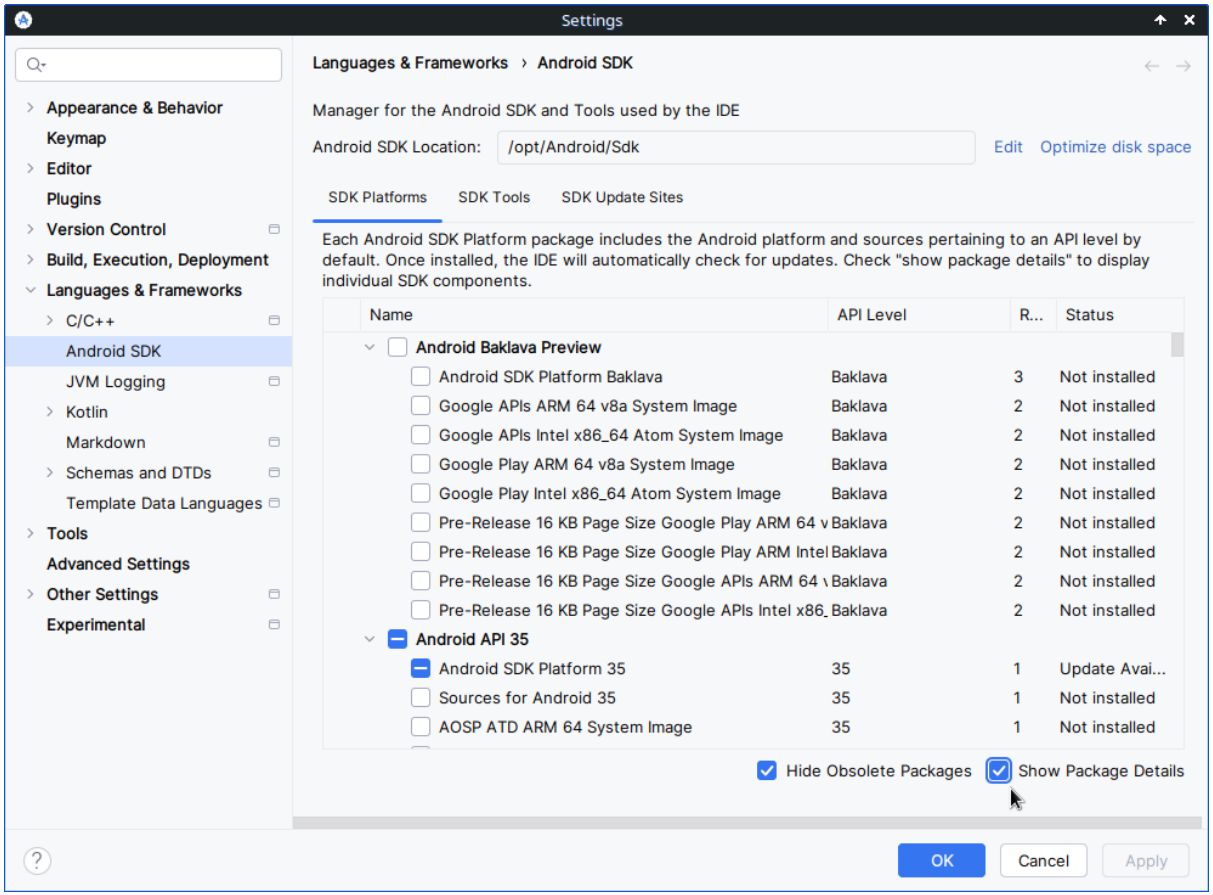
NB: dans la suite, certaines copies écran sont hors d'âge, mais je ne peux pas les refaire à chaque variante de Studio.

Tout est déjà installé à l'IUT et vous avez interdiction d'utiliser votre propre PC s'il n'a pas les capacités requises : 16Go de RAM, 25Go de SSD, écran 17" HD.

BY-NC-SA

SDK Manager

C'est le gestionnaire du SDK, une application qui permet de choisir les composants à installer et mettre à jour.



Pierre Nerzic
BY-NC-SA

Choix des éléments du SDK

Le gestionnaire permet de choisir les versions à installer, ex. :

- Android 15 (API 35)
- ...
- Android 7.0 (API 24)
- ...

Choisir celles qui correspondent aux tablettes qu'on vise, mais tout n'est pas à installer : il faut cocher `Show Package Details`, puis choisir élément par élément. Seuls ceux-là sont indispensables :

- `Android SDK Platform`
- `Intel x86 Atom_64 System Image`

Le reste est facultatif (Google APIs, sources, exemples et docs).

