Mobility Logger for Android

Danilo Valerio

Abstract

Collecting real traces of human mobility is a crucial prerequisite for several scientific studies. Recently, several applications have been published that allow the collection of GPS mobility traces from new generation smart-phones. However, it is hard to find reliable and stable applications that combine the collection of mobility data along with cellular and WiFi network details.

As part of a small project for one of my university lectures, I developed an application that permits to log GPS coordinates and cellular network details on any Android-based smart-phone. I explored deeply the Android telephony and location APIs, and developed a logger that parses, shows in a simple GUI, and saves in a CSV format any available radio and GPS information. Later, I enriched the application with several new core features and I optimized it focusing on efficiency, stability, and speed. In particular, I included the capability to monitor wifi access points' information, I added a module for logging neighboring cells in 2G/3G networks, and I extended the application with the possibility to output log files in kml format for easy viewing on Google Earth.

The final application is called "Mobility Logger for Android", in short "Mobilog", and it clearly belongs to the category of war-driving applications. Application binaries and source code (released under GPL version 2 license) can be freely downloaded from http://www.valerio.im/others.html#Mobilog.

1. Application Description

Mobilog is a war-driving application for Android smart-phones that logs the following data:

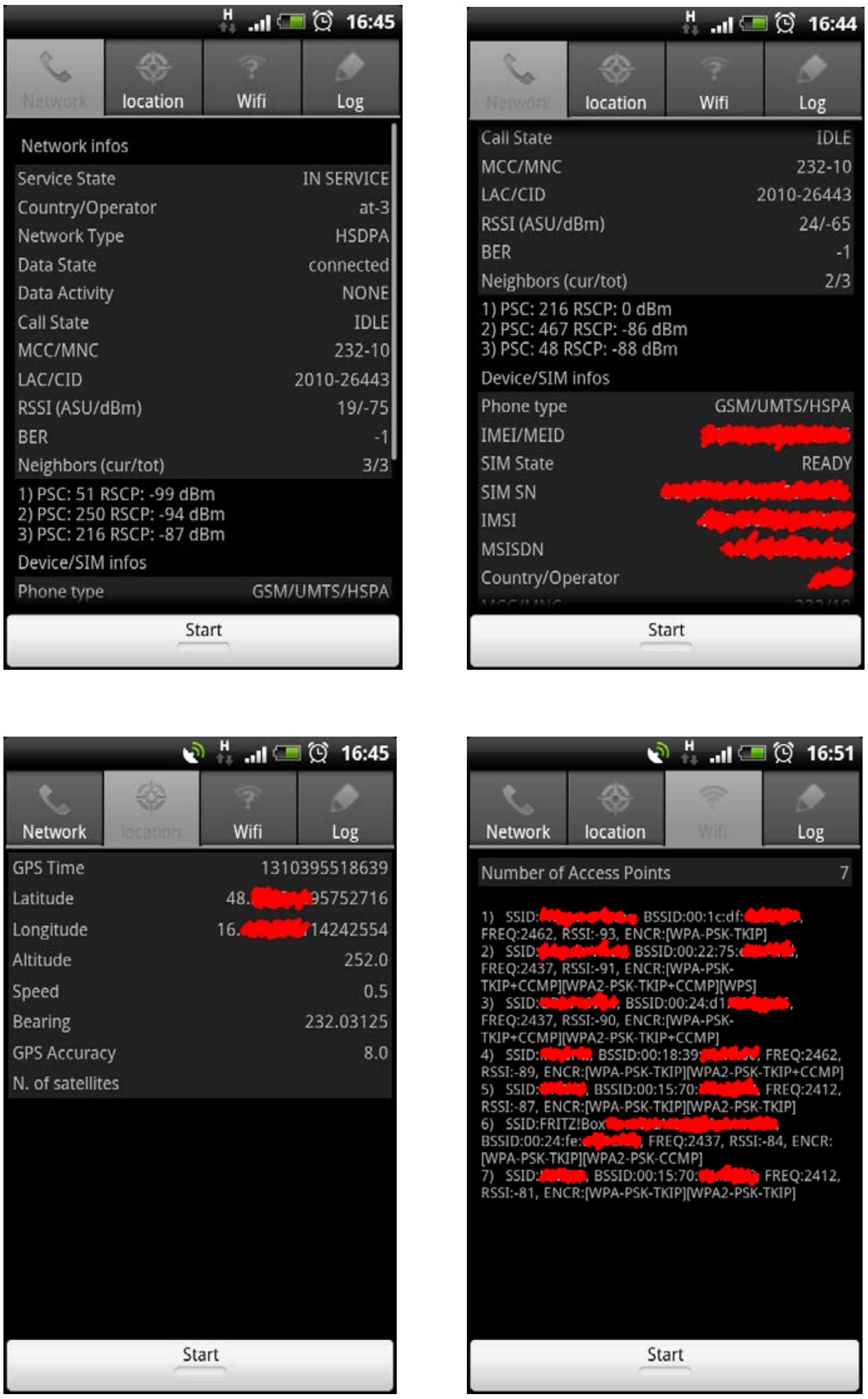
1) Information about cellular network, including cell-ids, signal strengths, neighboring cells, connection states, etc.,
2) Information about wifi access points, including SSID, BSSID, signal strengths, frequency, encryption, etc.,
3) Information from the GPS chipset, including latitude, longitude, altitude, speed, bearing, etc.

