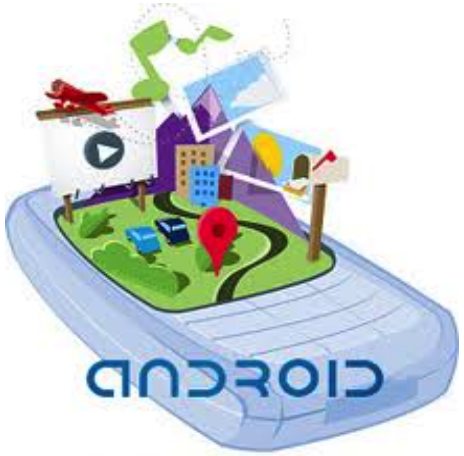
Développement d'Applications Mobiles sous Android

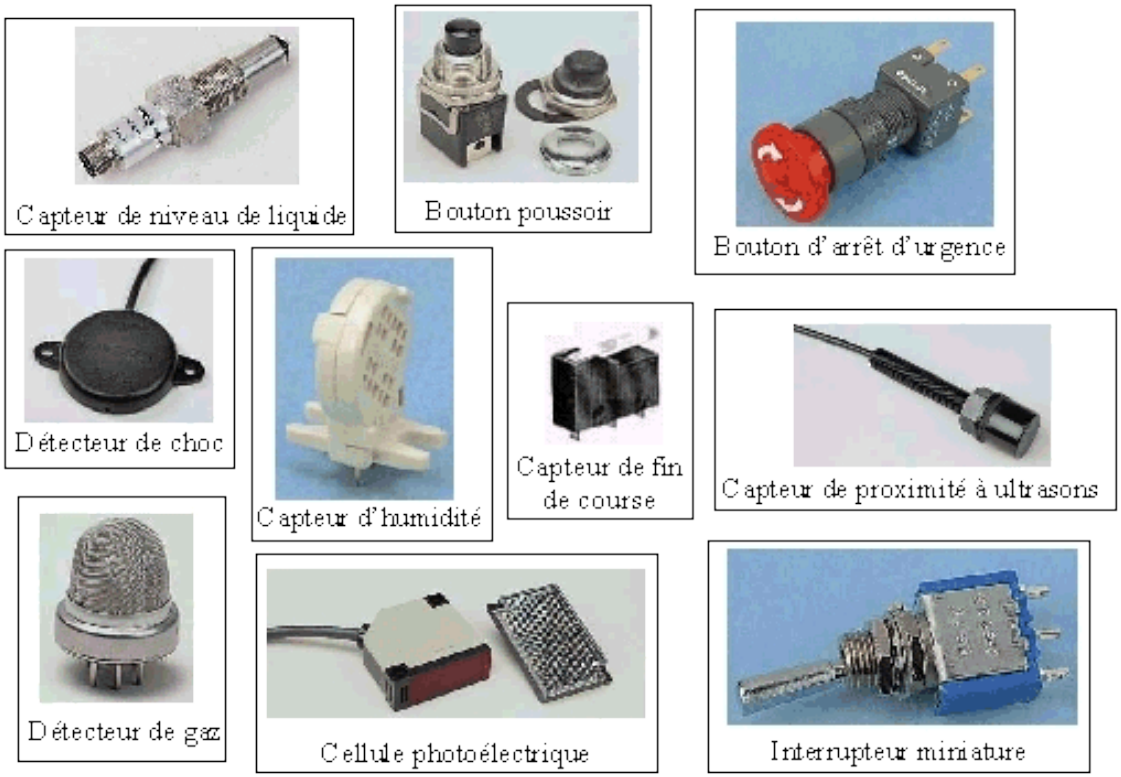


Abdelhak-Djamel Seriai

Les Capteurs

Présentation

- **Capteur :**
  - Est un dispositif transformant l'état d'une grandeur physique observée en une grandeur utilisable, telle qu'une tension électrique, une hauteur de mercure, une intensité ou la déviation d'une aiguille.



