INFO0306 - Programmation mobile

Sommaire

- Rappels
- Persistence des données
- La connectivité réseau
- Les capteurs
- La localisation

Rappels

- Interface d'une applications Android
- Layout d'applications
  - Agent de placement
  - Fichiers XML
  - Exemples : *LinearLayout, RelativeLayout, TableLayout, ScrollView*
- Composants graphiques d'une application
  - ***TextView***
  - ***Button***
  - ***ImageView*** & ***ImageButton***
  - ***ETc.***

Persistence des données

Persistence des données

Principe

Options de stockage	Description
Préférences partagées	- Enregistre les paramètres utilisateurs. (Préférences) - Stocker des données de type primitif. - Mécanisme de sauvegarde de clé/valeur.
Stockage de fichiers	- En interne - Pour la sauvegarde de données privées - Accessible uniquement à l'application
	- En externe - Pour la sauvegarde de données public - Utilisé pour des données conséquentes. - Accessible à tous.
Bases de données (Cf Partie 2)	- SQLite - Stockage de données structurées
Web	- Stockage des données sur Internet - Usage de java.net.* / android.net.*

Persistence des données

Les préférences partagées

- Objectif :
  - sauvegarder les préférences utilisateurs
  - exemples : notifications, thème, son, etc.
- La classe :
  - La classe *SharedPreferences* fournit tout ce qu'il faut pour sauvegarder ou récupérer des informations clés/valeurs de types standard.
  - Il sera possible de sauvegarder les types suivants et ceci même si votre application est arrêtée ou tuée par le système : *boolean*, *int*, *float*, *long* et *string*.
  - De plus ces préférences sont accessibles depuis plusieurs composants au sein d'une même application.

Persistence des données

Les préférences partagées

Méthodes	Description
getSharedPreferences (String name, int mode)	- A utiliser si plusieurs fichiers de préférences. - Le premier paramètre indique le nom du fichier
getPreferences (int mode)	- A utiliser si un seul fichier de préférences par activité est utilisé. - *Activity.getSharedPreferences(int).*
getDefaultSharedPreferences (Context context)	- *PreferencesManager.getSharedPreferences(String,int)* - Un moyen simple d'avoir accès à un static SharedPreferences.
Valeur du paramètre (int mode)	**Description**
Context.MODE_PRIVATE	- Le fichier n'est accessible que par l'application qui l'a créé.
Context.MODE_WORLD_READABLE	- Le fichier peut être lu par n'importe quelle application.
Context.MODE_WORLD_WRITEABLE	- Le fichier peut être lu et modifier pas n'importe quelle application.

Persistence des données

Modifications

- Les modifications apportées aux préférences partagées se gèrent avec un objet de type *SharedPreference.Editor*. Il est possible de le récupérer avec la méthode *edit() :*

```
SharedPreferences pref = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
SharedPreferences.Editor editor = pref.edit();
```

- L'ajout d'un couple clé/valeur s'effectue avec la méthode *putX(String cle, X valeur)* où *X* correspond à un type accepté *:*

```
editor.putBoolean("notification", true);
```

- Il est également possible de supprimer une ou toutes les préférences avec les méthodes respectives *removeString(String cle)* et *clear() :*

```
editor.remove("notification");
editor.clear();
```

- La dernière chose à faire est d'envoyer ces nouveaux paramètres au manager à l'aide de *commit() :*

```
editor.commit();
```

