Chapitre 8

Canaux de communication et matériel

Téléphonie

- L'API a été développée pour les appareils téléphoniques.

- Or, il y a aussi les tablettes et certaines d'entre elles qui ne peuvent pas effectuer un appel téléphonique.

- Pour ne restreindre les accès à votre application qu'aux appareils pouvant effectuer un appel téléphonique, il faudra ajouter ces lignes dans le fichier « Android Manifest » :

```
<uses-feature android:name="android.hardware.telephony"
                                    android:required="true"/>
```

- En déclarant « `android:required="true"` » cela signifie que l'application ne peut pas fonctionner, ou n'est pas conçue pour fonctionner, lorsque la fonctionnalité spécifiée n'est pas présente sur l'appareil.

- L'API permet de réaliser les opérations suivantes :
  - Effectuer des appels téléphoniques.
  - Obtenir des informations sur les propriétés du téléphone ainsi que le réseau qu'il utilise.
  - Gérer les appels entrants.
  - Envoyer et recevoir des SMS (ou MMS).
- La classe responsable de cette API est « TelephonyManager » :

  http://developer.android.com/reference/android/telephony/TelephonyManager.html

- Faire un appel téléphonique

  Déclarer un intent et lui associer :
  - Action : la tâche à réaliser, ici, effectuer un appel téléphonique.
  - Donnée : le numéro de téléphone sous la forme d'un URI.

```
Intent i = new
        Intent(android.content.Intent.ACTION_DIAL,
                Uri.parse("tel:+" + phoneNumber));
startActivity(i);
```

Quand l'intent est lancé, le combiné téléphonique sera affiché avec le numéro à appeler. L'utilisateur doit cliquer sur le bouton « appel » pour autoriser l'action.

Pour que cette tâche soit automatisée, il faudra utiliser « *ACTION_CALL* » à la place de « *ACTION_DIAL* » et ajoutez dans le fichier « Android Manifest », la possibilité à l'application d'effectuer directement un appel téléphonique :

```
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

À noter qu'il est déconseillé d'utiliser « *ACTION_CALL* » car il y a des restrictions sur les numéros d'appel. Par exemple, les appels urgents (911) ne sont pas autorisés.

Examiner l'exemple « **C08 : MakeCalls** ».

- Accéder à l'état du téléphone et aux propriétés de la téléphonie

```
TelephonyManager tm = ((TelephonyManager)
                  getSystemService(Context.TELEPHONY_SERVICE));
```

L'instance « tm » va vous donner accès aux propriétés du téléphone, du réseau utilisé, l'état de la connectivité, etc.

Consulter les différents états

Obtenir le type du téléphone (GSM, CDMA, SIP), son identifiant (IMEI ou MEID), la version du logiciel et le numéro de téléphone. Mais attention …

https://developer.android.com/training/articles/user-data-ids.html


```
//---Obtenir l'identifiant de la carte SIM---
String simID = tm.getSimSerialNumber();


//---Obtenir le numéro de téléphone---
String telNumber = tm.getLine1Number();

//---Obtenir le numéro IMEI--- (depuis oreo, avant getDeviceId)
String IMEI = tm.getImei();

//---Obtenir la nature du réseau (GPRS, CDMA, etc.)---
int nettype = tm.getNetworkType();


//---Obtenir le type du téléphone (GSM, CDMA, etc.)---
int phonetype = tm.getPhoneType();
```

Il fallait autoriser dans le fichier « Android Manifest » la lecture de ces informations :

```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

Examiner l'exemple « **C08 : SIMCardID** ».

Le numéro IMEI (International Mobile Equipment Identity) a besoin de plus de privilèges. Vous avez besoin d'inclure aussi cette permission dans le manifeste de l'application :

```
<uses-permission
android:name="android.permission.READ_PRIVILEGED_PHONE_STATE" />
```

Sauf que cette permission est autorisée uniquement pour les applications système. Afin de pouvoir tester cet exemple sur un émulateur, nous avons besoin d'ignorer cette erreur temporairement!

```
<uses-permission
android:name="android.permission.READ_PRIVILEGED_PHONE_STATE"
    tools:ignore="ProtectedPermissions" />
```


Surveiller le changement de l'état du téléphone

Nous aimerions surveiller l'état du téléphone : appel en cours, sonnerie, transferts d'appels, etc.

Le téléphone passe par 3 états possibles :

- Le téléphone n'est pas utilisé (« CALL_STATE_IDLE »).
- Le téléphone sonne (« CALL_STATE_RINGING »).
- Le téléphone est utilisé (« CALL_STATE_OFFHOOK »).

Pour connaître le statut de ces états, et même si la classe « PhoneStateListener », responsable de surveiller l'état du téléphone, est dépréciée depuis l'API 31, nous allons l'utiliser quand même pour expliquer les fonctionnalités intrinsèques du téléphone.

Le but de cette surveillance est de détecter les appels entrants et de réagir en conséquence.

Nous devons d'abord dériver de la classe « PhoneStateListener ».

```
public class PhoneReceiver extends PhoneStateListener { … }
```

Par la suite, nous devons redéfinir la méthode « onCallStateChanged() » responsable de détecter les changements.

```
public void onCallStateChanged(int state,String incomingNumber) {
    super.onCallStateChanged(state, incomingNumber);
    switch (state) {
    case TelephonyManager.CALL_STATE_IDLE:
        // faire quelque chose
        break;
    case TelephonyManager.CALL_STATE_RINGING:
        // faire quelque chose
        break;
    case TelephonyManager.CALL_STATE_OFFHOOK:
        // faire quelque chose
        break;
    }
}
```


Dans l'activité :

Dans la méthode « onCreate », on déclare une instance de la classe « TelephonyManager » et « PhoneReceiver » :

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    myPhoneStateListener = new PhoneReceiver(this);
    manager = ((TelephonyManager)
            getSystemService(Context.TELEPHONY_SERVICE));
}
```

Dans les méthodes « onResume » et « onPause », on se met à écouter les changements.

```
@Override
public void onResume() {
    super.onResume();
    manager.listen(myPhoneStateListener,
            PhoneStateListener.LISTEN_CALL_STATE);
}

@Override
public void onPause() {
    super.onPause();
    manager.listen(myPhoneStateListener,
            PhoneStateListener.LISTEN_NONE);
}
```

Si vous voulez que votre application continue de surveiller l'état de votre téléphone, même si l'application n'est pas en avant-plan, il faudra utiliser dans ce cas un récepteur.

On commence par déclarer une nouvelle classe qui va dériver de BroadcastReceiver.

```
public class PhoneStateReceiver extends BroadcastReceiver {…}
```

On redéfinit dans cette classe la méthode « onReceive ». Elle aura la tâche d'appeler la méthode « Listener » de « TelephonyManager ».

Nous allons définir aussi une classe interne qui va dériver de « PhoneStateListener », comme dans le précédent cas de figure. Dans cette classe, nous allons redéfinir la méthode « onCallStateChanged() ».

Il fallait ajouter aussi le récepteur dans le fichier « Android Manifest » :

```
<receiver android:name=".PhoneStateReceiver"
    android:exported="false">
    <intent-filter>
        <action
            android:name=
                "android.intent.action.PHONE_STATE" />
    </intent-filter>
</receiver>
```

Dans le fichier « Android Manifest », il fallait ajouter les permissions de lecture de l'état du téléphone ainsi que le journal des appels, :

```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
```

Examiner l'exemple « **C08 : Phone** ».

On peut utiliser aussi ce service pour surveiller les appels entrants :

Revoir l'exemple « **C02 : MyPhoneReceiver** ».

SMS

- Le SMS est la fonctionnalité la plus utilisée sur les téléphones portables.
- Elle repose sur l'envoi de messages textes courts.
- Par opposition, le MMS sert à envoyer des messages contenant des attachements multimédias (Photos, Vidéo, Son).
- Il y a deux manières pour envoyer des SMS :
  - En utilisant l'application SMS intégrée dans l'appareil.
  - En utilisant l'API associé à SmsManager.
- Comme dans le cadre de la téléphone, il faudra ajouter ces lignes dans le fichier « Android Manifest » :

