Android Development

Lecture 10
Sensors & Media Management

Marco Picone - 2012

Lecture Summary

- Sensors
  - Intro
  - Description
  - Framework
- Android & Multimedia
  - Framework
  - Camera

Sensors

- The modern Smartphone provides more than just the ability to send and receive communication in different ways.
- The addition of external sensors that can report information about the environment made the phone more powerful and useful for users and developers.
- Physical sensors available on the phone may include Accelerometer, Gyroscopes, Magnetic Field, Light Sensor, Proximity Sensor, Temperature and Pressure Sensor.
- With some of the sensors, the measurements can be combined or collected and the evaluation can be made over the collected data to show complex and interesting measurement.

Sensors

- Available sensors on an Android device are capable of providing raw data with high precision and accuracy, and are useful if you want to monitor three-dimensional device movement or positioning, or you want to monitor changes in the ambient environment near a device.
- For example, a game might track readings from a device's gravity sensor to infer complex user gestures and motions, such as tilt, shake, rotation, or swing.
- A weather application might use a device's temperature sensor and humidity and a travel application might use the geomagnetic field sensor and accelerometer to report a compass bearing.
- The Android platform supports three broad categories of sensors:
  - **Motion sensors**: These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.
  - **Environmental sensors:** These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.
  - **Position sensors:** These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

Sensors

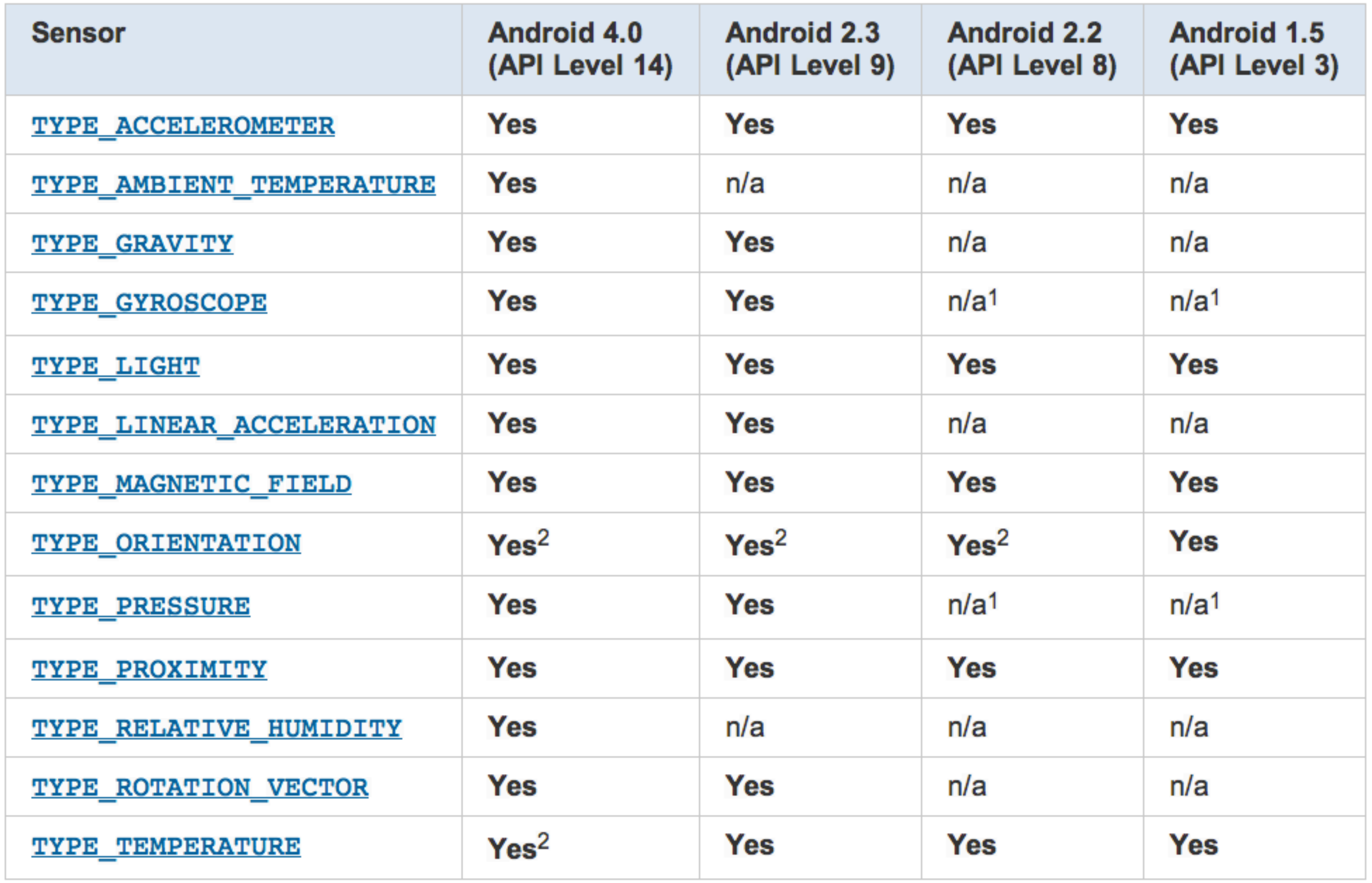
- The sensor framework provides several classes and interfaces that help you perform a wide variety of sensor-related tasks.
- For example, you can use the sensor framework to do the following:
  - Determine which sensors are available on a device.
  - Determine an individual sensor's capabilities, such 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.