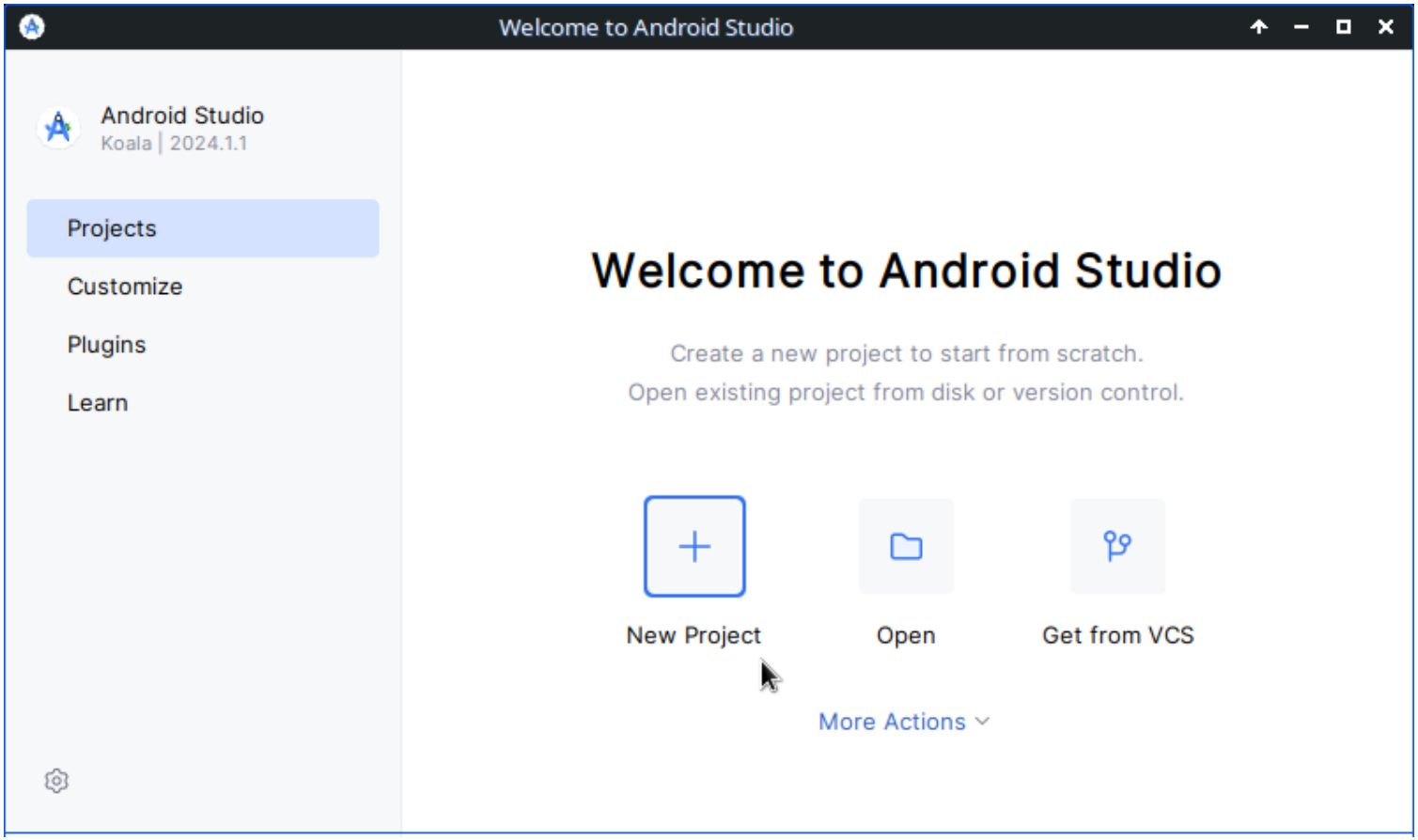
BY-NC-SA

Création d'une application

BY-NC-SA

Assistant de création d'application

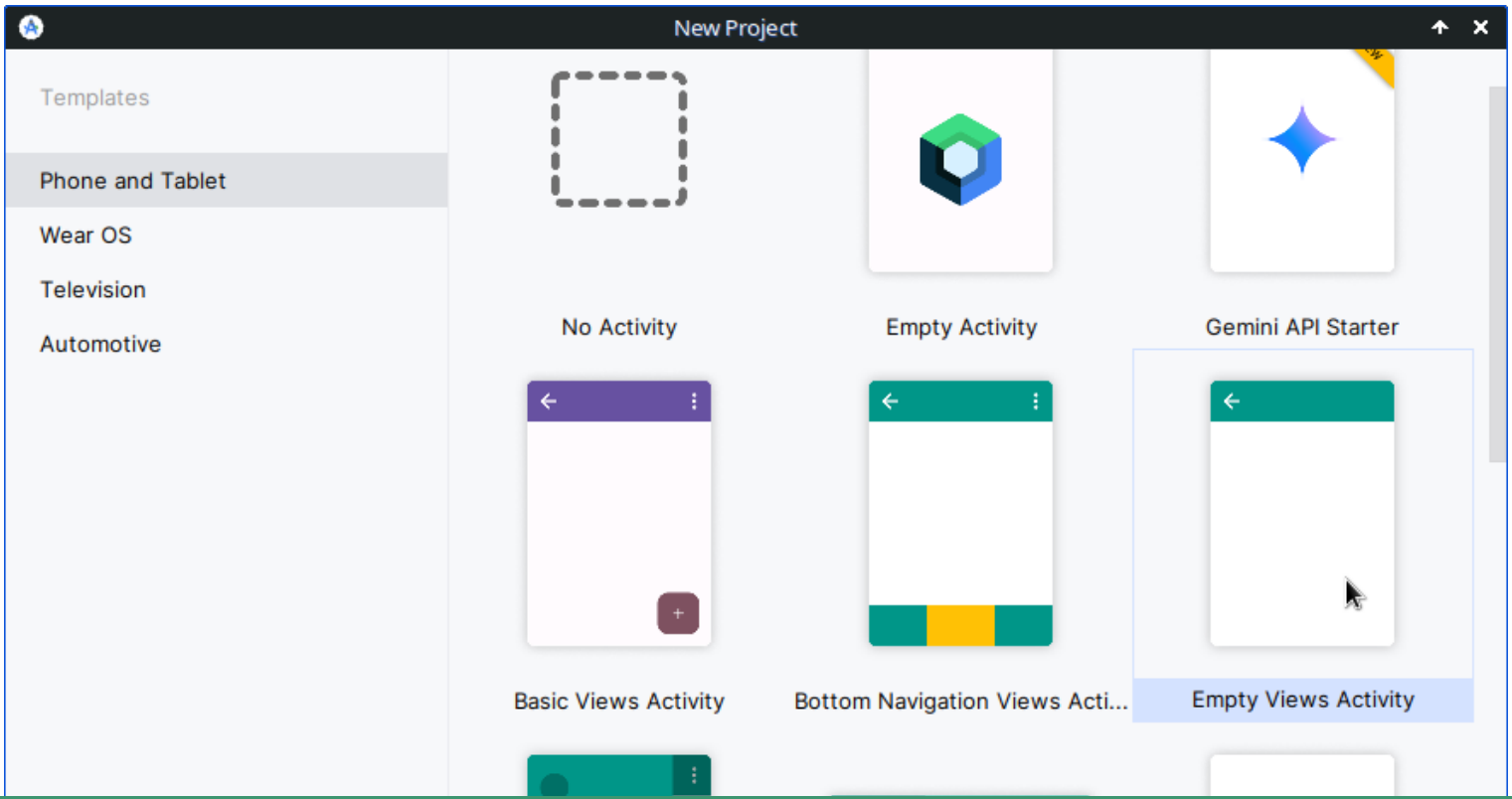
Android Studio contient un assistant de création d'applications :



Pierre Nerzic
BY-NC-SA

Modèle d'application

Android Studio propose plusieurs modèles de projet. En général, on part de celui appelé *Empty Activity* (Compose) ou *Empty Views Activity* (« legacy »).



