Android Sensor Framework

Xin Pan
CSCI5448 2011 Fall
Outline

- Introduction of Android System
  - Four primary application components
  - AndroidManifest.xml
- Introduction of Android Sensor Framework
  - Package
  - Interface
  - Classes
- Examples of Using Accelerometer
  - Using background Service
  - Using foreground Activity
Introduction

- Android Inc. is acquired by Google in 2005.
- Open Handset Alliance was established and Android was announced in 2007.
- The first Android handset and source code was released in 2008.
  - Open and comprehensive platform for mobile devices.
  - Platform is powered by Linux.
## Android Version History

<table>
<thead>
<tr>
<th>Version</th>
<th>Release Date</th>
<th>Linux Kernel</th>
<th>Selected Key Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.x</td>
<td>Sep 2008 – Sep 2009</td>
<td>2.6.23/27/29</td>
<td>Camera, WiFi, and Bluetooth supported.</td>
</tr>
<tr>
<td>2.x</td>
<td>Oct 2009 – Dec 2010</td>
<td>2.6.29/32/35</td>
<td>Bluetooth 2.1, API changes, system speed, memory, and performance optimizations, media support, video chat.</td>
</tr>
<tr>
<td>3.x (Honeycomb)</td>
<td>Feb 2011 – Jul 2011</td>
<td>2.6.36</td>
<td>The first SDK release for tablet computers. Motorola Xoom tablet is the first device featuring this version.</td>
</tr>
<tr>
<td>4.x (Ice Cream Sandwich)</td>
<td>Oct 2011</td>
<td>3.0.1</td>
<td>Face Unlock, Wi-Fi Direct. Galaxy Nexus is the first device featuring this version. added facial recognition, social networking, information sharing, and other features.</td>
</tr>
</tbody>
</table>
Android Architecture

- This diagram shows the major components of Android operating system.

Android Architecture

- Android software layers consists of:
  - Linux
    - Provides process and memory management, security, networking, and device drivers.
  - Libraries
  - Runtime
    - Dalvik VM
  - Application Framework
    - provides services to applications, such as notification and activity managers. These are all implemented as Java classes.
  - Applications
    - Component-oriented and integration-oriented
Android Application

- Written in Java programming language.
- Packaged into a .apk file.
- Runs isolated in its own VM.
- Composes of one or more application components.
- Starts the components when needed.
- Ends the components when no longer needed.
Application Components

- Android process has four primary components:
  - Activities
    - a component that provides a user interface, e.g. send an email.
  - Services
    - a component that can perform long-running background operations without user interface.
  - Content providers
    - a component that manages application data
  - Broadcast receivers
    - a component that responds to system-wide broadcast announcements.
Activities

- Android is sensitive to the lifecycle of an application and its components.
- Android provides callbacks to process state changes.
- Lifecycle callbacks for an activity
  - `onCreate()`
  - `OnStart()`
  - `OnRestart()`
    - `OnResume()`
    - `OnPause()`
  - `OnStop()`
  - `onDestory()`

http://developer.android.com/guide/topics/fundamentals/activities.html
Services

• A service runs in the background.
• A service needs to be declared in the mainifest

```xml
<manifest ... >
  ...
  <application ... >
    <service android:name=".ExampleService" />
  ...
</application>
</manifest>
```

• Services can be started with Context.startService() and Context.bindService().
• Service will only stop when Context.stopService() or stopSelf() is called.
• Context.bindService() can be used to obtain a persistent connection to a service.
Service Lifecycle

- Service lifecycle callback methods are used to monitor changes in a service’s state.
  - `onCreate()`
  - `onStartCommand()`
    - Or `onBind()` and `onUnbind()`
  - `onDestroy()`

http://developer.android.com/guide/topics/fundamentals/services.html
Content Providers

- Content providers store and retrieve data.
- android.provider package
- The information needed to query a content provider,
  - URI to identify the provider
    - A Uniform Resource Identifier that identifies an abstract or physical resource
  - The name of the data fields
  - The data types of the fields
    - Audio, video, images...
Broadcast Receivers

- BroadcastReceiver object is only valid during the call to onReceive().
- Once onReceive() returns, BroadcastReceiver is no longer active, and system will consider its process to be empty and kill the process.
- Therefore, for long-running operations, Service and BroadcastReceiver should be used together to keep the process active.
The Manifest File

• Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory.