Pers

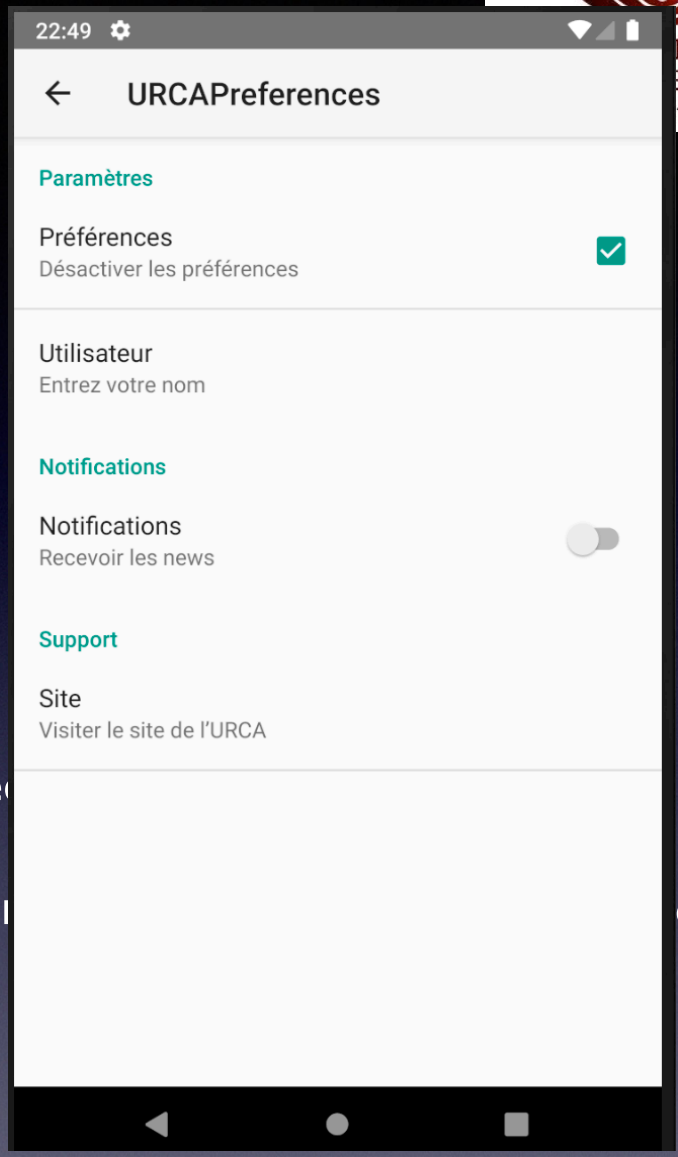
sateurs

- L'
- P

déf
ave

être géré
ces.xml.
veau fichier
ce

```
<PreferenceScreen xmlns:android="http://
schemas.android.com/apk/res/android" >
   <PreferenceCategory android:title="Paramètres" >
      <CheckBoxPreference
         android:defaultValue="true"
         android:key="preference"
         android:summaryOff="Activer les préférences"
         android:summaryOn="Désactiver les préférences"
         android:title="Préférences" />
      <EditTextPreference
         android:defaultValue="Billie Jean"
         android:dependency="preference"
         android:key="utilisateur"
         android:summary="Entrez votre nom"
         android:title="Utilisateur" />
   </PreferenceCategory>
   <PreferenceCategory android:title="Notifications" >
      <SwitchPreference
         android:key="notification"
         android:summary="Recevoir les news"
         android:title="Notifications" />
   </PreferenceCategory>
   <PreferenceCategory android:title="Support" >
      <Preference
         android:key="support"
         android:summary="Visiter le site de l'URCA"
         android:title="Site" >
         <intent
            android:action="android.intent.action.VIEW"
            android:data="https://www.univ-reims.fr" />
      </Preference>
   </PreferenceCategory>
</PreferenceScreen>
```




```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
   <boolean name="notification" value="false" />
   <string name="utilisateur">Marwane Ayaida</string>
   <boolean name="preference" value="true" />
</map>
```

Persistence des données

Les fichiers : Stockage interne

- Pour la gestion des fichiers, Android se base principalement sur les classes classiques de Java en simplifiant la gestion des permissions. Soit le fichier sera stocké dans le répertoire de l'application, soit à l'extérieur.
- Usage interne :
  - Dans : */data/data/<package>/files/FILE*
  - Sans autorisation
  - Suppression avec l'application
  - Eviter les fichiers volumineux

Persistence des données

Les fichiers : Stockage interne

- Lecture fichier : classique pour *JAVA*

```
FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(urca.getBytes());
fos.close();
```

- Ecriture fichier : classique pour *JAVA*

```
File fichier = new File(getFilesDir() + "/" + FILENAME);
byte fileContent[] = new byte[(int) fichier.length()];
fis.read(fileContent);
String content = new String(fileContent);
```

- Suppression fichier :

```
deleteFile(FILENAME)
myFile.delete();
```

Persistence des données

Les fichiers : Stockage externe

- Stockage externe :
  - Stockage partagé
  - Pas nécessairement sur carte SD
  - Lisibles par toutes les applications
  - Modifiables par toutes les applications
- Permissions :

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

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

- Vérification de la disponibilité du stockage externe :

```
String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state)) { ... }
```

- Récupération du stockage externe :

```
File externalDirectory = Environment.getExternalStorageDirectory();
```

Persistence des données

Les fichiers : Stockage externe

Constante	Description
DIRECTORY_ALARMS	- Emplacement des fichiers audio des alarmes.
DIRECTORY_DCIM	- L'emplacement initial pour les photos/vidéos provenant de l'appareil photo.
DIRECTORY_DOCUMENTS	- Emplacement des documents - Equivalent au « Mes documents » de Windows.
DIRECTORY_DOWNLOADS	- Emplacement des fichiers téléchargés. - Info : /Download et non pas /Downloads
DIRECTORY_MOVIES	- Emplacement des vidéos
DIRECTORY_MUSIC	- Emplacement des musiques
DIRECTORY_NOTIFICATIONS	- Emplacement des fichiers audio des notifications.
DIRECTORY_PICTURES	- Emplacement des photos/images
DIRECTORY_PODCASTS	- Emplacement des fichiers audio des podcats
DIRECTORY_RINGTONES	- Emplacement des sonneries.

Persistence des données

Les bases de données

- Objectif :
  - Sauvegarde d'éléments ordonnés
  - Recherche rapide et efficace
- Solution :
  - Android propose la possibilité de mise en oeuvre d'une *BD SQLite*
  - Choix du *SQLite* : *SGBD* très compact et s'avère très efficace (empreinte mémoire est réduite)
  - BD par application localisée : */data/data/<package>/databases/FILE_BD*
- Comment ça marche?
  - *BD* est un objet de la classe *android.database.sqlite.SQLiteDatabase*
    - Méthodes : *CREATE TABLE*, *SELECT*, *INSERT*, *UPDATE*, *DELETE*, etc.

Persistence des données

Les bases de données

- Création et mise à jour *BD* : classe *SQLiteOpenHelper*

```
public class UrcaDBHelper extends SQLiteOpenHelper {
        public UrcaDBHelper(Context context, String name, CursorFactory factory,
                        int version) {
                super(context, name, factory, version);
        }
        @Override
        public void onCreate(SQLiteDatabase db) {
                db.execSQL(Student.CREATE_TABLE);
                Log.i("create db", "ok");
        }
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
                db.execSQL(Student.DELETE_TABLE);
                onCreate(db);
        }
        @Override
        public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
                super.onDowngrade(db, oldVersion, newVersion);
                onUpgrade(db, oldVersion, newVersion);
        }
}
```

- Accès *BD* :

```
SQLiteDatabase db = mHelper.getReadableDatabase();

SQLiteDatabase db = mHelper.getWritableDatabase();
```

Persistence des données

Les bases de données

- Ajouter un élément de *BD* en *SQL* avec la méthode *INSERT* :

```
INSERT INTO nom_de_la_table (nom_de_la_colonne1, …) VALUES (valeur1, …) ;
```

- Ajouter un élément de *BD* en *JAVA* avec la méthode *insert(String table, String nullColumnHack, ContentValues values)* retournant le numéro de la ligne insérée :

Paramètre	Description
String table	- L'identifiant de la table dans laquelle insérer l'entrée.
String nullColumnHack	- Dans le cas d'une entrée vide, il faut indiquer le nom d'une des colonnes de la table avec la valeur null.
ContentValue values	- Un objet représentant l'objet à insérer.

```
long val = db.insert(TABLE_NAME, KEY_NAME, values);
```

- L'objet *ContentValue* : utilisé pour insérer une entrée dans la base.
  - Pour ajouter des valeurs à cet objet on utilise les méthodes *put()* :

```
ContentValues values = new ContentValues();
values.put(KEY_NAME, "AYAIDA");
values.put(KEY_FIRSTNAME, "Marwane");
```

