

# CS 4518 Mobile and Ubiquitous Computing

## Lecture 11: Maps & Sensors

**Emmanuel Agu**



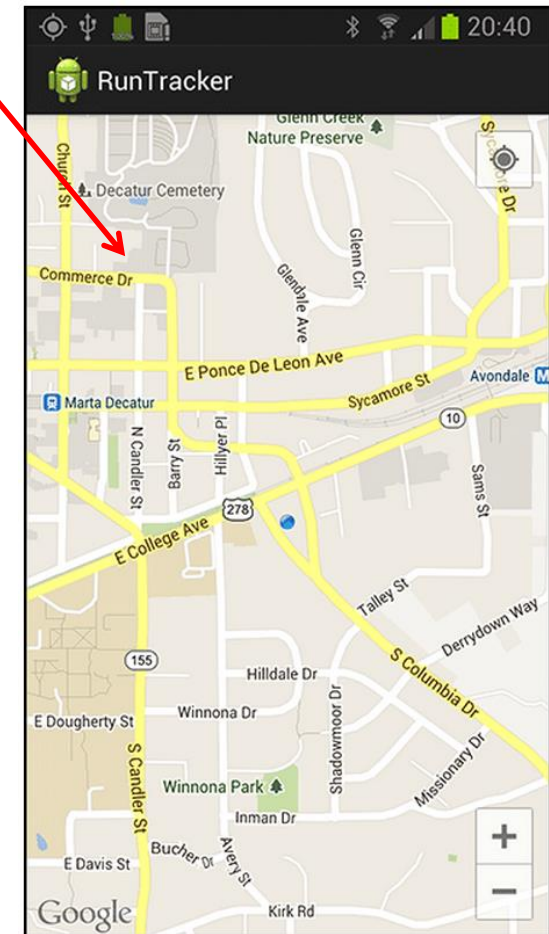


# Using Maps



# MapView and MapActivity

- **MapView:** UI widget that displays maps
- **MapActivity:** java class (extends Activity), handles map-related lifecycle and management for displaying maps.





# 7 Steps for using Google Maps Android API v2

<https://developers.google.com/maps/documentation/android-api/start>

1. Install Android SDK (Done already in zoolab!)
  - <https://developer.android.com/studio/index.html>
2. Add Google Play services to Android Studio
3. Create a Google Maps project
4. Obtain Google Maps API key
5. Hello Map! Take a look at the code
6. Connect an Android device
7. Build and run your app

# Step 2: Add Google Play Services to Android Studio

<https://developers.google.com/maps/documentation/android-api/start>



- Google Maps API v2 is part of Google Play Services SDK
- Use Android Studio SDK manager to download Google Play services

**Open SDK Manager**  
**Click on SDK Tools**

**Check Google Play Services, then Ok**

Default Settings

Appearance & Behavior > System Settings > Android SDK

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: C:\Users\emmanuel\AppData\Local\Android\Sdk [Edit](#)

SDK Platforms | **SDK Tools** | SDK Update Sites

Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

Name	Version	Status
<input checked="" type="checkbox"/> Android SDK Build-Tools		Installed
<input type="checkbox"/> CMake		Not Installed
<input type="checkbox"/> LLDB		Not Installed
<input type="checkbox"/> Android Auto API Simulators	1	Not installed
<input type="checkbox"/> Android Auto Desktop Head Unit Simulator	1.1	Not installed
<input checked="" type="checkbox"/> Android SDK Platform-Tools 25.0.3	25.0.3	Installed
<input checked="" type="checkbox"/> Android SDK Tools 25.2.5	25.2.5	Installed
<input checked="" type="checkbox"/> Documentation for Android SDK	1	Installed
<input type="checkbox"/> GPU Debugging tools	1.0.3	Not installed
<input type="checkbox"/> GPU Debugging tools	3.1.0	Not installed
<input type="checkbox"/> Google Play APK Expansion library	1	Not installed
<input type="checkbox"/> Google Play Billing Library	5	Not installed
<input type="checkbox"/> Google Play Licensing Library	1	Not installed
<input type="checkbox"/> Google Play services	38	Not installed
<input checked="" type="checkbox"/> Google USB Driver	11	Installed
<input type="checkbox"/> Google Web Driver	2	Not installed
<input checked="" type="checkbox"/> Intel x86 Emulator Accelerator (HAXM installer)	6.0.5	Installed
<input type="checkbox"/> NDK	13.1.3345770	Not installed
<input type="checkbox"/> Support Repository		
<input type="checkbox"/> ConstraintLayout for Android		Not installed

[Launch Standalone SDK Manager](#)