• AndroidManifest.xml defines all the components, contents and behavior of the application, e.g. activities and services.

```
<application>
  <activity/>
  <service/>
  <receiver/>
  <provider/>
</application>
```

• xml class will parse the contents
Introduction of Android Sensor Framework
Sensor Types

- Android supports multiple types of sensors
  - Light sensor
  - Proximity sensor
  - Temperature sensor
  - Pressure sensor
  - Gyroscope sensor
  - Accelerometer
  - Magnetic field sensor
  - Orientation sensor
  - Gravity sensor
  - Linear acceleration sensor
  - Rotation vector sensor
  - Near Field Communication (NFC) sensor
  - GPS (GPS is similar to a sensor, but not a sensor)
Android Sensor Framework

- Layers from bottom to top
  - Sensor driver
  - Sensor Hardware Module
  - Sensor JNI
  - Java Sensor Class
  - Java Application

Sensor Package and Classes

- Package: android.hardware
- Interface
  - SensorEventListener
- Classes:
  - Sensor
  - SensorEvent
  - SensorManager
Interface: SensorEventListener (I)

- Used for receiving notifications from the SensorManager when sensor values have changed.
- Public methods:
  - abstract void onSensorChanged(SensorEvent event)
  - abstract void onAccuracyChanged(Sensor sensor, int accuracy)
Interface: SensorEventListener (II)

- abstract void onSensorChanged(SensorEvent event)
  - This function is called by system when sensor values have changed.
  - This is an abstract function, need to be implemented by user.
  - The parameter of this function is an instance of Class SensorEvent (will introduce this class later), which holds information such as sensor type and sensor values.
Interface: SensorEventListener (III)

- abstract void onAccuracyChanged(Sensor sensor, int accuracy)
  - This function is called when the accuracy of a sensor has changed.
  - This is an abstract function, need to be implemented by user.
  - The parameters of this function are an instance of Class Sensor (will introduce this class later) and the new accuracy level (High(=3), Medium(=2), and Low(=1)).
Class: Sensor (I)

- Represents a sensor
- Use `getSensorList(int)` to get the list of available Sensors in Class SensorManager.
Class: Sensor (II)

- Class *Sensor* contains several constants to represent Android sensor type

<table>
<thead>
<tr>
<th>Constant</th>
<th>Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_ACCELEROMETER</td>
<td>an accelerometer sensor type</td>
</tr>
<tr>
<td>TYPE_ALL</td>
<td>all sensor types</td>
</tr>
<tr>
<td>TYPE_AMBIENT_TEMPERATURE</td>
<td>an ambient temperature sensor type</td>
</tr>
<tr>
<td>TYPE_GRAVITY</td>
<td>a gravity sensor type</td>
</tr>
<tr>
<td>TYPE_GYROSCOPE</td>
<td>a gyroscope sensor type</td>
</tr>
<tr>
<td>TYPE_LIGHT</td>
<td>an light sensor type</td>
</tr>
<tr>
<td>TYPE_LINEAR_ACCELERATION</td>
<td>a linear acceleration sensor type</td>
</tr>
<tr>
<td>TYPE_MAGNETIC_FIELD</td>
<td>a magnetic field sensor type</td>
</tr>
<tr>
<td>TYPE_PRESSURE</td>
<td>a pressure sensor type</td>
</tr>
<tr>
<td>TYPE_PROXIMITY</td>
<td>an proximity sensor type</td>
</tr>
<tr>
<td>TYPE_RELATIVE_HUMIDITY</td>
<td>a relative humidity sensor type</td>
</tr>
<tr>
<td>TYPE_ROTATION_VECTOR</td>
<td>a rotation vector sensor type</td>
</tr>
</tbody>
</table>
Class: Sensor (III)

- This class also includes a set of functions to get the properties of a sensor, such as
  - maximum range of the sensor in the sensor's unit.
  - name string of the sensor.
  - the power in mA used by this sensor while in use
  - resolution of the sensor in the sensor's unit.
  - generic type of this sensor.
  - vendor string of this sensor.
  - version of the sensor's module.
Class: SensorEvent (I)

- Represents a sensor event and holds information.
- Sensor event information includes:
  - The accuracy of the sensor data
  - The sensor that generated this event.
  - The time in nanosecond at which the event happened
  - Sensor data array. The length and contents of the values array depends on which sensor type is being monitored.
Class: SensorEvent (II)

- Sensor data Examples
  - Sensor type is Sensor.TYPE_ACCELEROMETER
    - Accelerometer has three directions: vertically, laterally, or longitudinally (X, Y, Z)
    - All values are in SI units (m/s^2)
    - values[0]: Acceleration minus Gx on the x-axis
    - values[1]: Acceleration minus Gy on the y-axis
    - values[2]: Acceleration minus Gz on the z-axis
  - Sensor type is Sensor.TYPE_GYROSCOPE
    - All values are in radians/second and measure the rate of rotation around the device's local X, Y and Z axis.
    - values[0]: Angular speed around the x-axis
    - values[1]: Angular speed around the y-axis
    - values[2]: Angular speed around the z-axis
Class: SensorManager (I)