```
<uses-feature android:name="android.hardware.telephony"
                                    android:required="true"/>
```

Envoyer un SMS avec l'application intégrée :

```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);
sendIntent.putExtra("sms_body", "default content");
sendIntent.setType("HTTP.PLAIN_TEXT_TYPE");
startActivity(sendIntent);
```

Il faut ajouter aussi l'autorisation d'envoyer un SMS dans le fichier « Android Manifest » :

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

Examiner l'exemple « **C08 : SMSLocal** ».

Envoyer un SMS avec SmsManager

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage(phoneNo, null, sms, null, null);
Toast.makeText(getApplicationContext(), "SMS Sent!",
                    Toast.LENGTH_LONG).show();
```

public void sendTextMessage (String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)

Il faut ajouter aussi l'autorisation d'envoyer un SMS dans le fichier « Android Manifest » :

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

Examiner l'exemple « C08 : SMSManager ».

Intercepter un SMS

Pour lire un SMS, nous pouvons utiliser l'application par défaut et dans ce cas, c'est le système qui va intercepter le message pour nous. Il va se charger de démarrer l'application appropriée, définie par défaut.

Nous pouvons aussi créer une application pour intercepter le SMS et l'afficher dans un format donné.

Nous allons utiliser le même principe que dans le cas de l'exemple « C02 : AirplaneMode ».

Nous allons déclarer un « BroadcastReceiver » qui va se charger d'écouter si un SMS a été envoyé. Si c'est le cas, il va le lire et l'afficher dans les logs de l'application.

Nous allons enregistrer ce « BroadcastReceiver » dans le fichier « Android Manifest » :

```
<receiver android:name="SMS_Catcher"/>
```

Dans le même fichier, nous allons autoriser l'application à lire les SMS entrants:

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

Si un intent correspondant à la réception d'un SMS a été enregistré (android.provider.Telephony.SMS_RECEIVED), nous allons décoder le contenu du message.

Examiner l'exemple « **C08 : TSTCatchSMS** ».

Attention, il vous faut une activité pour provoquer le déclenchement du BroadcastReceiver (comme dans l'exemple « AirplaneMode »).

Les notifications

https://developer.android.com/guide/topics/ui/notifiers/notifications.html

Une notification est un message affiché à l'utilisateur en dehors de l'application qui a le focus.

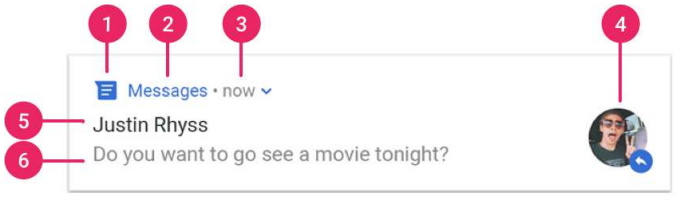
Les messages envoyés peuvent être en rapport avec de la publicité, des mises à jour, une alerte météo, etc.

Quand le système est informé d'une notification, cette dernière est affichée dans la zone des notifications.

Pour avoir le détail de la notification, il faudra ouvrir le tiroir des notifications en gardant le doigt (ou la souris pour l'émulateur) appuyé de haut en bas de l'écran sur la notification.

Les vues d'une notification

Une notification peut avoir une vue normale :



1. Icône petite, requise et configurée à l'aide de « setSamllIcon() »
2. Nom de l'application, fourni par le système
3. Heure d'émission de la notification, fournie par le système et configurable
4. Icône large, optionnelle, pour ajouter une photo par exemple. Elle ne sert pas pour mettre l'icône de l'application.
5. Titre, optionnel, configuré à l'aide de « setContentTitle() »
6. Texte, optionnel, configuré à l'aide de « setContentText() »

Depuis Android 4.1 (Jelly Bean), une notification peut avoir aussi une vue large. La vue normale est affichée par défaut. Il faudra ouvrir le tiroir des notifications pour avoir accès à la vue large. Elle a les mêmes propriétés qu'une vue normale. On ajoute seulement la possibilité d'afficher plus de détails avec un des styles prédéfinis.

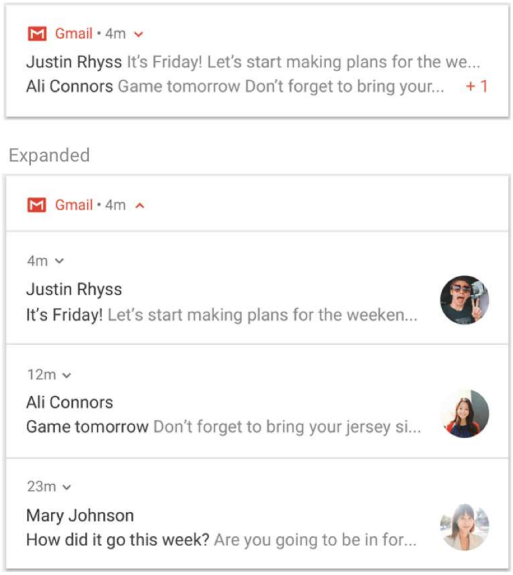
Depuis Android 5 (Lollipop), une notification apparaît brièvement en avant-plan d'une application. Elle peut apparaître aussi sur un écran verrouillé.

Depuis la version 8 (Oreo), un utilisateur peut désactiver toutes les notifications sur un écran verrouillé ou pour certains canaux (ce point sera introduit plus tard dans le document).

Un point coloré (« notification dot ») a été introduit depuis la version 8 pour signaler une notification en attente de lecture, pour une application donnée (« Gmail », « Skype », etc.).

Chaque notification doit ouvrir une application appropriée. On peut ajouter des actions à la notification. Depuis la version 7 (Nougat), on peut répondre directement sans ouvrir une nouvelle activité.

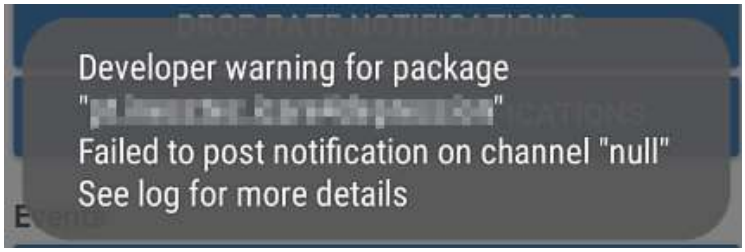
Pour éviter de bombarder l'utilisateur de notifications, il est préférable de se contenter de mettre à jour la notification. Si vous voulez quand même en envoyer plusieurs, il est conseillé vivement de les regrouper dans le même conteneur (disponible depuis Android 7).



Canaux de notifications

Dans Android 7 et moins, l'utilisateur peut gérer les notifications application par application. Chaque application n'a en réalité qu'un seul canal.

Depuis Android 8, toutes les notifications doivent être assignées à un canal sinon elles n'apparaîtront pas. Dans le cas contraire, vous allez obtenir un « toast » avec ce genre de message :



En créant plusieurs canaux, l'utilisateur aura la possibilité de bloquer qu'une catégorie en particulier de notifications et non pas toutes les notifications (comme c'était le cas auparavant).

https://developer.android.com/develop/ui/views/notifications/notification-permission

Depuis Android 13, nous avons besoin d'ajouter la permission de notification dans le fichier manifeste de l'application.