Logs can be synchronous (data is collected at specific time intervals) or asynchronous (data is collected based on GPS events or cell changes). For most use cases, the asynchronous collection means that a user can choose a value X, so that a point is logged whenever the position changes of more than X meters (see Section 4 for more details).

Tracks are saved into the "external storage" of the smartphone (for many devices this means the SD-CARD storage or the built-in non-system storage). The user can decide whether to just use plain CSV format, or to additionally include for each path a KML file for visual analysis on Google Earth and Google Maps.

The application is designed to be energy-efficient and simple. For this reason, I decided not to include map views or graphically enhanced GUIs. The interface is anyway intuitive, fluid and feels user-friendly. The following pictures present few screenshots taken from a smartphone while the application is running. As shown in the picture, data is grouped into four tabs, i.e.

Network, Location, WiFi, and Log. Each tab provides the related information in form of a scrollable table, continuously updated in real-time.



```
Network infos
Service State                IN SERVICE
Country/Operator                   at-3
Network Type                      HSDPA
Data State                    connected
Data Activity                      NONE
Call State                         IDLE
MCC/MNC                          232-10
LAC/CID                      2010-26443
RSSI (ASU/dBm)                   19/-75
BER                                  -1
Neighbors (cur/tot)                 3/3
1) PSC: 51 RSCP: -99 dBm
2) PSC: 250 RSCP: -94 dBm
3) PSC: 216 RSCP: -87 dBm
Device/SIM infos
Phone type                GSM/UMTS/HSPA
```

```
Call State                         IDLE
MCC/MNC                          232-10
LAC/CID                      2010-26443
RSSI (ASU/dBm)                   24/-65
BER                                  -1
Neighbors (cur/tot)                 2/3
1) PSC: 216 RSCP: 0 dBm
2) PSC: 467 RSCP: -86 dBm
3) PSC: 48 RSCP: -88 dBm
Device/SIM infos
Phone type                GSM/UMTS/HSPA
IMEI/MEID                    [redacted]
SIM State                         READY
SIM SN                       [redacted]
IMSI                         [redacted]
MSISDN                       [redacted]
Country/Operator             [redacted]
```

```
GPS Time                  1310395518639
Latitude           48.[redacted]95752716
Longitude          16.[redacted]14242554
Altitude                          252.0
Speed                               0.5
Bearing                       232.03125
GPS Accuracy                        8.0
N. of satellites
```

```
Number of Access Points                7

1)  SSID:[redacted] BSSID:00:1c:df:[redacted],
FREQ:2462, RSSI:-93, ENCR:[WPA-PSK-TKIP]
2)  SSID:[redacted] BSSID:00:22:75:[redacted],
FREQ:2437, RSSI:-91, ENCR:[WPA-PSK-
TKIP+CCMP][WPA2-PSK-TKIP+CCMP][WPS]
3)  SSID:[redacted], BSSID:00:24:d1:[redacted],
FREQ:2437, RSSI:-90, ENCR:[WPA-PSK-
TKIP+CCMP][WPA2-PSK-TKIP+CCMP]
4)  SSID:[redacted], BSSID:00:18:39:[redacted], FREQ:2462,
RSSI:-89, ENCR:[WPA-PSK-TKIP][WPA2-PSK-TKIP+CCMP]
5)  SSID:[redacted], BSSID:00:15:70:[redacted], FREQ:2412,
RSSI:-87, ENCR:[WPA-PSK-TKIP][WPA2-PSK-TKIP]
6)  SSID:FRITZ!Box[redacted],
BSSID:00:24:fe:[redacted], FREQ:2437, RSSI:-84, ENCR:
[WPA-PSK-TKIP][WPA2-PSK-CCMP]
7)  SSID:[redacted], BSSID:00:15:70:[redacted], FREQ:2412,
RSSI:-81, ENCR:[WPA-PSK-TKIP][WPA2-PSK-TKIP]
```

Fig. 1 - Main GUI (screenshots)

2. System Architecture

While coding, my first priorities were efficiency and stability, sometimes overlooking elegance and code reusability.

This application runs in a single process. When it starts, an activity is created (i.e. the GUI), that registers a few listeners to react to changes of location, signal-strengths, and cell-ids. In addition it registers a broadcast receiver that reacts to the reception of new WiFi scan results.

