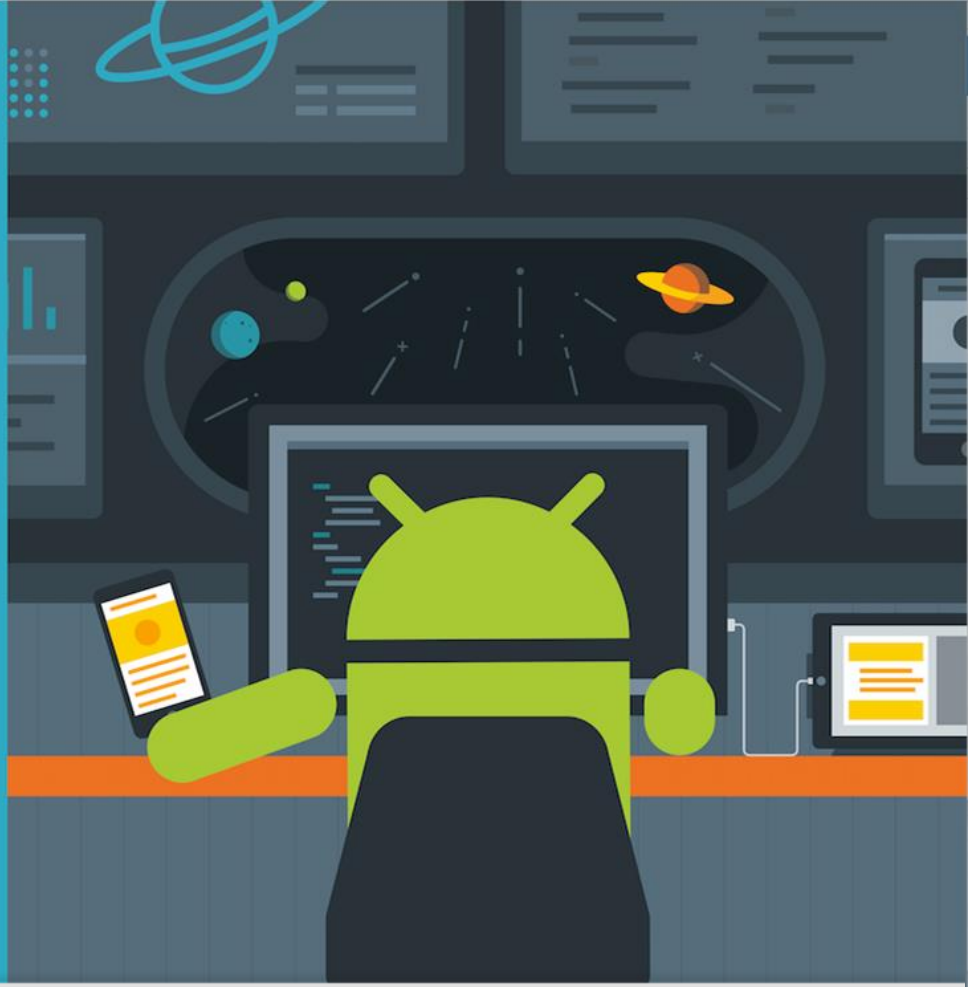
Advanced Android Development

Sensors

Lesson 3



3.1 Sensor basics

Measure motion, orientation, and environmental conditions

Contents

- Categories and types of sensors
- Emulating sensors
- Android sensor framework
- Discovering sensors and sensor capabilities
- Handling sensor configurations
- Monitoring sensor events

Categories and types of sensors

Sensor basics

Categories of sensors

- Motion sensors
- Environmental sensors
- Position sensors

Motion sensors

Measure device motion

- Accelerometers
- Gravity sensors
- Gyroscopes
- Rotational vector sensors

Environmental sensors

Measure environmental conditions

- Barometers
- Photometers (light sensors)
- Thermometers

Position sensors

Measure physical position of device

- Magnetometers (geomagnetic field sensors)
- Proximity sensors

Types of sensors

Sensor types supported by the Android platform

- Hardware-based sensors
- Software-based sensors

Hardware-based sensors

Physical component built into device

- Derives data by directly measuring specific properties
- Examples:
  light sensor, proximity sensor,
  magnetometer, accelerometer

Software-based sensors

Software: virtual or composite sensor

- Derives data from one or more hardware sensors
- Examples: linear acceleration, orientation.