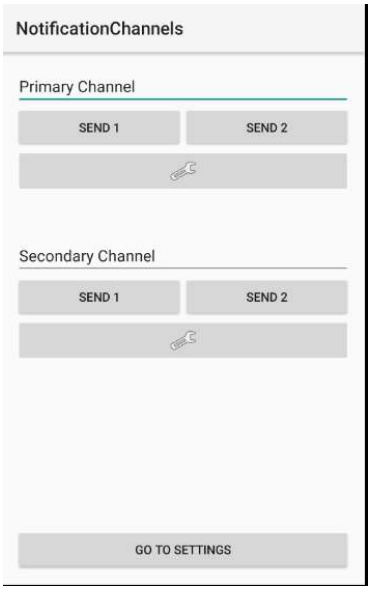
```
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
```


Un journal peut avoir plusieurs canaux, un pour les titres, un pour l'actualité nationale, un autre pour l'actualité internationale, un pour le sport, un pour les faits divers, etc. Vous pouvez choisir d'activer les notifications que pour les titres et désactiver les autres canaux.

Un canal peut avoir plusieurs paramètres : activé/désactivé, son, vibration, etc.

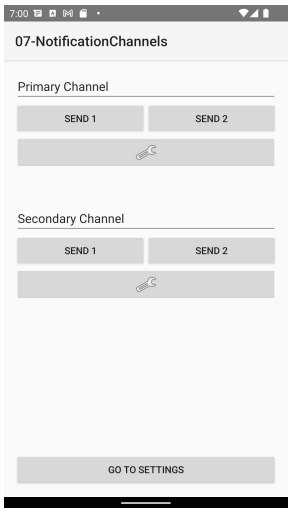
Dans ce qui suit, nous allons examiner l'exemple disponible sur cette page :

https://github.com/googlesamples/android-NotificationChannels



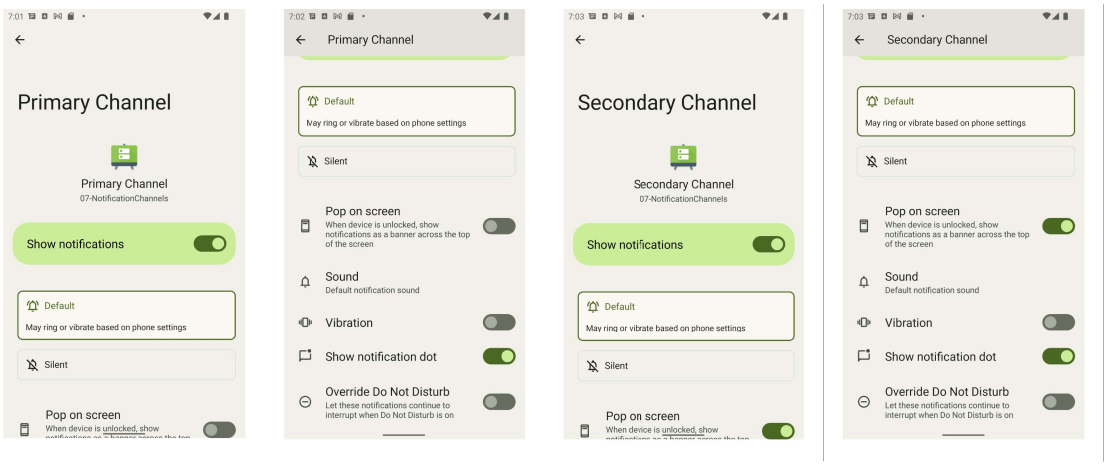
Examiner l'exemple « **C08 : android-NotificationChannels-master** ».

Nous allons créer deux canaux. Chacun des canaux peut envoyer divers types de notifications (« send 1 et send 2 »).



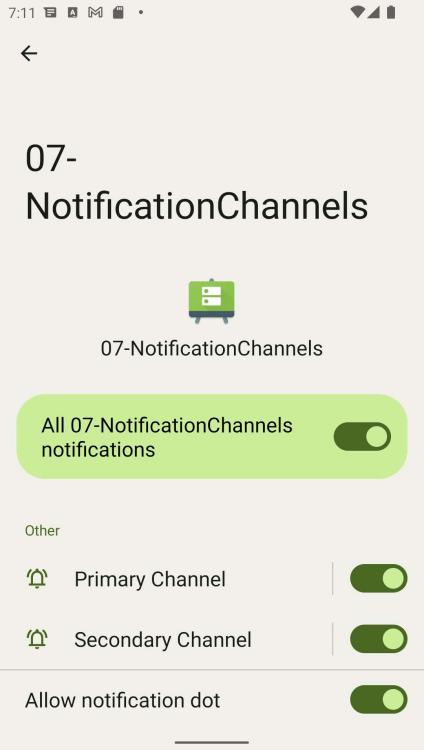
Les deux canaux diffèrent par le format de la notification.

Chaque canal a sa propre configuration. Certains éléments peuvent être fixes, d'autres peuvent être configurables par l'utilisateur. Dans cet exemple, nous avons fixé le fait qu'une notification génère un son, qu'elle a un niveau de priorité élevé. L'utilisateur peut par contre activé/désactivé le canal, émettre des vibrations, l'avis de l'arrivée d'une notification (« dot »), contourner la règle « ne pas déranger ».



L'application peut configurer aussi de manière globale les deux canaux. L'utilisateur peut activer/désactiver un canal, l'avis via le « dot » de l'arrivée d'une notification.

À noter qu'au niveau de l'application, les canaux sont listés dans la rubrique « Other ».



Le fichier « NotificationHelper.java » va contenir la création et la configuration des deux canaux.

Création d'une notification

On crée le canal « chan1 », ayant un nom, une couleur (si l'appareil supporte cette option) et il sera non visible si l'écran est verrouillé.

```
NotificationChannel chan1 = new NotificationChannel(PRIMARY_CHANNEL,
        getString(R.string.noti_channel_default),
        NotificationManager.IMPORTANCE_DEFAULT);
chan1.setLightColor(Color.GREEN);
chan1.setLockscreenVisibility(Notification.VISIBILITY_PRIVATE);
```

Par la suite, nous allons soumettre ce canal au gestionnaire des notifications :

```
getManager().createNotificationChannel(chan1);
```

On crée le canal « chan2 », ayant un nom, une couleur (si l'appareil supporte cette option), et il sera visible si l'écran est verrouillé.

```
NotificationChannel chan2 = new NotificationChannel(SECONDARY_CHANNEL,
        getString(R.string.noti_channel_second),
        NotificationManager.IMPORTANCE_HIGH);
chan2.setLightColor(Color.BLUE);
chan2.setLockscreenVisibility(Notification.VISIBILITY_PUBLIC);
```

Par la suite, nous allons soumettre ce canal au gestionnaire des notifications :

```
getManager().createNotificationChannel(chan2);
```


Création d'une notification

On commence par bâtir les caractéristiques d'une notification dans un objet du type « Notification.Builder », tout en l'assignant au canal approprié.

```
public Notification.Builder getNotification1(String title, String body) {
  return new Notification.Builder(getApplicationContext(), PRIMARY_CHANNEL)
            .setContentTitle(title)
            .setContentText(body)
            .setSmallIcon(getSmallIcon())
            .setAutoCancel(true);
}

public Notification.Builder getNotification2(String title, String body) {
  return new Notification.Builder(getApplicationContext(), SECONDARY_CHANNEL)
            .setContentTitle(title)
            .setContentText(body)
            .setSmallIcon(getSmallIcon())
            .setAutoCancel(true);
}
```

Pour déclencher une notification tout en l'associant aux caractéristiques préalablement définies :

```
public void notify(int id, Notification.Builder notification) {
    getManager().notify(id, notification.build());
}
```

Pour obtenir une instance du gestionnaire de notifications « NotificationManger » :

```
private NotificationManager getManager() {
    if (manager == null) {
        manager = (NotificationManager)
                    getSystemService(Context.NOTIFICATION_SERVICE);
    }
    return manager;
}
```

Le service associé aux notifications est « NOTIFICATION_SERVICE ».