Présentation

- **Capteurs et Android**
  - La majorité des Smartphones sont dotés de capteurs : accéléromètre, capteur de lumière, capteur d'orientation, etc.
  - Sont utiles pour construire différents types d'applications sensibles aux contextes (sports, tourisme, etc.), les jeux, etc.
  - Peu d'appareils possèdent tous les capteurs gérés par la plateforme Android.
    - La majorité des Smartphones et tablettes ont un accéléromètre et un magnétomètre mais peu possèdent un baromètre où un thermomètre.
  - Un appareil peut avoir plusieurs capteurs du même type.
    - Par exemple, un appareil peut avoir deux capteurs de gravité

Framework (API) d'Android pour gestion des capteurs

- **Types de capteurs gérés par Android** :
  - Capteurs de mouvements
    - Mesurent les forces d'accélération et de rotation autour de trois axes.
    - Inclut les capteurs de : accéléromètre, gravité, gyroscope et rotation.
  - Capteurs d'environnement
    - Mesurent différents paramètres environnementaux tels que la température ambiante et la pression, la luminosité et l'humidité.
    - Inclut les capteurs de : pression (baromètre), photomètre et thermomètre.
  - Capteurs de position
    - Mesurent la position physique de l'appareil.
    - Inclut les capteurs de : orientation et champs magnétique (magnétomètre).

Framework (API) d'Android pour gestion des capteurs

- Android propose un Framework (Une API) pour accéder aux capteurs disponibles sur un appareil et acquérir les données capturées par ces capteurs.

- Exemples de services :
  - Déterminer les capteurs disponibles sur un appareil.
  - Déterminer les propriétés d'un capteur donné telles que la portée, le constructeur, les besoins en énergie et la résolution.
  - Acquérir les données des capteurs.
  - Enregistrer et dé-enregistrer un écouteur d'évènement pour gérer les changements signalés par un capteur.

Capteur	Type	Description	Utilisation
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s^2that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s^2that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s^2that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.

Capteur	Type	Description	Utilisation
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in µT.	Creating a compass.
TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z).	Determining device position.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14	Monitoring temperatures.

Capteur	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
TYPE_ACCELEROMETER	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_AMBIENT_TEMPERATURE	**Yes**	n/a	n/a	n/a
TYPE_GRAVITY	**Yes**	**Yes**	n/a	n/a
TYPE_GYROSCOPE	**Yes**	**Yes**	n/a[1]	n/a[1]
TYPE_LIGHT	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_LINEAR_ACCELERATION	**Yes**	**Yes**	n/a	n/a
TYPE_MAGNETIC_FIELD	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_ORIENTATION	**Yes**[2]	**Yes**[2]	**Yes**[2]	**Yes**
TYPE_PRESSURE	**Yes**	**Yes**	n/a[1]	n/a[1]
TYPE_PROXIMITY	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_RELATIVE_HUMIDITY	**Yes**	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	**Yes**	**Yes**	n/a	n/a
TYPE_TEMPERATURE	**Yes**[2]	**Yes**	**Yes**	**Yes**

[1] Ce type de capteur a été ajouté à partir d'Android 1.5 (API Level 3), mais il est rendu utilisable que à partir de la version Android 2.3 (API Level 9).
[2] Ce capteur est disponible mais il est devenu obsolète.

Les principales classes de l'API

- **SensorManager :**
  - Utilisée pour créer une instance du service commun de gestion des capteurs.
  - Fournit :
    - Des méthodes pour accéder aux capteurs ou les lister, enregistrer et de-enregistrer les écouteurs etc..
    - Des constantes permettant d'accéder aux valeurs telles que la précision du capteur ou le taux d'acquisition des données, etc.
- **Sensor :**
  - Utilisée pour créer une instance d'un capteur donné.
  - Fournit des méthodes pour déterminer les propriétés d'un capteur.

Les principales classes de l'API

- **SensorEvent:**
  - Utilisée pour créer un objet évènement de capteur, qui fournit des informations sur l'évènement du capteur.
    - Cet objet inclut les informations suivantes : les données capturées par le capteur, le type du capteur qui a générer l'évènement, la précision des données et l'horodatage de l'événement.
    - Un évènement capteur apparaît chaque fois que le capteur détecte un changement du paramètre qu'il mesure.

