Hardware and software sensors

Sensors can be hardware- or software-based. Hardware-based sensors are physical components built into a handset or tablet device. Hardware sensors derive their data by directly measuring specific environmental properties, such as acceleration, geomagnetic field strength, or angular change.

Software-based sensors are not physical devices, although they mimic hardware-based sensors. Software-based sensors derive their data from one or more of the hardware-based sensors and are sometimes called *virtual sensors* or *composite sensors*. The proximity sensor and step counter sensors are examples of software-based sensors.

The Android sensor framework

You can access available sensors and acquire raw sensor data in your app with the Android sensor framework. The sensor framework, part of the `android.hardware` package, provides classes and interfaces to help you perform a wide variety of sensor-related tasks. With the Android sensor framework, you can:

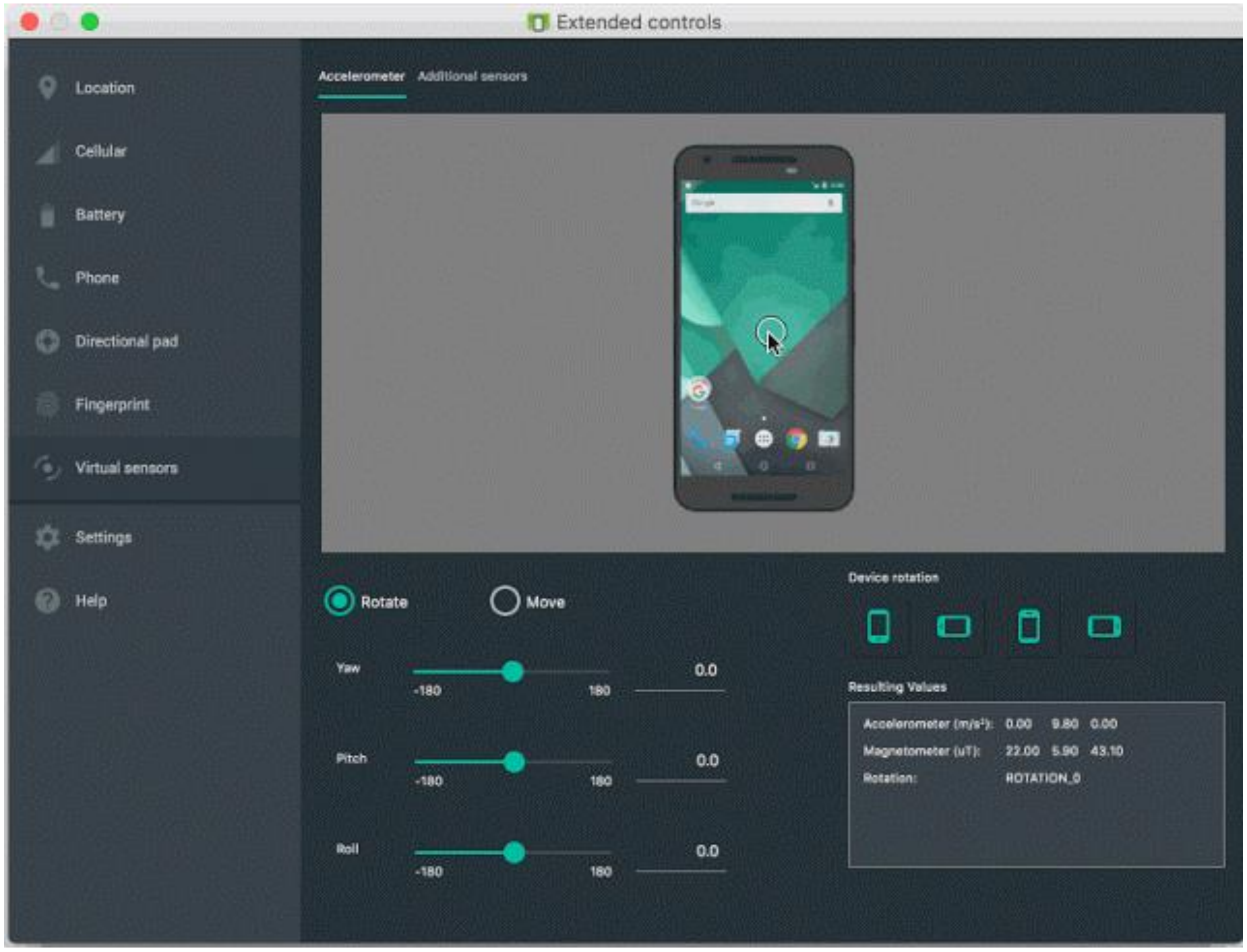
- Determine which sensors are available on a device.
- Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.
- Acquire raw sensor data and define the minimum rate at which you acquire sensor data.
- Register and unregister sensor event listeners that monitor sensor changes.

