Programming with Android: System Services

Federico Montori

Dipartimento di Informatica: Scienza e Ingegneria

Università di Bologna

System Services

- There is a wide list of services available

AccessibilityManager
AccountManager
ActivityManager
AlarmManager
AppOpsManager
AudioManager
BatteryManager
BluetoothManager
ClipboardManager
ConnectivityManager
DevicePolicyManager
DisplayManager
DownloadManager
DropBoxManager
FingerprintManager
InputMethodManager
InputManager
JobScheduler
KeyguardManager
LauncherApps
LayoutInflater
LocationManager
MediaProjectionManager
MediaRouter
MediaSessionManager
MidiManager
NetworkStatsManager
NfcManager
NotificationManager
NsdManager
PowerManager
PrintManager
RestrictionsManager
SearchManager
SensorManager
StorageManager,
SubscriptionManager
TelecomManager
TelephonyManager
TextServicesManager
TvInputManager
UiModeManager
UsageStatsManager
UsbManager
UserManager
Vibrator
WallpaperService
WifiManager
WifiP2pManager
WindowManager

Battery Manager

- Android runs on limited capabilities devices
- It is crucial to use the battery wisely
- The battery service gives us information about the power of the system
- Get it with:

```
BatteryManager bm = (BatteryManager) getSystemService(Context.BATTERY_SERVICE);
```

However you don't handle battery monitoring by calling directly its functions...

Battery Manager

- The BatteryManager broadcasts a **sticky intent** (that's why the receiver is null) accessed by :

```
Intent batteryStatus = context.registerReceiver(null,
      new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
```

- From there you extract monitoring data and beyond...

```
int status = batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
boolean isCharging = (status == BatteryManager.BATTERY_STATUS_CHARGING ||
                      status == BatteryManager.BATTERY_STATUS_FULL);

int chargePlug = batteryStatus.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
boolean usbCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_USB;
boolean acCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_AC;

float batteryPercent = batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1) * 100 /
      (float)batteryStatus.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
```

Battery Manager

- And obviously we can be notified whenever special conditions occur:

```
<receiver android:name=".BatteryLevelReceiver">
  <intent-filter>
    <action android:name="android.intent.action.ACTION_POWER_CONNECTED"/>
    <action android:name="android.intent.action.ACTION_POWER_DISCONNECTED"/>
    <action android:name="android.intent.action.BATTERY_LOW"/>
    <action android:name="android.intent.action.BATTERY_OKAY"/>
  </intent-filter>
</receiver>
```

Alarm Service

- Fires an Intent in the future
- Get it with

```
AlarmManager am = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
am.set(int type, long triggerAtTime, PendingIntent operation);
```

- type is one of:
  - ELAPSED_REALTIME
  - ELAPSED_REALTIME_WAKEUP
  - RTC
  - RTC_WAKEUP

SystemClock.elapsedRealTime()

Elapsed since sys boot.
Better for time slices

System.currentTimeMillis()

UTC Clock
Better for time of the day

Alarm Service

- Fire alarmIntent in **exactly** half an hour from now (otherwise inexact)

```
alarmMgr.setExact(AlarmManager.ELAPSED_REALTIME_WAKEUP,
      SystemClock.elapsedRealtime() + AlarmManager.INTERVAL_HALF_HOUR,
      alarmIntent);
```

- Fire alarmIntent every day at 14 starting from today, waking up the device if sleeping (WAKEUP) and clustering the alarm with others if present (Inexact).

```
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.set(Calendar.HOUR_OF_DAY, 14);
alarmMgr.setInexactRepeating(AlarmManager.RTC_WAKEUP,
      calendar.getTimeInMillis(), AlarmManager.INTERVAL_DAY, alarmIntent);
```

Alarm Service

- More methods
  - setRepeating(int type, long triggerAtTime, long interval, PendingIntent operation);
    - Can use INTERVAL_HOUR, INTERVAL_HALF_DAY
  - cancel(PendingIntent operation);
    - Match with filterEquals(Intent anotherIntent);
- [Best Practice warning] Sometimes is useful to set the alarms again if the device has rebooted