Le fichier « MainActivity.java » sert à la description de l'application et fait le lien avec le fichier de configuration des deux canaux.

Pour charger la configuration du système de notifications d'un canal donné :

```
public void goToNotificationSettings(String channel) {
    Intent i = new Intent(Settings.ACTION_CHANNEL_NOTIFICATION_SETTINGS);
    i.putExtra(Settings.EXTRA_APP_PACKAGE, getPackageName());
    i.putExtra(Settings.EXTRA_CHANNEL_ID, channel);
    startActivity(i);
}
```

Pour charger la configuration du système de notifications de l'application :

```
public void goToNotificationSettings() {
    Intent i = new Intent(Settings.ACTION_APP_NOTIFICATION_SETTINGS);
    i.putExtra(Settings.EXTRA_APP_PACKAGE, getPackageName());
    startActivity(i);
}
```

Localisation

Perdu dans la route, qui n'a pas voulu savoir son emplacement géographique exact et retrouver ainsi son chemin?

Dans un bled perdu, qui n'a pas voulu savoir le chemin vers son hôtel?

Qui n'a pas voulu savoir s'il y a un bon restaurant dans le coin?

Tout cela a été rendu possible grâce à la technique de localisation géographique.

Cette technique utilise le système de positionnement global (GPS).

Elle a été développée par le département de la défense américaine en 1978 et rendue accessible au civil en 1983.

Le système est complètement fonctionnel en 1995 avec ses 24 satellites opérationnels et quelques satellites de garde en cas de panne.

La géolocalisation repose sur le système de coordonnées (Longitude/Latitude). Pour avoir une idée de quoi s'agit-il :

http://youtu.be/swKBi6hHHMA

Et pour comprendre le fonctionnement du système GPS :

http://youtu.be/WoqpQbWdacQ

À noter que la localisation ne repose pas forcément que sur le système GPS. Elle peut se servir aussi du WIFI ou la 4G.

Android et la localisation

Pour obtenir la localisation d'un appareil Android, nous avons deux possibilités :

- En utilisant des librairies natives de l'API d'Android de « android.location.LocationListener ».

- Ou bien des librairies de l'API « Google Play Services API » « com.google.android.gms.location.LocationListener ».

Nous allons tester deux techniques de localisation : la première consiste à obtenir l'information du premier fournisseur disponible (GPS, WiFi, G4). Quant à la seconde, elle utilise strictement le GPS. L'utilisation de la librairie Criteria dans la première technique est maintenant dépréciée. Il est recommandé de cibler un fournisseur précis.

Pour ce dernier cas, il faut avoir un accès externe (le ciel soit disponible) pour faciliter les accès aux signaux émis par les satellites de géolocalisation.

Sachant que l'obtention des informations relatives aux coordonnées de géolocalisation n'est pas une opération synchrone, nous allons utiliser un service en arrière-plan pour demander la position de l'appareil.

Comme tout service qui a besoin d'un certain temps pour s'exécuter, nous allons utiliser un thread en parallèle au thread UI pour réaliser cette tâche.

Comme la personne peut bouger en permanence, nous devons donc mettre à jour constamment les informations GPS.

Le fichier « Android Manifest » doit contenir les autorisations nécessaires.

Pour une localisation précise, avec GPS, il faudra inclure cette permission :

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Pour une localisation approximative, par WiFi, G4, utilisez cette permission :

```
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Si la première permission (ACCESS_FINE_LOCATION) est nécessaire au bon fonctionnement de votre application, depuis Android 12 (API 32), vous avec l'obligation de déclarer aussi dans le manifeste de l'application, la permission (ACCESS_COARSE_LOCATION).

Nous allons examiner deux exemples :

- Utilisation de « android.location »

`Examiner l'exemple « C08 : TesterGPS ».`

Il faut définir une instance de « LocationManager ». Par la suite, il faut implémenter « LocationListener », et appeler « requestLocationUpdates » sur le « LocationManager ». Cette approche est maintenant dépréciée.

- Utilisation de « FusedLocation »

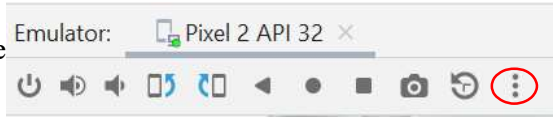
`Examiner l'exemple « C08 : LocationUpdates ».`

L'application a besoin de se connecter au « GooglePlayServicesClient ». L'activité a besoin d'implémenter ces deux interfaces :

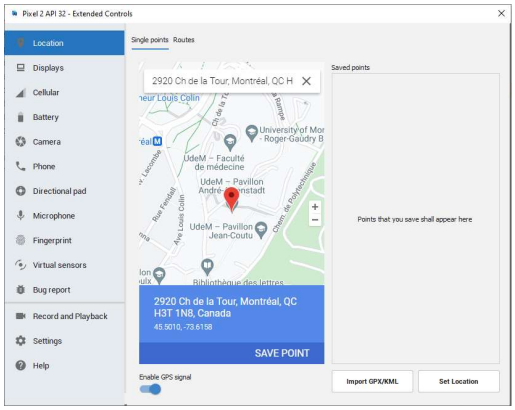
« GooglePlayServicesClient.ConnectionCallbacks »
« GooglePlayServicesClient.OnConnectionFailedListener »

Pour tester l'exemple du GPS, vous avez les deux solutions suivantes :

- Cliquer sur les « ... » dans la barre des outils associée à l'émulateur.



Puis, choisir « Location » afin d'inclure l'adresse du DIRO :



- Connectez-vous à l'appareil par « Telnet » et envoyez une commande « geo » avec la latitude et la longitude, comme suit :

```
telnet localhost 5554
geo fix 45.5010 -73.6158
```

On suppose que l'émulateur écoute sur le port 5554.

La première valeur représente la longitude, la seconde la latitude.

[geo fix longitude latitude]

Afficher des cartes (MapView)

Vous avez probablement tous, utilisé la version web de l'outil « Google Map ».

Cet outil est accessible aussi par programmation.

Il est donc possible de l'intégrer dans votre application Android.

On peut avoir accès à une carte à l'aide d'un intent.

`Examiner l'exemple « C08 : IntentMap ».`

Dans l'exemple, nous faisons appel à « Google Map » à travers une nouvelle activité :