- **SensorEventListener :**
  - Est une interface utilisée pour créer deux méthodes de callback
    - Ces méthodes reçoivent des notifications (événements du capteur) quand les valeurs ou la précision de ce capteur changent.

Utilisation des capteurs: Les étapes

1. Récupérer une instance du service de gestion des capteurs (SensorManager)

2. Vérifier si le capteur qui nous intéresse est disponible sur l'appareil→ utilisation des méthodes de SensorManager.

3. Si le capteur en question existe, récupérer un objet qui représente le capteur qui nous intéresse → instance de la classe Sensor.

4. S'abonner aux événements du capteur qui nous intéresse → la classe utilisateur doit implémenter l'interface SensorEventListener.

5. Dans les méthodes de callBack de l'interface SensorEventListener, utiliser les informations de l'objet, instance de la classe SensorEvent qui encapsule les données reçues du capteur et faire une action.

Utilisation des capteurs : SensorManager

- Récupérer une instance du service de gestion des capteurs (SensorManager)

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager)
                 getSystemService(Context.SENSOR_SERVICE);
```

- Lister les capteurs présents sur un appareil

```
final SensorManager sensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);

List<Sensor> sensorsList =
sensorManager.getSensorList(Sensor.TYPE_ALL);
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    listSensor();
}
```

```
private void listSensor() {
    List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
    StringBuffer sensorDesc = new StringBuffer();
for (Sensor sensor : sensors) {
        sensorDesc.append("New sensor detected : \r\n");
        sensorDesc.append("\tName: " + sensor.getName() + "\r\n");
        sensorDesc.append("\tType: " + getType(sensor.getType()) + "\r\n");
        sensorDesc.append("Version: " + sensor.getVersion() + "\r\n");
        sensorDesc.append("Resolution (in the sensor unit): " +
                          sensor.getResolution() + "\r\n");
        sensorDesc.append("Power in mA used by this sensor while in use" +
                          sensor.getPower() +"\r\n");
        sensorDesc.append("Vendor: " + sensor.getVendor() + "\r\n");
        sensorDesc.append("Maximum range of the sensor in the sensor's unit." +
                          sensor.getMaximumRange() + "\r\n");
        sensorDesc.append("Minimum delay allowed between two events in
                          microsecond » + " or zero if this sensor only returns a
                          value when the data it's measuring changes » +
                          sensor.getMinDelay() + "\r\n");
    }

Toast.makeText(this, sensorDesc.toString(), Toast.LENGTH_LONG).show();
 }
```

Utilisation des capteurs: SensorManager

- Déterminer la présence d'un capteur d'un type donné sur un appareil
  - Utilisation de la méthode ***getDefaultSensor()***
  - Avec un paramètre de type constante du capteur en question.
    - TYPE_GYROSCOPE, TYPE_LINEAR_ACCELERATION, TYPE_GRAVITY, …

  - Si l'appareil possède plus d'un capteur du même type, il doit être désigné comme capteur par défaut.

```
Sensor defaultProximitySensor =
      sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);

Log.v("SensorActivity", "defaultProximitySensor = " +
       defaultProximitySensor.getName());
```

Utilisation des capteurs: SensorManager

- Exemple du besoin de vérifier la présence d'un capteur
  - Une application de guidage (de navigation) peut utiliser les capteurs de température, de pression, capteur GPS, capteur de champs magnétique,
  - Si un appareil n'a pas de capteur de pression, l'application doit détecter dynamiquement l'absence de ce capteur et désactiver la partie de l'application qui utilise ce capteur

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager)
                 getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)
    !=null){
  // Success! There's a magnetometer.
  } else {
  // Failure! No magnetometer.}
```