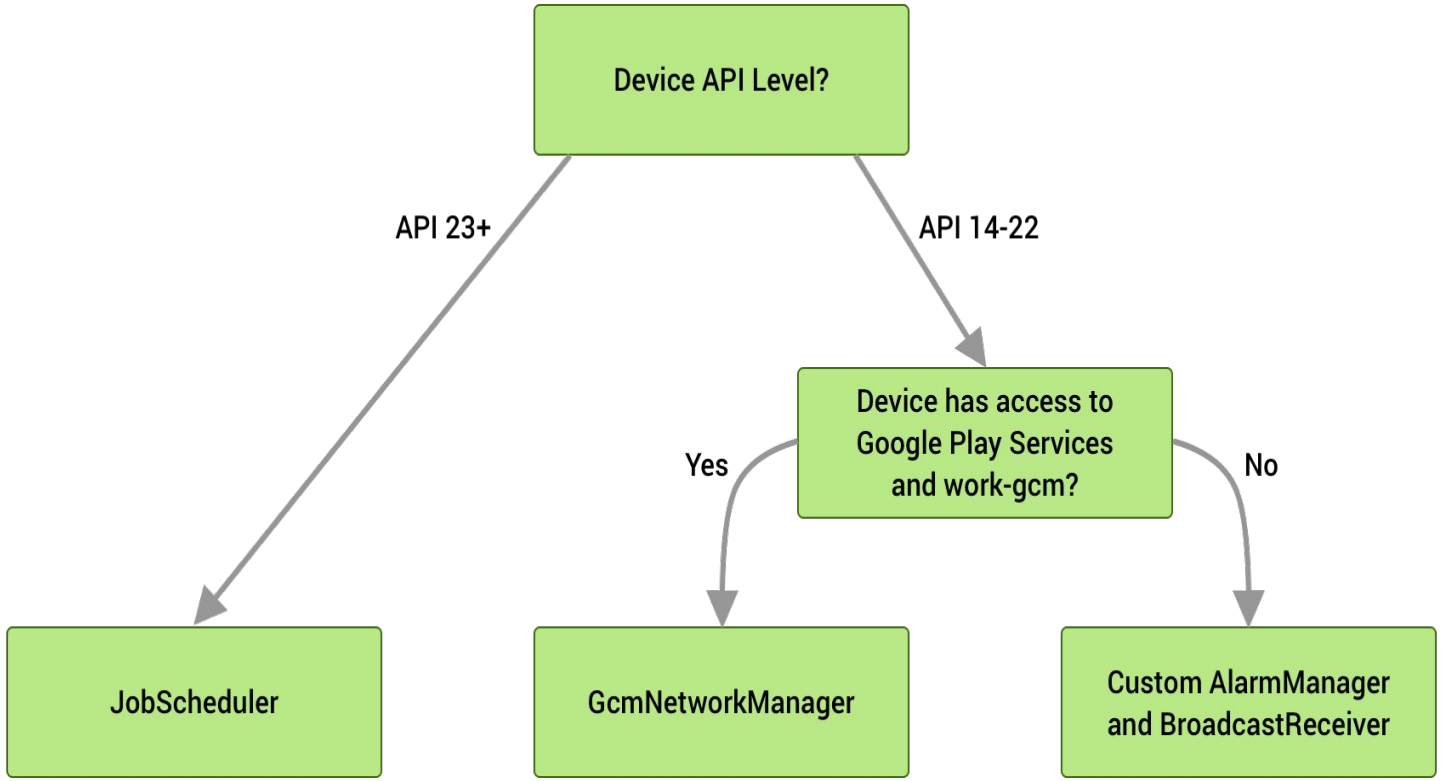
```
<uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>

[…]
      <action android:name="android.intent.action.BOOT_COMPLETED"></action>
```

WorkManager

WorkManager is an API that makes it easy to schedule deferrable, asynchronous tasks that are expected to run even if the app exits or the device restarts.