```
String MesCoordonnes = "geo:45.50113651565565,-73.61580998771994?z=20";
Uri gmmIntentUri = Uri.parse(MesCoordonnes);
Intent mapIntent = new Intent(Intent.ACTION_VIEW, gmmIntentUri);
mapIntent.setPackage("com.google.android.apps.maps");
startActivity(mapIntent);
```

- Il faut utiliser un émulateur avec l'API : « Google APIs (Google Inc.) ».

On constate que l'usage est très limité.

- Pour le zoom, nous avons ces options :

  1 : Le monde

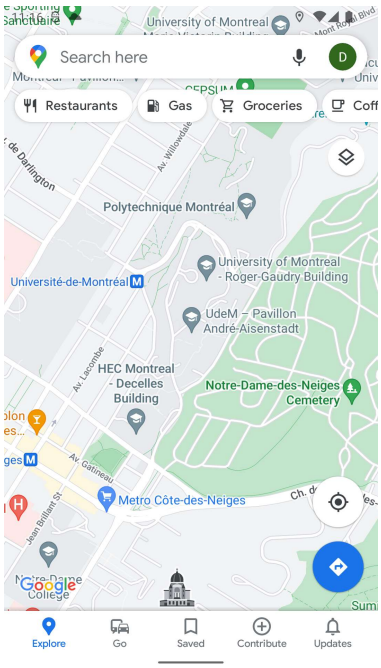
  5 : La masse continentale/le continent

  10 : La ville

  15 : Les rues

  20 : Les immeubles

- Le résultat obtenu pour z=15 :



Utilisation Google Map

Une autre approche que les intents est d'utiliser l'API Google Maps pour avoir accès à une carte.

Il y a plusieurs conditions à respecter pour faciliter l'intégration d'une carte dans une application Android.

- L'API à utiliser n'est pas l'API de base. En effet, comme l'API Google Maps ne fait pas partie du paquetage d'Android, il faudra donc l'inclure pour pouvoir l'utiliser.

- Pour une version de l'API que vous désirez utiliser, vérifiez que l'extension « Google APIs » est installée aussi. Dans le cas contraire, pensez à l'installer.

- Pour utiliser une carte, il faut avoir une clé personnalisée, comme dans l'exemple de la sauvegarde dans le nuage.

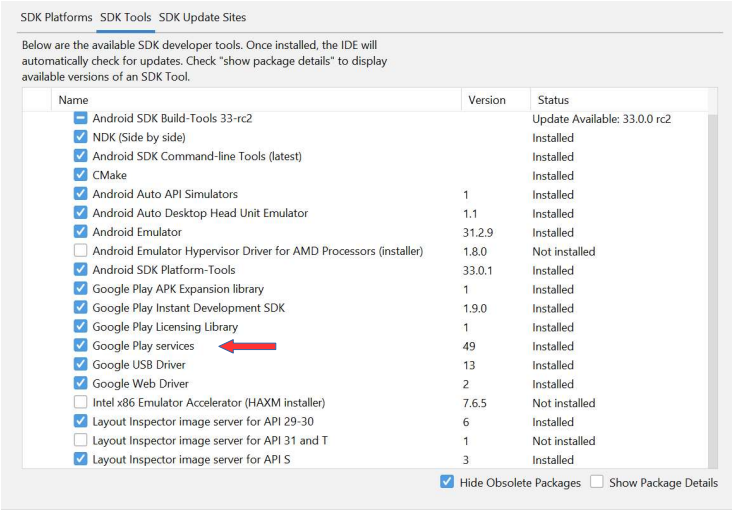
- L'API « Google Maps » est à sa 2e version. La première version a été déclarée officiellement désuète en décembre 2012. Et du coup, depuis mars 2013, il n'est plus possible d'obtenir une nouvelle clé pour utiliser la première version de l'API. La nouvelle clé n'ouvre l'accès qu'à la seconde version de l'API.

- La nouvelle API a ses détracteurs. Il est, par exemple, plus possible d'activer l'accès à « Google Map » sans partager cet accès avec d'autres applications (« Google now » est un exemple). Ainsi donc, si je cherche le mot Pizza en pensant faire une nouvelle recette magique, une des applications peut me suggérer toutes les pizzerias dans mon coin.

- Utiliser « Google Map » dans l'émulateur est un casse-tête. Mais, depuis la version kitkat, il est possible de tester l'affichage des cartes à travers cette API. Pour ce faire, il y a une série d'étapes à suivre. Nous allons les décrire dans les prochains exemples.

- Les exemples qui utilisent « Google Map » nécessitent d'inclure la librairie « Google Play Service » dans les exemples.

- On commence par installer la librairie « Google Play Services » : démarrez le gestionnaire « Android SDK », dans la section « SDK Tools », cochez « Google Play services » puis installez les paquetages associés.



- Il est possible de créer un projet Google Map prêt à l'emploi. Pour cela, vous pouvez choisir « Google Maps Activity ». Il va rester à le compléter avec la clé permettant d'utiliser le service « Google Maps ».

- Dans ce qui suit, nous allons partir de zéro, avec une activité vide et la compléter avec « Google Maps ».

`Examiner l'exemple « C08 : MapViewActivite ».`

L'utilisation de la 2e version de « Google Maps » passe par les étapes suivantes :

- Création d'un nouveau projet : pour tester cette API, vous allez créer un nouveau projet avec les paramètres par défaut, en lui incluant une activité. Soit « MapViewActivite » le nom de ce projet. Prenez note que ce projet fait partie du paquetage « ca.umontreal.iro.ift1155.mapviewactivite ». Choisissez la dernière API disponible.

- Il n'est pas conseillé d'utiliser toute la librairie « Google Play services », vu sa taille. Nous allons nous limiter à « play-services-maps ».

- Vous allez inclure dans la section dépendances du fichier « build.gradle » du module « MapViewActivite.app » la dernière version des librairies « **com.google.android.gms:play-services-location:** »
  et « **com.google.android.gms:play-services-maps:** » ainsi que « **androidx.legacy:legacy-support-v4:** » vu que nous utilisons des fragments dans l'application.

- Nous allons ajouter aussi au début du même fichier ces deux plug-ins :

```
plugins {
    id 'com.android.application'
    id 'com.google.android.libraries.mapsplatform.secrets-gradle-plugin'
}
```


- Dans le fichier « build.gradle » du projet « MapViewActivite », nous allons ajouter les dernières versions de ces dépendances :

```
classpath 'com.android.tools.build:gradle:'
classpath 'com.android.library:com.android.library.gradle.plugin :'
classpath 'com.google.android.libraries.mapsplatform.secrets-gradle-plugin:secrets-gradle-plugin :'
```

Activation de la clé « Google Maps API » du projet

- On commence par obtenir la clé SHA1 : ouvrez un terminal (xterm ou cmd), et localisez le fichier « debug.keystore ». Il est situé normalement dans le compte de l'utilisateur sous le répertoire « .android ». Ce dernier contient aussi les images AVD. Par la suite, exécutez cette commande :

  Linux :

```
keytool -list -v -keystore ~/.android/debug.keystore -alias androiddebugkey -storepass android -keypass android
```

Windows :

```
keytool -list -v –keystore
"C:\Users\your_user_name\.android\debug.keystore" –alias
androiddebugkey -storepass android -keypass android
```

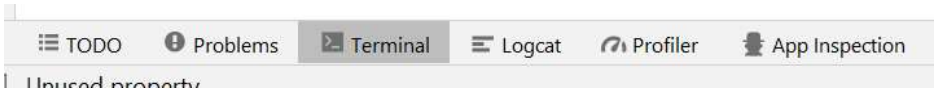
Si le système vous demande un mot de passe, tapez juste un retour de chariot. Vous allez obtenir ce qui suit :