Show Package Details

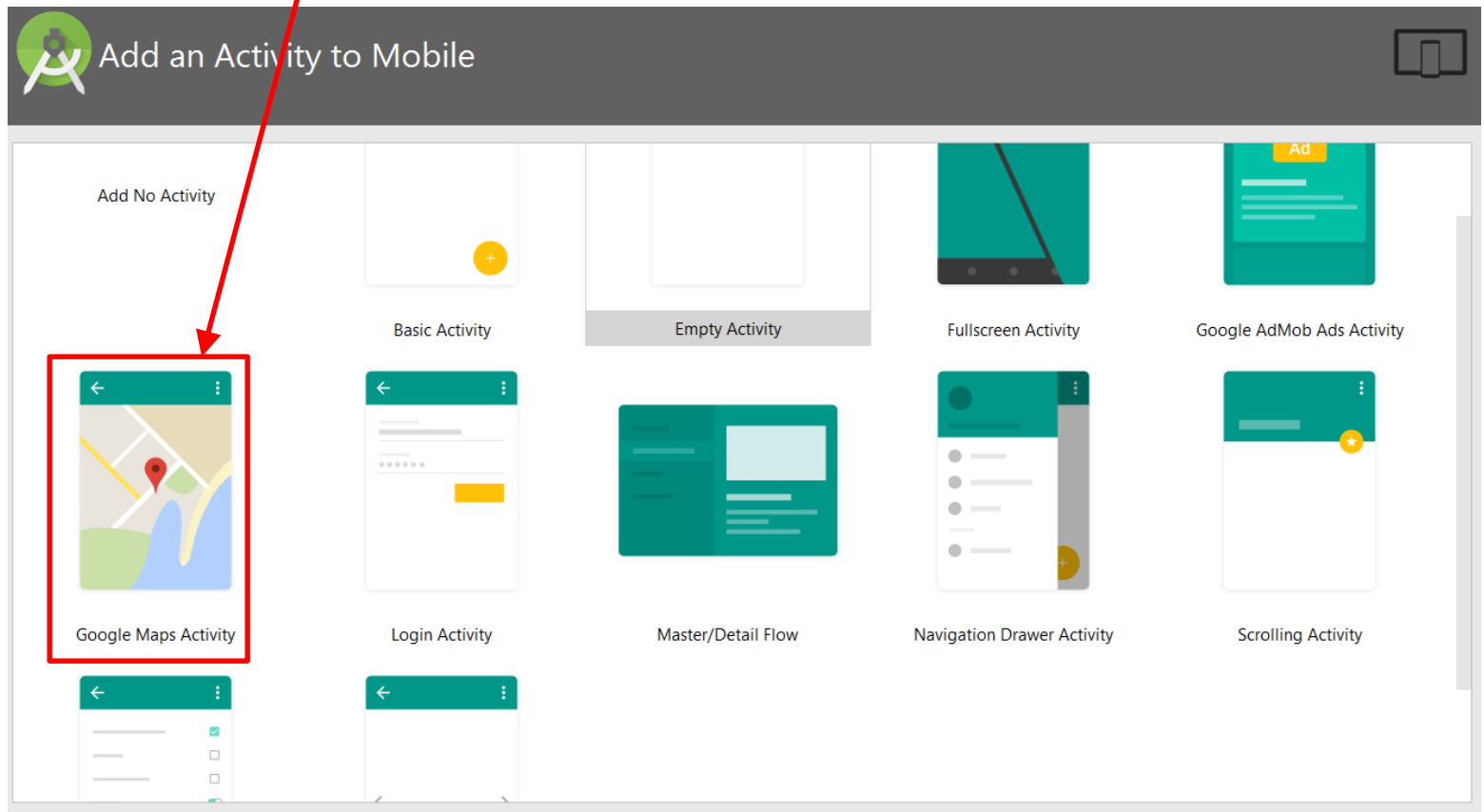
OK Cancel Apply Help

# Step 3: Create new Android Studio Project

<https://developers.google.com/maps/documentation/android-api/start>



- Select “Google Maps Activity, click Finish



# Step 4: Get Google Maps API key

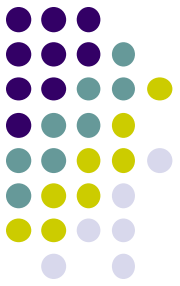
<https://developers.google.com/maps/documentation/android-api/start>



- To access Google Maps servers using Maps API, must add Maps API key to app
- Maps API key is free
- Android apps use Android-restricted API key
- **Background:** Before they can be installed, android apps must be signed with digital certificate (developer holds private key)
- Digital certificates uniquely identify an app, used in tracking:
  - Apps within Google Play Store and
  - App's use of resources such as Google Map servers

# Step 4a: Fast, Easy way to get Maps API Key

<https://developers.google.com/maps/documentation/android-api/start>



- Copy link provided in **google\_maps\_api.xml** of Maps template into browser
- Goes to Google API console, auto-fills form
- Creates Android-restricted API key

Register your application for Google Maps Android API in Google API Console

Google API Console allows you to manage your application and monitor API usage.

You have no existing projects. A new project named "My Project" will be created.

Please email me updates regarding feature announcements, performance suggestions, feedback surveys and special offers.