- It uses a mix of JobScheduler, AlarmManager and BroadcastReceiver
- It is **NOT** a replacement for scheduling tasks at exact time, for that you should use AlarmManager still.



WorkManager Includes

- It shall be imported as usual in the **build.gradle**

```
dependencies {
  def work_version = "2.5.0"

    // (Java only)
    implementation "androidx.work:work-runtime:$work_version"

    // Kotlin + coroutines
    implementation "androidx.work:work-runtime-ktx:$work_version"

    implementation "androidx.work:work-rxjava2:$work_version"
    implementation "androidx.work:work-gcm:$work_version"
    androidTestImplementation "androidx.work:work-testing:$work_version"
    implementation "androidx.work:work-multiprocess:$work_version"
  }
```

WorkManager Create Worker

- This declares what the deferrable task is (in the doWork)
  - It will run on a background thread once enqueued

```
public class UploadWorker extends Worker {
   public UploadWorker(
       @NonNull Context context,
       @NonNull WorkerParameters params) {
       super(context, params);
   }

   @Override
   public Result doWork() {
                           // Do the work here--in this case, upload the images.
     uploadImages();
                           // Indicate whether the work finished successfully with the Result
     return Result.success();
   }
}
```

WorkManager Create Worker

- Then we should instantiate the object by stating implicitly what kind of job is

```
WorkRequest uploadWorkRequest =
   new OneTimeWorkRequest.Builder(UploadWorker.class)
       .build();
```

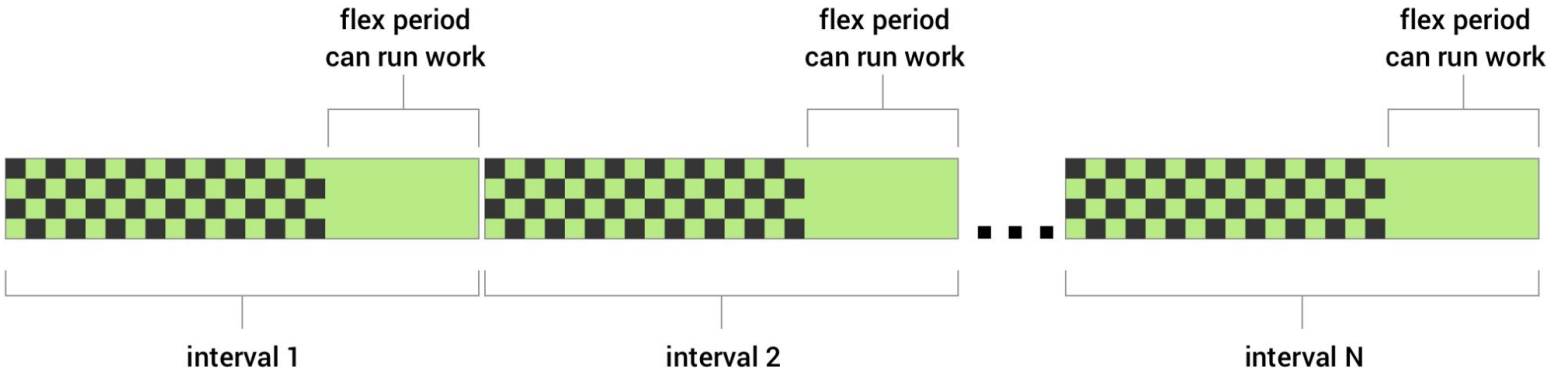
- Then we need to get the reference to the WorkManager and submit the job

```
WorkManager
    .getInstance(myContext)
    .enqueue(uploadWorkRequest);
```

- From now on, the job will be executed on top of the constraints declared while building the WorkRequest.
  - There are many parameters and constraints (e.g. retries, network types…)
  - https://developer.android.com/topic/libraries/architecture/workmanager/how-to/define-work