```
Alias name: androiddebugkey
Creation date: Jan 01, 2013
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 4aa9b300
Valid from: Mon Jan 01 08:04:04 UTC 2013 until: Mon Jan 01 18:04:04
PST 2033
Certificate fingerprints:
MD5:  AE:9F:95:D0:A6:86:89:BC:A8:70:BA:34:FF:6A:AC:F9
SHA1: BB:0D:AC:74:D3:21:E1:43:07:71:9B:62:90:AF:A1:66:6E:44:5D:75
Signature algorithm name: SHA1withRSA
Version: 3
```

Notez la clé « SHA1 ».

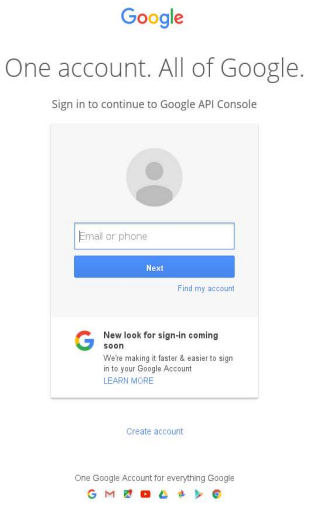
- Il est possible aussi d'obtenir directement cette information à partir d'Android Studio, comme suit :

  Ouvrir un terminal et taper la commande « ./gradlew signingReport ».



  Ou bien dans Android Studio, sélectionner la fenêtre « Gradle » située en à droite de votre interface. Par la suite, cliquer sur l'éléphant afin d'exécuter une nouvelle requête gradle. Puis taper la commande « gradle signingReport ».

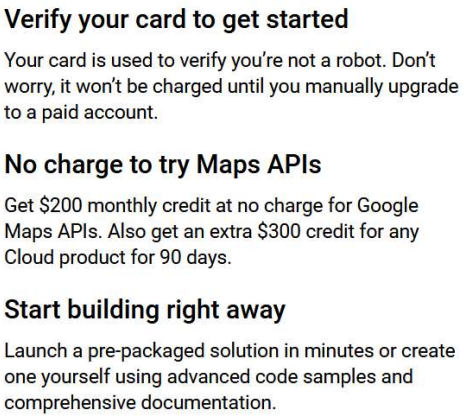
- Création d'un compte Google : ouvrez un nouveau compte chez Google, si vous n'en avez pas déjà un.



- Génération de clé Google : à l'aide de votre compte Google, connectez-vous au site

https://console.cloud.google.com/project/_/google/maps-apis/credentials

La création d'un compte nécessite maintenant l'association d'une carte de crédit. Après l'ajout de la carte de crédit, vous avez droit quand même à un crédit gratuit d'utilisation.



https://developers.google.com/maps/billing-and-pricing/billing?hl=fr#maps-product

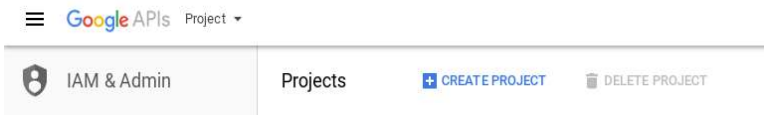
Premier compte

Si le premier compte de facturation Cloud que vous créez est utilisé pour un projet dans lequel les API ou les SDK Google Maps Platform sont activés, l'essai à hauteur de 300 USD de Google Cloud Platform et le crédit mensuel récurrent de 200 USD de Google Maps Platform s'appliquent tous les deux.

Le principe est le suivant : pendant l'essai, les frais sont d'abord déduits du crédit mensuel récurrent de 200 USD de Google Maps Platform. Si les frais dépassent 200 USD au cours d'un mois donné, le montant excédentaire est déduit du montant restant sur les 300 USD de l'essai de Google Cloud Platform.

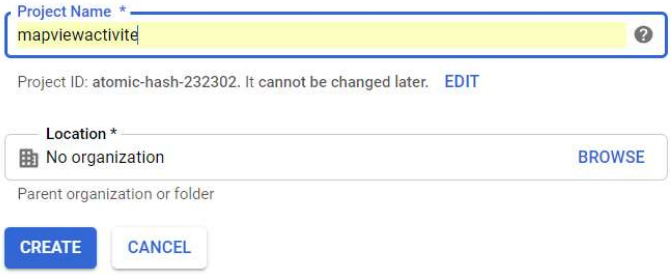
Comme indiqué ci-dessus, au plus tard à la date de fin de l'essai, vous devez mettre à niveau votre premier compte de facturation Cloud vers un compte payant. Une fois la mise à niveau effectuée, le crédit mensuel de 200 USD continuera à être appliqué à votre compte de facturation Cloud, même après la fin de l'essai.

- Puis cliquez sur « Create project… » ou sur « Project » puis « Create project » :

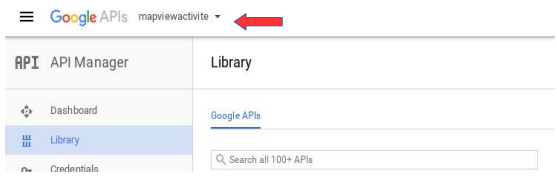


Attention, le nombre de projets (gratuits) dont vous avez droit est limité.

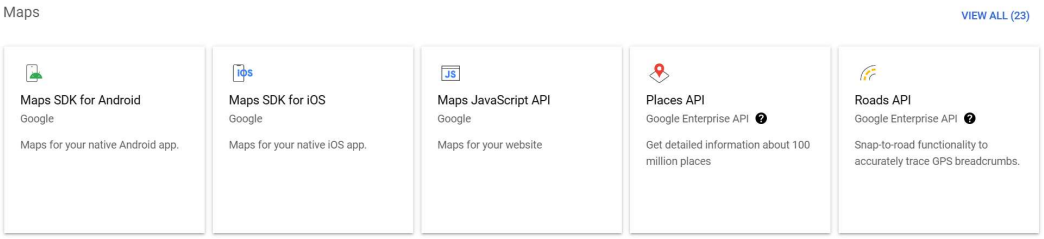
Compléter le nom du projet et cocher que vous êtes en accord avec les droits d'utilisation, puis cliquer sur « Create » :



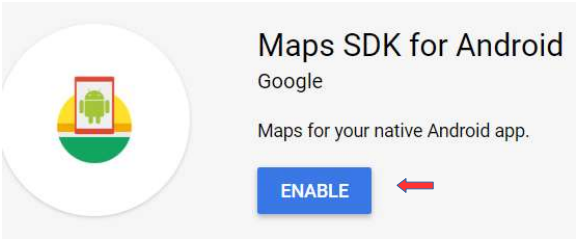
- Il faut vous assurer que le projet créé est bien sélectionné :



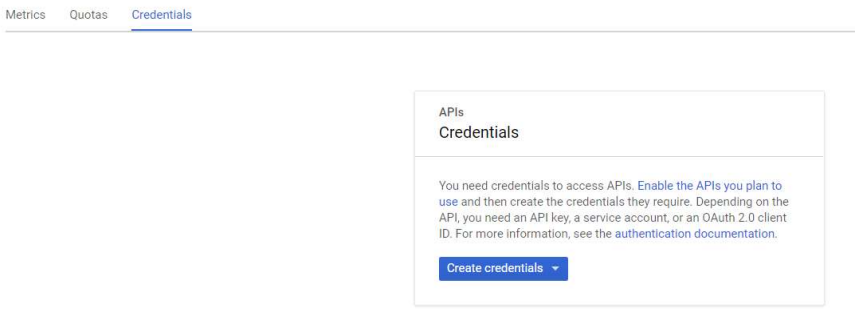
- Dans le menu de gauche, sélectionner « Library », puis cliquer sur « Google Maps Android API » dans la colonne « Google Maps APIs » :



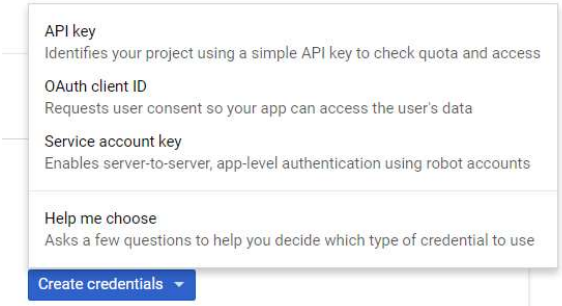
- Cliquer par la suite sur « Enable » :



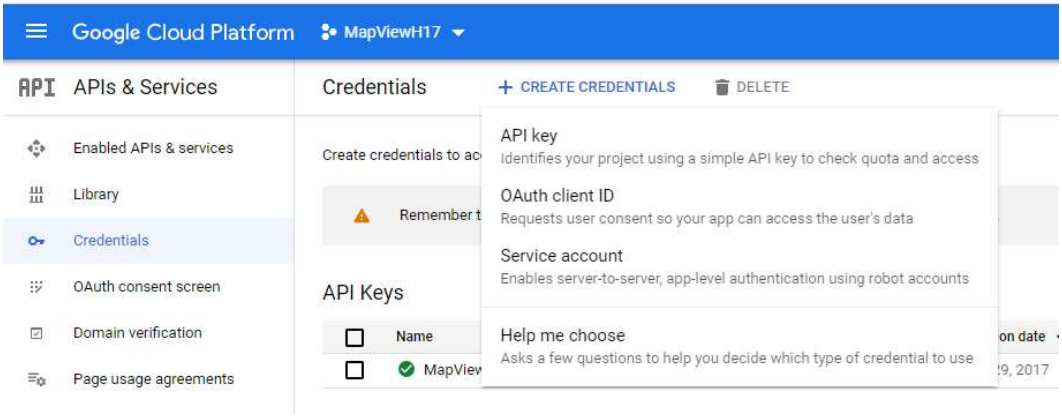
- Comme il est indiqué, il faut maintenant créer les autorisations nécessaires pour pouvoir utiliser cette API. On peut cliquer soit sur « Credentials » dans le menu :



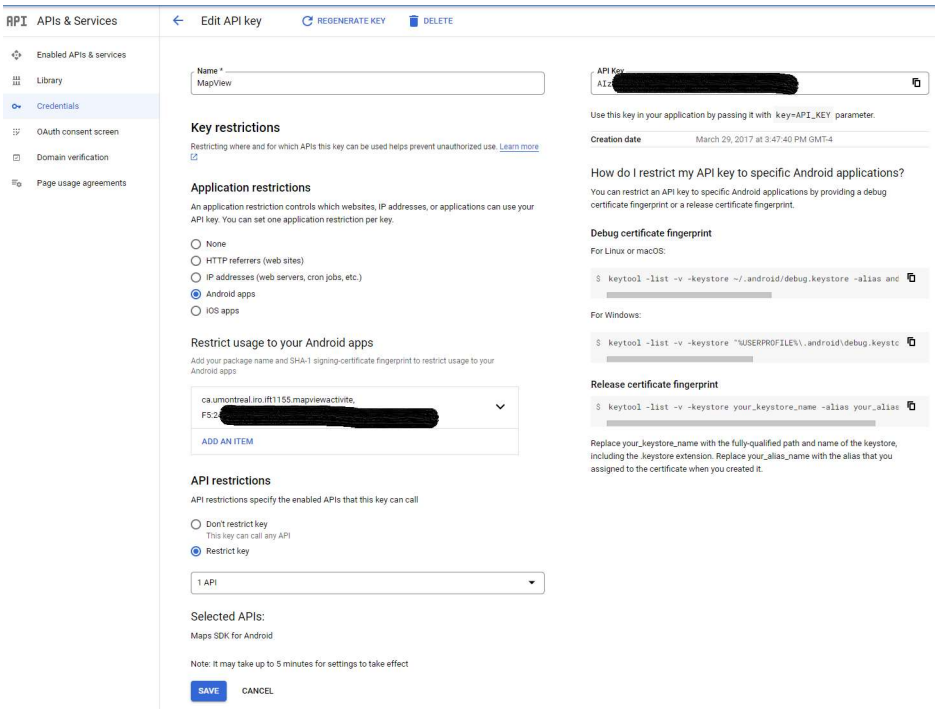
- Cliquer sur « Create credentials », vous allez obtenir les différentes options disponibles :



- Choisir « API key » :



- Compléter les différents champs :



Donner un nom à la clé dans le champ « Name », ici « MapView ».

Nous allons ajouter des restrictions à cette clé pour mieux contrôler son usage.

Dans « Application restrictions », cocher « Android apps ».

Dans « Restrict usage to your Android apps », ajouter un item. L'item va représenter le nom du paquetage associé à votre application (ici « ca.umontreal.iro.ift1155.mapviewactivite ») et la clé « SHA-1 » générée précédemment.

Dans « API restriction », cocher « Restrict key » et choisir « Maps SDK for Android ».

Finalement, noter la clé dans le champ « API Key ».

Il faut sauvegarder le tout et comme il est mentionné durant cette étape, il faut attendre quelques minutes avant de pouvoir utiliser la nouvelle clé.

- Se positionner maintenant dans votre fichier « AndroidManifest.xml » et ajouter ce segment dans « application » :

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="${MAPS_API_KEY}" />
```

