Lab 15: Sensors

Android supports sensors which provide the measurements of certain properties such as device motion, orientation and rotation, pressure, temperature, light, magnetic field, proximity and humidity. Using android sensor framework, you can implement in your android app the features that use sensors and its raw data. [Android Developer Documentation]

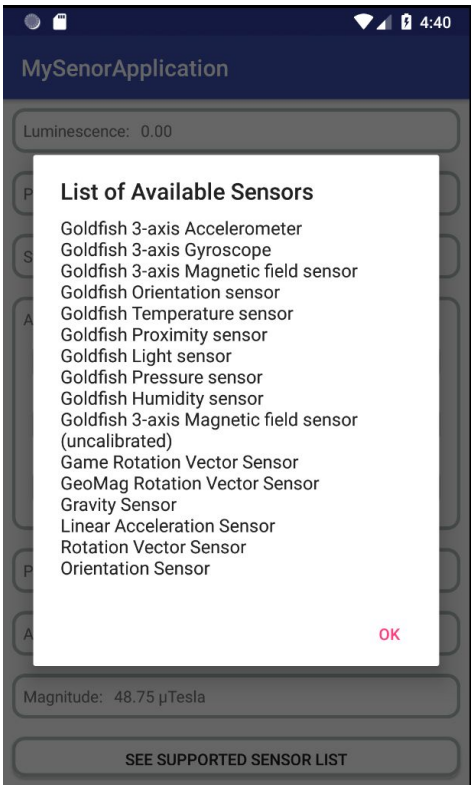
In this lab you will:

- Query your device for a list of available sensors (i..e motion, environmental, and position) and display the results in a AlertDialog screen.
- Listen to sensor events and display numerical values representing: 1) light, proximity, acceleration, magnitude, pressure

Result:

Starting the app will display sensor data in a linear list (Figure a). On tapping the button at the bottom, an Alert Dialog (Figure b) will pop up with the list of available sensors.

a.



b.

Initial Steps

- Create a new project (with the Empty Activity template). Give it a package name `com.techexchange.mobileapps.lab15`.
- We will cheat and not use a Fragment this time; all our widgets will be directly in the MainActivity. Open activity_main.xml and overwrite it with XML depicting the view hierarchy of the list of elements in Figure a. As shown below, one row consist of two TextViews (e.g. one for the Label, and one for the sensor value) laid out horizontally, and each row is laid out vertically in a list. You may need to add a layer of nesting to accommodate the three Accelerometer values.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  android:id="@+id/sensors_layout">

  <LinearLayout
     android:layout_width="match_parent"
     android:layout_margin="8dp"
     android:orientation="horizontal"
     android:layout_height="wrap_content"
     android:background="@drawable/row_layout_style">

     <TextView
        android:id="@+id/light_sensor"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/luminescence" />


     <TextView
        android:id="@+id/luminescence_value"
        android:layout_marginStart="10dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/luminescence"/>

  </LinearLayout>

        :
        :
```

```
  <Button
     android:id="@+id/button_id"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:text="See Supported Sensor List"
     android:layout_margin="8dp"
     android:background="@drawable/row_layout_style"/>

  </LinearLayout>
```

(The text views are set to completely transparent)

Part 1, customize row elements and button with drawable, String Resources

- Create a new drawable resource file named "row_layout_style". Have it be a rectangular outline with rounded corners:

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
  <solid android:color="#FFFFFF"/>
  <stroke android:width="3dp" android:color="#B1BCBE" />
  <corners android:radius="10dp"/>
  <padding android:left="10dp" android:top="10dp" android:right="10dp"
android:bottom="10dp" />
</shape>
```

- Open strings.xml in the /res folder and create labels for each sensor shown in Figure a.

```
<resources>
  <string name="app_name">MySensorApplication</string>
  <string name="luminescence">Luminescence:</string>
  <string name="proximity">Proximity:</string>
  <string name="steps">Steps:</string>
  <string name="accelerometer">Accelerometer - </string>
  <string name="azimuth">Azimuth:</string>
  <string name="pitch">Pitch:</string>
  <string name="roll">Roll:</string>
```

```
  <string name="Pressure">Pressure:</string>
  <string name="temperature">Ambient Temperature:</string>
  <string name="magnetic">Magnitude:</string>
  <string name="sensorList">List of Available Sensors</string>
  <string name="OK">OK</string>
</resources>
```

- Override the row_layout_style background color for the Button and use your favorite color.

Part 2, Declaring variables and references to Widgets

- Declare a *private* class variable **TextView** for each widget defined in the activity_main.xml.
- Declare *private* **Senso**r variables for the following sensors: light, proximity, rotation, step counter, magnetic, pressure, and temperature.
- Declare a *private* variable **SensorManager**. The **SensorManager** provides methods:
    - to get sensor instance,
    - to access and list sensors,
    - to register and unregister sensor listeners etc.
- Declare a private **String** variable to hold the list of available sensors returned from the SensorManager.

Part 3, Wiring up, and getting references to Widgets in the MainActivity - onCreate()

- Override onCreate() and assign references to all widgets to your local variables.
- Set the onClickListener for the Button to be **onButtonPressed**.
- Get a reference to the SensorManager.