Periodic Work

You can schedule periodic work pretty easily and WorkManager is powerful enough to set a flexible period.




```
WorkRequest saveRequest =
        new PeriodicWorkRequest.Builder(SaveImageToFileWorker.class,
                1, TimeUnit.HOURS,
                15, TimeUnit.MINUTES)
            .build();
```

- In this example the job gets executed every hour with a 15-minutes tolerance

Monitor and Chain

You can observe changes on your work by using a LiveData

```
workManager.getWorkInfoByIdLiveData(saveRequest.id)
        .observe(getViewLifecycleOwner(), workInfo -> {
    if (workInfo.getState() != null &&
            workInfo.getState() == WorkInfo.State.SUCCEEDED) {
        // YOUR REACTION HERE
    }
});
```

You can also chain works

```
WorkManager.getInstance(myContext)
            // Candidates to run in parallel
    .beginWith(Arrays.asList(plantName1, plantName2, plantName3))
            // Dependent work (only runs after all previous work in chain)
    .then(cache)
    .enqueue();
```

Sensor Service

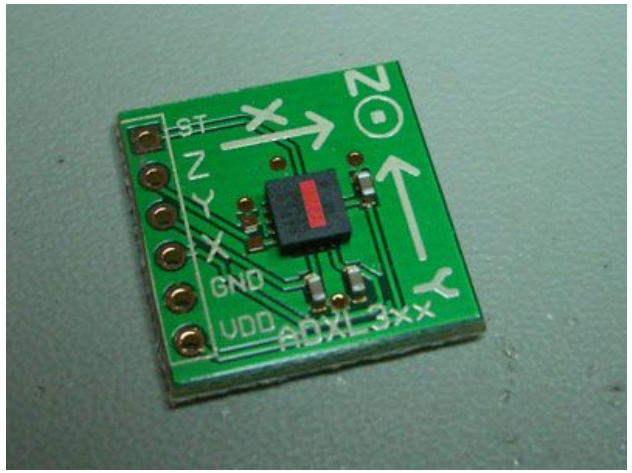
- Interaction with sensors
- Get it with

