Service Notifications

• Once running, a service can notify the user of events using
  – Toast notifications are messages that appear on the surface of the current window for a moment then disappear
  – Status bar notifications provide an icon in the status bar with a message, the user can select it to take an action
    • This is the best technique when some background work has completed
Status Bar Notifications

- Used by background services to notify the occurrence of an event without interrupting the operations of the foreground activities
  - Display an icon on the Status Bar
  - Display a message in the Notification Window
  - Fire an event in case the user selects the notification
Toast Notifications

• A Toast Notification is a message that pops up on the surface of the window, and automatically fades out
  — Typically created by the foreground activity
  — Display a message text and then fades out
  — Does not accept events! (use Status Bar Notifications instead)
Inter-process communication

• AIDL is similar to other IDLs
• Used to define the programming interface that both the client and service agree upon to communicate with each other
  – One process cannot normally access the memory of another process
  – Actors need to decompose their objects into primitives that the operating system can understand, and marshal the objects across the boundary
  – Android handles it for us with AIDL
System services
Many different services

- Power Service
- KeyGuard Service
- Vibrator Service
- Alarm Service
- Sensor Service
- Audio Service
- Telephony Service
- Connectivity Service
- Wi-Fi Service
Power Service

- The battery of our phones must be used wisely

```java
PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
PowerManager.WakeLock wl = pm.newWakeLock(
    PowerManager.PARTIAL_WAKE_LOCK, "My Tag")
wl.acquire();
  // screen will stay off during this section
wl.release();
```
- Proper permissions in application's manifest
- Methods: isScreenOn, isPowerSaveMode, ...
- Device battery life will be significantly affected by the use of this API
  - Do not acquire locks unless you really need them
  - Be sure to release them as soon as possible.

<table>
<thead>
<tr>
<th>Flag Value</th>
<th>CPU</th>
<th>Screen</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTIAL_WAKE_LOCK</td>
<td>On*</td>
<td>Off</td>
<td>Off</td>
</tr>
</tbody>
</table>
Vibrator Service

Vibrator  

vibrator = (Vibrator)
    getSystemService(Context. VIBRATOR_SERVICE);

• Some methods:
  — hasVibrator()
  — vibrate(long time);
  — cancel();
  — vibrate(long[] pattern, int repeat);

• Needs android.permission.VIBRATE
Alarm Service

• Allows one to schedule an application to be run in the future
  – When an alarm goes off, the Intent that had been registered for it is broadcast by the system and the target application is run automatically
  – Registered alarms are retained while the device is asleep (and can optionally wake it up)
    • They are cleared if it is turned off and rebooted

• Beginning with API 19, the OS shifts alarms to minimize wakeups and battery use
  – There are new APIs to support applications that need strict delivery guarantees
Alarm Manager

```java
AlarmManager as = (AlarmManager) getSystemService(Context.ALARM_SERVICE);

• set(Exact)(int type, long triggerAtTime, PendingIntent operation)
  — Schedule an alarm
• set(Inexact)Repeating(int type, long triggerAtTime, long interval, PendingIntent operation)
  — Schedule a repeating alarm
• cancel(PendingIntent operation)
  — Remove all alarms that match
• setWindow(int type, long windowStartMillis, long windowLengthMillis, PendingIntent operation)
  — Schedule an alarm to be delivered within a given window of time
```
Sensors

• Android supports three broad categories of sensors
  — Motion sensors: accelerometers, gravity sensors, gyroscopes, and rotational vector sensors
  — Environmental sensors: barometers, photometers, and thermometers
  — Position sensors: orientation sensors and magnetometers

• Sensor framework helps
  — Determine which sensors are available on a device
  — Determine an individual sensor's capabilities
  — Acquire raw sensor data
  — Register and unregister sensor event listeners that monitor sensor changes

• Key elements
  — SensorManager, Sensor, SensorEvent, and SensorEventListener
<table>
<thead>
<tr>
<th>Sensor</th>
<th>Type</th>
<th>Description</th>
<th>Common Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_ACCELEROMETER</td>
<td>Hardware</td>
<td>Measures the acceleration force in m/s² that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.</td>
<td>Motion detection (shake, tilt, etc.).</td>
</tr>
<tr>
<td>TYPE_AMBIENT_TEMPERATURE</td>
<td>Hardware</td>
<td>Measures the ambient room temperature in degrees Celsius (°C). See note below.</td>
<td>Monitoring air temperatures.</td>
</tr>
<tr>
<td>TYPE_GRAVITY</td>
<td>Software or Hardware</td>
<td>Measures the force of gravity in m/s² that is applied to a device on all three physical axes (x, y, z).</td>
<td>Motion detection (shake, tilt, etc.).</td>
</tr>
<tr>
<td>TYPE_GYROSCOPE</td>
<td>Hardware</td>
<td>Measures a device’s rate of rotation in rad/s around each of the three physical axes (x, y, and z).</td>
<td>Rotation detection (spin, turn, etc.).</td>
</tr>
<tr>
<td>TYPE_LIGHT</td>
<td>Hardware</td>
<td>Measures the ambient light level (illumination) in lx.</td>
<td>Controlling screen brightness.</td>
</tr>
<tr>
<td>TYPE_LINEAR_ACCELERATION</td>
<td>Software or Hardware</td>
<td>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.</td>
<td>Monitoring acceleration along a single axis.</td>
</tr>
<tr>
<td>TYPE_MAGNETIC_FIELD</td>
<td>Hardware</td>
<td>Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.</td>
<td>Creating a compass.</td>
</tr>
<tr>
<td>TYPE_ORIENTATION</td>
<td>Software</td>
<td>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.</td>
<td>Determining device position.</td>
</tr>
</tbody>
</table>
Sensor Service