Pierre Nerzic
BY-NC-SA

Résultat de l'assistant

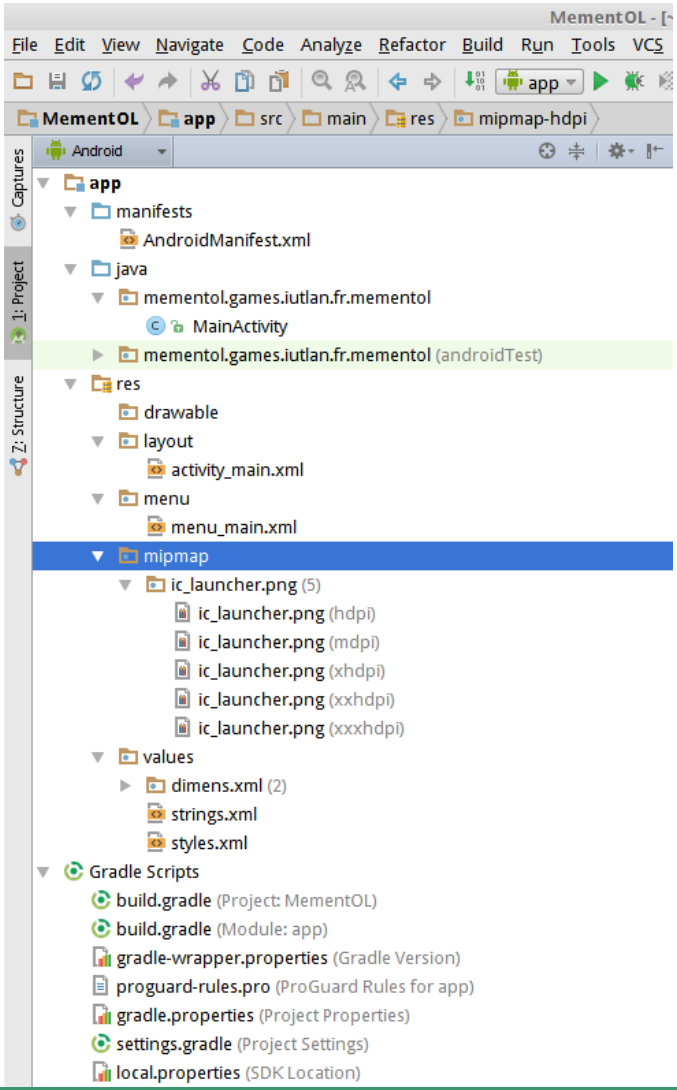
L'assistant crée de nombreux éléments :

- `manifests` : description et liste des classes de l'application
- `java` : les sources, rangés par paquetage,
- `res` : ressources = fichiers XML et images de l'interface, il y a des sous-dossiers :
  - `layout` : interfaces (disposition des vues sur les écrans)
  - `menu` : menus contextuels ou d'application
  - `mipmap` et `drawable` : images, icônes de l'interface
  - `values` : valeurs de configuration, textes...
- `Gradle scripts` : c'est l'outil de compilation du projet.

NB: ne pas chercher à tout comprendre dès le début.

BY-NC-SA

Fenêtre du projet



Pierre Nerzic
BY-NC-SA

Éditeurs spécifiques

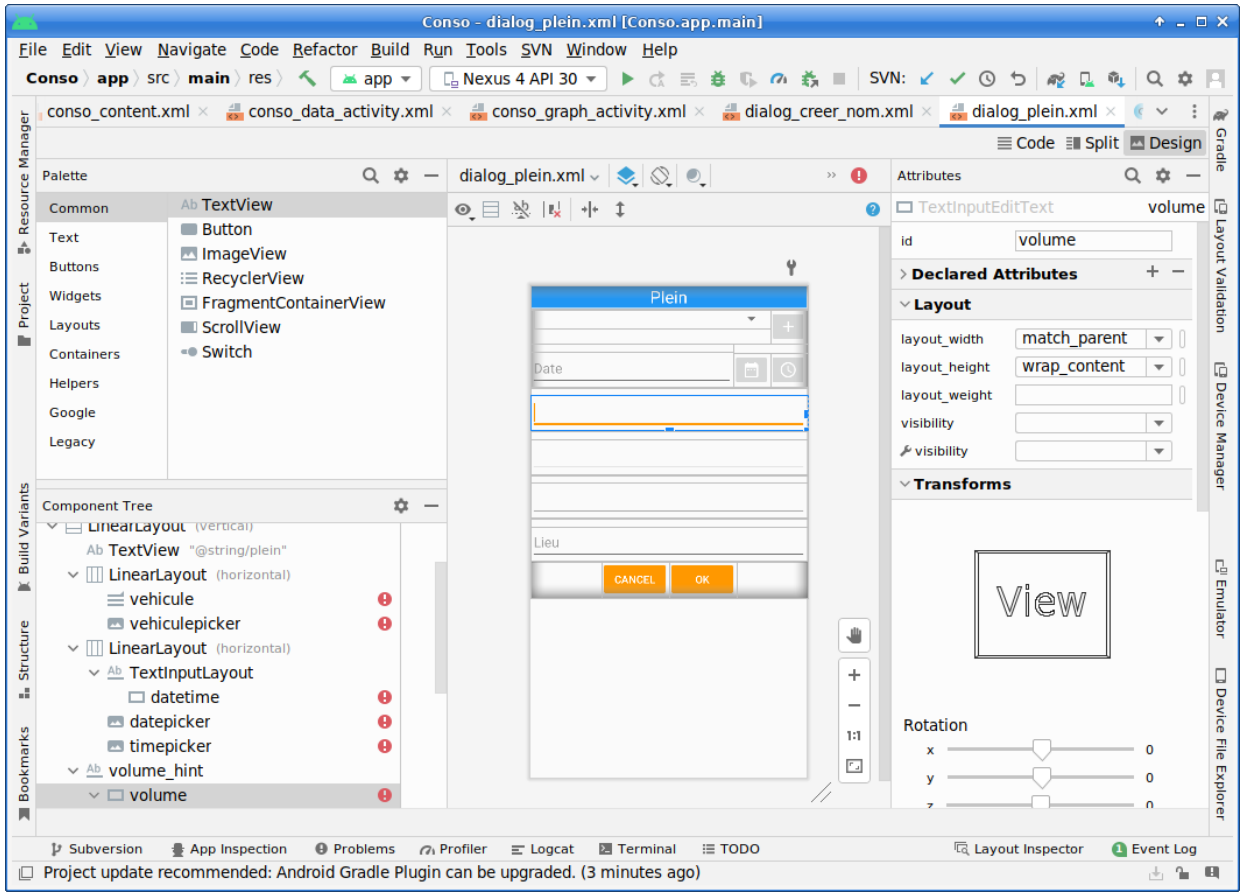
Les ressources (disposition des vues dans les interfaces, menus, images vectorielles, textes...) sont définies à l'aide de fichiers XML.