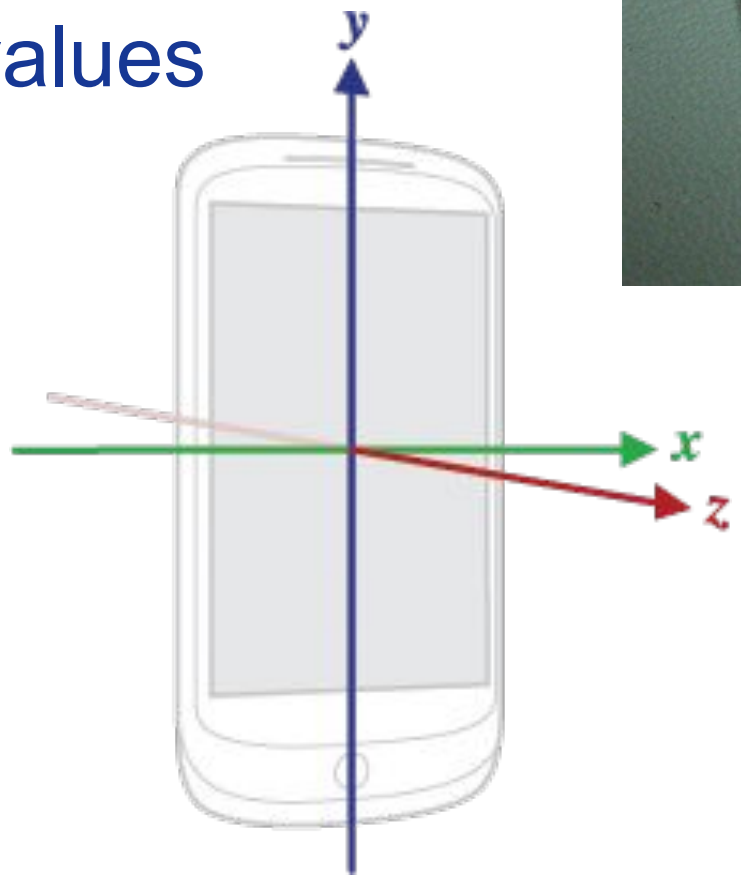
```
SensorManager sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

- Various kind of sensors
  - Motion (accelerometer, gyroscope, ...)
  - Environment (barometer, thermometer, photometer, ...)
  - Position (compass, magnetometer, ...)

Accelerometer

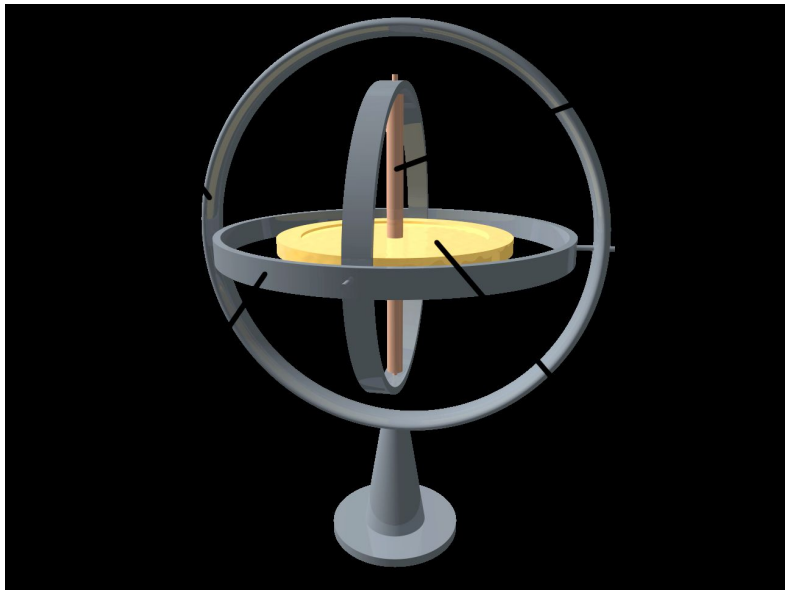
- To measure acceleration
- Given with 3-axes values
- Useful to inspect movements





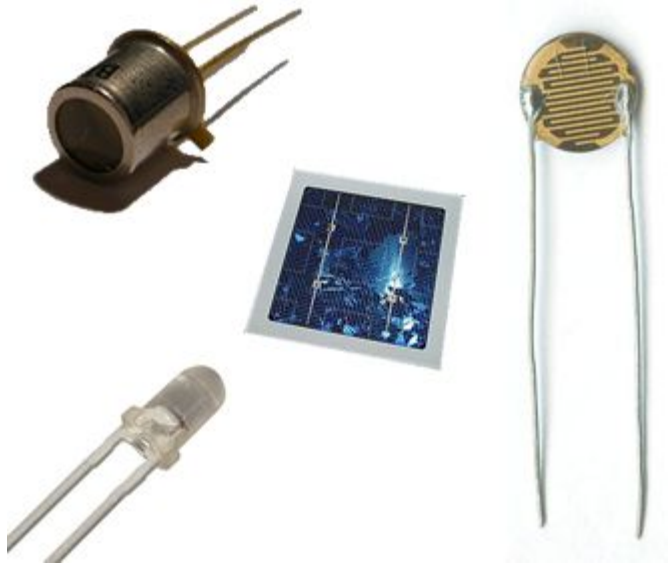
Gyroscope

- To measure orientation
- Usually a spinning wheel or a spinning disk
- Gives angular speed
- Now more common in smartphones

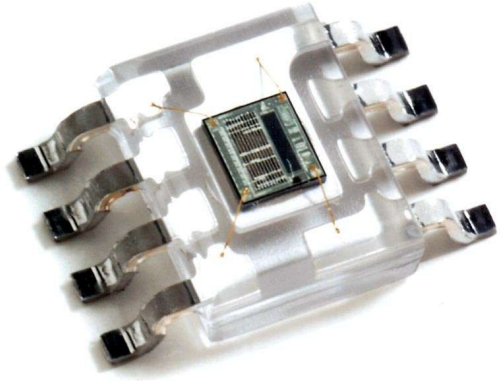


Light sensor

- Usually a photodiode



- When exposed to light, they create a current
- More current, more light



Proximity sensor

- To measure distance from objects

- Useful to understand when the smartphone is in, for instance, a pocket
- Used to switch off screen during calls

Sensors List

- public List<Sensor> getSensorList(int type); (can be **Sensor.TYPE_ALL**)

Sensor	Type (Hardware/Software)	Used for
TYPE_ACCELEROMETER	Hardware	Acceleration along three axes (+ gravity)
TYPE_AMBIENT_TEMPERATURE	Hardware	Temperature
TYPE_GRAVITY	Can be both	Motion Detection
TYPE_GYROSCOPE	Hardware	Rotation
TYPE_LIGHT	Hardware	Ambient brightness
TYPE_LINEAR_ACCELERATION	Can be both	Acceleration along three axes (no gravity)
TYPE_MAGNETIC_FIELD	Hardware	Compass, indoor navigation
TYPE_ORIENTATION	Software	Obtaining device position
TYPE_PRESSURE	Hardware	Obtaining the height from sea level
TYPE_PROXIMITY	Hardware	Setting off the screen
TYPE_RELATIVE_HUMIDITY	Hardware	Humidity
TYPE_ROTATION_VECTOR	Can be both	Motion and Rotation detection

Sensors

- Not all smartphones are created equal
- Some carry a set of sensors some others don't
- Also different vendors offer different sensors with different capabilities…
  - getResolution()
  - getMaximumRange()
  - getPower()
  - getVendor()
  - getMinDelay()

How to "use" a Sensor

- Each Sensor contains information about the vendor, type and others
- Implement SensorEventListener
  - onAccuracyChanged(Sensor sensor, int accuracy)
  - onSensorChanged(SensorEvent event)
  - registerListener(SensorEventListener listener, Sensor sensor, int rate)

    [do this in the onResume (and the unregisterListener in the onPause)]

    - rate is one of
      - SENSOR_DELAY_NORMAL
      - SENSOR_DELAY_FASTEST (default)

Requesting sensor updates