Yes  No

I agree that my use of any [services and related APIs](#) is subject to my compliance with the applicable [Terms of Service](#).

Yes  No

Agree and continue



The API is enabled

The project has been created and Google Maps Android API has been enabled.

Next, you'll need to create an API key in order to call the API.

Create API key





# Step 4a: Fast, Easy way to get Maps API Key

<https://developers.google.com/maps/documentation/android-api/start>

- If successful, Maps API key generated

API key created

Use this key in your application by passing it with the `key=API_KEY` parameter.

Your API key

AIzaSyCc0\_1EEjP11TLnPkVsX10YIY7oBa9XsXs

⚠ Restrict your key to prevent unauthorized use in production.

CLOSE

RESTRICT KEY

- Copy key, put it in `<string>` element in `google_maps_api.xml` file

```
<string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">AIzaSyCc0_1EEjP11TLnPkVsX10YIY7oBa9XsXs</string>
```



## Step 4b: Longer (older) way to API key

- If easy way doesn't work, older way to obtain a Maps API key
- Follow steps at:
  - See: <https://developers.google.com/maps/documentation/android-api/signup>

# Step 5: Examine Code Generated by Android Studio Maps Template



- XML file that defines layout is in **res/layout/activity\_maps.xml**

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/map"
  tools:context=".MapsActivity"
  android:name="com.google.android.gms.maps.SupportMapFragment" />
```

# Step 5: Examine Code Generated by Android Studio Maps Template



- Default Activity file is **MapActivity.java**

```
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
            .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }

    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        // Add a marker in Sydney, Australia, and move the camera.
        LatLng sydney = new LatLng(-34, 151);
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }
}
```

# Steps 6, 7



- **Step 6:** Connect to an Android device (smartphone)
- **Step 7:** Run the app
  - Should show map with a marker on Sydney Australia
- More code examples at:
  - <https://github.com/googlemaps/android-samples>



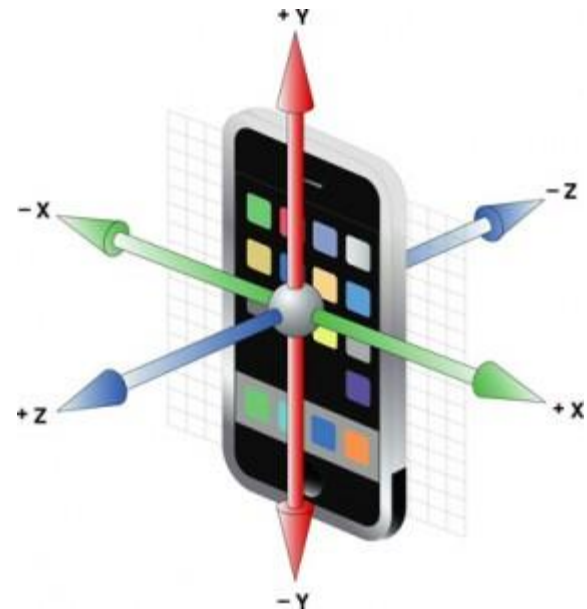
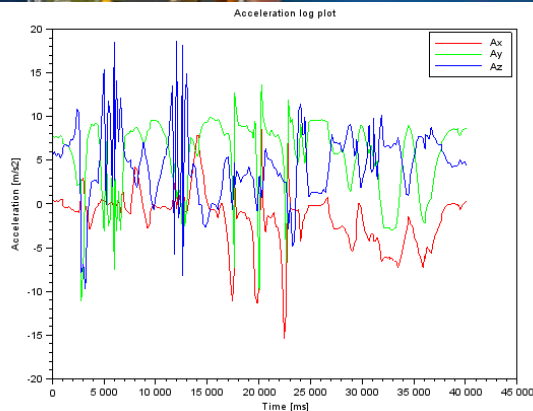
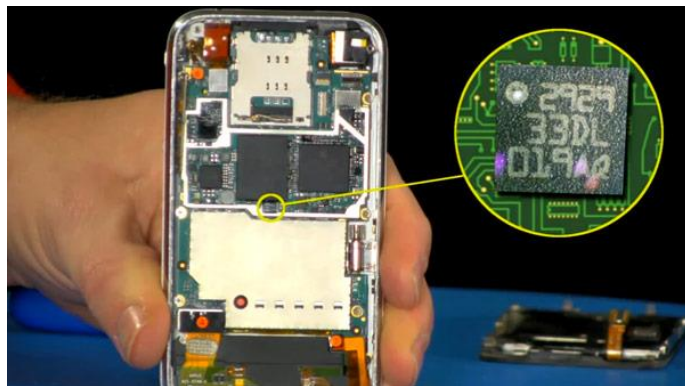


# Android Sensors



# What is a Sensor?

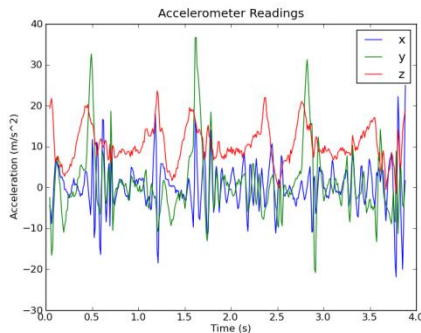
- Converts physical quantity (e.g. light, acceleration, magnetic field) into a signal
- **Example:** accelerometer converts acceleration along X,Y,Z axes into signal



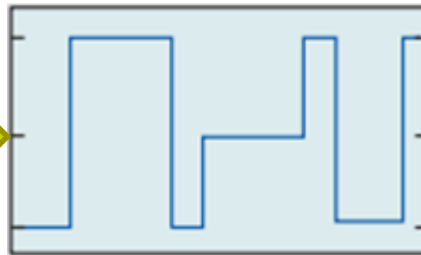


# So What?

- Raw sensor data can be processed into useful info
- **Example:** Raw accelerometer data can be processed/classified to infer user's activity (e.g. walking running, etc)
- Audio samples can be processed/classified to infer stress level in speaker's voice



Raw accelerometer readings



Machine learning  
Feature extraction  
and classification



Walking  
Running  
Jumping  
Step count  
Calories burned  
Falling





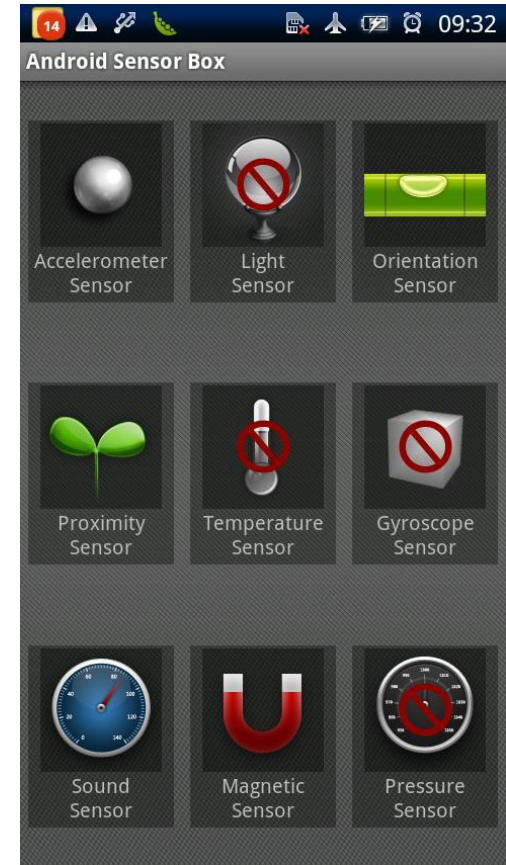


# Android Sensors

- Microphone (sound)
  - Camera
  - Temperature
  - Location (GPS, A-GPS)
  - Accelerometer
  - Gyroscope (orientation)
  - Proximity
  - Pressure
  - Light
- **Different phones do not have all sensor types!!**



**AndroSensor**



**Android Sensor Box**

# Android Sensor Framework

[http://developer.android.com/guide/topics/sensors/sensors\\_overview.html](http://developer.android.com/guide/topics/sensors/sensors_overview.html)



- Enables apps to:
  - Access sensors available on device and
  - Acquire raw sensor data
  
- Specifically, using the Android Sensor Framework, you can:
  - Determine **which sensors** are available on phone
  - Determine **capabilities of sensors** (e.g. max. range, manufacturer, power requirements, resolution)
  - **Register and unregister** sensor event listeners
  - **Acquire raw sensor data** and define data rate

[http://developer.android.com/guide/topics/sensors/sensors\\_overview.html](http://developer.android.com/guide/topics/sensors/sensors_overview.html)

# Android Sensor Framework

[http://developer.android.com/guide/topics/sensors/sensors\\_overview.html](http://developer.android.com/guide/topics/sensors/sensors_overview.html)



- Android sensors can be either hardware or software
- **Hardware sensor:**
  - physical components built into phone,
  - **Example:** temperature
- **Software sensor (or virtual sensor):**
  - Not physical device
  - Derives their data from one or more hardware sensors
  - **Example:** gravity sensor

# Sensor Types Supported by Android



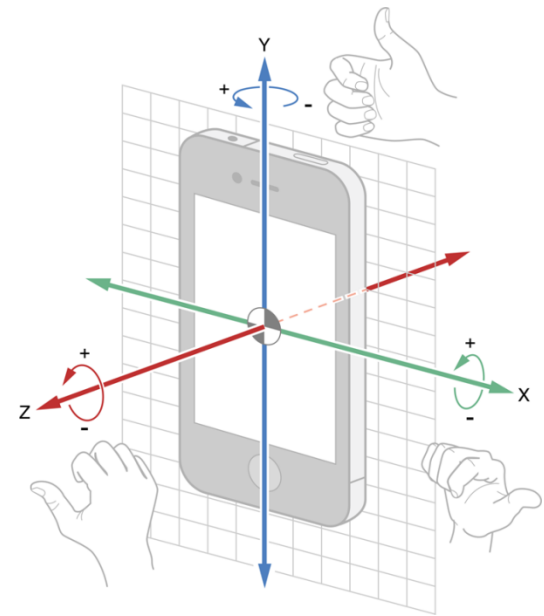
- TYPE\_PROXIMITY

- Measures an **object's proximity to device's screen**
- **Common uses:** determine if handset is held to ear



- TYPE\_GYROSCOPE

- Measures device's **rate of rotation** around X,Y,Z axes in rad/s
- **Common uses:** rotation detection (spin, turn, etc)



# Types of Sensors



Sensor	HW/SW	Description	Use
TYPE_ACCELEROMETER	HW	Rate of change of velocity	Shake, Tilt