Sensors

- The Android sensor framework lets you access many types of sensors. Some of these sensors are hardware-based and some are software-based.
- Hardware-based sensors are physical components built into a handset or tablet device. They 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 synthetic sensors. The linear acceleration sensor and the gravity sensor are examples of software-based sensors.
- Few Android-powered devices have every type of sensor. For example, most handset devices and tablets have an accelerometer and a magnetometer, but fewer devices have barometers or thermometers. Also, a device can have more than one sensor of a given type. For example, a device can have two gravity sensors, each one having a different range.

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	HW	Measures the acceleration force in m/s^2 that 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	HW	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	HW / SW	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	HW	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	HW	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	HW / SW	Measures the acceleration force in m/s2 that 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.
TYPE_MAGNETIC_FIELD	HW	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.
TYPE_ORIENTATION	SW	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the getRotationMatrix() method.	Determining device position.
TYPE_PRESSURE	HW	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	HW	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	HW	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	HW / SW	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	HW	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.

Sensors Availability



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[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

Sensor Framework

- You can access these sensors and acquire raw sensor data by using the Android sensor framework. The sensor framework is part of the android.hardware package and includes the following classes and interfaces:
- SensorManager: You can use this class to create an instance of the sensor service. This class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.
- Sensor: You can use this class to create an instance of a specific sensor. This class provides various methods that let you determine a sensor's capabilities.
- SensorEvent: The system uses this class to create a sensor event object, which provides information about a sensor event. A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.
- SensorEventListener: You can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

Sensor Framework

- In a typical application you use these sensor-related APIs to perform two basic tasks:
  - **Identifying sensors and sensor capabilities**: Identifying sensors and sensor capabilities at runtime is useful if your application has features that rely on specific sensor types or capabilities. For example, you may want to identify all of the sensors that are present on a device and disable any application features that rely on sensors that are not present. Likewise, you may want to identify all of the sensors of a given type so you can choose the sensor implementation that has the optimum performance for your application.
  - **Monitor sensor events:** Monitoring sensor events is how you acquire raw sensor data. A sensor event occurs every time a sensor detects a change in the parameters it is measuring. A sensor event provides you with four pieces of information:
    - the name of the sensor that triggered the event
    - the timestamp for the event
    - the accuracy of the event
    - the raw sensor data that triggered the event.

Identify Sensors & Capabilities

- The Android sensor framework provides several methods that make it easy for you to determine at runtime which sensors are on a device. The API also provides methods that let you determine the capabilities of each sensor, such as its maximum range, its resolution, and its power requirements.
- To identify the sensors that are on a device you first need to get a reference to the sensor service. To do this, you create an instance of the SensorManager class by calling the getSystemService() method and passing in the SENSOR_SERVICE argument.

```
private SensorManager mSensorManager;

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

Identify Sensors & Capabilities

- Next, you can get a listing of every sensor on a device by calling the getSensorList() method and using the TYPE_ALL constant.

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

- If you want to list all of the sensors of a given type, you could use another constant instead of TYPE_ALL such as TYPE_GYROSCOPE, TYPE_LINEAR_ACCELERATION, or TYPE_GRAVITY.

- You can also determine whether a specific type of sensor exists on a device by using the getDefaultSensor() method and passing in the type constant for a specific sensor. If a device has more than one sensor of a given type, one of the sensors must be designated 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.

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

Multiple Sensor Management