Sensor availability

Sensor availability varies from device to device, it can also vary between Android versions

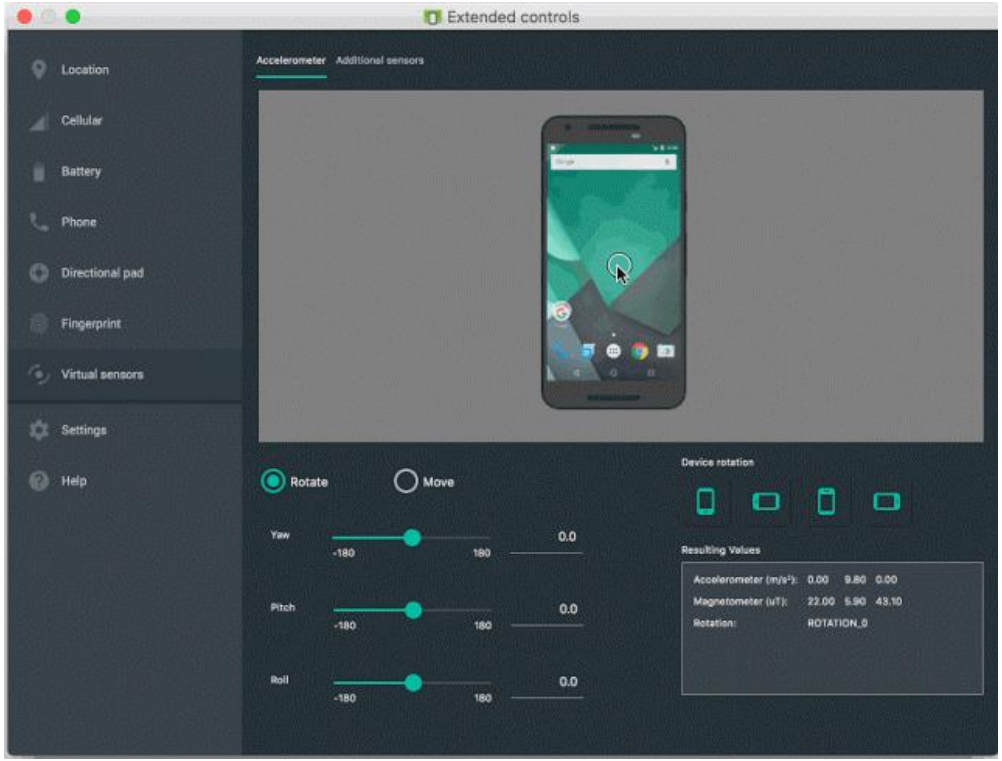
- Most devices have accelerometer and magnetometer
- Some devices have barometers or thermometers
- Device can have more than one sensor of a given type
- Availability varies between Android versions

Emulating sensors

Sensors and Android emulator

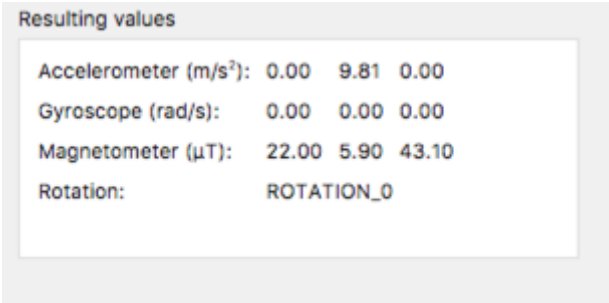
- Virtual sensor controls for testing:
  In emulator, use the panel on the right side, select **... > Virtual sensors**
  - **Accelerometer** tab:
    Test app for changes in device position and/or orientation
  - **Additional sensors** tab:
    Simulate position and environment sensors

Sensors and Android emulator



Emulator: Accelerometer tab

- Simulates device motion such as tilt and rotation
- Simulates the way accelerometers and magnetometers respond
- **Resulting Values** fields show values app can access



Emulator: Additional sensors tab

- Ambient temperature
- Magnetic field at the *x*-axis, *y*-axis, and *z*-axis. Values are in microtesla (μT)
- Proximity: Distance of device from object
- Light: Measures illuminance
- Pressure: Measures ambient air pressure
- Relative humidity

Android sensor framework

Framework classes and interfaces

SensorManager

- Access and listen to sensors
- Register and unregister sensor event listeners
- Acquire orientation information
- Provides constants for accuracy, data acquisition rates, and calibration

Important framework classes

- `Sensor`: Determine specific sensor's capabilities
- `SensorEvent`: Info about event, including raw sensor data
- `SensorEventListener`: Receives notifications about sensor events
  - When sensor has new data
  - When sensor accuracy changes