TYPE_AMBIENT_TEMPERATURE	HW	Room temperature	Monitor Room temp
TYPE_GRAVITY	SW/HW	Gravity along X,Y,Z axes	Shake, Tilt
TYPE_GYROSCOPE	HW	Rate of rotation	Spin, Turn
TYPE_LIGHT	HW	Illumination level	Control Brightness
TYPE_LINEAR_ACCELERATION	SW/HW	Acceleration along X,Y,Z – g	Accel. Along an axis
TYPE_MAGNETIC_FIELD	HW	Magnetic field	Create Compass
TYPE_ORIENTATION	SW	Rotation about X,Y,Z axes	Device position
TYPE_PRESSURE	HW	Air pressure	Air pressure
TYPE_PROXIMITY	HW	Any object close to device?	Phone close to face?
TYPE_RELATIVE_HUMIDITY	HW	% of max possible humidity	Dew point
TYPE_ROTATION_VECTOR	SW/HW	Device's rotation vector	Device's orientation
TYPE_TEMPERATURE	HW	Phone's temperature	Monitor temp

## 2 New Hardware Sensor in Android 4.4



- TYPE\_STEP\_DETECTOR
  - Triggers sensor event each time user takes a step
  - Delivered event has value of 1.0 + timestamp of step
  
- TYPE\_STEP\_COUNTER
  - Also triggers a sensor event each time user takes a step
  - Delivers total ***accumulated number of steps since this sensor was first registered by an app,***
  - Tries to eliminate false positives
- **Common uses:** Both used in step counting, pedometer apps
- Requires hardware support, available in Nexus 5
- Alternatively available through Google Play Services (more later)



# Sensor Programming

- Sensor framework is part of **android.hardware**
- Classes and interfaces include:
  - **SensorManager**
  - **Sensor**
  - **SensorEvent**
  - **SensorEventListener**
- These sensor-APIs used for 2 main tasks:
  - Identifying sensors and sensor capabilities
  - Monitoring sensor events

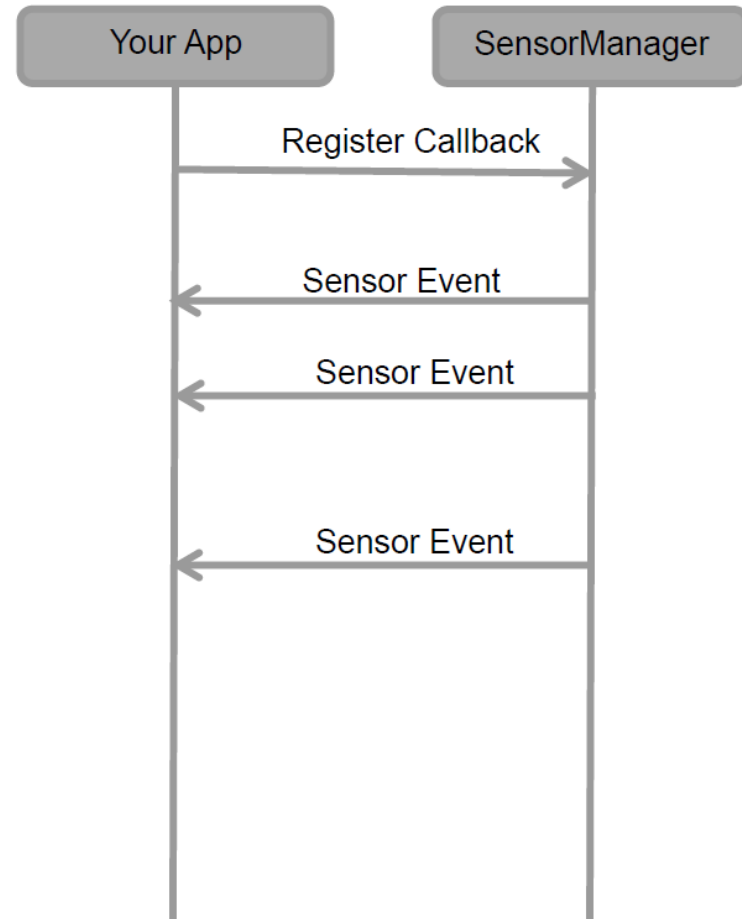
# Sensor Events and Callbacks



- App sensors send events asynchronously, when new data arrives



- General approach:
  - App registers callbacks
  - **SensorManager** notifies app of sensor event whenever new data arrives (or accuracy changes)





# Sensor

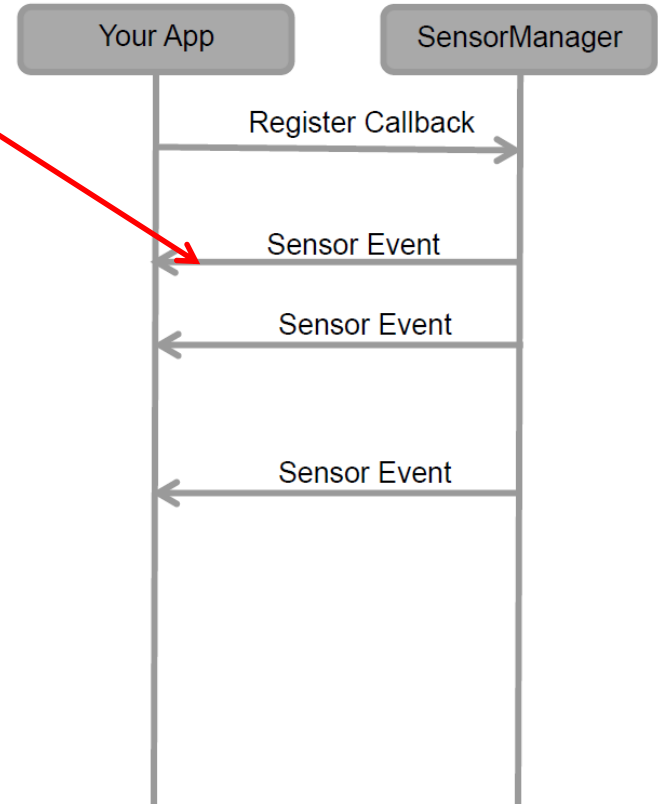


- A class that can be used to create instance of a specific sensor
- Has methods used to determine a sensor's capabilities

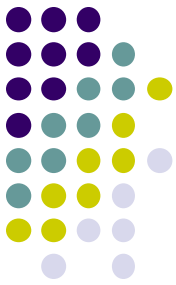


# SensorEvent

- Android system sensor event information as a **sensor event object**
- **Sensor event object** includes:
  - **Sensor:** Type of sensor that generated the event
  - **Values:** Raw sensor data
  - **Accuracy:** Accuracy of the data
  - **Timestamp:** Event timestamp



Sensor	Sensor event data	Description	Units of measure
TYPE_ACCELEROMETER	<code>SensorEvent.values[0]</code>	Acceleration force along the x axis (including gravity).	$m/s^2$
	<code>SensorEvent.values[1]</code>	Acceleration force along the y axis (including gravity).	
	<code>SensorEvent.values[2]</code>	Acceleration force along the z axis (including gravity).	
TYPE_GRAVITY	<code>SensorEvent.values[0]</code>	Force of gravity along the x axis.	$m/s^2$
	<code>SensorEvent.values[1]</code>	Force of gravity along the y axis.	
	<code>SensorEvent.values[2]</code>	Force of gravity along the z axis.	
TYPE_GYROSCOPE	<code>SensorEvent.values[0]</code>	Rate of rotation around the x axis.	rad/s
	<code>SensorEvent.values[1]</code>	Rate of rotation around the y axis.	
	<code>SensorEvent.values[2]</code>	Rate of rotation around the z axis.	
TYPE_GYROSCOPE_UNCALIBRATED	<code>SensorEvent.values[0]</code>	Rate of rotation (without drift compensation) around the x axis.	rad/s