Persistence des données

Les bases de données

- Supprimer un élément de *BD* en *JAVA* avec la méthode *delete(String table, String whereClause, String[] whereArgs)* retournant le numéro de la ligne supprimée :

Paramètres	Description
String table	- Le nom de la table
String whereClause	- Correspond au WHERE en SQL - Exemple : « id = ? »
String[] whereArgs	- Tableau de valeurs remplaçant les ? de whereClause

```
int val = db.delete(TABLE_NAME, _ID + " = ?", new String[] {String.valueOf(id)});
```

- Mettre à jour un élément de BD en *JAVA* avec la méthode *update(String table, ContentValues values, String whereClause, String[] whereArgs)* :

```
ContentValues value = new ContentValues();
value.put(KEY_FIRSTNAME, student.getPrenom());
db.update(TABLE_NAME, value, _ID  + " = ?", new String[] {String.valueOf(student.getId())});
```

Persistence des données

Les bases de données

- Sélectionner un élément de *BD* en *SQL* avec la méthode *rawQuery()* :

```
String query = "SELECT * FROM " + TABLE_NAME + " WHERE " + _ID + " = ?";
    SQLiteDatabase db = open();
    Cursor cursor = db.rawQuery(query, new String[] {Long.toString(id)});
```

- Sélectionner un élément de *BD* en *JAVA* avec la méthode *query(boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)* retournant un *Cursor* :

Paramètre	Description
Boolean distinct	- Vrai si on ne souhaite pas de résultat en double.
String table	- Le nom de la table
String[] columns	- Tableau de colonnes à sélectionner
String selection	- Identique à whereClause / WHERE en SQL - Mettre null si on souhaite séléctionner toutes les lignes
String[] selectionArgs	- Identique à whereArgs - Null si pas de ?
String having	- Identique HAVING en SQL - Indique les groupes de lignes à retourner
String groupBy	- Identique à GROUP BY en SQL - Permet de regrouper les résultats.
String orderBy	- Identique à ORDER BY en SQL - Permet de trier le résultat.
String limit	- Fixe le nombre maximum de résultat retourné.

Persistence des données

Les bases de données

- *Cursor* : représente les lignes contenant le résultat de la requête *SELECT*

Méthode	Type de retour	Description
moveToFirst()	Boolean	- On se positionne au début - Retourne false si le curseur est vide
moveToNext()	Boolean	- On passe à la ligne suivante - Retourne false si on est à la fin.
getCount()	Int	- Retourne le nombre de lignes dans le curseur.
getX **(int columnIndex)**	String, int, Long, Float, Short,etc.	- Récupère la valeur d'un champ suivant son index.
getColumnIndex (String nom)	Int	- Retourne l'index d'un champ suivant son nom.
isAfterLast()	Boolean	- Indique si le curseur est après la dernière ligne.
getColumnNames	String[]	- Récupère le nom des colonnes
close()	Void	- Libère les ressources occupées par le curseur.

```
String query = "SELECT * FROM " + TABLE_NAME;
    SQLiteDatabase db = open();
    Cursor cursor = db.rawQuery(query,null);
    if (cursor != null) cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
       Student s = new Student(cursor.getLong(0), cursor.getString(1), cursor.getString(2));
        cursor.moveToNext(); }
    cursor.close();
    db.close();
```

La connectivité réseau

La connectivité réseau

Les permissions

- Comme pour tout accès dans Android :
  - il faut demander l'autorisation d'accès au réseau avant
  - dans le fichier : ***AndroidManifest.xml***
  - Principales autorisations réseau :

```
<!-- Autoriser l'accès à Internet -->
<uses-permission android:name="android.permission.INTERNET" />

<!-- Autorise la vérification de l'état du réseau -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

La connectivité réseau

L'accès réseau (1/3)

- Avant de tenter une connexion réseau :
  - Il faut vérifier si une connexion réseau est disponible
  - Raisons d'absence de connexion :
    - Hors de portée d'un réseau mobile ou WIFI
    - L'utilisateur peut avoir désactivé le WIFI (Ex : Pour préserver la batterie)
    - L'utilisateur peut avoir désactivé les données mobile (Ex : Pour éviter un surcoût sur le forfait mobile).
    - Etc.
  - Il existe un gestionnaire de connexions réseau : ***ConnectivityManager***
    - Récupération : avec la méthode ***getSystemService()***
    - Comme tout le reste dans Android : les capteurs, le vibreur, l'alarme, la localisation, etc.

```
ConnectivityManager connectivityManager = (ConnectivityManager) getSystemService(CONNECTIVITY_SERVICE);
```

La connectivité réseau

L'accès réseau (2/3)

- La suite :
  - Appel à la méthode : ***getActiveNetworkInfo()***
    - permet de récupérer un objet ***NetworkInfo***
    - si objet ***null*** :
      - aucun réseau n'est disponible
      - annuler les accès réseau
      - prévenir l'utilisateur?

```
NetworkInfo networkInfo = connectivityManager.getActiveNetworkInfo();
```

  - Sinon appel à la méthode : ***getNetworkInfo()***
    - permet de vérifier l'état de chaque interface

```
NetworkInfo mobile = connectivityManager.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
NetworkInfo wifi = connectivityManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
```

La connectivité réseau

L'accès réseau (3/3)

- La suite :
  - On peut vérifier l'état du réseau maintenant :
    - ***isConnected()*** : réseau connecté
    - ***isConnectedOrConnecting()*** : réseau connecté ou en cours de connexion

```
boolean mobile_isConnected = mobile != null && mobile.isConnected();
boolean wifi_isConnected = wifi != null && wifi.isConnected();
```

```
if (wifi_isConnected || mobile_isConnected ) {
    // Disponible + Faire un travail
} else {
    // Non disponible + Prévenir l'utilisateur
}
```

La connectivité réseau

Surveillance du réseau (1/3)

- Objectifs de la surveillance réseau :
  - Privilégier le réseau WIFI plutôt que le réseau mobile :
    - Réseau mobile : lent + coûteux
  - Arrêter les téléchargements en Mobile et poursuivre en WIFI
  - Arrêter car plus de réseau tout simplement
  - Etc.
- —> il faut surveiller les différents changements de connectivité :
  - Le système averti les applications :
    - inscrites aux changements d'état du réseau
    - évènement : ***android.net.conn.CONNECTIVITY_CHANGE***
  - On utilise pour cela le ***BroadcastReceiver*** :
    - il faut instancier un récepteur d'intentions
    - il faut l'inscrire à cet évènement

La connectivité réseau

Surveillance du réseau (2/3)

- 2 méthodes de souscription :
  1. Dans le manifest :

