DOCUMENTATION

Mobile-Sensing Library

Dennis Lawo

dennis.lawo@uni-siegen.de

Table of Contents

[Kurzbeschreibung 2](#_Toc512788414)

[Verwendete Libraries 2](#_Toc512788415)

[Sensoren 2](#_Toc512788416)

[Speicherung 2](#_Toc512788417)

[Events 2](#_Toc512788418)

[Funktionsbeschreibung 2](#_Toc512788419)

[Sensoren 2](#_Toc512788420)

[Location 2](#_Toc512788421)

[Google Activity 2](#_Toc512788422)

[Google Activity Transition 3](#_Toc512788423)

[Network 3](#_Toc512788424)

[Clustering 3](#_Toc512788425)

[Track 4](#_Toc512788426)

[RunningApplication 4](#_Toc512788427)

[ScreenOn 4](#_Toc512788428)

[EventBus 4](#_Toc512788429)

[Speicherung 4](#_Toc512788430)

[Upload 4](#_Toc512788431)

[Nutzung der Library 5](#_Toc512788432)

[SensingManager 5](#_Toc512788433)

[UploadManager 5](#_Toc512788434)

[Upload Manager 5](#_Toc512788435)

[Datenadapter Beispiel 5](#_Toc512788436)

# Brief Description

The Android Library contains different sensors and manages the handling of the data from collection to storage. The SensingManager-Class is the interface to the parent app, which includes the library. Furthermore, events are used to communicate within the library but also with the parent app. Those Events can be intercepted to display or upload the collected data.

# Included Libraries

## Sensors

* Intel Context Sensing SDK (https://software.intel.com/en-us/documentation/context-sensing-sdk-for-android-states-datasheet)
* Google Location (https://developers.google.com/android/reference/com/google/android/gms/location/package-summary)
* Google Activity Recognition & Google Activity Transition (https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionClient)
* TNME / Clustering (https://github.com/martinste1n/TNME/tree/ProximityRideSharing)

## Storage

* ObjectBox (http://objectbox.io/)

## Events

* GreenRobot Event Bus (http://greenrobot.org/eventbus/)

# Functioning

## Sensors

In the following, the implemented sensors are briefly described and their resulting objects are presented.

### Location

This sensor is implemented using the GoogleLocationAPI. The current GPS position of the device is recorded at a given interval.

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Type of Data** | **Description** |
| Id | Long | ID in the DB |
| Timestamp | Long | Time of collection in milliseconds |
| Lat | Double | Latitude Coordinate |
| Lng | Double | Longitude Coordinate |
| Speed | Float | Speed |
| isClustered | Boolean | Clustered Yes/No |
| parentCluster | Long | ID of the resulting Cluster |

Table 1 Description of the LocationObject

### Google Activity

This sensor is implemented using Google Activity Recongition. The current activity (IN\_VEHICLE, ON\_BICYCLE, ON\_FOOT, STILL, UNKNOWN, TILTING, WALKING, RUNNING) is recorded in a given interval.

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Type of Data** | **Description** |
| Id | Long | ID in the DB |
| Timestamp | Long | Time of collection in milliseconds |
| Activity | String | Name of the Activity |
| Probability | int | Probability of the Activity |

Table 2 Description of the GoogleActivityObjects

### Google Activity Transition

This sensor is implemented using Google Activity Recongition. As an extension to the regular storage of activities (IN\_VEHICLE, ON\_BICYCLE, ON\_FOOT, STILL, UNKNOWN, TILTING, WALKING, RUNNING), this sensor outputs a value when a transition occurs between two activities.

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Type of Data** | **Description** |
| Id | Long | ID in the DB |
| Timestamp | Long | Start of the Activity |
| Endtime | Long | End of the Activity |
| Activity | String | Name of the Activity |

Table 3 Description of the GoogleActivityTransitionObjects

### Network

This sensor is implemented using the Intel Context Sensing SDK. The current network status is collected at a given interval. If the network type (e.g. LTE or WIFI) differs from the previous type, a new NetworkObject is created.

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Type of Data** | **Description** |
| Id | Long | ID in the DB |
| Timestamp | Long | Time of collection in milliseconds |
| NetworkType | String | Network Type |

Table 4 Description of the NetworkObjects

### Clustering

This sensor (from https://github.com/martinste1n/TNME/tree/ProximityRideSharing) calculates ClusterLocations every 24 hours for the available locations.

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Type of Data** | **Description** |
| Id | Long | ID in the DB |
| date | Long | Date of collection |
| firstseen | Long | First occurrence of the Location |
| count | int | Location Count |
| lat | double | Latitude Coordinate |
| lng | double | Longitude Coordinate |
| Metajson | String | ClusterMetaData as JSON |

Table 5 Description of the ClusterObjects

The MetaJson describes the metadata of the cluster, which is needed for the internal calculations of the sensor and contains among other things the convex hull of locations around the cluster as well as the cluster type.

### Track

With each newly recorded location, this sensor checks whether there is an overlap with a cluster, with the result "in cluster" or "outside the cluster". Based on these results, a route ("Outside the Cluster") is interpreted as the time and locations between two "In Cluster" results.

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Type of Data** | **Description** |
| Id | Long | ID in the DB |
| startTimestamp | Long | Starttime of the route |
| endTimestamp | Long | Endtime of the route |

Table 6 Description of the TrackObjects

### RunningApplication

This sensor checks the current foreground app at a given interval, when the smartphone is awake, and stores it.