	<code>SensorEvent.values[1]</code>	Rate of rotation (without drift compensation) around the y axis.	
	<code>SensorEvent.values[2]</code>	Rate of rotation (without drift compensation) around the z axis.	
	<code>SensorEvent.values[3]</code>	Estimated drift around the x axis.	
	<code>SensorEvent.values[4]</code>	Estimated drift around the y axis.	
	<code>SensorEvent.values[5]</code>	Estimated drift around the z axis.	



# Sensor Values Depend on Sensor Type

# Sensor Values Depend on Sensor Type



Sensor	Sensor event data	Description	Units of measure
TYPE_LINEAR_ACCELERATION	<code>SensorEvent.values[0]</code>	Acceleration force along the x axis (excluding gravity).	m/s <sup>2</sup>
	<code>SensorEvent.values[1]</code>	Acceleration force along the y axis (excluding gravity).	
	<code>SensorEvent.values[2]</code>	Acceleration force along the z axis (excluding gravity).	
TYPE_ROTATION_VECTOR	<code>SensorEvent.values[0]</code>	Rotation vector component along the x axis ( $x * \sin(\theta/2)$ ).	Unitless
	<code>SensorEvent.values[1]</code>	Rotation vector component along the y axis ( $y * \sin(\theta/2)$ ).	
	<code>SensorEvent.values[2]</code>	Rotation vector component along the z axis ( $z * \sin(\theta/2)$ ).	
	<code>SensorEvent.values[3]</code>	Scalar component of the rotation vector ( $(\cos(\theta/2))$ ). <sup>1</sup>	
TYPE_SIGNIFICANT_MOTION	N/A	N/A	N/A
TYPE_STEP_COUNTER	<code>SensorEvent.values[0]</code>	Number of steps taken by the user since the last reboot while the sensor was activated.	Steps
TYPE_STEP_DETECTOR	N/A	N/A	N/A



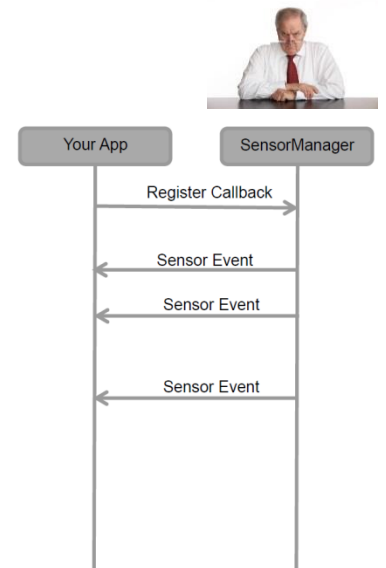
# SensorEventListener

- Interface used to create 2 callbacks that receive notifications (sensor events) when:
  - Sensor values change (**onSensorChange( )**) or
  - When sensor accuracy changes (**onAccuracyChanged( )**)



# SensorManager

- A class that provides methods for:
  - Accessing and listing sensors
  - Registering and unregistering sensor event listeners
- Can be used to create instance of sensor service
- Also provides sensor **constants** used to:
  - Report sensor accuracy
  - Set data acquisition rates
  - Calibrate sensors

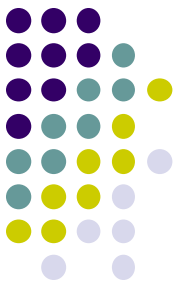




# Sensor API Tasks

- **Sensor API Task 1: Identifying sensors and their capabilities**
- Why identify sensor and their capabilities at runtime?
  - Disable app features using sensors not present, or
  - Choose sensor implementation with best performance
- **Sensor API Task 2: Monitor sensor events**
- Why monitor sensor events?
  - To acquire raw sensor data
  - Sensor event occurs every time sensor detects change in parameters it is measuring

# Sensor Availability

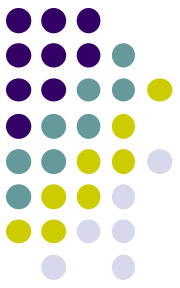


- Different sensors are available on different **Android versions**

Sensor	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 <sup>1</sup>	n/a <sup>1</sup>
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 <sup>2</sup>	Yes <sup>2</sup>	Yes <sup>2</sup>	Yes
TYPE_PRESSURE	Yes	Yes	n/a <sup>1</sup>	n/a <sup>1</sup>
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 <sup>2</sup>	Yes	Yes	Yes



# Identifying Sensors and Sensor Capabilities



- First create instance of **SensorManager** by calling **getSystemService( )** and passing in **SENSOR\_SERVICE** argument