```
<receiver
    android:name=".MyConnectionReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    </intent-filter>
</receiver>
```

  2. Dynamiquement dans l'activité en JAVA :

```
@Override
protected void onResume() {
    super.onResume();
    MyConnectionReceiver receiver = new MyConnectionReceiver();
    IntentFilter intentFilter = new IntentFilter("android.net.conn.CONNECTIVITY_CHANGE");
    registerReceiver(receiver, intentFilter);
}
@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
}
```

La connectivité réseau

Surveillance du réseau (3/3)

- Il reste à programmer le ***BroadcastReceiver***

```
public class MyConnectionReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        ConnectivityManager cm =
                (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
        boolean isConnected = activeNetwork != null &&
                activeNetwork.isConnectedOrConnecting();
        if (isConnected) {
            Toast.makeText(context, "Réseau connecté", Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(context, "Réseau changé ou reconnecté", Toast.LENGTH_LONG).show();
        }
    }
}
```

La connectivité réseau

Les pages Web

- Charger du HTML :
  - Possible dans un ***TextView***, mais limité (ex : pas d'image)
  - Il faut utiliser les ***WebView*** : gérées par ***WebKit***
  - Avec : ***loadDataWithBaseURL*** (***String Url***, ***String data***, ***String mimeType***, ***String encoding***, ***String historyUrl***)

```
WebView webview = (WebView) findViewById(R.id.webview);
webview.loadDataWithBaseURL(null, "<html>"
          + "<body>La connectivité réseau</body></html>", "text/html",
          "UTF-8", null);
```

- Charger une page Internet :
  - Il faut utiliser aussi les ***WebView***
  - Avec : ***loadURL*** (***String url***)
  - Pensez bien à indiquer l'adresse complète avec les ***http://*** et les ***www***

```
webview.loadUrl("https://www.android.com");
```

La connectivité réseau

Les Requêtes HTTP (1/3)

- Les clients ***HTTP*** dans Android :
  - ***HttpClient*** :
    - fourni par Apache
    - très utilisé pour les accès PC
    - adapté à Android par Apache
  - ***HttpURLConnection*** :
    - fourni par Android pour ***Gingerbread*** et plus (API 9, Android 2.3)
    - un client léger et adapté pour le mobile
    - privilégié par Android
- Les requêtes sur le réseau :
  - peuvent être longues (téléchargement fichier multimédia)
  - peuvent entrainer des retards imprévisibles
  - éviter de bloquer l'interface utilisateur (ANR)
  - il est conseillé de toujours effectuer ces opérations dans un thread séparé

La connectivité réseau

Les Requêtes HTTP (2/3)

- Etapes de connexion et de chargement :
  1. transformer la chaine de caractère en objet URL

```
URL url = new URL("https://www.android.com");
```

  2. ouvrir une connexion dessus avec ***openConnection()***

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
```

  3. vérifier si le serveur est accessible

```
if (conn.getResponseCode() == HttpURLConnection.HTTP_OK) {
    // Chargement infos
}
```

  4. récupérer le contenu du flux

```
InputStream is = conn.getInputStream();
```

La connectivité réseau

Les Requêtes HTTP (3/3)

- Les méthodes de la classe ***HttpURLConnection*** :

Méthodes	Description
setConnectTimeout(int)	- Fixe le temps maximum de la connexion - Exprimé en milliseconde
setReadTimeout(int)	- Fixe le temps maximum de chargement de la page - Exprimé en milliseconde
setUsesCaches (boolean)	- Usage ou non du cache
setRequestMode(String)	- Fixe la méthode HTTP à utiliser - GET/POST
setDoOutput(boolean)	- Autorise ou non les flux sortants - Utile quand on souhaite envoyer des informations vers un serveur.

Les capteurs

Les capteurs

Diversité des capteurs

- Un grand nombre d'appareils Android disposent de capteurs :
  - Mouvement
  - Orientation
  - Conditions environnementales
- Des informations avec une grande précision, proposer aux utilisateurs des :
  - nouvelles possibilités
  - de nouveaux services
- Capteurs divers et variés :
  - des accéléromètres
  - des gyroscopes
  - des capteurs de luminosité
  - des capteurs de température
  - des capteurs de pression
  - détecteur d'empreinte digitale, Etc.

Les capteurs

Prise en charge des capteurs

- Framework des capteurs dans Android :
  - Contient :
    - plusieurs classes
    - plusieurs interfaces
  - Permet :
    - détecter la présence ou non d'un capteur
    - reconnaitre ses capacités
    - recueillir ses données
- Types de capteurs :
  - capteurs matériels : physiquement présents sur l'appareil
  - capteurs logiciels (virtuels) : calculent leurs valeurs en fonction d'un ou plusieurs autres capteurs

Les capteurs

Liste des capteurs

Nom	Description et usage	Type
Accéléromètre	–TYPE_ACCELEROMETER –Mesure la force d'accélération sur les 3 axes (x,y,z). –Capable d'en déduire la force de gravitation en m/s². –Détecter les mouvements (secousse, inclinaison, etc.)	Matériel
Gyroscope	–TYPE_GYROSCOPE –Mesure la rotation sur les 3 axes (x, y, z) en rad/s. –Détecter l'orientation de l'appareil.	Matériel
Orientation	–TYPE_ORIENTATION –Mesure la rotation sur les 3 axes (x, y, z) en degrés. –Préférer l'usage de getOrientation() depuis API 8. –Détecter l'orientation de l'appareil.	Logiciel
Proximité	–TYPE_PROXIMITY –Mesure la proximité d'un objet en cm. –Détecter si l'utilisateur porte l'appareil à son oreille.	Matériel
Magnétomètre	–TYPE_MAGNETIC_FIELD –Mesure le champ géomagnétique sur les 3 axes en micro-telsa (µT). –Création d'une boussole	Matériel
Photomètre	–TYPE_LIGHT –Mesure le niveau de lumière ambiante en lux. –Contrôler la luminosité de l'écran.	Matériel
Baromètre	–TYPE_PRESSURE –Mesure la pression de l'air ambiant. –Surveiller le changement de pression atmosphérique.	Matériel
Thermomètre	–TYPE_TEMPERATURE / TYPE_AMBIANT_TEMPERATURE (>API 14) –Mesure la température en degré Celsius (°C). –Contrôler la température.	Matériel
Tout	–TYPE_ALL –Liste : http://developer.android.com/reference/android/hardware/Sensor.html	Matériel Logiciel

