Sensor Topics

- Overview
    - Most Android devices have built-in sensors that measure motion, orientation, and environmental conditions.
    - Provide raw data with high precision and accuracy.
        - Need to account for environmental factors when dealing with this data
- What are sensors and why are they important?
    - Sensors are either a hardware or composite software implementation that uses physical device(s) to measure a property
    - There are usually many of these sensors in today's devices
        - There purpose is to improve user experience by providing a way for them to control certain aspects of a virtual space
        - We can collect data from the real world (temp, light, motion, etc.) to adjust settings within the device to improve user experience
            - Adjust screen brightness
            - Throttle / increase power consumption when needed
            - Stabilize the camera view
            - Step counters
            - Speed and acceleration data
        - We can use these sensors to interact with a virtual world, using tilting, shaking, rotation, etc.
            - Allows us to play games using physical movement rather than touch controls
- Reporting Mode
    - Continuous
        - Events are generated at a constant rate defined by sampling_period_ns
    - On-change
        - Events are only generated if the measured values have changed
        - The sampling_period_ns is passed to the batch function and is used to set the minimum time between consecutive events
    - One-shot
        - When an event occurs, the sensor deactivates itself and sends a signal event through the HAL (Hardware Abstraction Layer: a standard interface for hardware vendors to implement)
        - Order matters to avoid race conditions
            - Sensor must be deactivated in order to report event through HAL
        - No other event is sent until the sensor is reactivated
    - Special
        - Specific to the sensor

- Types of sensors

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	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	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	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	Hardware	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	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s^2 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	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.
TYPE_ORIENTATION	Software	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	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	Hardware	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	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	Hardware	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.

- Motion Sensors
  - Measure acceleration and rotational forces along 3 axes (X, Y, Z)
  - In other words they monitor device movement such as tilt, shake, rotation, or swing
  - Some examples: steering a car game with your phone, counting steps, significant motion sensor(i.e sitting in a moving car)

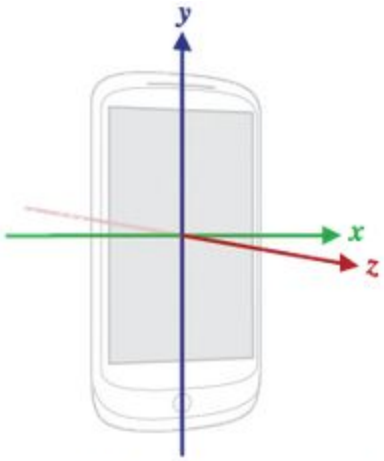


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

-
- The axes do not change when the screen orientation changes
  - Same behaviour as the openGL coordinate system
- Cannot assume default orientation is portrait
  - Most tablets are defaulted to landscape
- If your application matches sensor data to the display you need to use getRotation() to determine screen rotation and use the remapCoordinateSystem() to map the data accordingly

  - Types of sensors in this class
    - Accelerometers
    - Gravity sensors
    - Gyroscopes
    - Rotational vector sensors

NOTE: The availability of the software-based sensors is more variable because they often rely on one or more hardware sensors to derive their data.

  - Return a multidimensional array of sensor values for each sensor event
    - These can be identified in further detail by searching the respective sensor
  - Android open source project sensors(AOSP) provides three software-based motion sensors
    - Gravity
    - Linear acceleration sensor
    - Rotation vector sensor

    - Environmental Sensors
        - Measure various environmental parameters
            - Measures
                - Ambient air temperature
                - Ambient air pressure
                - Illumination
                - Humidity
                - NOTE: ALL HARDWARE BASED SENSORS NOT EVERY PHONE WILL HAVE THEM
            - Types of sensors in this class
                - Barometers
                - Photometers
                - Thermometers
        - Return a single sensor value for each data event
        - Usually do not require any filtering or data processing
        -
    - Position Sensors
        - Measure the physical position of a device
        - Types of sensors in this class
            - Orientation sensors
            - Magnetometers
        - Android provided sensors
            - Geomagnetic field sensor
            - Accelerometer
            - Proximity sensor (how close an object is)
        - Used to determine a device's physical position in the world's frame of reference
        - E.g implementations use geomagnetic field sensor with accelerometer to determine distance from north pole
        - Return a multidimensional array of sensor values for each sensor event
            - These can be identified in further detail by searching the respective sensor
- Category of Sensor: Hardware vs. Software
    - Hardware
        - Physical components built into the device
        - They get data by directly measuring specific environmental properties
            - Acceleration
            - Ambient temperature
            - Gyroscope
            - Light
            - Geomagnetic field strength
            - Pressure
            - Proximity