```
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.
  }
}
```

Manage multiple sensors for the same type on the device

Monitoring Sensor Events

- To monitor raw sensor data you need to implement two callback methods that are exposed through the SensorEventListener interface:
  - onAccuracyChanged(): A sensor's accuracy changes. The method provides you the reference to the Sensor object that changed and the new accuracy of the sensor. Accuracy is represented by one of four status constants:SENSOR_STATUS_ACCURACY_LOW, SENSOR_STATUS_ACCURACY_MEDIUM, SENSOR_STATUS_ACCURACY_HIGH, or SENSOR_STATUS_UNRELIABLE.
  - onSensorChanged(): A sensor reports a new value. The method provides you the new SensorEvent object. A SensorEvent object contains information about the new sensor data, including: the accuracy of the data, the sensor that generated the data, the timestamp at which the data was generated, and the new data that the sensor recorded.

Sensor Event Management

```
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);
  }
```

Retrieve the SensorManager instance and get the Light Sensor reference

```
  @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.
  }
```

Listener Methods to receive updates from registered sensors

```
 ...
```

Sensor Event Management

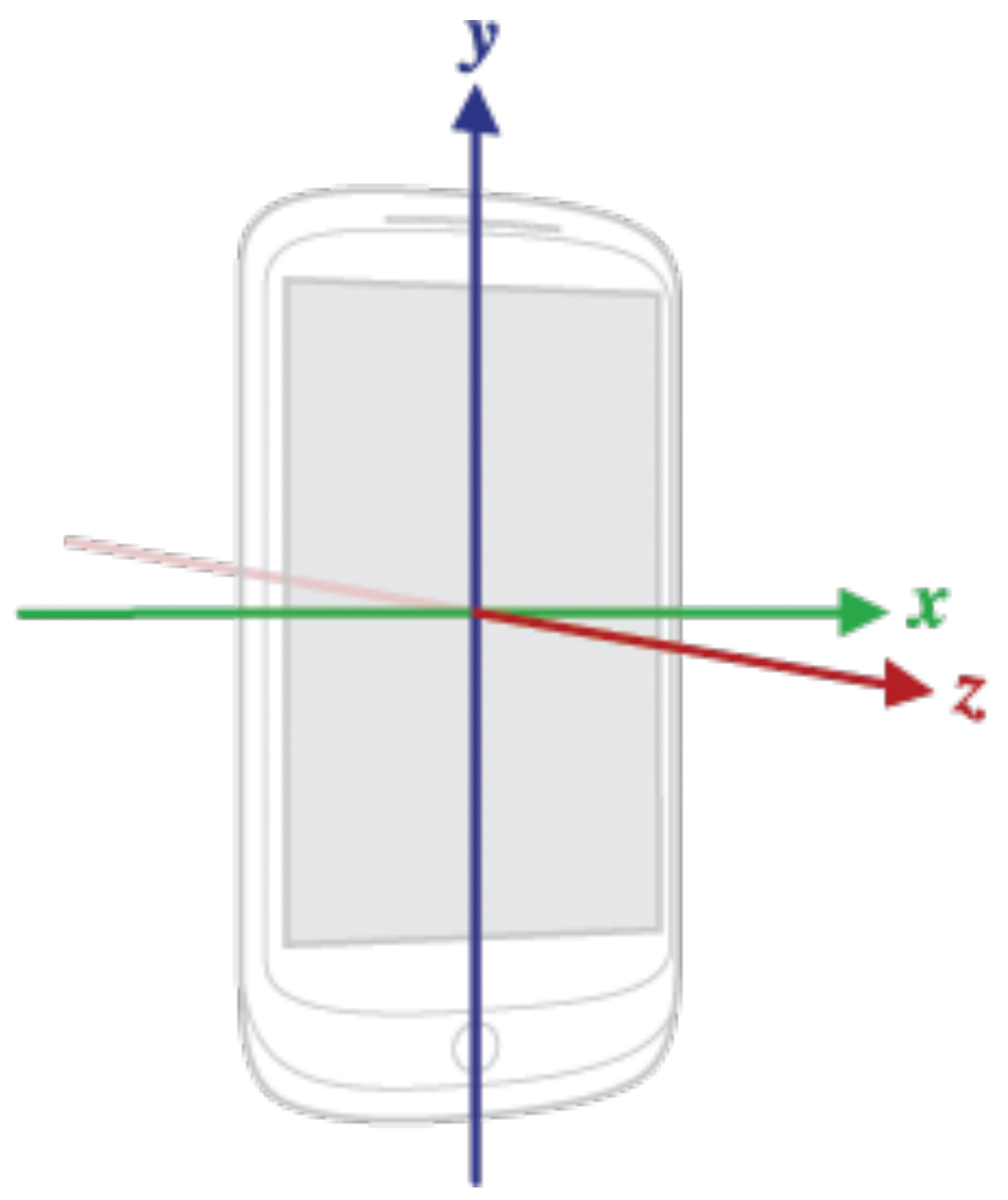
```
 @Override
  protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
  }