Les capteurs

Filtre Google Play Store

- Lors de la mise en ligne de l'application :
  - filtrer son affichage uniquement en présence de certains capteurs nécessaire à son bon fonctionnement
  - on utilise la balise ***<uses-feature>*** dans le manifest
  - exemple :

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

  - Les attributs :
    - name :
      - spécifie le nom du capteur
      - Liste dans le manifest : accéléromètre, baromètre, magnétomètre, gyroscope, photomètre et proximité
    - required :
      - spécifie la présence du capteur est indisponible ou pas
      - sinon : il faut désactiver les parties du capteur si non dispo

Les capteurs

Le Framework des capteurs

- Framework des capteurs dans Android fournit :
  - L'identification des capteurs et des capacités de détection
  - Surveiller les évènements des capteurs

Classe/interface	Description
SensorManager	–Création de l'instance du service du capteur –Accès aux capteurs, listings des capteurs –Constante spécifiant la précision du capteur et l'étalonnage –Permet l'ajout ou la suppression d'un écouteur d'événements.
Sensor	–Création d'un capteur spécifique (en fonction du capteur) –Permet de déterminer les capacités du capteur.
SensorEvent	–Création d'un événement de capteur –Informations sur un événement de capteur –Utiliser par le système pour publier les données de capteur –Fourni les données des capteurs, le type de capteur ayant généré l'événement, la précision des données et l'horodatage de l'événement.
SensorEventListener	–Interface disposant de deux méthodes de rappels ✓Une pour détecter le changement de précision ✓Une pour détecter le changement de valeurs

Les capteurs

Disp

Capteur	Android 4.0	Android 2.3	Android 2.2	Android 1.5
TYPE_ACCELEROMETER	Oui	Oui	Oui	Oui
TYPE_AMBIENT_TEMPERATURE	Oui	Non	Non	Non
TYPE_GRAVITY	Oui	Oui	Non	Non
TYPE_GYROSCOPE	Oui	Oui	Non[2]	Non[2]
TYPE_LIGHT	Oui	Oui	Oui	Oui
TYPE_LINEAR_ACCELERATION	Oui	Oui	Non	Non
TYPE_MAGNETIC_FIELD	Oui	Oui	Oui	Oui
TYPE_ORIENTATION	Oui[1]	Oui[1]	Oui[1]	Oui
TYPE_PRESSURE	Oui	Oui	Non[2]	Non[2]
TYPE_PROXIMITY	Oui	Oui	Oui	Oui
TYPE_RELATIVE_HUMIDITY	Oui	Non	Non	Non
TYPE_ROTATION_VECTOR	Oui	Oui	Non	Non
TYPE_TEMPERATURE	Oui[1]	Oui	Oui	Oui

Légende

Dispo

Non Dispo

Dispo déprécié

Dispo 1.5 utilisable 2.3

Les capteurs

Identifier les capteurs

1. Récupérer le manager des capteurs

```
sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
```

2. Récupérer la liste des capteurs ou un capteur en particulier

```
List<Sensor> sensorList = sensorManager.getSensorList(Sensor.TYPE_ALL);
Sensor accelerationSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

3. vérifier si le capteur est disponible

```
if (accelerationSensor!=null) {
    // un accéléromètre est disponible
}
else{
    // aucun accéléromètre n'est disponible
}
```

Les capteurs

Surveiller les capteurs (1/3)

- Afin de surveiller les capteurs :
  - On utilise l'interface : ***SensorEventListener***
    - Méthodes :
      - ***onAccuracyChanged()***
      - ***onSensorChanged()***

Méthode	Description
onAccuracyChanged (Sensor, int)	-Appelée quand la précision du capteur change. -*Sensor* : Capteur -*int* : précision du capteur
onSensorChanged (SensorEvent)	-Appelé quand il y a un événement sur le capteur -*SensorEvent* : un événement ➡*accuracy* pour indiquer la précision de la mesure ➡*sensor* comme référence au capteur ayant pris la mesure ➡*timestamp* pour l'instant de prise de mesure ➡*values*, la plus importante, pour les valeurs (tableau d'entiers)

Les capteurs

Surveiller les capteurs (2/3)

- La précision du capteur peut prendre :
  - ***SensorManager.SENSOR_STATUS_ACCURACY_LOW*** : Faible précision.
  - ***SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM*** : Précision moyenne.
  - ***SensorManager.SENSOR_STATUS_ACCURACY_HIGH*** : Précision maximale.
  - ***SensorManager.SENSOR_STATUS_ACCURACY_UNRELIABLE*** : il ne faut pas faire confiance à ce capteur par manque de fiabilité des données ou dû à un mauvais étalonnage par exemple on utilisera.
- Il faut déclarer quel capteur nous voulons surveiller :