Utilisation des capteurs: classe Sensor

- Déterminer les propriétés d'un capteur : utilisation des méthodes de la classe Sensor
  - Permet d'adapter dynamiquement les services d'une application en fonction de la disponibilité ou non de certains capteurs sur l'appareil
  - Exemples
    - getResolution() et getMaximumRange() : retournent respectivement les mesures de la résolution et de la portée maximale.
    - getPower() retourne la mesure du besoin énergétique (batterie) d'un capteur.
    - getVendor() et getVersion().
- Exemple :
  - Chercher si un capteur de gravité est disponible et dont le fabricant est Google Inc. et la version est 3. Si ce capteur n'est pas présent, utilisation de l'accéléromètre

```
private SensorManager mSensorManager;
private Sensor mSensor;

...

mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

if (mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null){
  List<Sensor> gravSensors = mSensorManager.getSensorList(Sensor.TYPE_GRAVITY);
  for(int i=0; i<gravSensors.size(); i++) {
    if ((gravSensors.get(i).getVendor().contains("Google Inc.")) &&
        (gravSensors.get(i).getVersion() == 3)){
      // Use the version 3 gravity sensor.
      mSensor = gravSensors.get(i);
    }
  }
}
else{
  // Use the accelerometer.
  if (mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null){
    mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
  }
  else{
    // Sorry, there are no accelerometers on your device.
    // You can't play this game.
  }
}
```

Utilisation des capteurs: classe Sensor

- Déterminer la sensibilité d'un capteur
  - La sensibilité d'un capteur est l'intervalle de temps minimum (en microsecondes) qu'un capteur utilise pour acquérir ses données.
  - **getMinDelay()** retourne la sensibilité d'un capteur :
    - Un capteur qui retourne une valeur différente de zéro est un capteur de flux continu (streaming sensor).
    - Un capteur qui retourne la valeur 0, n'est pas un capteur de flux continu mais ce capteur qui avertit ses écouteurs en cas de changement.
- Remarque : la valeur de sensibilité retournée par un capteur n'est pas nécessairement celle de livraison des données vers une application.
  - L'API livre les données à une application via des événements dont le ration dépend d'autres facteurs.

Agir par rapport aux événements d'un capteur

- Pour obtenir les données à partir d'un capteur, il est nécessaire de :
  1. S'enregistrer auprès du service comme écouteur d'événements
     - Utilisation de la méthode registerListener().

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

Boolean supported = sensorMgr.registerListener(this,
                            sensorMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
                            SensorManager.SENSOR_DELAY_UI);
if (!supported) {
    sensorMgr.unregisterListener
                    (this,sensorMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER));

((TextView)findViewById(R.id.acc)).setText("Pas d'accelerometre");

}
```

Agir par rapport aux événements d'un capteur

- Pour obtenir les données à partir d'un capteur, il est nécessaire de :
  1. s'enregistrer auprès du service comme écouteur d'événements
     - Utilisation de la méthode registerListener().

     - « data delay ou rate » est un des paramètres de registerListener() = l'intervalle de temps, souhaité par l'utilisateur, entre chaque envoi de données (événements) par le capteur à l'application via la méthode onSensorChanged().

     - Plus le taux est élevé, plus les données sont rafraîchies et précises, et plus la consommation d'énergie est élevée.

Agir par rapport aux événements d'un capteur

- Pour obtenir les données à partir d'un capteur, il est nécessaire de :
  1. s'enregistrer auprès du service comme écouteur d'événements
     - Utilisation de la méthode registerListener().
     - « data delay ou rate » :
       - SENSOR_DELAY_FASTEST : le taux de rafraîchissement le plus élevé. Les données sont récupérées aussi vite que possible par le système ;
       - SENSOR_DELAY_GAME : taux utilisable pour des applications nécessitant des données précises telles que les jeux ; 20,000 microsecondes
       - SENSOR_DELAY_NORMAL : taux normal (par défaut) utilisé par exemple par le système pour détecter les changements d'orientation ; 200,000 microsecondes
       - SENSOR_DELAY_UI : le taux le plus bas utilisé dans les applications ne nécessitant qu'une faible précision. 60,000 microsecondes
     - A partir de Android 3.0 (API Level 11), il est possible de spécifier le taux de rafraichissement par une valeur en microsecondes.

Agir par rapport aux événements d'un capteur

- Pour obtenir les données à partir d'un capteur, il est nécessaire de :
  1. s'enregistrer auprès du service comme écouteur d'événements
     - Utilisation de la méthode registerListener().
     - « data delay ou rate » :
       - Par exemple la valeur par défaut (NORMAL) est souhaitable pour contrôler les changements d'orientation de l'écran.