                - Relative humidity
                - Temperature
        - Software (virtual / synthetic sensors)
            - Mimic hardware based sensors but are not physical
            - They get data from one or more of the hardware-based sensors
                - Linear acceleration
                - Gravity sensor
                - Orientation
                - Rotation vector
            - Availability
                - Some sensors were not introduced until later android versions
                    - Began in Android 1.5(3) and continued until Android 4.0(14)
- Android Sensor Framework
    - Allows access to sensors available on the device so you can acquire the raw sensor data
    - Part of the **android.hardware** package
        - Provides several classes and interfaces that help the user perform many sensor related tasks
        - Classes:
            - SensorManager
                - Used to create an instance of the sensor service
                - Provides methods for:
                    - Accessing sensors
                    - Listing sensors
                    - Registering / Unregistering sensor event listeners
                    - Acquiring orientation information
                - Provides constants for reporting:
                    - Sensor accuracy
                    - Setting data acquisition rates
                    - Calibrating sensors
            - Sensor
                - Used to create an instance of a specific sensor
                - Provides classes to use the sensors capabilities
            - SensorEvent
                - Used to create a sensor event object
                    - Provides information about a sensor event
                        - Raw data
                        - Type of sensor
                        - Accuracy
                        - Timestamp of event
            - SensorEventListener

        - Used to create 2 callback methods that receive notifications (sensor events) when the sensor values or accuracy changes
        - Delay
            - You should specify the largest delay that you can as the system will normally use a smaller delay and if the specified delay is to small it puts to much load on the system
                - Optimization
                    - Best practice to disable sensors you do not need
            - You should not need to alter the delay when it is set but if you do you need to unregister and then re-register it
    - Configurations
        - Android does not specify sensor configurations and so manufacturers can incorporate any sensor configuration they want
        - If your application requires a specific sensor and version ensure it is present on the device prior to accessing it
        - Uses-feature
            - You can filter devices that can see your application by adding the <uses-feature> of the sensor your app uses to the manifest
            - Set the descriptor, android:required="true", if your application depends entirely on that sensor

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

- Sensor API's
    - Used to:
        - **Identify sensors and sensor capabilities**

This is the general setup to determining if the device has a certain sensor

```
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.
}
```

Get a list of all the sensors on the device

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

This is how you check to see if a specific sensor and version is on the device

```
private SensorManager mSensorManager;
private Sensor mSensor;

...

mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = null;

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 LLC")) &&
           (gravSensors.get(i).getVersion() == 3)){
            // Use the version 3 gravity sensor.
            mSensor = gravSensors.get(i);
        }
    }
}
if (mSensor == null){
    // 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.
    }
}
```

- **Monitor sensor events -** two main uses for the Android sensor framework
    - onAccuracyChanged()
        - Accuracy is represented by one of four constants:
            - SENSOR_STATUS_ACCURACY_LOW
            - SENSOR_STATUS_ACCURACY_MEDIUM
            - SENSOR_STATUS_ACCURACY_HIGH
            - SENSOR_STATUS_ACCURACY_UNRELIABLE
    - onSensorChanged()
        - Provides a sensor event containing:
            - Accuracy of data
            - Sensor that generated the data
            - Timestamp
            - New data the sensor recorded