```
@Override
protected void onResume() {
    super.onResume();
    sensorManager.registerListener(accelerationEventListener, accelerationSensor,
SensorManager.SENSOR_DELAY_UI);
}
@Override
protected void onPause() {
   super.onPause();
   sensorManager.unregisterListener(accelerationEventListener);
}
```

Les capteurs

Surveiller les capteurs (3/3)

- Délai de rafraîchissement :
  - Entier qui peut prendre ces valeurs :

Constante	Description
SENSOR_DELAY_NORMAL	-Rafraichissement normal -0,2 seconde
SENSOR_DELAY_UI	-Rafraichissement lent -0,6 seconde -Convient bien aux interfaces utilisateurs
SENSOR_DELAY_GAME	-Rafraichissement rapide -0,02 seconde -Convient bien aux jeux
SENSOR_DELAY_FASTEST	-Rafraichissement le plus rapide possible -0 seconde -Consommatrice d'énergie

Les capteurs

Système de coordonnées des capteurs (1/2)

- Un ***SensorEvent*** indique les données des capteurs :
  - un système de coordonnées standard à 3 axes, où ***X***, ***Y*** et ***Z*** sont représentés par ***values[0]***, ***values[1]*** et ***values[2]***
  - Pour certains capteurs ne délivrant qu'une seule information :
    - seule ***values[0]*** est utilisée
    - exemple : lumière, température, proximité, pression, etc.

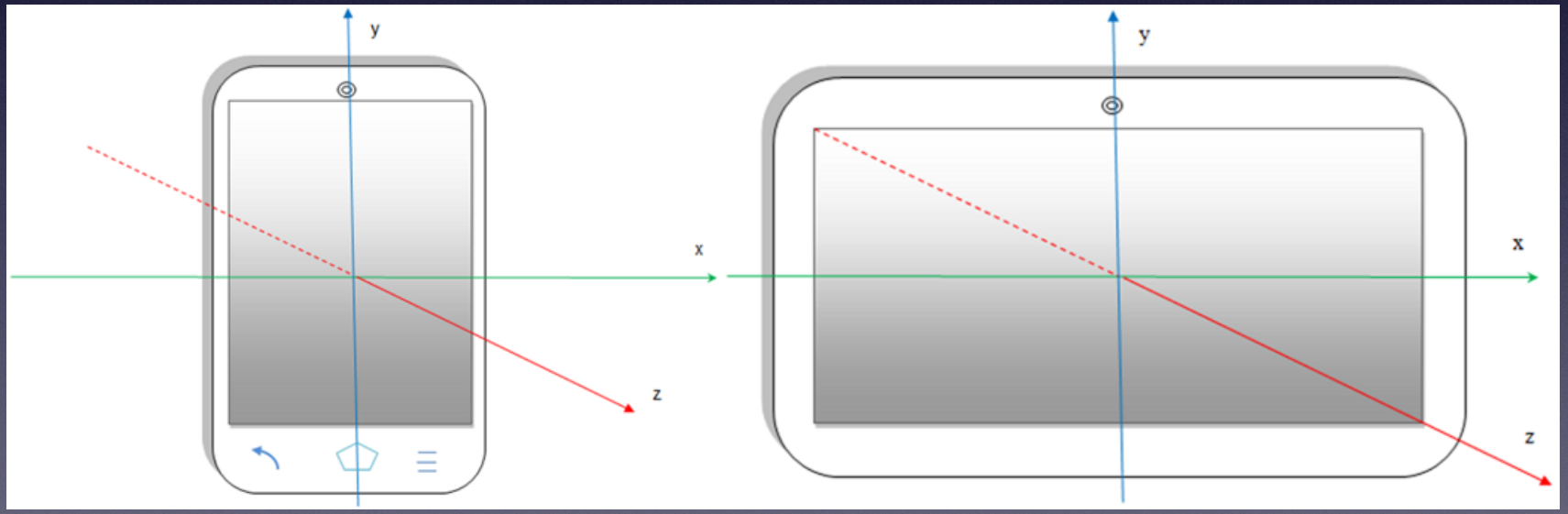
```
private SensorEventListener accelerationEventListener = new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent event) {
        float[] values = event.values;
        float Ax = values[0];
        float Ay = values[1];
        float Az = values[2];
        // traiter les données
    }
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {

    }
};
```

Les capteurs

Système de coordonnées des capteurs (2/2)

- Orientation :
  - Smartphone : mode portrait
  - Tablette : mode paysage

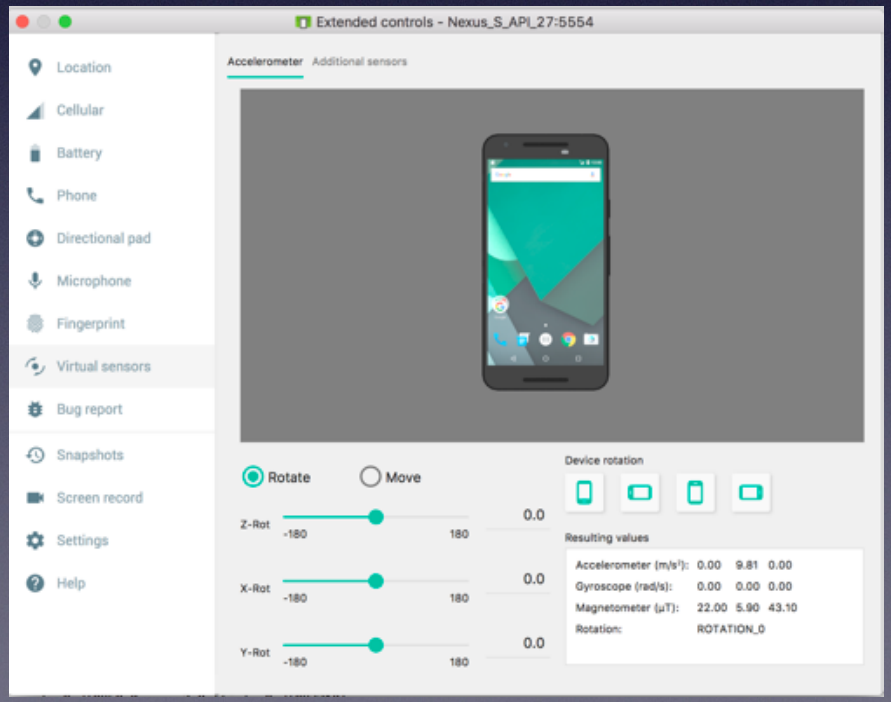


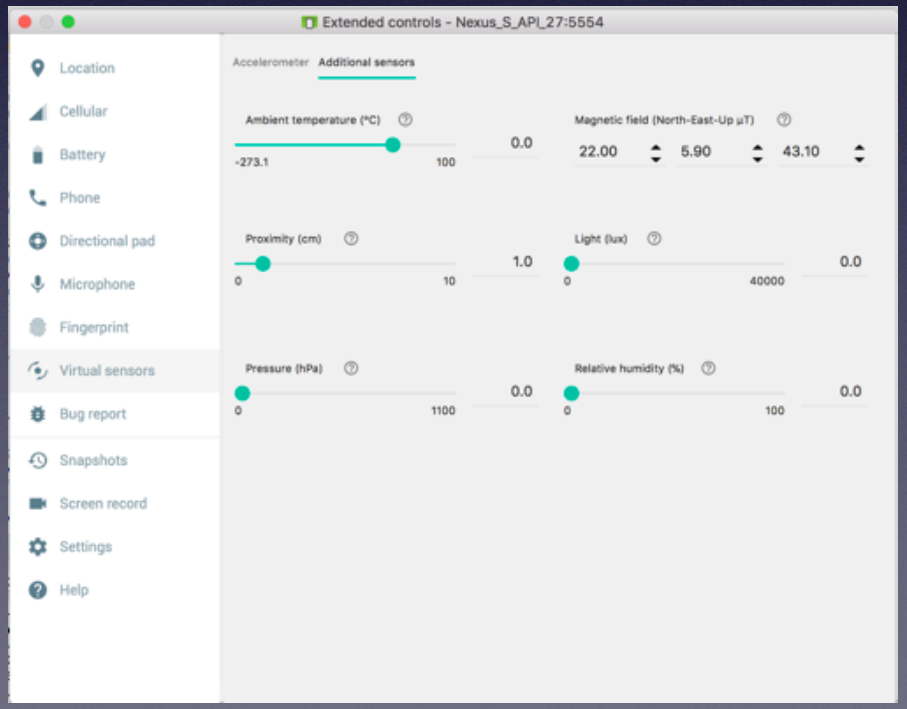
- Attention : le système des coordonnées du capteurs ne change jamais même lorsque l'appareil bouge ou change d'orientation!

Les capteurs

Tester les capteurs

- Tests :
  - Smartphones réels
  - Émulateurs : pas possible avant sur Eclipse, mais possible sur Android Studio





Les capteurs

Les bonnes pratiques

- Se désinscrire :
  - de la surveillance d'un ou de plusieurs capteurs
  - quand plus besoin
  - avec la méthode ***unregisterListener()***
  - souvent dans la méthode ***onPause()***
- Vérifier les capteurs :
  - toujours vérifier la présence des capteurs !
  - même avec la mise en œuvre d'un filtre Google Play
- Rafraichissement :
  - bien choisir le type de délai correspondant à votre application
  - sinon consommation excessive de la batterie…

La localisation

La localisation

Généralités

- La localisation ou plutôt géolocalisation :
  - un procédé permettant de positionner un objet sur une carte ou sur un plan
  - basé sur les coordonnées géographiques
  - peut enrichir grandement une application :
    - application intelligente
    - offrir à vos utilisateurs des informations adaptées en fonction de la position de l'appareil Android.
  - la solution la plus efficace est l'usage du GPS :
    - il est également possible de déterminer une position à l'aide du WIFI
    - ou d'un réseau GSM

La localisation

Les permissions

- Deux types de permissions :
  - géolocalisation par GPS (***GPS_PROVIDER***)
  - une localisation moins précise par le WIFI et les antennes relais (***NETWORK_PROVIDER***)
  - la première englobe la deuxième

```
<!-- Autorise la gélocalisation à partir du GPS -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<!-- Autorise la gélocalisation à partir du Cellulaire et du Wifi -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

La localisation

Mise à jour de la position (1/2)

1. Récupérer le manager de la localisation

```
locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

2. Préparer le LocationListener

```
private LocationListener locationListener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        float lat = (float) (location.getLatitude());
        float lng = (float) (location.getLongitude());
    }
    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {
    }
    @Override
    public void onProviderEnabled(String provider) {
    }
    @Override
    public void onProviderDisabled(String provider) {
    }
};
```

La localisation

Mise à jour de la position (2/2)

3. s'enregistrer pour récupérer les mises à jour de la localisation

```
@Override
protected void onResume() {
    super.onResume();
    locationManager.requestLocationUpdates(provider, 1000 * 60 * 5, 1000 * 2, locationListener);
}
```

4. se désinscrire si plus besoin de la position

```
@Override
protected void onPause() {
    super.onPause();
    locationManager.removeUpdates(locationListener);
}
```

La localisation

Les paramètres de la méthode ***requestLocationUpdates***

Paramètres	Description
String provider	– LocationManager.GPS_PROVIDER – LocationManager.NETWORK_PROVIDER
Long minTime	– Intervalle minimum entre 2 mises à jour – Exprimé en millisecondes. – Attention : plus la fréquence est basse plus la consommation d'énergie est importante.
Float minDistance	– Distance minimale entre 2 mises à jour – Exprimée en mètres. – Attention : plus la distance est courte plus la consommation d'énergie est importante.
LocationListener listener	– L'écouteur (listener) appelé à chaque mise à jour d'emplacement.

La localisation

Les bonnes pratiques (1/2)

- Quand commencer l'écoute ?
  - Problématique :
    - Une longue écoute du GPS consommera beaucoup de batterie
    - Une courte période d'écoute du GPS manquera de précision
  - Durée d'écoute dépend du type d'application (navigation Vs. Store Locator)
  - Choix de ***minTime*** et ***minDistance*** a un impact très important sur la consommation de la batterie
  - Donc compromis entre précision et consommation!
  - En général, démarrage dans ***onResume()***
- Quand arrêter l'écoute ?
  - Dès que possible (plus besoin)
  - En général, arrêt dans ***onPause()***

La localisation

Les bonnes pratiques (2/2)

- Une solution rapide :
  - récupérer dernière position connue même si géolocalisation à l'arrêt
  - position disponible dans le cache récupérée grâce à ***getLastKnownLocation()***
  - plus rapide

```
Location location = locationManager.getLastKnownLocation(provider);
```

La localisation

Les alertes de proximités (1/2)

- Description :
  - Les alertes de proximités vous permettent d'avertir les utilisateurs dès qu'ils entrent ou sortent d'une zone
  - Cette zone sera définie par un cercle dont on doit préciser son centre et son rayon