Studio fournit des éditeurs spécialisés pour ces fichiers, par exemple :

- Formulaires pour :
  - `res/values/strings.xml` : textes de l'interface.
- Éditeurs graphiques pour :
  - `res/layout/*.xml` : disposition des contrôles sur l'interface.

BY-NC-SA

Exemple `res/layout/main.xml`



Pierre Nerzic
BY-NC-SA

Source XML sous-jacent

Ces éditeurs sont beaucoup plus confortables que le XML brut, mais ne permettent pas de tout faire (widgets custom).

Assez souvent, il faut éditer le source XML directement : 

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
</RelativeLayout>
```

Notez le *namespace* des éléments et le préfixe de chaque attribut.

 BY-NC-SA

Reconstruction du projet

Chaque modification d'un source ou d'une ressource fait reconstruire le projet (compilation des sources, transformation des XML et autres). C'est automatique.

Dans de rares circonstances, mauvaise mise à jour des sources (partages réseau ou gestionnaire de version) :

- il peut être nécessaire de reconstruire manuellement. Il suffit de sélectionner le menu `Build/Rebuild Project`,
- il faut parfois nettoyer le projet. Sélectionner le menu `Build/Clean Project`.

Ces actions lancent l'exécution de *Gradle*.

BY-NC-SA

Gradle

Gradle est un outil de construction de projets comme Make (projets C++ sur Unix), Ant (projets Java dans Eclipse) et Maven.

De même que `make` se sert d'un fichier `Makefile`, Gradle se sert de fichiers nommés `build.gradle` pour construire le projet.

C'est assez compliqué car AndroidStudio fait une distinction entre le projet global et l'application. Donc il y a deux `build.gradle` :

- un script `build.gradle` dans le dossier racine du projet. Il indique quelles sont les dépendances générales (noms des dépôts Maven contenant les librairies utilisées).
- un dossier `app` contenant l'application du projet.
- un script `build.gradle` dans le dossier `app` pour compiler l'application.

BY-NC-SA

Structure d'un projet AndroidStudio

Un projet AndroidStudio est constitué ainsi :

```
.
+-- app/
|   +-- build/                          FICHIERS COMPILÉS
|   +-- build.gradle                    SPÉCIF. COMPILATION
|   `-- src/
|       +-- androidTest/                TESTS UNITAIRES ANDROID
|       +-- main/
|       |   +-- AndroidManifest.xml     DESCR. DE L'APPLICATION
|       |   +-- java/                   SOURCES
|       |   `-- res/                    RESSOURCES (ICONES...)
|       `-- test/                       TESTS UNITAIRES JUNIT
+-- build/                              FICHIERS TEMPORAIRES
+-- build.gradle                        SPÉCIF. PROJET
`-- gradle/                             FICHIERS DE GRADLE
```


BY-NC-SA

Utilisation de bibliothèques

Certains projets font appel à des bibliothèques externes. On les spécifie dans le `build.gradle` du dossier app, dans la zone `dependencies` :