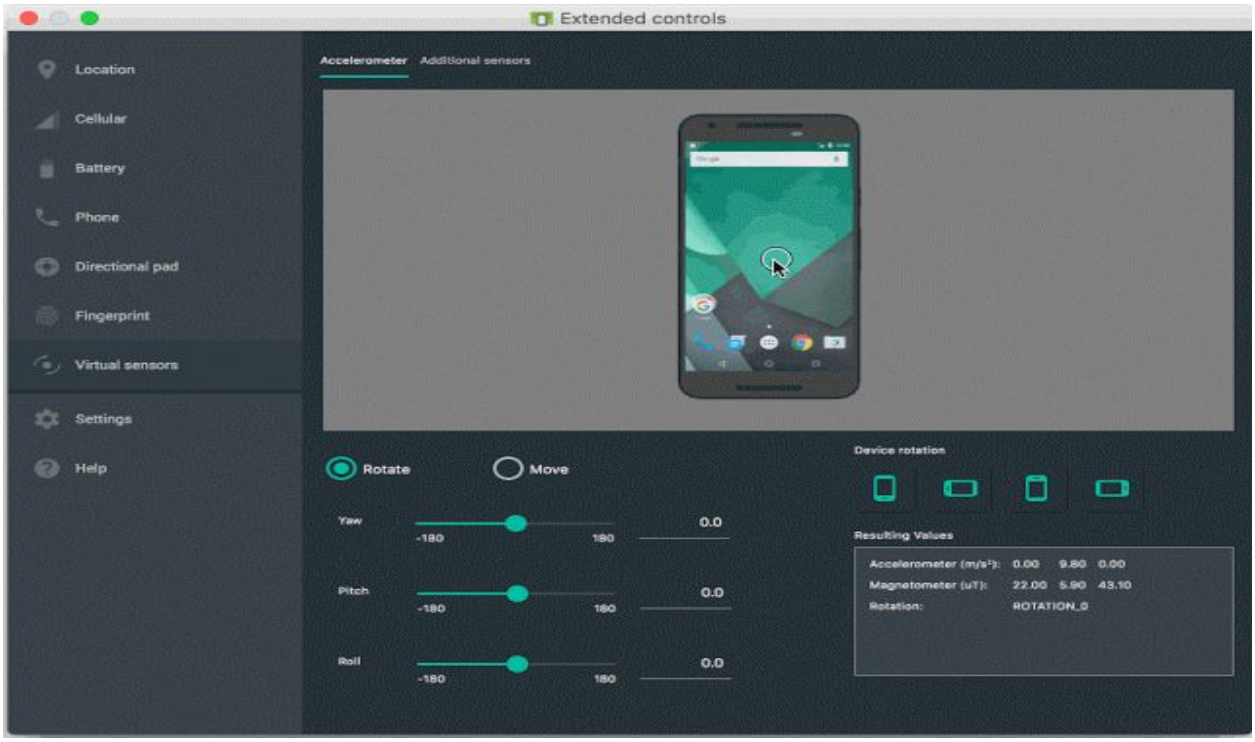
```java
public class SensorActivity extends Activity implements SensorEventListener {
    private SensorManager mSensorManager;
    private Sensor mLight;

    @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.
    }

    @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();
        mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NOR
    }

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

    - Capabilities
        - Determine which sensors are available on a device
        - Determine an individual sensor's capabilities, such as 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
- Best Practices

- Only gather sensor data in the foreground
- Android 9 (28) has the following restrictions:
  - Sensors that use the continuous reporting mode (accelerometers, gyroscopes) don't receive events
  - Sensors that use the on-change or one-shot reporting modes don't receive events
- Be sure to unregister sensor listeners when they are no longer needed or when the activity pauses

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

- Test with emulator
  - Has built in virtual sensor controls that allow you to test accelerometer, ambient temperature, magnetometer, etc.



- Don't block the onSensorChanged() method
  - Do any filtering work outside this method
- Avoid using deprecated methods or sensor types
- Verify sensors prior to using them
- Choose sensor delays carefully
  - Choose the delivery rate that suites the applications needs, if you go overboard you are wasting resources

- Handling different sensor configurations
  - Detect sensors at runtime
  - Use google play filters to target devices with specific sensor configurations
-

Sensor availability by platform:

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

★ [1] This sensor type was added in Android 1.5 (API Level 3), but it was not available for use until Android 2.3 (API Level 9).

★ [2] This sensor is available, but it has been deprecated.

User Location

- Overview
  - Google Play provides many location APIs to facilitate adding location awareness to the application with automated:

★ **Note:** The documentation here uses the Google Play services location APIs over the Android framework APIs (android.location). To setup your app to access these APIs, read Set Up Google Play Services.

  - Location tracking
  - Geofencing
  - Activity recognition

- Fused Location Provider API
    - Combines the signals from GPS, Wi-Fi, and cell networks, as well as some sensors
- Optimization
    - Android 8.0 (26) introduced Background Location limits
        - This throttles background location gathering and so location is only computed and delivered a few times per hour
        - Wi-fi scans are more conservative and location updates are not computed when the device stays connected to the same static access point
        - Geofencing responsiveness changes from tens of seconds to about 2 minutes
            - Big battery performance improvement (up to 10x)
    - Google location APIs are more optimized than framework location APIs
        - Better battery performance and higher accuracy
        - Also steals info ;)
    - Increased accuracy = more battery drain
    - Increased frequency = more battery drain
    - Decreased latency = more battery drain
    - Use removeLocationUpdates() in the onPause() and onStop() methods
        - Need to requestlocationUpdates() in the onResume() and onStart() methods
    - Set timeouts - ensures the updates do not continue forever
    -
- Last known location
    - Use the fused location provider to retrieve the device's last known location
    - Setting up google play services
        - Download and install google play services component in the SDK Manager and add the library to the project
        - Use this link to set up google play services
            - https://developers.google.com/android/guides/setup
    - App permissions
        - Android offers 2 location permissions (needed in Android 6.0 and up)
            - ACCESS_COARSE_LOCATION
                - Accuracy of about a city block
            - ACCESS_FINE_LOCATION

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.google.android.gms.location.sample.basiclocationsample" >

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

    - Create location services client and get last known location