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

Register the Listener for a specific type of sensor. You can UnRegister the Listener for all sensors (as in the example) or for a specific type if it is not needed anymore.

The data delay (or sampling rate) controls the interval at which sensor events are sent to your application via the onSensorChanged() callback method. The default data delay is suitable for monitoring typical screen orientation changes and uses a delay of 200,000 microseconds. You can specify other data delays, such as SENSOR_DELAY_GAME (20,000 microsecond delay), SENSOR_DELAY_UI (60,000 microsecond delay), or SENSOR_DELAY_FASTEST (0 microsecond delay). As of Android 3.0 (API Level 11) you can also specify the delay as an absolute value (in microseconds).

Sensor Coordinate System

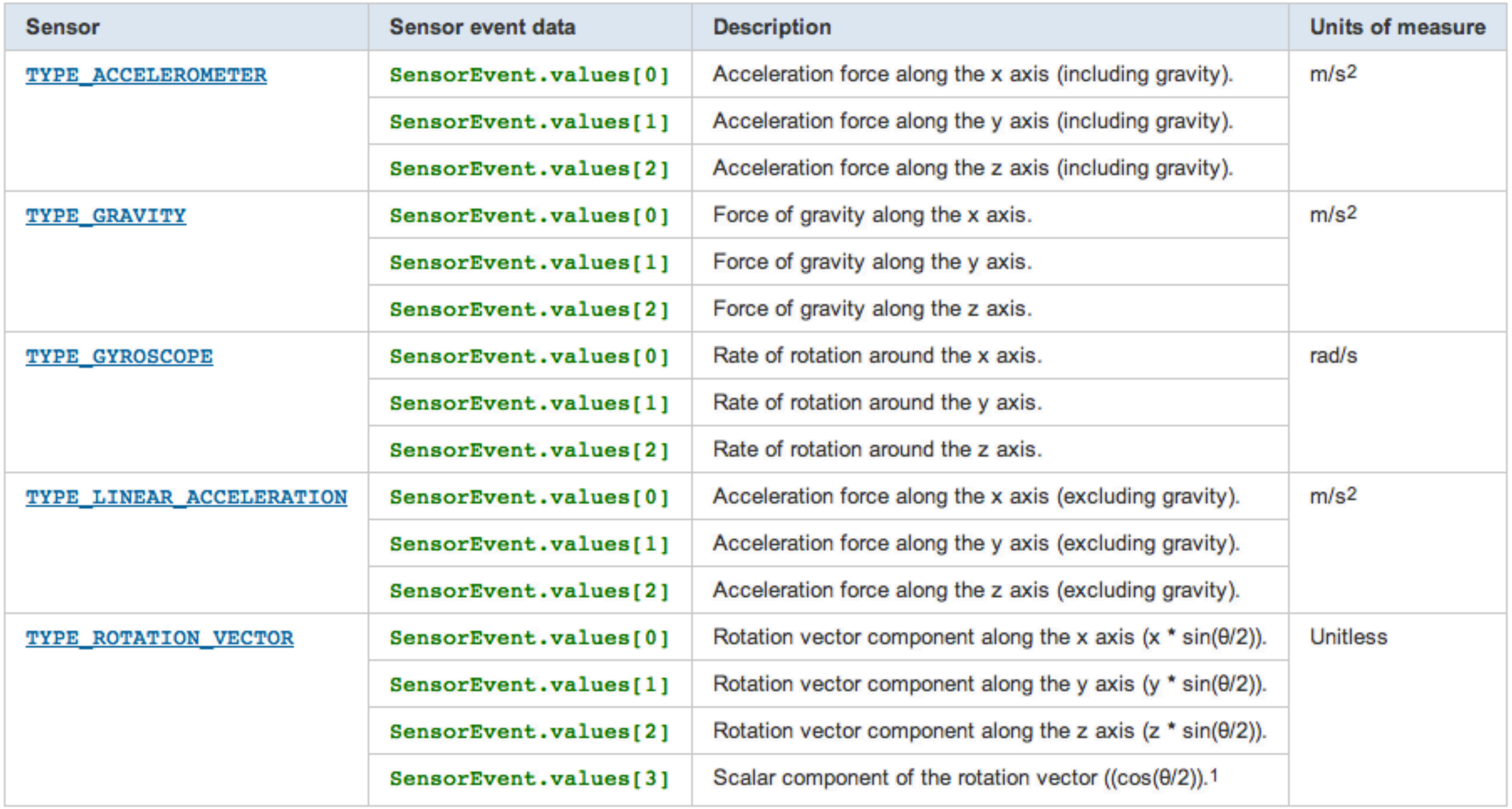
- In general, the sensor framework uses a **standard 3-axis coordinate system** to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation.
- When a device is in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values.
- This coordinate system is used by the following sensors:
  - Acceleration sensor
  - Gravity sensor
  - Gyroscope
  - Linear acceleration sensor
  - Geomagnetic field sensor