```
dependencies {
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)
}
```

Les bibliothèques indiquées sont automatiquement téléchargées.

BY-NC-SA

Exécution de l'application

BY-NC-SA

Simulateur ou smartphone

L'application est prévue pour tourner sur un appareil (smartphone ou tablette) réel ou simulé (virtuel).

Le SDK Android permet de :

- Installer l'application sur une vraie tablette connectée par USB
- Simuler l'application sur une tablette virtuelle *AVD*

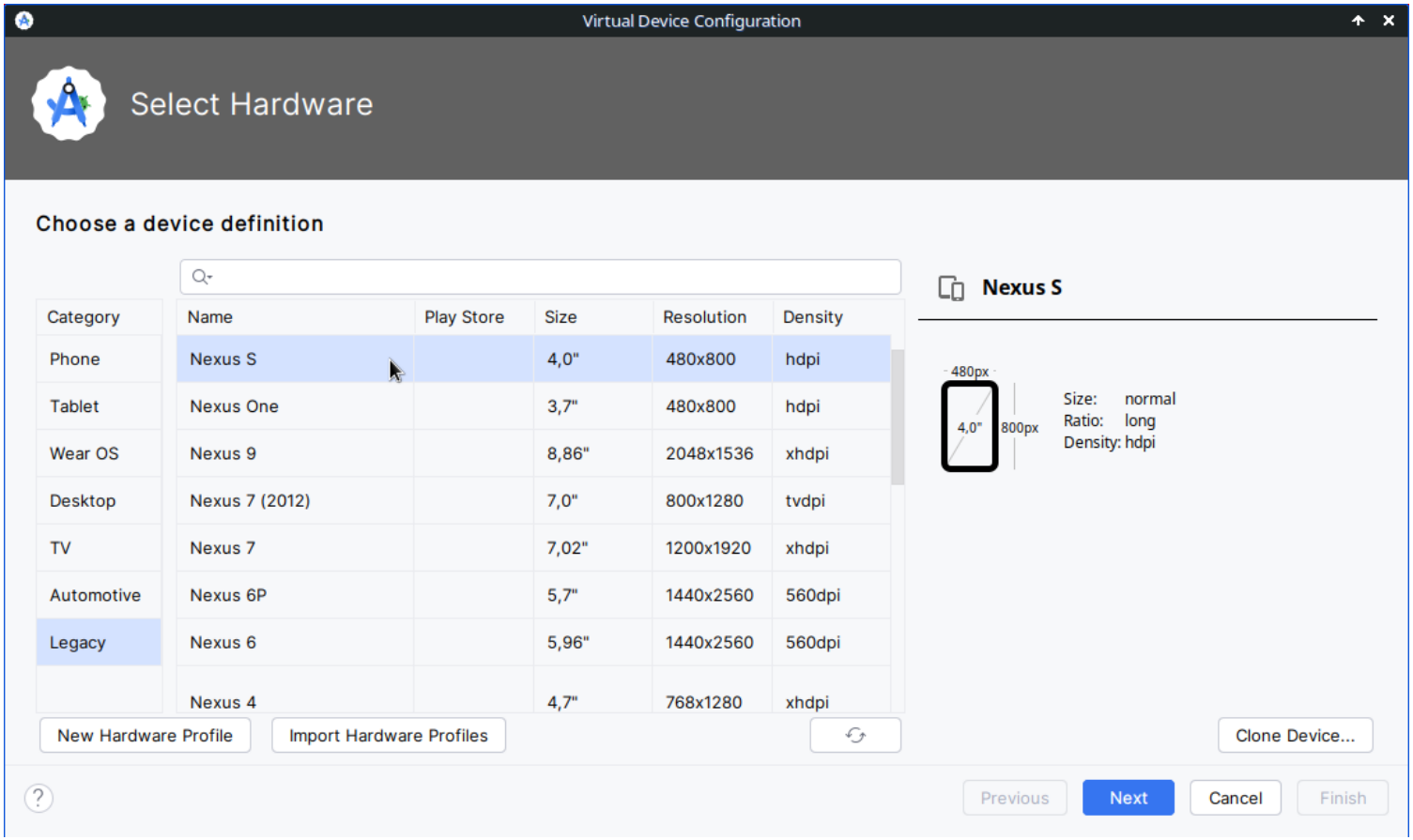
AVD = Android Virtual Device

C'est une machine virtuelle comme celles de VirtualBox et VMware, mais basée sur QEMU.

QEMU est en licence GPL, il permet d'émuler toutes sortes de CPU dont des ARM7, ceux qui font tourner la plupart des tablettes Android.

BY-NC-SA

Assistant de création d'une tablette virtuelle



 BY-NC-SA

Caractéristiques d'un AVD

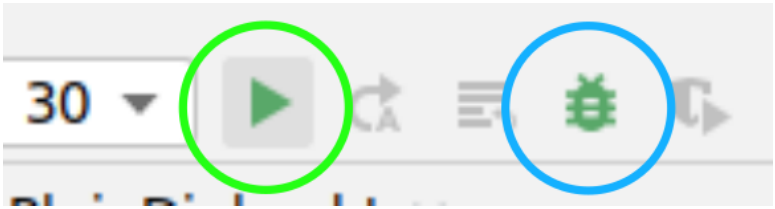
L'assistant de création de tablette demande :

- Modèle de tablette ou téléphone à simuler,
- Version du système Android,
- Orientation et densité de l'écran
- Options avancées :
  - `RAM` : mémoire à allouer, mais est limitée par votre PC,
  - `Internal storage` : capacité de la flash interne,
  - `SD Card` : capacité de la carte SD simulée supplémentaire (optionnelle).

BY-NC-SA

Lancement d'une application

Bouton vert pour exécuter, bleu pour déboguer :

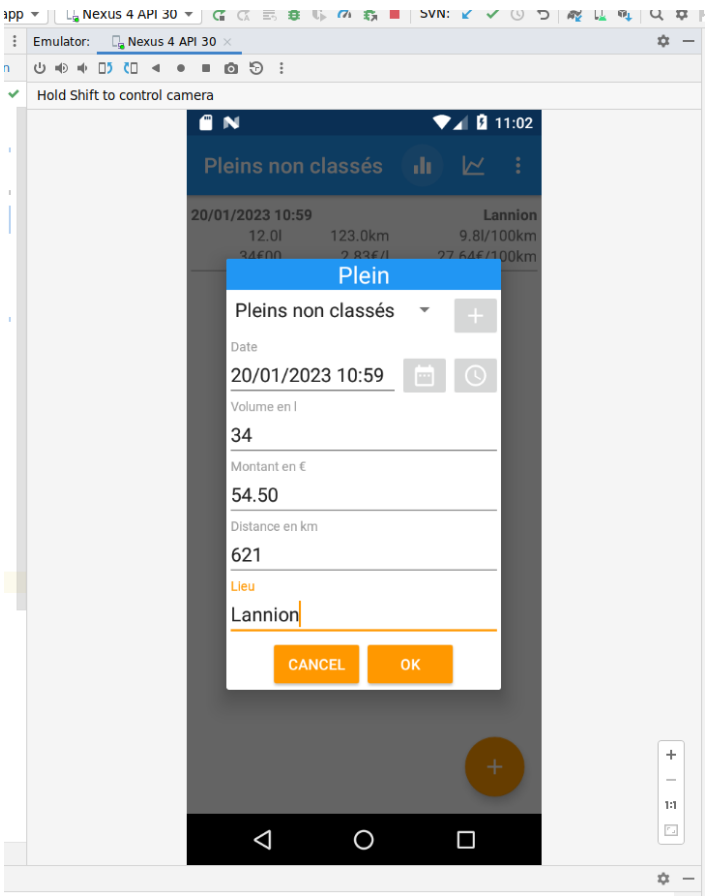