```
private SensorManager mSensorManager;
```

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

- Then list sensors available on device by calling **getSensorList( )**

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

- To list particular type, use **TYPE\_GYROSCOPE, TYPE\_GRAVITY, etc**

[http://developer.android.com/guide/topics/sensors/sensors\\_overview.html](http://developer.android.com/guide/topics/sensors/sensors_overview.html)



# Determining if Device has at least one of particular Sensor Type

- Device may have multiple sensors of a particular type.
  - E.g. multiple magnetometers
- If multiple sensors of a given type exist, one of them must be designated “the default sensor” of that type
- To determine if specific sensor type exists use **getDefaultSensor( )**
- **Example:** To check whether device has at least one magnetometer

```
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.
}
```

A red arrow originates from the text 'getDefaultSensor( )' in the list above and points to the corresponding method call in the code snippet.



# Determining Capabilities of Sensors

- Some useful methods of **Sensor** class methods:
  - **getResolution( )**: get sensor's resolution
  - **getMaximumRange( )**: get maximum measurement range
  - **getPower( )**: get sensor's power requirements
  - **getMinDelay( )**: min time interval (in microseconds) sensor can use to sense data. Return values:
    - **0 value**: Non-streaming sensor, reports data only if sensed parameters change
    - **Non-zero value**: streaming sensor



# Monitoring Sensor Events

- To monitor raw sensor data, 2 callback methods exposed through **SensorEventListener** interface need to be implemented:
- **onSensorChanged:**
  - Invoked by Android system to report new sensor value
  - Provides **SensorEvent** object containing information about new sensor data (accuracy, sensor, timestamp, data)
- **onAccuracyChanged:**
  - invoked when accuracy of sensor being monitored changes



# Example: Monitoring Light Sensor Data

- **Goal:** Monitor light sensor data using `onSensorChanged()`, display it in a `TextView` defined in `main.xml`