The following classes are the key parts of the Android sensor framework:

- `SensorManager`: Represents the Android sensor service. This class provides methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants. The constants are used to represent sensor accuracy, data acquisition rates, and sensor calibration.
- `Sensor`: Represents a specific sensor. This class provides methods that let you determine sensor's capabilities.
- `SensorEvent`: Represents information about a sensor event. A sensor event includes the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.
- `SensorEventListener`: An interface that includes callback methods to receive notifications (sensor events) when a sensor has new data or when sensor accuracy changes.

Sensors and the Android emulator

The Android emulator includes a set of virtual sensor controls that let you to test sensors such as the accelerometer, the ambient temperature sensor, the magnetometer, the proximity sensor, the light sensor, and more.



Yaw: 0.0
Pitch: 0.0
Roll: 0.0

```
Accelerometer (m/s²):  0.00   9.80   0.00
Magnetometer (uT):     22.00  5.90   43.10
Rotation:              ROTATION_0
```

The **Accelerometer** tab lets you test your app against changes in device position, orientation, or both. For example, you can simulate device motion such as tilt and rotation. The control simulates the way accelerometers and magnetometers respond when you move or rotate a real device. As you adjust the device in the emulators, the **Resulting Values** fields change accordingly. These fields show the values that an app can access.

The **Additional sensors** tab can simulate various position and environment sensors. In this tab you adjust the following sensors to test them with your app:

- Ambient temperature: This environmental sensor measures ambient air temperature.
- Magnetic field: This position sensor measures the ambient magnetic field at the *x* -axis, *y* -axis, and *z* -axis. The values are in microtesla (μT).

- Proximity: This position sensor measures the distance of the device from an object. For example, this sensor can notify a phone that a face is close to the phone to make a call.
- Light: This environmental sensor measures illuminance.
- Pressure: This environmental sensor measures ambient air pressure.
- Relative humidity: This environmental sensor measures ambient relative humidity.

Discovering sensors and sensor capabilities

The Android sensor framework lets you determine at runtime which sensors are available on a device. The framework also provides methods that let you determine the capabilities of a sensor, such as its maximum range, its resolution, and its power requirements.

Identifying sensors

To identify the device sensors you must first access the sensor manager, an Android system service. Create an instance of the `SensorManager` class by calling the `getSystemService()` method and passing in the `SENSOR_SERVICE` argument.

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

To get a listing of all the device sensors, use the `getSensorList()` method and the sensor type `TYPE_ALL`.

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

To list all of the sensors of a specific type, use another sensor type constant, for example `TYPE_PROXIMITY`, `TYPE_GYROSCOPE`, or `TYPE_GRAVITY`.

To determine whether a specific type of sensor exists on a device, and to get a `Sensor` object that represents that sensor, use the `getDefaultSensor()` method and pass in the type constant for a specific sensor. If a device has more than one sensor of a given type, the system designates one of the sensors as the default sensor. If a default sensor does not exist for a given type of sensor, the method call returns `null`, which means the device does not have that type of sensor.

For example, the following code checks whether there's a magnetometer on a device:

```
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.
}
```