```
@Override
  protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mLight,
                                    SensorManager.SENSOR_DELAY_NORMAL);
  }
}
```

Agir par rapport aux événements d'un capteur

- Pour obtenir les données à partir d'un capteur, il est nécessaire de :
  1. s'enregistrer auprès du service comme écouteur d'événements
     - Après utilisation de ces données ou avant de quitter l'application, il est indispensable de se dé-senregistrer
       - Utilisation de la méthode unregisterListener.

```
if (accelSupported)
sensorMgr.unregisterListener
                    (this,sensorMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER));
```

Agir par rapport aux événements d'un capteur

- Pour obtenir les données à partir d'un capteur, il est nécessaire de :
  1. S'enregistrer auprès du service comme écouteur d'événements
  2. Implémenter deux méthodes de callback définies dans l'interface
     - **onAccuracyChanged() :** Appelée quand la précision du capteur change
       - Le système fournit à cette méthode une référence vers l'objet Sensor concerné et la nouvelle valeur de la précision
       - La précision a une des 4 valeurs suivantes :
         - SENSOR_STATUS_ACCURACY_LOW,
         - SENSOR_STATUS_ACCURACY_MEDIUM,
         - SENSOR_STATUS_ACCURACY_HIGH,
         - SENSOR_STATUS_UNRELIABLE.

Agir par rapport aux événements d'un capteur

```
public class MainActivity extends Activity  implements SensorEventListener_{

    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        switch(accuracy){
            case SensorManager.SENSOR_STATUS_ACCURACY_HIGH:
            case SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM:
            case SensorManager.SENSOR_STATUS_ACCURACY_LOW:
            case SensorManager.SENSOR_STATUS_UNRELIABLE:
        }

        Log.d("Sensor",sensor.getType()+":"+accuracy);
    }
```

Agir par rapport aux événements d'un capteur

- Pour obtenir les données à partir d'un capteur, il est nécessaire de :
  1. s'enregistrer auprès du service comme écouteur d'événements
  2. Implémenter deux méthodes de callback définies dans l'interface
     - onAccuracyChanged()
     - **onSensorChanged() :** Appelée quand le capteur retourne une nouvelle donnée.
       - Le système fournit à cette méthode l'objet **SensorEvent** qui contient les informations suivantes :
         - La donnée générée;
         - Le capteur qui a généré cette donnée;
         - L'horodatage de la donnée générée;
         - La précision de cette donnée.

```
public class MainActivity extends Activity  implements SensorEventListener {

public void onSensorChanged(SensorEvent event) {
        switch(event.sensor.getType()){
            case Sensor.TYPE_ACCELEROMETER:
                onAccelerometerChanged(event);
                break;
            case Sensor.TYPE_ORIENTATION:
                onOrientationChanged(event);
                Break;
                case Sensor.TYPE_MAGNETIC_FIELD:
                    onMagneticFieldChanged(event);
                    break;
                case Sensor.TYPE_TEMPERATURE:
                    onTemperatureChanged(event);
                    break;
                case Sensor.TYPE_PROXIMITY:
                    onProximityChanged(event);
                    break;
                case Sensor.TYPE_LIGHT:
                    onLightChanged(event);
                    break;
                case Sensor.TYPE_PRESSURE:
                    onPressureChanged(event);
                    break;
                case Sensor.TYPE_GYROSCOPE:
                    onGyroscopeChanged(event);