```java
private FusedLocationProviderClient mFusedLocationClient;

// ..

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...

    mFusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
}
```

```java
mFusedLocationClient.getLastLocation()
        .addOnSuccessListener(this, new OnSuccessListener<Location>() {
            @Override
            public void onSuccess(Location location) {
                // Got last known location. In some rare situations this can be null.
                if (location != null) {
                    // Logic to handle location object
                }
            }
        });
```

        - The accuracy of the location returned by this is determined by what is specified in the manifest
        - Watch out for null if location settings are off, device is brand new or factory reset, google play services crashes
        -
- Location settings
    - When there is a request for location or receive permission updates the app specifies the required level of accuracy and power consumption and desired update interval, then the device automatically makes the appropriate changes to the system settings
    - Location request set up
        - Create a locationRequest to add parameters for the fused location provider
            - Update interval
                - setInterval()
                    - Sets the rate in ms at which the app prefers to receive location updates (may be faster)
            - Fastest update interval
                - setFastestInterval()
                    - Sets the fastest rate in ms at which your app can handle location updates
            - Priority
                - setPriority()
                    - Sets the priority of the request, helps google play services decide which location sources to use

- PRIORITY_BALANCED_POWER_ACCURACY
    - City block accuracy (100m) (coarse)
    - Cell tower and wifi for positioning
- PRIORITY_HIGH_ACCURACY
    - Most precise location
    - More likely to use gps for location
- PRIORITY_LOW_POWER
    - 10KM radius accuracy
- PRIORITY_NO_POWER
    - Gets updates from other apps

```
protected void createLocationRequest() {
    LocationRequest mLocationRequest = new LocationRequest();
    mLocationRequest.setInterval(10000);
    mLocationRequest.setFastestInterval(5000);
    mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
}
```

- Current location settings
    - Create LocationSettingsRequest.Builder and check if current location settings are satisfied

```
LocationSettingsRequest.Builder builder = new LocationSettingsRequest.Builder()
     .addLocationRequest(mLocationRequest);
```

```
LocationSettingsRequest.Builder builder = new LocationSettingsRequest.Builder();

// ...

SettingsClient client = LocationServices.getSettingsClient(this);
Task<LocationSettingsResponse> task = client.checkLocationSettings(builder.build());
```

- Prompt user to change location settings

```
task.addOnSuccessListener(this, new OnSuccessListener<LocationSettingsResponse>() {
    @Override
    public void onSuccess(LocationSettingsResponse locationSettingsResponse) {
        // All location settings are satisfied. The client can initialize
        // location requests here.
        // ...
    }
});

task.addOnFailureListener(this, new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        if (e instanceof ResolvableApiException) {
            // Location settings are not satisfied, but this can be fixed
            // by showing the user a dialog.
            try {
                // Show the dialog by calling startResolutionForResult(),
                // and check the result in onActivityResult().
                ResolvableApiException resolvable = (ResolvableApiException) e;
                resolvable.startResolutionForResult(MainActivity.this,
                        REQUEST_CHECK_SETTINGS);
            } catch (IntentSender.SendIntentException sendEx) {
                // Ignore the error.
            }
        }
    }
});
```

- Location updates
    - Used to provide the user with up to date information including device sensor readings
        - Available in the Location object, retrieved from the fused location listener
    - **Before requesting updates, the application needs to connect to location services and make a location request
    - Call requestLocationUpdates() and pass it the instance of the locationRequest and a LocationCallback
        - Define a startLocationUpdates() method as shown