Sensor Coordinate System

- The most important points to understand are
  - the axes are not swapped when the device's screen orientation changes—that is, the sensor's coordinate system never changes as the device moves. This behavior is the same as the behavior of the OpenGL coordinate system.
  - your application must not assume that a device's natural (default) orientation is portrait. The natural orientation for many tablet devices is landscape. And the sensor coordinate system is always based on the natural orientation of a device.
- Finally, if your application matches sensor data to the on-screen display, you need to use the getRotation() method to determine screen rotation, and then use the remapCoordinateSystem() method to map sensor coordinates to screen coordinates. You need to do this even if your manifest specifies portrait-only display.
- For more information about the sensor coordinate system, including information about how to handle screen rotations: http://android-developers.blogspot.it/2010/09/one-screen-turn-deserves-another.html

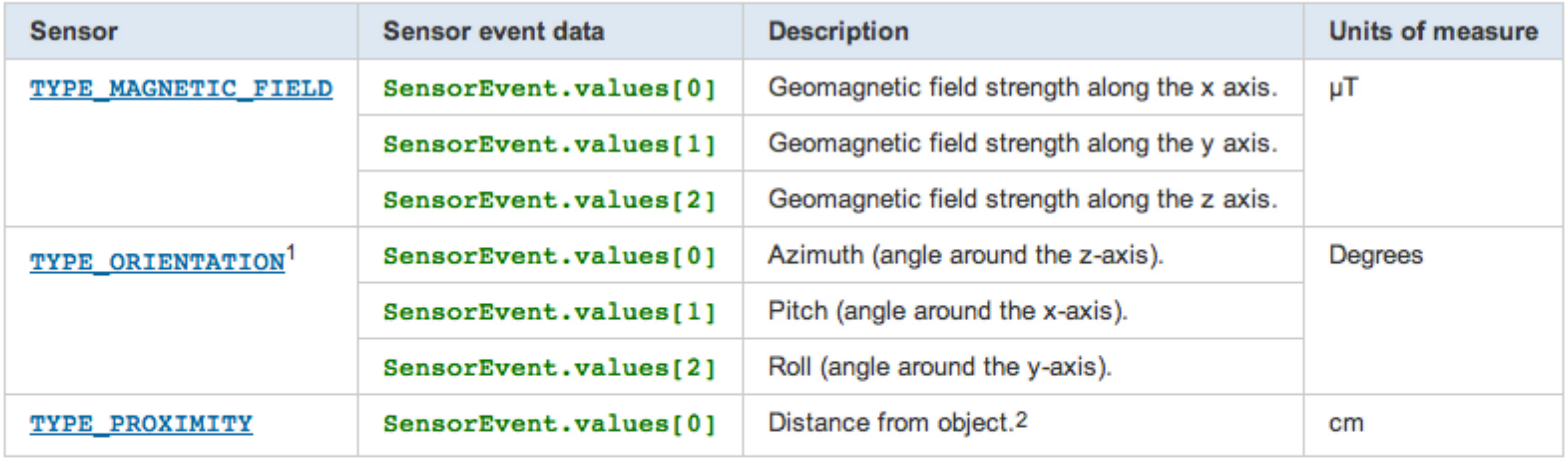
Motion Sensor



Sensor	Sensor event data	Description	Units of measure
TYPE_ACCELEROMETER	SensorEvent.values[0]	Acceleration force along the x axis (including gravity).	m/s2
	SensorEvent.values[1]	Acceleration force along the y axis (including gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (including gravity).	
TYPE_GRAVITY	SensorEvent.values[0]	Force of gravity along the x axis.	m/s2
	SensorEvent.values[1]	Force of gravity along the y axis.	