        break;
}}
//...
```

Agir par rapport aux événements d'un capteur

- Pour obtenir les données à partir d'un capteur, il est nécessaire de :
  1. s'enregistrer auprès du service comme écouteur d'événements
  2. Implémenter deux méthodes de callback définies dans l'interface
     - Chaque capteur renvoie un vecteur de données sous forme de flottants.

Nom	Dimension du vecteur	Unité	Sémantique	Valeurs[]
Accelerometer	3	m/s2	Mesure de l'accélération (gravité incluse)	[0] axe x [1] axe y [2] axe z
Gyroscope	3	Radian/seconde	Mesure la rotation en termes de vitesse autour de chaque axe	[0] vitesse angulaire autour de x [1] vitesse angulaire autour de y [2] vitesse angulaire autour de z
Light	1	Lux	Mesure de la luminosité	[0]valeur
Magnetic_Field	3	µTesla	Mesure du champ magnétique	[0] axe x [1] axe y [2] axe z
Orientation	3	degrés	Mesure l'angle entre le nord magnétique	[0] Azimut entre l'axe y et le nord [1] Rotation autour de l'axe x (-180,180) [2] Rotation autour de l'axe y (-90,90)
Pressure	1	KPascal	Mesure la pression	[0]valeur
Proximity	1	mètre	Mesure la distance entre l'appareil et un objet cible	[0]valeur
Temperature	1	Celsius	Mesure la température	[0]valeur

```
public class SensorActivity extends Activity implements SensorEventListener {
  private SensorManager mSensorManager;
  private Sensor mLight;

  @Override
  public final void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
  }

  @Override
  public final void onAccuracyChanged(Sensor sensor, int accuracy) {
    // Do something here if sensor accuracy changes.
  }

  @Override
  public final void onSensorChanged(SensorEvent event) {
    // The light sensor returns a single value.
    // Many sensors return 3 values, one for each axis.
    float lux = event.values[0];
    // Do something with this sensor value.
  }
```

Agir par rapport aux événements d'un capteur

```
@Override
  protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mLight,
                                    SensorManager.SENSOR_DELAY_NORMAL);
  }

  @Override
  protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
  }
}
```

Agir par rapport aux événements d'un capteur

```
public class SensorAccelerationActivity extends Activity implements SensorEventListener {
SensorManager sensorManager;
Sensor accelerometer;

@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

        accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
}

@Override
    protected void onPause() {
        // unregister the sensor (désenregistrer le capteur)
        sensorManager.unregisterListener(this, accelerometer);
        super.onPause();
    }

@Override
    protected void onResume() {
sensorManager.registerListener(this, accelerometer, SensorManager.SENSOR_DELAY_UI);
        super.onResume();
    }
```

Agir par rapport aux événements d'un capteur

```
@Override
public void onAccuracyChanged(Sensor sensor, intaccuracy) {
    // Rien à faire la plupart du temps
}

@Override
public void onSensorChanged(SensorEvent event) {
// Récupérer les valeurs du capteur
floatx, y, z;
if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
    x = event.values[0];
    y = event.values[1];
    z = event.values[2];
}
}
```

Utilisation du filtre Google Play pour configurer une application avec capteur

- Pour vérifier qu'un capteur est disponible sur un appareil.
  - Détecter le capteur dynamiquement et activer/désactiver certaines fonctionnalités d'une application
  - Utilisation d'un filtre Google Play pour cibler des appareils avec des configurations données de capteurs

Utilisation du filtre Google Play pour configurer une application avec capteur

- Si une application est publiée sur Google play
  - Utilisation de l'élément **<uses-feature>** dans le manifest pour filtrer les appareils qui ont la configuration de capteurs requise par l'application.
  - L'élément **<uses-feature>** a plusieurs descripteurs matériels qui permettent à une application de définir une configuration requise de capteurs.
    - La liste des capteurs inclut :
      - accelerometer,
      - barometer,
      - compass (geomagnetic field),
      - gyroscope,
      - light,
      - proximity.

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

Utilisation du filtre Google Play pour configurer une application avec capteur

- Utilisation de la description `android:required="true"` seulement si l'application a besoin de ce capteur pour fonctionner.
  - Un appareil qui ne dispose pas de ce capteur ne peut pas installer l'application en question

- Utilisation de la description `android:required="false"` si l'application utilise le capteur en question uniquement pour certaines fonctionnalités.
  - Dans ce cas, un appareil peut installer une application même si elle ne dispose pas de ce capteur
  - Pour le développeur de l'application, il est nécessaire de détecter l'absence de ce capteur sur l'appareil et désactiver les services liés à ce capteur

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android2ee.android.tuto.sensor.light"android:versionCode="1"
    android:versionName="1.0">

    <uses-sdkandroid:minSdkVersion="10"/>

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

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

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

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

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

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

 </manifest>
```