```
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.
    }
}
```

**Create instance of Sensor manager**

**Get default Light sensor**

# Example: Monitoring Light Sensor Data (Contd)



```
@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.
}
```

Get new light sensor value

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

Register sensor when app becomes visible

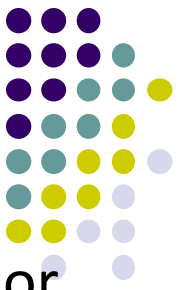
```
@Override
protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
}
}
```

Unregister sensor if app is no longer visible to reduce battery drain



# Handling Different Sensor Configurations

- Different phones have different sensors built in
- **E.g.** Motorola Xoom has pressure sensor, Samsung Nexus S doesn't
- If app uses a specific sensor, how to ensure this sensor exists on target device? Two options
  - **Option 1:** Detect device sensors at runtime, enable/disable app features as appropriate
  - **Option 2:** Use Google Play filters so only devices possessing required sensor can download app



# Option 1: Detecting Sensors at Runtime

- Following code checks if device has at least one pressure sensor

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



## Option 2: Use Google Play Filters to Target Specific Sensor Configurations



- Can use **<uses-feature>** element in AndroidManifest.xml to filter your app from devices without required sensors
- **Example:** following manifest entry ensures that only devices with accelerometers will see this app on Google Play

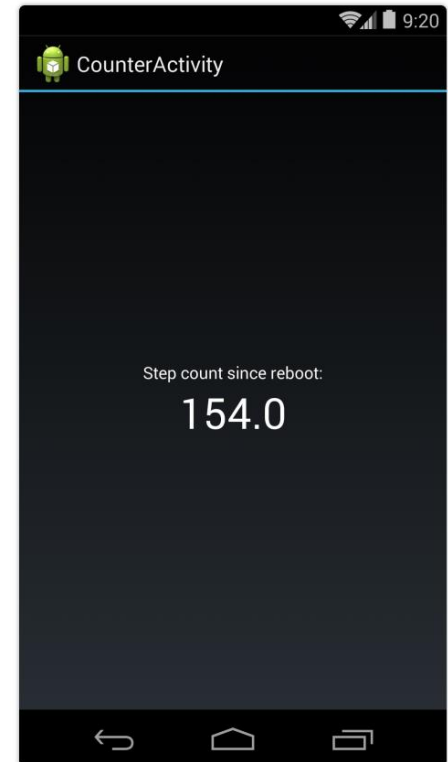
```
<uses-feature android:name="android.hardware.sensor.accelerometer"  
            android:required="true" />
```

# Example Step Counter App



- **Goal:** Track user's steps, display it in TextView
- **Note:** Phone hardware must support step counting

```
1 package com.starboardland.pedometer;
2
3 import android.app.Activity;
4 import android.content.Context;
5 import android.hardware.*;
6 import android.os.Bundle;
7 import android.widget.TextView;
8 import android.widget.Toast;
9
10 public class CounterActivity extends Activity implements SensorEventListener {
11
12     private SensorManager sensorManager;
13     private TextView count;
14     boolean activityRunning;
15
16     @Override
17     public void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.main);
20         count = (TextView) findViewById(R.id.count);
21
22         sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
23     }
```





# Example Step Counter App (Contd)

```
25     @Override
26     protected void onResume() {
27         super.onResume();
28         activityRunning = true;
29         Sensor countSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
30         if (countSensor != null) {
31             sensorManager.registerListener(this, countSensor, SensorManager.SENSOR_DELAY_UI);
32         } else {
33             Toast.makeText(this, "Count sensor not available!", Toast.LENGTH_LONG).show();
34         }
35     }
36
37
38     @Override
39     protected void onPause() {
40         super.onPause();
41         activityRunning = false;
42         // if you unregister the last listener, the hardware will stop detecting step events
43     //     sensorManager.unregisterListener(this);
44     }
```

<https://theelfismike.wordpress.com/2013/11/10/android-4-4-kitkat-step-detector-code/>

# Example Step Counter App (Contd)



```
46     @Override
47     public void onSensorChanged(SensorEvent event) {
48         if (activityRunning) {
49             count.setText(String.valueOf(event.values[0]));
50         }
51     }
52 }
53
54     @Override
55     public void onAccuracyChanged(Sensor sensor, int accuracy) {
56     }
57 }
```



# References

- Android Sensors Overview, [http://developer.android.com/guide/topics/sensors/sensors\\_overview.html](http://developer.android.com/guide/topics/sensors/sensors_overview.html)
- Busy Coder's guide to Android version 6.3
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014