- Dans le fichier « local.properties », ajouter cette ligne avec la clé obtenue précédemment :

```
MAPS_API_KEY =AIz....
```

- Cette manière de disposer la clé permet de transmettre le projet tout en masquant la clé d'accès.

- Déployer l'application sur un émulateur ayant « Google API ».

- Sauf que parfois ça ne marche pas …. examiner les logs dans Android Studio pour trouver les raisons.

- Voici un exemple de message si vous n'obtenez pas la carte :

2022-04-01 19:39:57.588 1543-1637/ca.umontreal.iro.ift1155.positionmap E/Google Maps Android API: Authorization failure. Please see https://developers.google.com/maps/documentation/android-api/start for how to correctly set up the map.
2022-04-01 19:39:57.590 1543-1637/ca.umontreal.iro.ift1155.positionmap E/Google Maps Android API: In the Google Developer Console (https://console.developers.google.com)
Ensure that the "Google Maps Android API v2" is enabled.
Ensure that the following Android Key exists:
API Key: =AIz...
Aiz...
Android Application (<cert_fingerprint>;<package_name>): F5:...;ca.umontreal.iro.ift1155.positionmap

Il y avait un « = » de trop!

- Par la suite, déployer votre application. Vous allez obtenir ce qui suit :

- Nous n'avions pas besoin d'ajouter des permissions dans l'application!

- Le fichier manifeste de l'activité va contenir ce qui suit :

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/the_map"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:name="com.google.android.gms.maps.MapFragment" />
```

Le code fait référence à un fragment d'une carte. Nous allons identifier cette carte afin de l'utiliser directement à partir du code Java. Nous pouvons inclure une panoplie d'options dans le fichier XML, comme nous pouvons les définir à partir du code Java.

- Votre activité va contenir ce qui suit :

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    //…
}
```

- Si vous déployez l'application sur un émulateur, assurez-vous d'utiliser un émulateur avec l'API : « Google APIs (Google Inc.) ».

Ajouter des marqueurs

Examiner l'exemple « C08 : PositionMap ».

- On ajoute les mêmes informations dans les différents fichiers de configuration comme pour l'exemple de « MapViewActivite ».

- On ajoute aussi ces deux permissions dans le fichier manifeste de l'application :

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

- Par la suite, on récupère la carte déclarée dans le fichier XML de l'activité :

```
SupportMapFragment mapFragment =
  (SupportMapFragment)
      getSupportFragmentManager().findFragmentById(R.id.map);

if (mapFragment!=null)
      mapFragment.getMapAsync(this);
```


On déclare une coordonnée, on la positionne sur la carte et on lui ajoute un marqueur avec quelques informations.

```
LatLng sfLatLng = new LatLng(45.500910, -73.615656);
googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
googleMap.addMarker(new MarkerOptions()
        .position(sfLatLng)
        .title("Pavillon André-Aisenstadt")
        .snippet("DIRO!")
        .icon(BitmapDescriptorFactory.defaultMarker(
                        BitmapDescriptorFactory.HUE_AZURE)));
```

On permet à l'utilisateur certaines actions sur la carte

```
googleMap.getUiSettings().setCompassEnabled(true);
googleMap.getUiSettings().setZoomControlsEnabled(true);
googleMap.getUiSettings().setMyLocationButtonEnabled(true);
```

On fixe le zoom :

```
float cameraZoom = 15;
```

La valeur « 1 » correspond à une vue de la terre entière.