- `getDefaultSensor()` returns default sensor of a given type
  - If a default sensor does not exist, the method call returns null, which means the device does not have that type of sensor

```java
if (sm.getDefaultSensor(Sensor.TYPE_GRAVITY) != null) {
    List<Sensor> gravSensors = sm.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 version 3 gravity sensor.
            s = gravSensors.get(i);
        }
    }
}
```
How to monitor sensor events

• Implement two callback methods that are exposed through the SensorEventListener interface
  – onAccuracyChanged()
    • 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 SensorEvent object contains information about the new sensor data
      ─ 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
Example (I)

```java
public class SensorActivity extends Activity implements SensorEventListener {
    private SensorManager mSenMan;
    private Sensor mLight;

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

        ms = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mLight = ms.getDefaultSensor(Sensor.TYPE_LIGHT);
    }

    @Override
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Do something here if sensor accuracy changes.
    }
}
```
Example (II)

```java
@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.
}

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

@Override
protected void onPause() {
    super.onPause();
    mSenMan.unregisterListener(this);
}
```
Telephony Service

• Provides access to information about the telephony services on the device
  — Applications can use the methods in this class to determine telephony services and states, as well as to access some types of subscriber information
  — Applications can also register a listener to receive notification of telephony state changes

• Some methods
  — getCallState(), getDataState(), getDataActivity(), getNetworkType(), getCellLocation(), getPhoneType(), and isNetworkRoaming()
  — Through usual acquisition of a TelephonyManager
Text messages

- `android.telephony.SmsManager` vs `android.telephony.gsm.SmsManager`
- Only the default SMS app (selected by the user in system settings) is able to write to interact with it (through Intents)
- `SMSManager` manages SMS operations
  - `getCarrierConfigValues()`, `divideMessage()`, `sendDataMessage()`, `sendTextMessage()`
    - sent and delivery: two intents to be fired when the message is sent and/or delivered

http://android-developers.blogspot.it/2013/10/getting-your-sms-apps-ready-for-kitkat.html
Connectivity Service

• Checks the state of network connectivity

```java
ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
```

• Monitors Wi-Fi, GPRS, UMTS, ecc.

• Sends broadcast intents when network connectivity changes

• Attempts to "fail over" to another network when connectivity to a network is lost

• Provides an API that allows applications to query the state of the available networks

• Needs proper permissions
Wi-Fi Service

- Manages all aspects of Wi-Fi connectivity
  - `getWifiState()`
    - Returns `WIFI_STATE_DISABLED`, `WIFI_STATE_DISABLING`, `WIFI_STATE_ENABLED`, `WIFI_STATE_ENABLING`, `WIFI_STATE_UNKNOWN`
  - `isWifiEnabled() / setWifiEnabled()`
  - `getConfiguredNetworks()`
  - `addNetwork(WifiConfiguration config)`
  - `updateNetwork(WifiConfiguration config)`
  - `removeNetwork(int netid)`
  - `startScan()`
  - `getScanResults()`
Summary

• Android Applications are structured as a single Activity or a group of Activities
  — Intents to call other activities
  — Layout and Views to setup the GUI
  — Events to manage the interactions with the user

• Activities execute only in foreground
  — Updates in background mode
    • A Service provides a robust environment for background tasks
  — Notifications in case of changes
Hello Map

```java
import com.google.android.gms.maps.*;
import com.google.android.gms.maps.model.*;
import android.app.Activity;
import android.os.Bundle;

public class MapPane extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.map_activity);

        // Get a handle to the Map Fragment
        GoogleMap map = ((MapFragment) getFragmentManager().
            findFragmentById(R.id.map)).getMap();

        Latlng sydney = new Latlng(-33.867, 151.206);
        map.setMyLocationEnabled(true);
        map.moveCamera(CameraUpdateFactory.newLatLngZoom(sydney, 13));
        map.addMarker(new MarkerOptions()
            .title("Sydney")
            .snippet("The most populous city in Australia.")
            .position(sydney));
    }
}
```
Many other things ...