NB: les icônes, styles et emplacements, varient d'une version d'AndroidStudio à l'autre.

Ces deux boutons installent l'application sur l'AVD ou le smartphone et la démarrent.

Pierre Nerzic
BY-NC-SA

Application sur l'AVD



L'apparence change d'une version à l'autre du SDK.

Pierre Nerzic
BY-NC-SA

Communication AVD - Android Studio

Pierre Nerzic
BY-NC-SA

Fenêtres Android

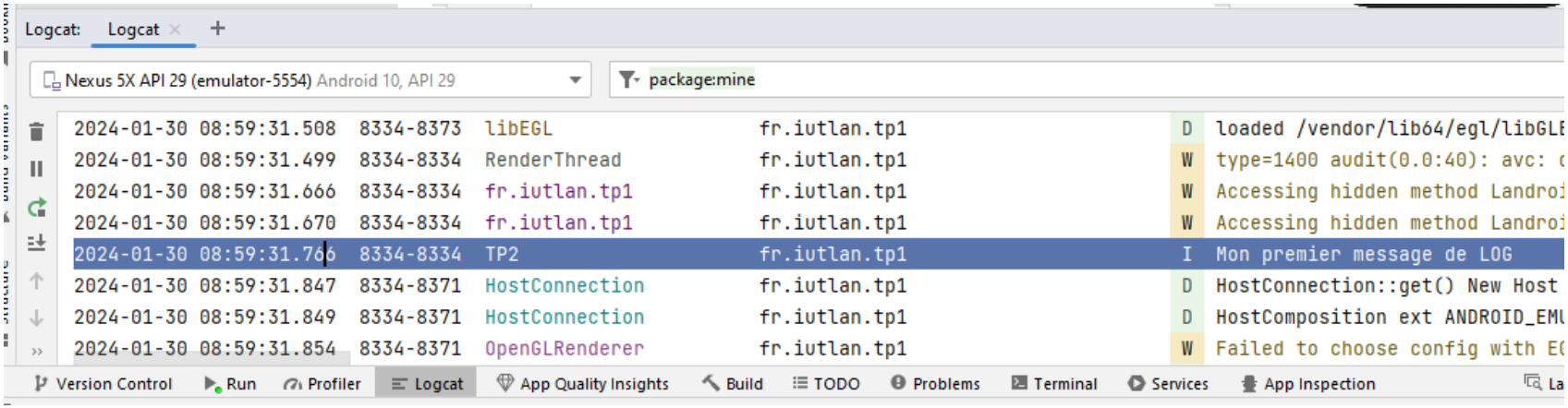
Android Studio affiche plusieurs fenêtres utiles indiquées dans l'onglet tout en bas :

Logcat	Affiche tous les messages émis par la tablette courante
Messages	Messages du compilateur et du studio
Terminal	Shell unix permettant de lancer des commandes dans le dossier du projet.

BY-NC-SA

Fenêtre Logcat

Des messages détaillés sont affichés dans la fenêtre LogCat :



Ils sont émis par les applications : debug, infos, erreurs... comme syslog sur Unix : date, heure, gravité, source (code de l'émetteur) et message.

BY-NC-SA

Filtrage des messages

Il est commode de définir des *filtres* pour ne pas voir la totalité des messages de toutes les applications de la tablette :

- sur le niveau de gravité : `verbose`, `debug`, `info`, `warn`, `error` et `assert`,
- sur l'étiquette *TAG* associée à chaque message,
- sur le *package* de l'application qui émet le message.

BY-NC-SA

Émission d'un message vers LogCat

Une application émet un message par ces instructions :