	SensorEvent.values[2]	Force of gravity along the z axis.	
TYPE_GYROSCOPE	SensorEvent.values[0]	Rate of rotation around the x axis.	rad/s
	SensorEvent.values[1]	Rate of rotation around the y axis.	
	SensorEvent.values[2]	Rate of rotation around the z axis.	
TYPE_LINEAR_ACCELERATION	SensorEvent.values[0]	Acceleration force along the x axis (excluding gravity).	m/s2
	SensorEvent.values[1]	Acceleration force along the y axis (excluding gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (excluding gravity).	
TYPE_ROTATION_VECTOR	SensorEvent.values[0]	Rotation vector component along the x axis (x * sin(θ/2)).	Unitless
	SensorEvent.values[1]	Rotation vector component along the y axis (y * sin(θ/2)).	
	SensorEvent.values[2]	Rotation vector component along the z axis (z * sin(θ/2)).	
	SensorEvent.values[3]	Scalar component of the rotation vector ((cos(θ/2)).[1]	

http://developer.android.com/guide/topics/sensors/sensors_motion.html

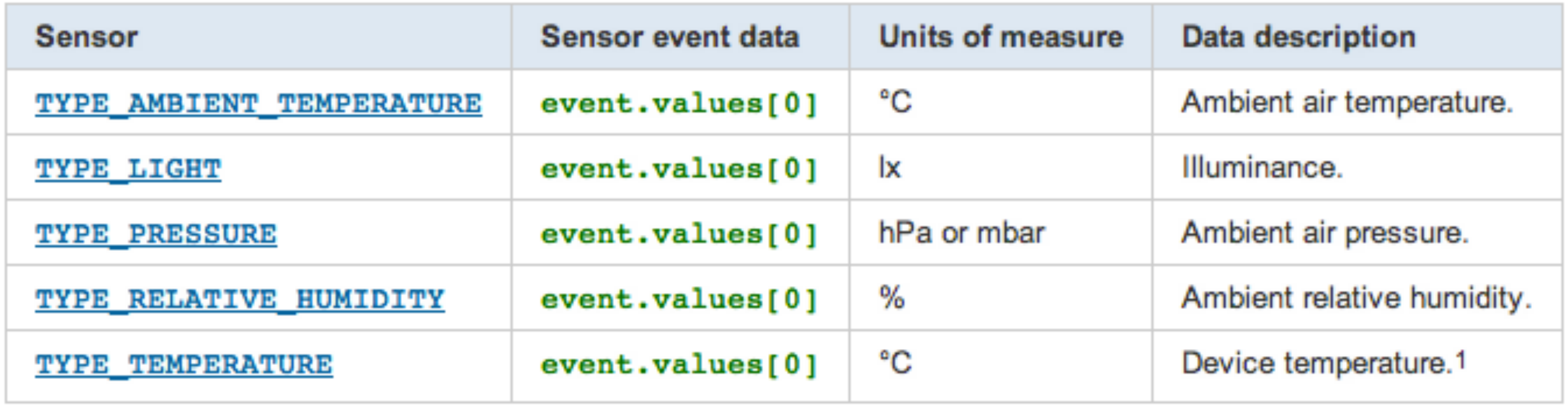
Position Sensor



Sensor	Sensor event data	Description	Units of measure
TYPE_MAGNETIC_FIELD	SensorEvent.values[0]	Geomagnetic field strength along the x axis.	μT
	SensorEvent.values[1]	Geomagnetic field strength along the y axis.	
	SensorEvent.values[2]	Geomagnetic field strength along the z axis.	
TYPE_ORIENTATION[1]	SensorEvent.values[0]	Azimuth (angle around the z-axis).	Degrees
	SensorEvent.values[1]	Pitch (angle around the x-axis).	
	SensorEvent.values[2]	Roll (angle around the y-axis).	
TYPE_PROXIMITY	SensorEvent.values[0]	Distance from object.[2]	cm