```
@Override
protected void onResume() {
    super.onResume();
    if (mRequestingLocationUpdates) {
        startLocationUpdates();
    }
}

private void startLocationUpdates() {
    mFusedLocationClient.requestLocationUpdates(mLocationRequest,
            mLocationCallback,
            null /* Looper */);
}
```

    - Defining location update callback
        - Implement the LocationCallBack interface get the timestamp of the location update and display new data

```
private LocationCallback mLocationCallback;

// ...

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...

    mLocationCallback = new LocationCallback() {
        @Override
        public void onLocationResult(LocationResult locationResult) {
            if (locationResult == null) {
                return;
            }
            for (Location location : locationResult.getLocations()) {
                // Update UI with location data
                // ...
            }
        };
    };
}
```

    - Stopping and starting location updates

```java
@Override
protected void onPause() {
    super.onPause();
    stopLocationUpdates();
}

private void stopLocationUpdates() {
    mFusedLocationClient.removeLocationUpdates(mLocationCallback);
}
```

```java
@Override
protected void onResume() {
    super.onResume();
    if (mRequestingLocationUpdates) {
        startLocationUpdates();
    }
}
```

    - Need to save the state incase of and change to devices configuration
        - Store instance state in a bundle object

```java
@Override
protected void onSaveInstanceState(Bundle outState) {
    outState.putBoolean(REQUESTING_LOCATION_UPDATES_KEY,
            mRequestingLocationUpdates);
    // ...
    super.onSaveInstanceState(outState);
}
```

    - Restore the saved values from previous instance if available

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    // ...
    updateValuesFromBundle(savedInstanceState);
}

private void updateValuesFromBundle(Bundle savedInstanceState) {
    if (savedInstanceState == null) {
        return;
    }

    // Update the value of mRequestingLocationUpdates from the Bundle.
    if (savedInstanceState.keySet().contains(REQUESTING_LOCATION_UPDATES_KEY)) {
        mRequestingLocationUpdates = savedInstanceState.getBoolean(
                REQUESTING_LOCATION_UPDATES_KEY);
    }

    // ...

    // Update UI to match restored state
    updateUI();
}
```

- Display location
    - Convert the longitude and latitude (geographic location) into an address
    - Define an intent service to fetch the address

- Tasking so call it from a background thread

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.google.android.gms.location.sample.locationaddress" >
    <application
        ...
        <service
            android:name=".FetchAddressIntentService"
            android:exported="false"/>
    </application>
    ...
</manifest>
```

- Create a geocoder to do the reverse geocoding and pass it the Locale to ensure proper results

```
@Override
protected void onHandleIntent(Intent intent) {
    Geocoder geocoder = new Geocoder(this, Locale.getDefault());
    // ...
}
```

- Retrieve the address, define constants to contain the values

```
public final class Constants {
    public static final int SUCCESS_RESULT = 0;
    public static final int FAILURE_RESULT = 1;
    public static final String PACKAGE_NAME =
        "com.google.android.gms.location.sample.locationaddress";
    public static final String RECEIVER = PACKAGE_NAME + ".RECEIVER";
    public static final String RESULT_DATA_KEY = PACKAGE_NAME +
        ".RESULT_DATA_KEY";
    public static final String LOCATION_DATA_EXTRA = PACKAGE_NAME +
        ".LOCATION_DATA_EXTRA";
}
```

- Return to requester

```
public class FetchAddressIntentService extends IntentService {
    protected ResultReceiver mReceiver;
    // ...
    private void deliverResultToReceiver(int resultCode, String message) {
        Bundle bundle = new Bundle();
        bundle.putString(Constants.RESULT_DATA_KEY, message);
        mReceiver.send(resultCode, bundle);
    }
}
```

- Start the intent service

```
public class MainActivity extends AppCompatActivity implements
        ConnectionCallbacks, OnConnectionFailedListener {

    protected Location mLastLocation;
    private AddressResultReceiver mResultReceiver;

    // ...