```
import android.util.Log;

const val TAG = "democours"

class MainActivity : ComponentActivity() {

    fun maMethode() {
        Log.i(TAG, "appel de maMethode()")
    }
```

Fonctions `Log.*` du package `android.util.Log` :

- `Log.i(tag: String, message: String)` affiche une info,
- `Log.w(tag: String, message: String)` affiche une alerte,
- `Log.e(tag: String, message: String)` affiche une erreur.

Pierre Nerzic
BY-NC-SA

Logiciel ADB

Android Studio contient donc un éditeur de sources et de ressources, un compilateur Kotlin et de quoi construire une application prête à être installée. Il y a aussi un outil de communication entre une tablette/smartphone (réel ou virtuel) et votre PC : *Android Debug Bridge*.

ADB est un logiciel qui fournit :

- Serveur de connexion des tablettes : `shell`
- Commandes de communication : `push`, `pull`, `install`

Ce sont les mêmes concepts que :

- FTP : transfert de fichiers,
- SSH : connexion à un shell.

Pierre Nerzic
BY-NC-SA

Mode d'emploi de ADB

En ligne de commande : `adb commande paramètres...`

- `adb devices` : liste les appareils connectés
- `adb shell` : connexion à l'appareil

Exemple :

```
~/CoursAndroid/$ adb devices
List of devices attached
emulator-5554   device
c1608df1b170d4f device
~/CoursAndroid/$ adb shell
$ pwd
/
$
```


Pierre Nerzic
BY-NC-SA

Système de fichiers Android

On retrouve l'architecture des dossiers Unix, avec des variantes :

- Dossiers Unix classiques : `/usr`, `/dev`, `/etc`, `/lib`, `/sbin`...
- Les volumes sont montés dans `/mnt`, par exemple `/mnt/sdcard` (SDcard amovible)
- Les applications sont dans :
  - `/system/app` pour les pré-installées
  - `/data/app` pour les applications normales
- Les données des applications sont dans `/data/data/`*nom.du.paquetage.java*
  Ex: `/data/data/fr.iutlan.helloworld/`...

NB : il y a des restrictions d'accès sur un vrai smartphone, car vous n'y êtes pas *root*.

BY-NC-SA

Mode d'emploi, suite

- Pour échanger des fichiers avec une tablette :
  - `adb push` *nom_du_fichier_local* */nom/complet/dest*