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

Using android **SensorManager**, you can get list of all available sensors on a device by calling *getSensorList* on **SensorManager** object passing sensor type all argument to it. The method returns list of **Sensor** objects such as sensor *type, name, vendor, power, delay* between sensor events and maximum *range,* etc.

- Create a private method that returns a **String** in the Activity called - **getSensorList()**

```
private String getSensorList() {
```

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

    String sensorInfo = "";
    for (Sensor s : sensorList){
        sensorInfo= sensorInfo + s.getName()+ "\n";
    }
    return sensorInfo;
}
```

- Create another private method called - **void getSensorAvailabilty(),** that determines the availability of a specific sensor and initializes the TextViews for each sensor to “N/A” if the specific sensor is not available. Below is sample code for checking just the Light Sensor. Please complete the remaining checks for each type below:
    - TYPE_PROXIMITY
    - TYPE_GAME_ROTATION_VECTOR
    - TYPE_STEP_COUNTER
    - TYPE_MAGNETIC_FIELD
    - TYPE_PRESSURE
    - TYPE_AMBIENT_TEMPERATURE

```
private void getSensorAvailabilty(){

    if ((lightSensor = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)) == null)
        lightSen_value.setText("N/A");
            :
            :
            :

}
```

- In **onCreate()**, call **getSensorAvailability()** to identify what sensors are available. Then call **getSensorList()** and assign the result to the private **String** variable declared to hold the list of available sensors.
- Now we need to implement the **ButtonPressed()** method to create an *AlertDialog* popup to display the contents of the **sensorList** using the code below.

```
@Override
public void onButtonPressed(){
```

```
  new AlertDialog.Builder(this).setMessage(sensorList)
        .setTitle(getString(R.string.sensorList))
        .setPositiveButton(getString(R.string.OK), (arg0, arg1) -> { } ).show();
}
```

Part 4, Sensor Event Listener

To listen to sensors events, you need to implement **SensorEventListener** interface. SensorEventListener has two callback methods, **onAccuracyChanged()** and **onSensorChanged()**. The callback method onAccuracyChanged is called when accuracy of measurement changes and is passed a sensor object which caused the event and accuracy status. Method **onSensorChanged** is called when new sensor data is available and it is passed SensorEvent object which contains sensor data.

- Your MainActivity should implement the **SensorEventListener**, and add the following two methods: **onAccuracyChanged()** and **onSensorChanged()**

```
@Override
public final void onAccuracyChanged(Sensor sensor, int accuracy) {

}

@Override
public final void onSensorChanged(SensorEvent event) {

}
```

- Now provide the implementation for the **onSensorChanged()** method.

Handling value changes of sensors is easy because in the sensor event listeners you can get between one - three values that the sensor measures based on the sensor type, then you can take action based on the value(s). The example below shows how to use light sensor and set the luminance value displayed in a TextView.Notice how the liminascent value is stored in *event.values[0]*.

```
@Override
public final void onSensorChanged(SensorEvent event) {

      switch(event.sensor.getType()) {
```

```
                case Sensor.TYPE_LIGHT:

                    float valueZ = event.values[0];
                    lightSen_value.setText(String.format("%.2f", valueZ));
                     break;

                case Sensor.TYPE_XXXXX:

                     break;

                case Sensor.TYPE_YYYY:
                      :
                      :
}
```

Look here to understand how to retrieve values for postion, motion, and environmental sensors. You will have to do some calculating for Magnitude and Azmith, Pitch, and Roll. The magnetic field sensor will return values on three axes : X, Y and Z. To determine the magnetic field value, you will need to apply a mathematical formula :

Magnetic Value = sqrt(Value_X * Value_X + Value_Y * Value_Y + Value_Z * Value_Z)

Thus, the magnetic value is equal to the square root of the sum of squared values on each axes.

Using rotation (i.e. Sensor.TYPE_GAME_ROTATION_VECTOR) sensors, you can find values of azimuth, roll and pitch which can be used to determine device rotation. Azimuth is angle between magnetic north and y-axis, roll represents tilt of z-axis toward positive x-axis and pitch represents tilt of z-axis toward positive y-axis. These values can be used to determine device rotation and implement app features based on it.

I have provided the code for that calculation:

```
float[] rotMatrix = new float[9];
float[] rotVals = new float[3];

SensorManager.getRotationMatrixFromVector(rotMatrix, event.values);
SensorManager.remapCoordinateSystem(rotMatrix,
SensorManager.AXIS_X, SensorManager.AXIS_Y, rotMatrix);

SensorManager.getOrientation(rotMatrix, rotVals);
```

```
float azimuth = (float) Math.toDegrees(rotVals[0]);
float pitch = (float) Math.toDegrees(rotVals[1]);
float roll = (float) Math.toDegrees(rotVals[2]);
```

It is important to unregister and reregister the sensor listener in **onPause** and **onResume** callback methods of the activity so that when app is not visible it releases sensors and reuses them on resume or when it comes to foreground. For onResume please duplicated the ***registerListerner()*** method for each sensor you are displaying in the app.

```
@Override
protected void onResume() {
   super.onResume();

   sensorManager.registerListener(this, lightSensor,
SensorManager.SENSOR_DELAY_NORMAL);


}
```

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

That's it! Have fun with the app.