Zoom sur la réception des données

Les capteurs

- Réception des données
  - Utilisation de l'accéléromètre

```
private void onAccelerometerChanged(SensorEvent event){
float x,y,z;
    x = event.values[0];
    y = event.values[1];
    z = event.values[2];

((TextView)findViewById(R.id.axex)).setText("Axe X: "+x+"m/s^2");
((TextView)findViewById(R.id.axey)).setText("Axe Y: "+y+"m/s^2");
((TextView)findViewById(R.id.axez)).setText("Axe Z: "+z+"m/s^2");
}
```

Les capteurs

- Réception des données
  - Utilisation du capteur d'orientation

```
private void onOrientationChanged(SensorEvent event){
float azimuth,pitch,roll;
    azimuth = event.values[0];
    pitch = event.values[1];
    roll = event.values[2];

((TextView)findViewById(R.id.azimuth)).setText("Azimuth: "+azimuth+"°");
((TextView)findViewById(R.id.pitch)).setText("Pitch: "+pitch+"°");
((TextView)findViewById(R.id.roll)).setText("Roll: "+roll+"°");
}
```

Les capteurs

- Réception des données
  - Utilisation du capteur de température

```
private void onTemperatureChanged(SensorEvent event){
float temperature;
    temperature = event.values[0];
    ((TextView)findViewById(R.id.temp)).setText("Temperature: "+temperature
+"°F");
}
```

Les capteurs

- Réception des données
  - Utilisation du capteur de champ magnétique

```
private void onMagneticFieldChanged(SensorEvent event){
float uTx,uTy,uTz;
    uTx = event.values[0];
    uTy = event.values[1];
    uTz = event.values[2];

((TextView)findViewById(R.id.uTx)).setText("Axe X: " + uTx + "µT");
((TextView)findViewById(R.id.uTy)).setText("Axe Y: " + uTy + "µT");
((TextView)findViewById(R.id.uTz)).setText("Axe Z: " + uTz + "µT");
}
```

Les capteurs

- Réception des données
  - Utilisation du capteur de lumière

```
private void onLightChanged(SensorEvent event){
float illuminance;
    illuminance = event.values[0];
    ((TextView)findViewById(R.id.litex)).setText("Light: "+ illuminance +
"lux");
}
```

Les capteurs

- Réception des données
  - Utilisation du capteur de proximité

```
private void onProximityChanged(SensorEvent event){
float distance;
    distance = event.values[0];

    ((TextView)findViewById(R.id.prox)).setText("Proximity: " + distance +
"cm");
}
```

Les capteurs

- Utilisation du capteur de pression
  - Réception des données

```
private void onPressureChanged(SensorEvent event){
float pressure;
    pressure = event.values[0];

((TextView)findViewById(R.id.pressex)).setText("Pressure: " + pressure + "psi");
}
```

Les capteurs

- Réception des données
  - Utilisation du gyroscope

```
private void onGyroscopeChanged(SensorEvent event){
float x, y, z;
    x = event.values[0];
    y = event.values[1];
    z = event.values[2];

((TextView)findViewById(R.id.gyrox)).setText("Axe X: " + x + "°");
((TextView)findViewById(R.id.gyroy)).setText("Axe Y: " + y + "°");
((TextView)findViewById(R.id.gyroz)).setText("Axe Z: " + z + "°");
}
```