```
sm = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
Sensor sensorLight = sm.getDefaultSensor(Sensor.TYPE_LIGHT);
sm.registerListener(this, sensorLight, SensorManager.SENSOR_DELAY_NORMAL);

public void onSensorChanged(SensorEvent event) {
      if (event.sensor.getType() == Sensor.TYPE_LIGHT) {
                  // doSomething
      } else if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
                  // doSomething
      }
}
```

Sensor can report updates with different speeds:

- ***SENSOR_DELAY_FASTEST:*** *as fast as possible*
- ***SENSOR_DELAY_GAME:*** *suitable for games*
- ***SENSOR_DELAY_UI:*** *for interface changes*
- ***SENSOR_DELAY_NORMAL:*** *for all other uses*

Virtual Sensor

- In addition to the hardware sensors, there are a number of possible virtual sensors
  - Gravity
  - Linear Acceleration
  - Orientation
  - Rotation
- Readings from hardware sensors are computed to offer aggregated data

Sensor Challenges

- Bias
  - Sensor reading is off by a constant value
- Drift
  - Data is shifted without cause
- Settling time
  - Initial sensor readings may be inaccurate
- Noise
  - Data can’t report a reliable and steady value
- Interference
  - From the environment

Detecting User's activities

- Detecting the user activity is of paramount importance
  - Start vehicle related apps while the user is driving
  - Start tracking distances if the user is walking
  - Activate fitness apps
- How?
  - Reading raw values and use machine learning models
  - Exploit Activity Recognition API
- Permission