On fixe la nature de la vue par défaut

```
googleMap.setMapType(mapType);
googleMap.animateCamera(CameraUpdateFactory.newLatLngZoom(
                                    cameraLatLng, cameraZoom));
```

On déclare un menu permettant de choisir les différents types d'affichage :

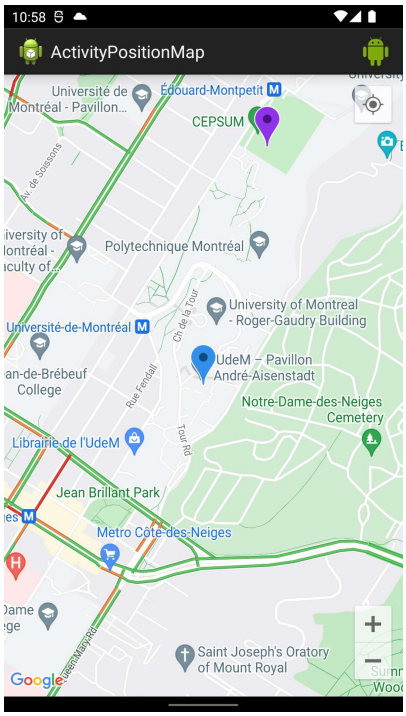
```
if(item.getItemId() == R.id.normal_map)
      mapType = GoogleMap.MAP_TYPE_NORMAL;
else if (item.getItemId() == R.id.satellite_map)
      mapType = GoogleMap.MAP_TYPE_SATELLITE;
else if (item.getItemId() == R.id.terrain_map)
      mapType = GoogleMap.MAP_TYPE_TERRAIN;
else if (item.getItemId() == R.id.hybrid_map)
      mapType = GoogleMap.MAP_TYPE_HYBRID;
```


On sauvegarde l'état de l'activité en cas de rotation par exemple :

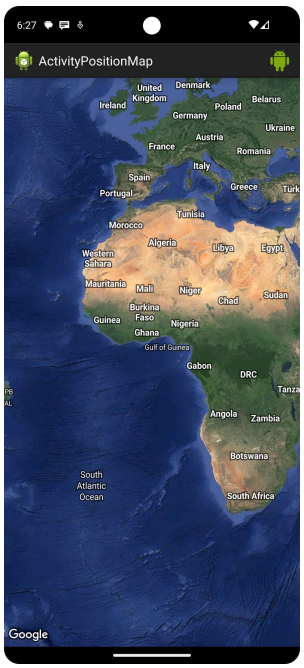
```
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);

    LatLng cameraLatLng = googleMap.getCameraPosition().target;
    float cameraZoom = googleMap.getCameraPosition().zoom;
    outState.putInt("map_type", mapType);
    outState.putDouble("lat", cameraLatLng.latitude);
    outState.putDouble("lng", cameraLatLng.longitude);
    outState.putFloat("zoom", cameraZoom);
}
```

- Si vous déployez l'application sur un émulateur, assurez-vous de :
    - Il faut utiliser un émulateur avec l'API : « Google APIs (Google Inc.) ».

- Vous allez obtenir ce qui suit :



Si l'application a de la difficulté à obtenir les autorisations ou les permissions nécessaires, vous allez obtenir ce type de carte :



Utilisation Open Street Map

- « Open Street Map » est une alternative à « Google Map ».

Open Street Map
http://www.openstreetmap.org/

osmdroid-android
https://github.com/osmdroid/osmdroid

slf4j-android (pour les logs)
http://www.slf4j.org

- L'application peut s'exécuter sans problème dans un émulateur.

- Il faut inclure les dernières versions des librairies « osmdroid » et « slf4j » dans le fichier « gradle » associé à l'application :

```
dependencies {
    implementation 'androidx.legacy:legacy-support-v4:
    implementation 'org.osmdroid:osmdroid-android:'
    implementation 'org.slf4j:slf4j-android:'
}
```

En date d'aujourd'hui, les versions disponibles sont :

```
implementation 'androidx.legacy:legacy-support-v4:1.0.0'
implementation 'org.osmdroid:osmdroid-android:6.1.18'
implementation 'org.slf4j:slf4j-android:1.7.36'
```

`Examiner l'exemple « C08 : OpenStreetMap ».`

- N'oubliez pas d'ajouter les permissions nécessaires dans le fichier « Android Manifest » :

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    tools:ignore="ScopedStorage" />
```

La dernière permission est critique. Nous avons ignoré l'erreur afin de pouvoir déployer l'application sur un émulateur.

- La déclaration et l'utilisation de la carte se font grosso modo de la même manière que dans le cadre d'une carte du type Google Map.

- On déclare une carte dans le fichier XML de l'activité :

```
<org.osmdroid.views.MapView
    android:id="@+id/openmapview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
```

- La carte est par la suite traitée dans le code Java :

```
Configuration.getInstance().setUserAgentValue(BuildConfig.APPLICATION_ID);
MapView myOpenMapView = findViewById(R.id.openmapview);
myOpenMapView.getZoomController().setVisibility(CustomZoomButtonsController.Visibility.NEVER);
myOpenMapView.setMultiTouchControls(true);
IMapController myMapController = myOpenMapView.getController();
myMapController.setZoom(4.0);
```

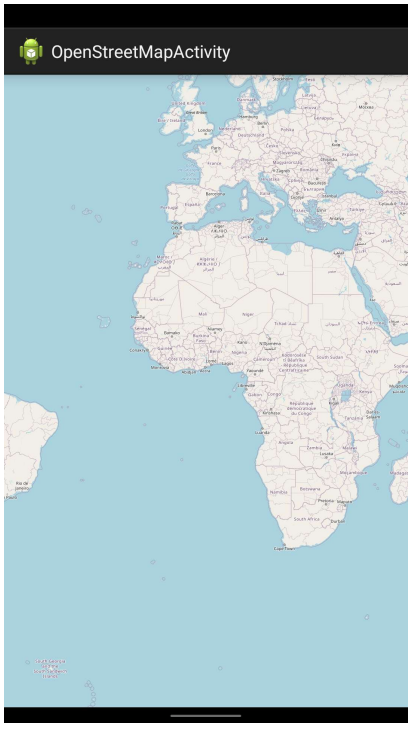
- L'accès au serveur qui héberge les cartes est soumis à des règles strictes. Il vous faut utiliser, un autre hébergeur de cartes. Pour ce faire, vous devez ajouter cette ligne dans le programme :

```
myOpenMapView.setTileSource(TileSourceFactory.MAPNIK);
```

https://operations.osmfoundation.org/policies/tiles/

- Il n'est pas nécessaire d'utiliser un émulateur configuré avec l'API de Google vu qu'on n'utilise pas la cartographie de Google.

- Le résultat obtenu est comme suit :



Bibliographies

Chapitre 4, Telephony, Android Application Development Cookbook: 93 Recipes for Building Winning Apps, Wei-Meng Lee.

Chapitre 17, Téléphonie et SMS, Android 4, Développement d'Applications Avancées, Reto Meier.

Exemples d'envoi de SMS
http://www.mkyong.com/android/how-to-send-sms-message-in-android/

Exemple de réception d'un SMS
Matos, Cleveland University

Chapitre 23, Android Notifications, Matos, C.U, [Attention : ces notifications sont pour une API 10 et -].

Chapitre 08, Personnalisation et gestion des événements, Notifications, Android 4, Nazim Benbourahla.

Notifications dans Nougat (liens non actifs, voir sur https://web.archive.org/)
https://willowtreeapps.com/ideas/mobile-notifications-part-1-a-gateway-to-app-engagement

https://willowtreeapps.com/ideas/mobile-notifications-part-2-some-useful-android-notifications

Working with Google Maps (examiner les 3 liens)
http://mobile.tutsplus.com/tutorials/android/android-sdk-working-with-google-maps-application-setup/

Google Maps Android API v2
https://developers.google.com/maps/documentation/android/?hl=fr

L'exemple PositionMap
https://github.com/codebybrian/mapsample

Obtenir Longitude et Latitude
https://getlatlong.net/

2 vidéos en rapport avec Gogle Maps v2
(ANG) : http://youtu.be/awX5T-EwLPc
(ESP) : http://youtu.be/mRnEkP2q4fo

Source de l'exemple sur « OpenStreetMap » avec Android :
https://github.com/osmdroid/osmdroid/wiki

Using OpenStreetMap
https://github.com/IanDarwin/Android-Cookbook-Examples/tree/master/OSMIntro

et

https://github.com/osmdroid/osmdroid/wiki