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Type of Data** | **Description** |
| Id | Long | ID in the DB |
| timestamp | Long | Time of collection in milliseconds |
| ApplicationName | String | Name of the APp |

Table 7 Description of the RunningApplicationObjects

### ScreenOn

This sensor checks the current status of the screen at a given interval, when the smartphone is awake.

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Type of Data** | **Description** |
| Id | Long | ID in the DB |
| timestamp | Long | Time of collection in milliseconds |
| screenOn | boolean | Screen On/Off |

Table 8 Description of the ScreenOnObjects

## EventBus

Within the app, the EventBus from GreenRobot is used for communication.

The SensorDataEvent is fired by the sensors when they have collected a new value. The event contains the sensor object.

The UploadEvent is fired by the UploadManager at a given interval and contains a SensorTimeseries, which is ready for upload.

## The TrackEndEvent is fired by the TrackSensor if it has a new track and can be used to show this to the user or to open a “RatingActivity”.

## Storage

## The storage is implemented using the SensorObjects, described above and the "ObjectBox" library. The sensor objects are accumulated on a daily basis in the so-called "SensorTimeseries" classes.

## Upload

The app using the SensingLibrary must implement a data adapter for the upload. This is a class that catches the UploadEvent and uploads the SensorTimeseries in the desired format.

# Usage of the Library

## SensingManager

The SensingManager is required to start the sensors. This can be used as follows.

Module.*init*(*context*, "USERNAME");  
*sensMang* = Module.*getSensingManager*();  
*sensMang*.setSensingSetting(SensingManager.SensorNames.*Activity*,true);  
*sensMang*.setSensingSetting(SensingManager.SensorNames.*GPS*,true);  
*sensMang*.setSensingSetting(SensingManager.SensorNames.*WLANUpload*,true);  
*sensMang*.setSensingSetting(SensingManager.SensorNames.*ScreenOn*,true);  
*sensMang*.setSensingSetting(SensingManager.SensorNames.*Apps*,true);  
*sensMang*.setSensingSetting(SensingManager.SensorNames.*Network*,true);  
*sensMang*.setSensingSetting(SensingManager.SensorNames.*Track*,true);  
*sensMang*.setSensingSetting(SensingManager.SensorNames.*Cluster*,true);  
*startSensing*();

## In addition to the On/Off settings, it is also possible to set the tracking interval and other options using the "AdvancedSettings".

## UploadManager

### Upload Manager

uplMang = Module.*getUploadManager*();

uplMang.setDailyUpload(*context*);  
*uploader* = new ParseUploader();

### Data Adapter Example

public class ParseUploader {  
  
 public ParseUploader()  
 {  
 EventBus.*getDefault*().register(this);  
 }  
  
 // This method will be called when a MessageEvent is posted (in the UI thread for Toast)  
 @Subscribe(threadMode = ThreadMode.*BACKGROUND*)  
 public void onMessageEvent(UploadEvent event) {  
 uploadToParse(event.data);  
 }  
  
 public void uploadToParse(SensorTimeseries st)  
 {  
 try {  
 ObjectBoxAdapter oba = new ObjectBoxAdapter();  
 //GLocationif (st.getClass().getName().equals(GLocationTimeseries.class.getName())) {  
 GLocationTimeseries glocTimeseries = (GLocationTimeseries) st;  
 ParseObject po = new ParseObject("Location");  
 po.put("basetime",glocTimeseries.getTimestamp\_day());  
 po.put("meta", new JSONObject());  
 po.put("name", "Location");  
 po.put("icon", "Icons/smarthome/default\_18.png");  
 JSONArray ja = new JSONArray();  
 JSONObject jo = new JSONObject();  
 jo.put("name", "GeoJSON");  
 jo.put("type","Geo");  
 ja.put(jo);  
 JSONObject jo2 = new JSONObject();  
 jo2.put("name","Timestamp");  
 jo2.put("type","Number");  
 ja.put(jo2);  
 po.put("valueTypes",ja );  
 po.put("user", ParseUser.*getCurrentUser*().getUsername());  
 JSONArray values = new JSONArray();  
 for(GLocationsObject gLocationsObject: glocTimeseries.getValues()){  
   
 JSONObject value = new JSONObject();  
 value.put("type","Feature");  
 JSONObject geo = new JSONObject();  
 geo.put("type","Point");  
 JSONArray coords = new JSONArray();  
 coords.put(gLocationsObject.getLat());  
 coords.put(gLocationsObject.getLng());  
 geo.put("coordinates",coords);  
 value.put("geometry",geo);  
 JSONObject props = new JSONObject();  
 props.put("timestamp",gLocationsObject.getTimestamp());  
 value.put("properties",props);  
 values.put(value);  
 }  
 po.put("values",values);  
 glocTimeseries.setUploaded(true);  
 oba.updateSensorTimeseries(glocTimeseries);  
 po.saveInBackground(updateSensorTimeseriesUpdated(glocTimeseries));  
 }  
 }catch(Exception e){  
 undoUpdated(st);  
 }  
 }  
  
 public SaveCallback updateSensorTimeseriesUpdated(final SensorTimeseries st){  
 return new SaveCallback() {  
 @Override  
 public void done(ParseException e) {  
 if(e == null){  
   
 }else{undoUpdated(st);  
 }  
 }  
 };  
 }  
 public void undoUpdated(SensorTimeseries st){  
 ObjectBoxAdapter oba = new ObjectBoxAdapter();  
 //GLocation  
 if (st.getClass().getName().equals(GLocationTimeseries.class.getName())) {  
 GLocationTimeseries glt = (GLocationTimeseries) st;  
 glt.setUploaded(false);  
 oba.updateSensorTimeseries(glt);  
 }  
}