```
<uses-permission android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION" />
```

Requesting Activity notifications

```
List<ActivityTransition> transitions = new ArrayList<>();

transitions.add(
  new ActivityTransition.Builder()
    .setActivityType(DetectedActivity.IN_VEHICLE)
    .setActivityTransition(ActivityTransition.ACTIVITY_TRANSITION_ENTER)
    .build());

transitions.add(
  new ActivityTransition.Builder()
    .setActivityType(DetectedActivity.IN_VEHICLE)
    .setActivityTransition(ActivityTransition.ACTIVITY_TRANSITION_EXIT)
    .build());

transitions.add(
  new ActivityTransition.Builder()
    .setActivityType(DetectedActivity.WALKING)
    .setActivityTransition(ActivityTransition.ACTIVITY_TRANSITION_EXIT)
    .build());
```

Requesting Activity notifications

Build the request

```
ActivityTransitionRequest request = new ActivityTransitionRequest(transitions);
```

And register it

```
Task<Void> task =
 ActivityRecognition.getClient(context).requestActivityTransitionUpdates(request, myPendingIntent);

 task.addOnSuccessListener(new OnSuccessListener<Void>() {
    @Override
    public void onSuccess(Void result) { // do something }
  }
 );
 task.addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(Exception e) { // do something }
  }
 );
```

Receiving Activity notifications

- Performed as a Broadcast Receiver

```
if (ActivityTransitionResult.hasResult(intent)) {
        ActivityTransitionResult result = ActivityTransitionResult.extractResult(intent);
        for (ActivityTransitionEvent event : result.getTransitionEvents()) {
            // chronological sequence of events....
        }
    }
```

- Notifications are ordered

- Remember to de-register notifications

```
Task<Void> task = ActivityRecognition.getClient(context).removeActivityTransitionUpdates(myPendingIntent);
```

Audio Service

- Able to
  - select a stream and control sound
  - adjust the volume
  - change ring type
  - play effects

Telephony Service

- Interacts with calls
- Get it with

```
TelephonyManager tm = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
```

- Ask the device about call information
  - getCallState()
  - getDataState()
  - getDataActivity()
  - getNetworkType()
  - isNetworkRoaming()

SMS Service

- Send text messages
- Get it with

```
SmsManager sms = SmsManager.getDefault();
```

- To send a message call:
  - sendTextMessage(String dest, String sc, String text, PendingIntent sent, PendingIntent delivery);
    - sent and delivery: two intents to be fired when the message is sent and/or delivered

Connectivity Service

- Check device network state
- Get it with

```
String serId = Context.CONNECTIVITY_SERVICE;
ConnectivityManager cm = (ConnectivityManager) Context.getSystemService(serId);
```

- Check WI-FI, GPRS, LTE
- Notify connection changes
- Needs
  - android.permission.ACCESS_NETWORK_STATE
  - android.permission.CHANGE_NETWORK_STATE

Wi-Fi Service

- Manages the Wi-Fi connection
- Get it with

```
WifiManager wfm = (WifiManager) getSystemService(Context.WIFI_SERVICE)
```

- Check Wi-Fi
  - getWifiState()
    - Returns WIFI_STATE_DISABLED, WIFI_STATE_DISABLING, WIFI_STATE_ENABLED, WIFI_STATE_ENABLING, WIFI_STATE_UNKNOWN
  - isWifiEnabled() / setWifiEnabled()
- Lists all the configured wifi connections
  - getConfiguredNetworks()

Wi-Fi Service

- Check/edit wi-fi connection
  - addNetwork(WifiConfiguration config)
  - updateNetwork(WifiConfiguration config)
  - removeNetwork(int netid)
- Scan for wi-fi networks
  - startScan()
- Be notified about wi-fi changes
  - Broadcast Intent: SCAN_RESULTS_AVAILABLE_ACTION
    - Call getScanResults()