    envoi du fichier local sur la tablette
  - `adb pull` */nom/complet/fichier*
    récupère ce fichier de la tablette
- Pour gérer les logiciels installés :
  - `adb install paquet.apk`
  - `adb uninstall` *nom.du.paquetage.java*
- Pour archiver les données de logiciels :
  - `adb backup -f` *fichier_local* *nom.du.paquetage.java* ...
    enregistre les données du/des logiciels dans le fichier local
  - `adb restore` *fichier_local*
    restaure les données du/des logiciels d'après le fichier.

BY-NC-SA

Création d'un paquet installable

Pierre Nerzic
BY-NC-SA

Paquet

Un paquet Android est un fichier `.apk`. C'est une archive signée (authentifiée) contenant les binaires, ressources compressées et autres fichiers de données.

La création est relativement simple avec Studio :

1. Menu `Build...`, choisir `Generate Signed Bundle/APK`,
2. Signer le paquet à l'aide d'une *clé privée*,
3. Définir l'emplacement du fichier `.apk`.

Le résultat est un fichier `.apk` dans le dossier spécifié.

BY-NC-SA

Signature d'une application

Lors de la mise au point, Studio génère une clé qui ne permet pas d'installer l'application ailleurs. Pour distribuer une application, il faut une *clé privée*.

Les clés sont stockées dans un *keystore* = trousseau de clés. Il faut le créer la première fois. C'est un fichier crypté, protégé par un mot de passe, à ranger soigneusement.

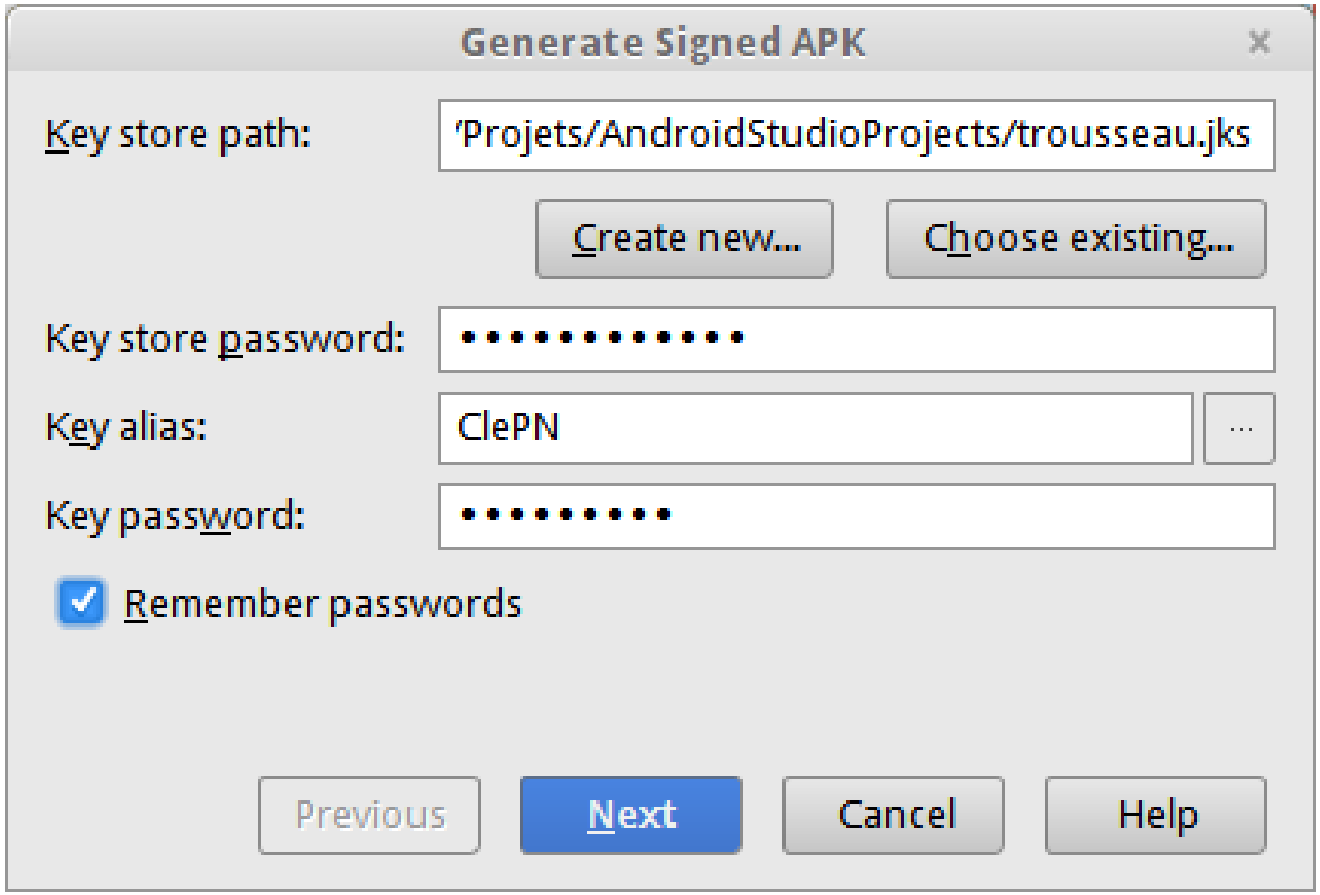
Ensuite créer une *clé privée* :

- `alias` = nom de la clé, mot de passe de la clé
- informations personnelles complètes : prénom, nom, organisation, adresse, etc.

Les mots de passe du trousseau et de la clé seront demandés à chaque création d'un `.apk`. **Ne les perdez pas**.

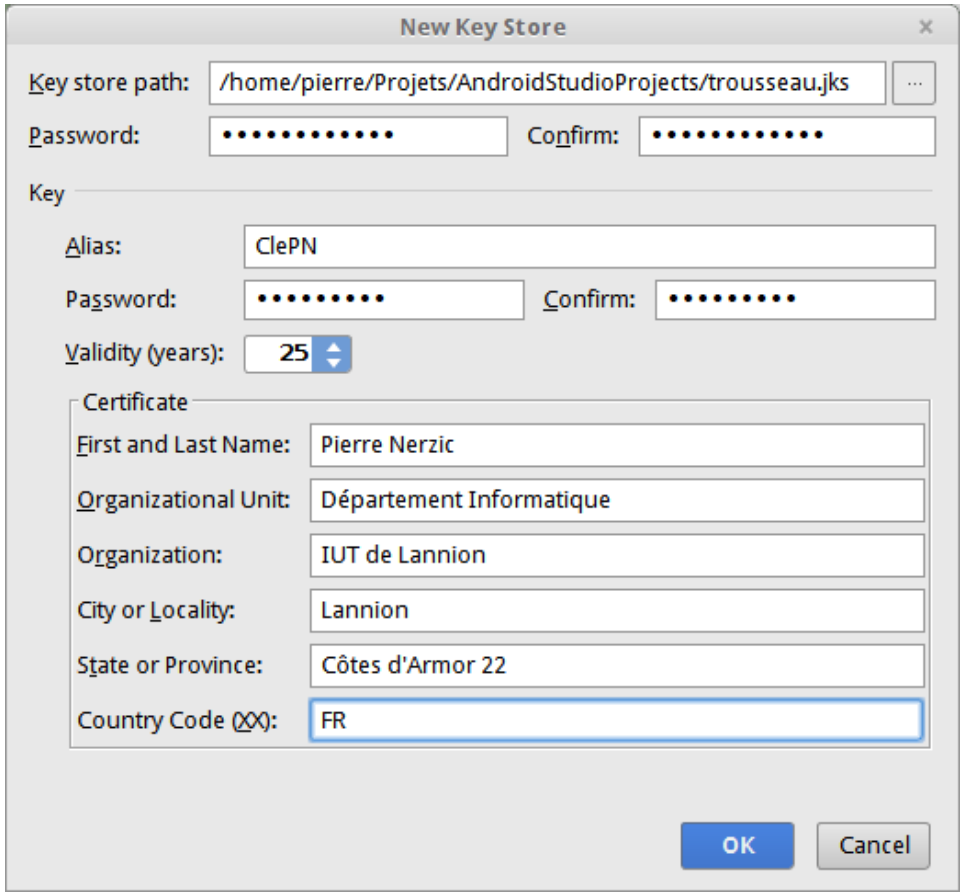
BY-NC-SA

Création du *keystore*



 BY-NC-SA

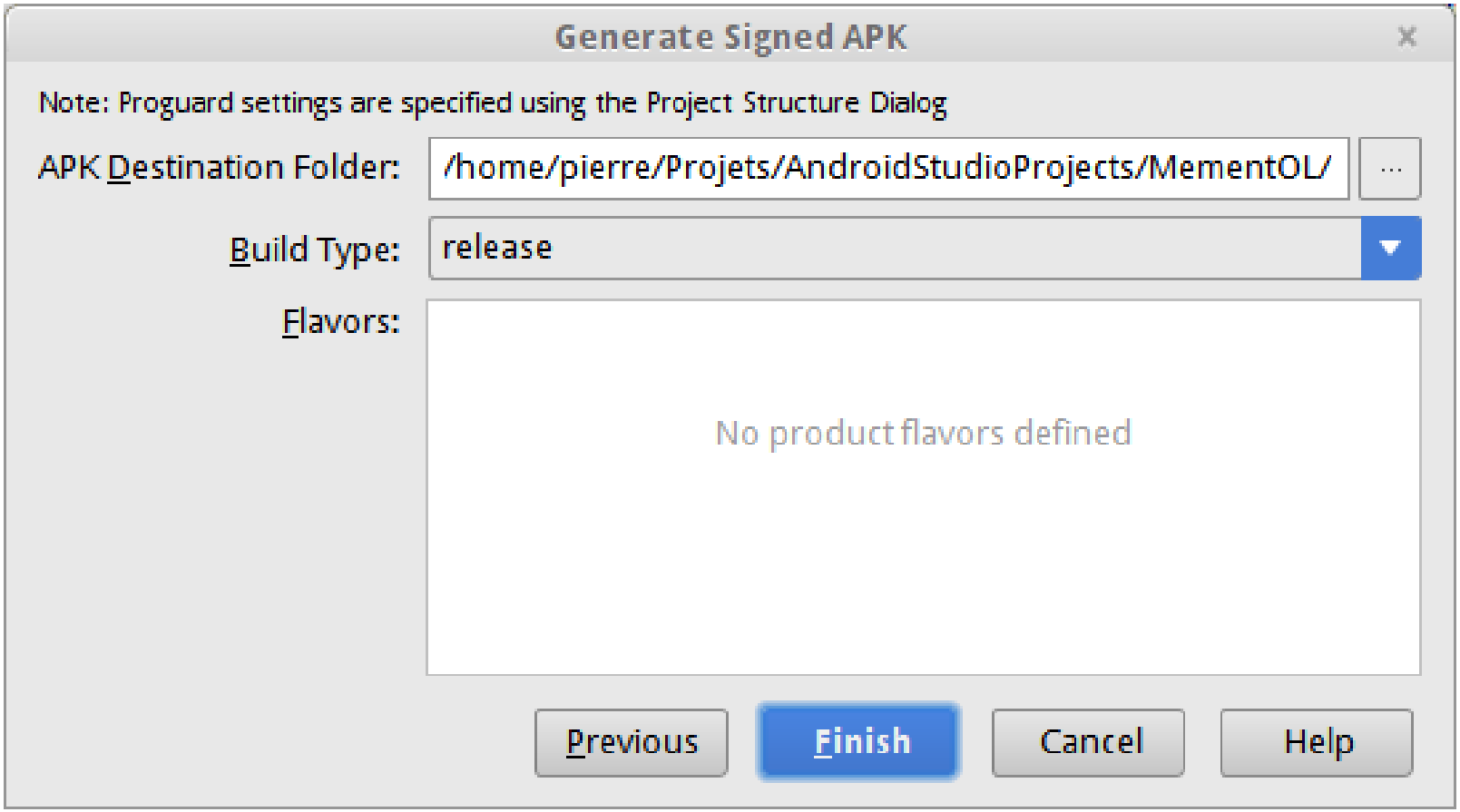
Création d'une clé



BY-NC-SA

Création du paquet

Ensuite, Studio demande où placer le `.apk` :



Pierre Nerzic
BY-NC-SA

Et voilà

C'est fini pour cette semaine, rendez-vous en TD/TP pour la mise en pratique.

Pierre Nerzic
BY-NC-SA