http://developer.android.com/guide/topics/sensors/sensors_position.html

Environment Sensor



Sensor	Sensor event data	Units of measure	Data description
TYPE_AMBIENT_TEMPERATURE	event.values[0]	°C	Ambient air temperature.
TYPE_LIGHT	event.values[0]	lx	Illuminance.
TYPE_PRESSURE	event.values[0]	hPa or mbar	Ambient air pressure.
TYPE_RELATIVE_HUMIDITY	event.values[0]	%	Ambient relative humidity.
TYPE_TEMPERATURE	event.values[0]	°C	Device temperature.[1]

http://developer.android.com/guide/topics/sensors/sensors_position.html

Android & Multimedia

- The Android multimedia framework includes support for capturing and playing audio, video and images in a variety of common media types, so that you can easily integrate them into your applications. You can play audio or video from media files stored in your application's resources, from standalone files in the file system, or from a data stream arriving over a network connection, all using the MediaPlayer or JetPlayer APIs. You can also record audio, video and take pictures using the MediaRecorder and Camera APIs if supported by the device hardware.
- Main Multimedia Topics are
  - Media Playback: How to play audio and video in your application.
  - JetPlayer: How to play interactive audio and video in your application using content created with JetCreator.
  - Camera: How to use a device camera to take pictures or video in your application.
  - Audio Capture: How to record sound in your application.

Media Playback

- The Android multimedia framework includes support for playing variety of common media types, so that you can easily integrate audio, video and images into your applications.
- You can play audio or video from media files stored in your application's resources (raw resources), from standalone files in the filesystem, or from a data stream arriving over a network connection, all using MediaPlayer APIs.
- Main classes are:
  - MediaPlayer: This class is the primary API for playing sound and video.
  - AudioManager: This class manages audio sources and audio output on a device.

Media Player

- One of the most important components of the media framework is the **MediaPlayer** class. An object of this class can **fetch**, **decode**, and **play** both audio and video with minimal setup. It supports several different media sources such as:
  - Local resources
  - Internal URIs, such as one you might obtain from a Content Resolver
  - External URLs (streaming)
- Supported Media Formats are:
  - RTSP (RTP, SDP)
  - HTTP/HTTPS progressive streaming
  - HTTP/HTTPS live streaming
  - 3GPP
  - MPEG-4
  - MP3
  - ...

MediaPlayer

- Whit this example you can play an audio file as a local raw resource from res/raw/ directory:

```
MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.sound_file_1);
mediaPlayer.start(); // no need to call prepare(); create() does that for you
```

- You can also load a local content through an URI Object (obtained with a Content Resolver)

```
Uri myUri = ....; // initialize Uri here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(getApplicationContext(), myUri);
mediaPlayer.prepare();
mediaPlayer.start();
```

- You can also play a content from a remote URL via HTTP streaming.

```
String url = "http://........"; // your URL here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(url);
mediaPlayer.prepare(); // might take long! (for buffering, etc)
mediaPlayer.start();
```

Camera

- The Android framework includes support for various cameras and camera features available on devices, allowing you to capture pictures and videos in your applications.
- Android applications that use the camera should consider a few questions about how they intend to use this hardware feature.
- **Camera Requirement** - Is the use of a camera so important to your application that you do not want your application installed on a device that does not have a camera? If so, you should declare the camera requirement in your manifest.
- **Quick Picture or Customized Camera** - How will your application use the camera? Are you just interested in snapping a quick picture or video clip, or will your application provide a new way to use cameras? (Use Existing Camera App or Build a custom Camera Application)
- **Storage** - Are the images or videos your application generates intended to be only visible to your application or shared so that other applications such as Gallery or other media and social apps can use them? Do you want the pictures and videos to be available even if your application is uninstalled?