    protected void startIntentService() {
        Intent intent = new Intent(this, FetchAddressIntentService.class);
        intent.putExtra(Constants.RECEIVER, mResultReceiver);
        intent.putExtra(Constants.LOCATION_DATA_EXTRA, mLastLocation);
        startService(intent);
    }
}
```

- Receive the results, if successful ResultData contains the address

```
public class MainActivity extends AppCompatActivity {

    // ...

    class AddressResultReceiver extends ResultReceiver {
        public AddressResultReceiver(Handler handler) {
            super(handler);
        }

        @Override
        protected void onReceiveResult(int resultCode, Bundle resultData) {

            if (resultData == null) {
                return;
            }

            // Display the address string
            // or an error message sent from the intent service.
            mAddressOutput = resultData.getString(Constants.RESULT_DATA_KEY);
            if (mAddressOutput == null) {
                mAddressOutput = "";
            }
            displayAddressOutput();

            // Show a toast message if an address was found.
            if (resultCode == Constants.SUCCESS_RESULT) {
                showToast(getString(R.string.address_found));
            }

        }
    }
}
```

- Detect activities
  - Detect if the user starts an activity such as walking, biking, driving
    - Use the Activity Recognition Transition API
      - Must declare a dependency to Google Location and Activity Recognition (API 12+)

- Specify the below permission

1. To declare a dependency to the API, add a reference to the Google maven repository and add an implementation entry to `com.google.android.gms:play-services-location:12.0.0` to the dependencies section of your app `build.gradle` file. For more information, see Set Up Google Play Services.
2. To specify the `com.google.android.gms.permission.ACTIVITY_RECOGNITION` permission, add a `<uses-permission>` element in the app manifest, as shown in the following example:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.example.myapp">

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

- Need an ActivityTransitionRequest object specifying the type of activity
- Need a PendingIntent callback to receive 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());
```

- Request task, Register task, Process task, Deregister task

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

```java
// myPendingIntent is the instance of PendingIntent where the app receives callbacks.
Task<Void> task =
  ActivityRecognition.getClient(context).requestActivityTransitionUpdates(request, myPendingIntent);

  task.addOnSuccessListener(
    new OnSuccessListener<Void>() {
      @Override
      public void onSuccess(Void result) {
        // Handle success
      }
    }
  );

  task.addOnFailureListener(
    new OnFailureListener() {
      @Override
      public void onFailure(Exception e) {
        // Handle error
      }
    }
  );
```

```java
@Override
protected void onReceive(Context context, Intent intent) {
  if (ActivityTransitionResult.hasResult(intent)) {
    ActivityTransitionResult result = ActivityTransitionResult.extractResult(intent);
    for (ActivityTransitionEvent event : result.getTransitionEvents()) {
        // chronological sequence of events....
    }
  }
}
```

```java
// myPendingIntent is the instance of PendingIntent where the app receives callbacks.
Task<Void> task =
  ActivityRecognition.getClient(context).removeActivityTransitionUpdates(myPendingIntent);

task.addOnSuccessListener(
  new OnSuccessListener<Void>() {
    @Override
    public void onSuccess(Void result) {
      myPendingIntent.cancel();
    }
  });

task.addOnFailureListener(
  new OnFailureListener() {
    @Override
    public void onFailure(Exception e) {
      Log.e("MYCOMPONENT", e.getMessage());
    }
  });
```

- Add maps (Google Maps Android API)
    - Allow users to:
        - Identify locations with custom markers

        - Augment the map data with image overlays
        - Embed one or more maps as fragments
        - Etc.
    - Google Maps Android API v2 allows embedded maps into activities as fragments with simple XML
    - Add markers to the map
        - Define color and icons
        - Draw polylines and polygons to indicate paths and regions
    - Ability to control rotation, tilt, zoom, pan properties
    - Embedded street view for 360 degree views

References

- Sensors

https://developer.android.com/guide/topics/sensors/
https://developer.android.com/guide/topics/sensors/sensors_overview#java
https://developer.android.com/guide/topics/sensors/sensors_motion
https://developer.android.com/guide/topics/sensors/sensors_position
https://developer.android.com/guide/topics/sensors/sensors_environment
https://developer.android.com/guide/topics/sensors/sensors_position

- Location and Tracking

https://developer.android.com/training/location/
https://developer.android.com/guide/topics/location/battery
https://developer.android.com/training/location/retrieve-current
https://developer.android.com/guide/topics/location/transitions
https://developer.android.com/training/maps/