```
void addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent intent)
```

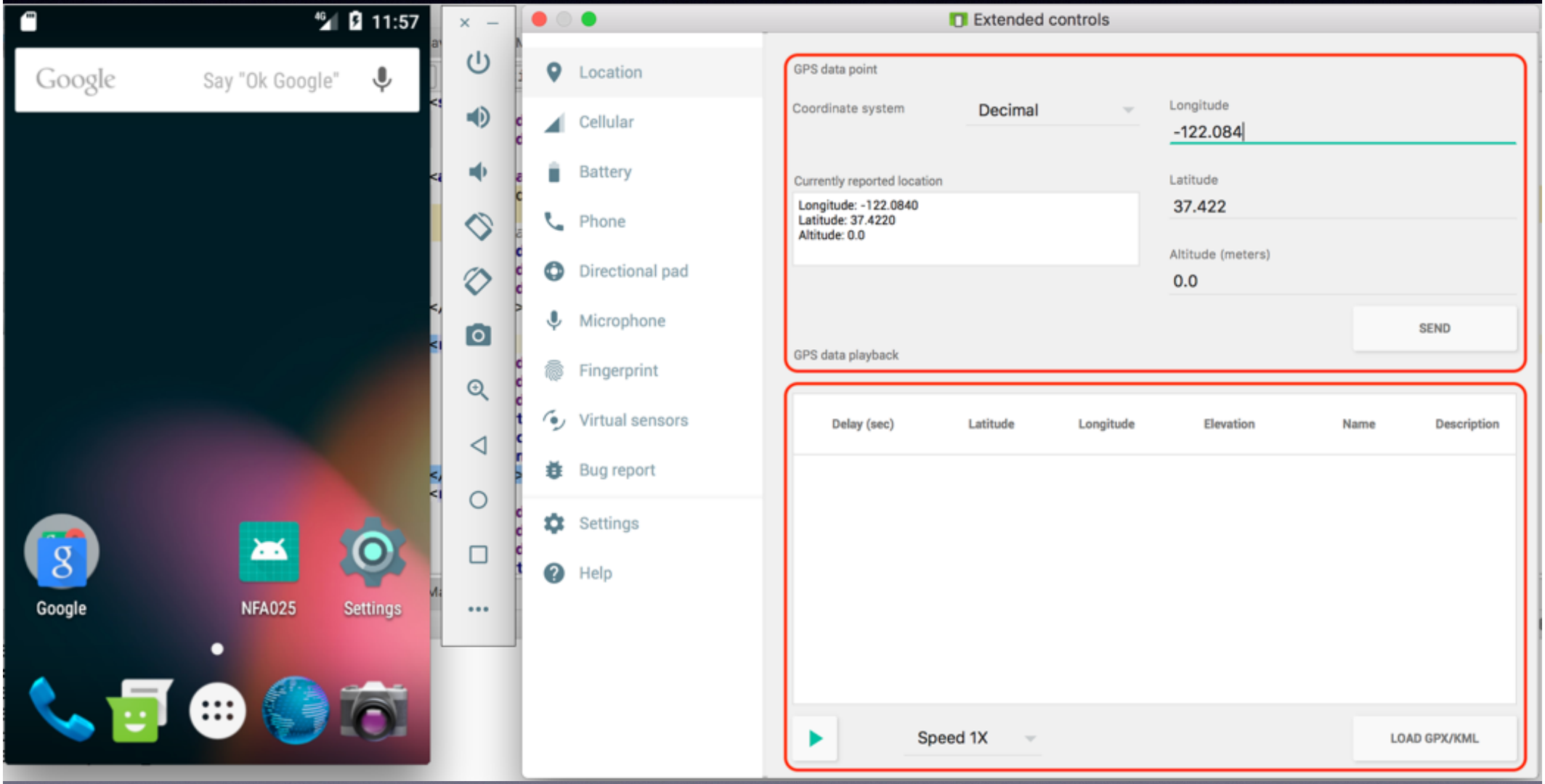
La localisation

Les alertes de proximités (2/2)

Paramètres	Description
Double latitude	-Latitude du centre
Double longitude	-Longitude du centre
Float radius	-Rayon de la zone (du cercle) -Exprimé en mètre
Long expiration	-Durée pour laquelle l'alerte est valable. -Exprimé en milliseconde -Valeur négative = aucune expiration.
PendingIntent intent	-Le pendingIntent qui sera lancé quand l'alerte sera déclenchée.

- Une alerte de proximité est envoyée dès lors que l'utilisateur entre ou sort d'une zone uniquement (même au démarrage)
- Pour limiter la consommation de la batterie, il est, conseillé de fixer une durée de validité

La localisation



La localisation

Tester la localisation (2/2)

- Comme pour les capteurs :
  - Smartphones réels
  - Émulateurs : 2 méthodes
    1. Android Studio
       - …
    2. Telnet
       - En utilisant la ligne de commande, c'est plus drôle !
       - Connectez-vous à l'appareil avec :
         $ telnet localhost 5554
       - Puis envoyez vos coordonnées avec :
         $ geo fix <latitude> <longitude>
       - Exemple :
         $ geo fix 49.244178 4.058911
       - La commande retourne ok en cas de réussite et ko en cas d'échec

La connectivité réseau / Les capteurs / La localisation

Post-it

- Pour afficher une page web ou effectuer une requête HTTP, il est indispensable de demander l'autorisation à INTERNET dans le manifest
- Une bonne pratique reste à vérifier l'état de la connexion et d'avertir l'utilisateur.
- Une requête doit toujours charger ses informations dans un thread séparé afin d'éviter les ANR
- Le manager des capteurs facilite beaucoup leur utilisation dans Android
- Il existe des capteurs physiques et d'autres virtuels
- Il faut vérifier toujours la présence ou non d'un capteur avant de l'utiliser
- Il est possible de surveiller et de récupérer les données capteurs via le service de gestion des capteurs
- Les appareils Android peuvent déterminer leur position avec plus ou moins de précision
- La géolocalisation consomme beaucoup d'énergie et a un impact important sur l'autonomie de la batterie
- Il est conseillé de :
✓ maximiser l'intervalle de temps et de distance entre deux mises à jour de la position.
✓ utiliser la dernière position connue sans réactiver le GPS.
✓ stopper l'écoute dès qu'on en a plus besoin.