Sensor class types and typical uses

TYPE_ACCELEROMETER	Detecting motion (shake, tilt, etc.)
TYPE_AMBIENT_TEMPERATURE	Monitoring air temperature
TYPE_GRAVITY	Detecting motion (shake, tilt, etc.)
TYPE_GYROSCOPE	Detecting rotation (spin, turn, etc.)
TYPE_LIGHT	Controlling screen brightness
TYPE_LINEAR_ACCELERATION	Monitoring acceleration along single axis
TYPE_MAGNETIC_FIELD	Creating a compass

Using sensors

- Determine which sensors are available on device
- Determine an individual sensor's capabilities
  - Maximum range, manufacturer, power requirements, resolution
- Register sensor event listeners
- Acquire raw sensor data
  - Also define minimum rate for acquiring sensor data
- Unregister sensor event listeners

Discovering sensors and capabilities

Identify sensors

Create an instance of `SensorManager`

- Call `getSystemService()`
- Pass in `SENSOR_SERVICE` argument

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

Get list of sensors

Use getSensorList()

- To get all device sensors, use TYPE_ALL constant

```
List<Sensor> deviceSensors =
        mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

- To get sensors of specific type, use a constant such as TYPE_PROXIMITY, TYPE_GYROSCOPE, or TYPE_GRAVITY

Identify sensor features

`Sensor` class methods

- `getResolution()` for sensor resolution
- `getMaximumRange()` for maximum range of measurement
- `getPower()` for sensor's power requirements
- `getVendor()` and `getVersion()` to optimize for different sensors or different versions of sensor
- `getMinDelay()` to determine maximum rate at which sensor can acquire data

Example: Identify magnetometer sensor

```
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.
}
```

Handling different sensor configurations

Use Google Play filters to target devices

- Google Play filters target specific sensor configurations
  - Filter app from devices that don't have sensor configuration
  - `<uses-feature>` in Android manifest

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

Detecting sensors at runtime

- Detect sensors at runtime to turn off app features as appropriate
- Use `getDefaultSensor()` and pass in type constant for specific sensor such as `TYPE_PROXIMITY`, `TYPE_GYROSCOPE`, or `TYPE_GRAVITY`
- If there are more than one sensor for a given type, system designates one as default
- If none of that type exist, method returns `null`

Monitoring sensor events

Register listener for sensor event

- App must register listener for sensor event
- Register in activity `onStart()` and unregister in `onStop()`
  - **Don't** register in `onCreate()`, `onResume()`, or `onPause()`
  - Ensures sensors use power only when app is in foreground
  - Sensors continue running even if app is in multi-window mode

Register listener in onStart()

- Register sensor event listener for specific sensor

```
@Override
protected void onStart() {
    super.onStart();
    if (mIsLightSensorPresent) {
        mSensorManager.registerListener(this, mSensorLight,
                                        SensorManager.SENSOR_DELAY_UI);
    }
}
```

Unregister listener in onStop()

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

Monitor sensor events

1. Implement `SensorEventListener` interface with callbacks
   - `onSensorChanged(SensorEvent event)`
   - `onAccuracyChanged(Sensor sensor, int accuracy)`
2. Get sensor types and values from `SensorEvent` object
3. Update app accordingly

SensorEventListener and callbacks

```
public class SensorActivity extends Activity
                                    Implements SensorEventListener {
    // …
     @Override
     public void onSensorChanged(SensorEvent sensorEvent) {
         // Do something here if sensor data changes.
     }
     @Override
     public final void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Do something if sensor accuracy changes.
     }
}
```

onAccuracyChanged()

- `onAccuracyChanged()` called when a sensor accuracy changes
- `Sensor` object identifies sensor that changed accuracy
- Accuracy status constant:
  - `SENSOR_STATUS_ACCURACY_LOW`
  - `SENSOR_STATUS_ACCURACY_MEDIUM`
  - `SENSOR_STATUS_ACCURACY_HIGH`
  - `SENSOR_STATUS_UNRELIABLE`
  - `SENSOR_STATUS_NO_CONTACT`

onSensorChanged()

`onSensorChanged()` called when sensor reports new data, passing in a `SensorEvent`

A `SensorEvent` object contains information about the new sensor data

- `sensor`: Sensor that generated the event (`Sensor` object)
- `values`: Data that the sensor generated, as an array of `float` values. Different sensors provide different amounts and types of data.

Example: Changes to light sensor

```
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    int sensorType = sensorEvent.sensor.getType();
    float currentValue = sensorEvent.values[0];

    if (sensorType == Sensor.TYPE_LIGHT) {
        // Get light sensor string and fill data placeholder.
        mTextSensorLight.setText(getResources().getString(
            R.string.label_light, currentValue));
    }
}
```

What's next?

- Concept chapter: 3.1 Sensor basics
- Practical: 3.1 Working with sensor data

END