- SensorManager provides sensor management services to other applications on the device.
  - Provides a sensor selector package
  - Provides a standard way to all supported sensors
  - Provides an interface to list and invoke the sensors
- Get an instance of this class by calling `Context.getSystemService()` with the argument `SENSOR_SERVICE`. 
An important Function

- `registerListener (SensorEventListener listener, Sensor sensor, int rate)`
  - Registers a SensorEventListener for the given sensor.
  - You can make a single SensorManager, but for each sensor you want to track, you need to make a unique SensorEventListener, and Sensor.
  - To avoid the unnecessary usage of battery, you should register the listener in the onResume method and unregister in the onPause method when overriding Activity methods.
  - Listener- A SensorEventListener object.
  - Sensor - The Sensor to register to.
  - Rate - The rate sensor events are delivered at.
Class: SensorManager (III)

- Delivering rate for sensor events must be one of:

<table>
<thead>
<tr>
<th>Constants</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SENSOR_DELAY_FASTEST</td>
<td>get sensor data as fast as possible</td>
</tr>
<tr>
<td>SENSOR_DELAY_GAME</td>
<td>rate suitable for games</td>
</tr>
<tr>
<td>SENSOR_DELAY_NORMAL</td>
<td>rate (default) suitable for screen orientation changes</td>
</tr>
<tr>
<td>SENSOR_DELAY_UI</td>
<td>rate suitable for the user interface</td>
</tr>
</tbody>
</table>
Examples of Reading Accelerometer
Read Accelerometer

- It can be read in background service or foreground activity.
- We will look at both examples:
  - Service – write accelerometer data into log file
  - Activity – display accelerometer data on screen
Write Accelerometer into Log

Steps:

- Create an accelerometer **Service** and implement a SensorEventListener
- Implement `onAccuracyChanged` and `onSensorChanged` method
- Create variables for `SensorManager` and `Sensor`
- Get Object of `SensorManager` using system service
- Get Object of Acc Sensor from `SensorManager`
- Register a SensorEventListener for the accelerometer sensor
Step 1

- Create an accelerometer **Service** and implement a **SensorEventListener** interface to process sensor data and sensor accuracy change.

```java
class AccServcie extends Service implements SensorEventListener {

    public void onSensorChanged(SensorEvent event) {
        // deal with sensor data
        mNewValue = (int) event.values[0] * 10;
        Log.d(TAG, Integer.toString(mNewValue));
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // deal with sensor accuracy change
    }
}
```
Step2

- Create and get instants of SensorManager and Sensor

```java
Class AccService extends Service implements SensorEventListener {

    @Override
    public void onCreate() {

        SensorManager sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

        Sensor accSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    }
}
```
Step 3

- Register a SensorEventListener for the accelerometer sensor.

```java
Class AccService extends Service implements SensorEventListener {

    @Override
    public void onCreate() {
        SensorManager sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
        Sensor accSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

        sensorManager.registerListener(this, accSensor, SensorManager.SENSOR_DELAY_NORMAL);
    }
}
```
Read Accelerometer in Foreground

Steps:

- Add a main.xml in /res/layout folder
  - main.xml describe the layout of the screen display
- Similar to that of Service,
  - Create an accelerometer Activity and implement a SensorEventListener interface to process sensor data and sensor accuracy change
  - Create and get instants of SensorManager and Sensor, and register a SensorEventListener
- Implement activity life cycle management for sensor reading
Step1

- main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:id="@+id.textView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Shake to get a toast and to switch color"
    />
</LinearLayout>
```
Step2

- Create an accelerometer Activity and implement a SensorEventListener interface to process sensor data and sensor accuracy change

```java
Class AccActivity extends Activity implements SensorEventListener {

    public void onSensorChanged(SensorEvent event) {
        // deal with sensor data
        TextView tvX = (TextView) findViewById(R.id.x_axis);
        mNewValue = (int) event.values[0]*10;
        ...
        tvX.setText(Float.toString(mNewValue));
        ...
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // deal with sensor accuracy change
    }
}
```
Step3

- Create and get instants of SensorManager and Sensor, and register a SensorEventListener for the accelerometer sensor

```java
public class AccActivity extends Activity implements SensorEventListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

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

        Sensor accSensor =
                sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

        // Register for events.
        sensorManager.registerListener(
                this, accSensor, SensorManager.SENSOR_DELAY_NORMAL);
    }
}
```
Step 4

- For activity life cycle management, onResume and onPause need to be overridden.
  - Register a listener when receiving data from accelerometer
  - Turn off the listener when not listening

```java
Class AccActivity extends Activity implements SensorEventListener {
    protected void onResume() {
        super.onResume();
        mSensorManager.registerListener(this, mAccelerometer,
                                      SensorManager.SENSOR_DELAY_NORMAL);
    }

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

 unregister the listener to save energy
Reference

- [http://developer.android.com](http://developer.android.com)
Thank you