Camera API Permission

- **Camera Permission** - Your application must request permission to use a device camera.

```
<uses-permission android:name="android.permission.CAMERA" />
```

- **Storage Permission** - If your application saves images or videos to the device's external storage (SD Card), you must also specify this in the manifest.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

- **Audio Recording Permission** - For recording audio with video capture, your application must request the audio capture permission.

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

- **Location Permission** - If your application tags images with GPS location information, you must request location permission:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Using Native Camera App

- A quick way to enable taking pictures or videos in your application without a lot of extra code is to use an Intent to invoke an existing Android camera application.
- A camera intent makes a request to capture a picture or video clip through an existing camera app and then returns control back to your application. This section shows you how to capture an image or video using this technique.
- The procedure for invoking a camera intent follows these general steps:
  - **Compose a Camera Intent** - Create an Intent that requests an image or video, using one of these intent types:
    - **MediaStore.ACTION_IMAGE_CAPTURE** - Intent action type for requesting an image from an existing camera application.
    - **MediaStore.ACTION_VIDEO_CAPTURE** - Intent action type for requesting a video from an existing camera application.
  - **Start the Camera Intent** - Use the startActivityForResult() method to execute the camera intent. After you start the intent, the Camera application user interface appears on the device screen and the user can take a picture or video.
  - **Receive the Intent Result** - Set up an onActivityResult() method in your application to receive the callback and data from the camera intent. When the user finishes taking a picture or video (or cancels the operation), the system calls this method.

Image Capture Intent

- Capturing images using a camera intent is quick way to enable your application to take pictures with minimal coding. An image capture intent can include the following extra information:

- MediaStore.EXTRA_OUTPUT - This setting requires a Uri object specifying a path and file name where you'd like to save the picture. This setting is optional but strongly recommended. If you do not specify this value, the camera application saves the requested picture in the default location with a default name, specified in the returned intent's Intent.getData() field.

```
private static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE = 100;
private Uri fileUri;

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

    // create Intent to take a picture and return control to the calling application
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

    fileUri = getOutputMediaFileUri(MEDIA_TYPE_IMAGE); // create a file to save the image
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri); // set the image file name

    // start the image capture Intent
    startActivityForResult(intent, CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
}
```

Video Capture Intent

- Capturing video using a camera intent is a quick way to enable your application to take videos with minimal coding.
- A video capture intent can include the following extra information:
  - MediaStore.EXTRA_OUTPUT - This setting requires a Uri specifying a path and file name where you'd like to save the video. This setting is optional but strongly recommended. If you do not specify this value, the Camera application saves the requested video in the default location with a default name, specified in the returned intent's Intent.getData() field.
  - MediaStore.EXTRA_VIDEO_QUALITY - This value can be 0 for lowest quality and smallest file size or 1 for highest quality and larger file size.
  - MediaStore.EXTRA_DURATION_LIMIT - Set this value to limit the length, in seconds, of the video being captured.
  - MediaStore.EXTRA_SIZE_LIMIT - Set this value to limit the file size, in bytes, of the video being captured.

Video Capture Intent

```
private static final int CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE = 200;
private Uri fileUri;

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

    //create new Intent
    Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);

    fileUri = getOutputMediaFileUri(MEDIA_TYPE_VIDEO);  // create a file to save the video
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);  // set the image file name

    intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 1); // set the video image quality to high

    // start the Video Capture Intent
    startActivityForResult(intent, CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE);
}
```

Camera Intent Result

```
private static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE = 100;
private static final int CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE = 200;
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            // Image captured and saved to fileUri specified in the Intent
            Toast.makeText(this, "Image saved to:\n" +
                     data.getData(), Toast.LENGTH_LONG).show();
        } else if (resultCode == RESULT_CANCELED) {
            // User cancelled the image capture
        } else {
            // Image capture failed, advise user
        }
    }
    if (requestCode == CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            // Video captured and saved to fileUri specified in the Intent
            Toast.makeText(this, "Video saved to:\n" +
                     data.getData(), Toast.LENGTH_LONG).show();
        } else if (resultCode == RESULT_CANCELED) {
            // User cancelled the video capture
        } else {
            // Video capture failed, advise user
        }
    }
}
```

Receive the result with the file system location of the new Image.

Receive the result with the file system location of the new Video.

Android Development

Lecture 10
Sensors & Media Management