As soon as the <Start> button is pressed, an "Android Service" is created, that registers independent listeners and broadcast receivers. In this way, the GUI can be closed while the application still logs in background the necessary information.[1] Services in android have higher priority over other activities, making them perfect for this kind of logging tasks.

The software architecture is depicted in Fig. 2.

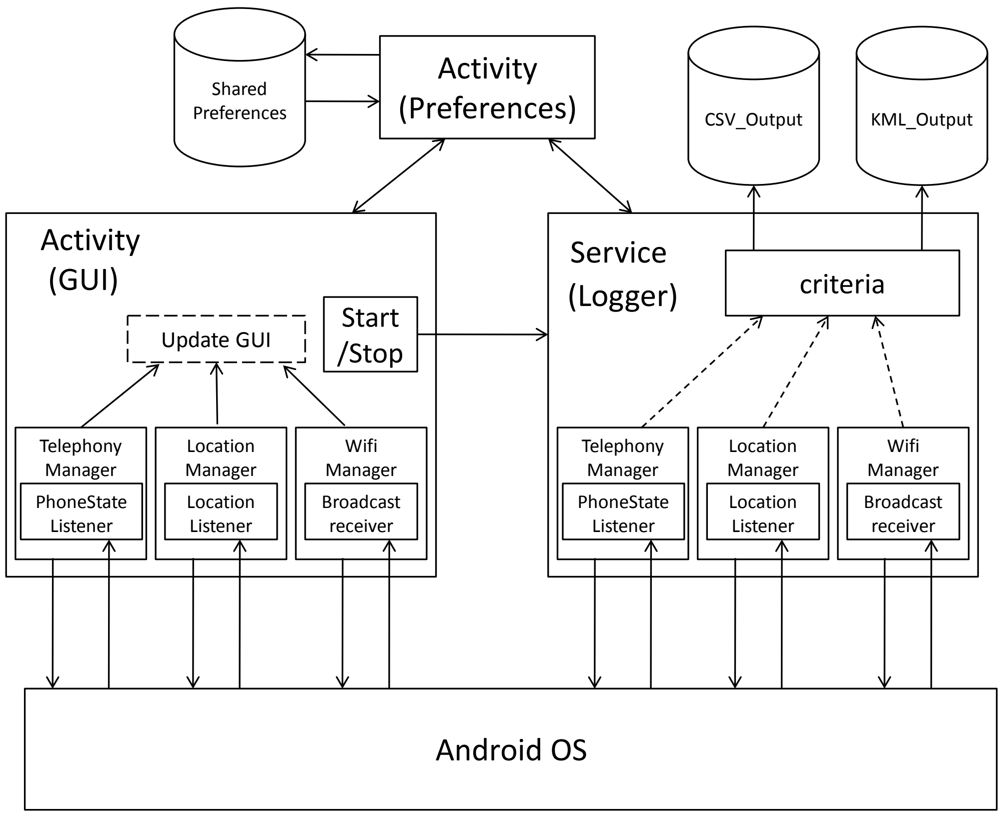


Fig. 2 - Software components

3. System Prerequisites

The application runs on any Android device that meets following requirements:

- Android OS 2.1 or higher
- GPS chipset
- External storage (or internal built-in storage)
- A file browser (for reading/copying the logs)

Please understand that, due to the large number of Android smart-phones on the market, I could not test all the functionalities in all smart-phones. However the application has been

[1] If the application is closed or put into background before pressing <start>, then all listeners are unregistered and the application will not consume any resource.

developed and tested thoroughly on two devices, i.e. Samsung Galaxy I9000 and HTC Desire Bravo. The Samsung Galaxy was running Android OS version 2.2 Froyo and the HTC Desire was running Android OS 2.3 Gingerbread. Both tests have been successful and every function ran smoothly. No bugs have been found in any of the tested devices.

4. Installing and Call Syntax

The installation of Mobilog is as simple as for any Android application. The downloadable binary consists of a file with extension .apk. By running this file **directly on the smart-phone**, the installation process begins. If the installation finishes successfully, the Mobilog icon will appear in the list of installed applications.

Using the application:

Using the application is very intuitive and can be summarized in the following steps:

1) Launch the application,

2) Be sure to choose your options by going to menu→Settings (see below),

3) Press <Start> and optionally close the app to avoid continuous refreshes of the screen and save some battery,

4) Drive, walk, bike, jog, or sit in a public transport,

5) Reopen the app (if closed) and press <Stop>.

6) You will find your log into /<SD-CARD>/Mobilog

IMPORTANT: The file will be closed (and the memory will be flushed) only when you press <Stop>. Accessing the file while monitoring should be harmless but it is not suggested, as buffering is used for writing into the sd-card file system.

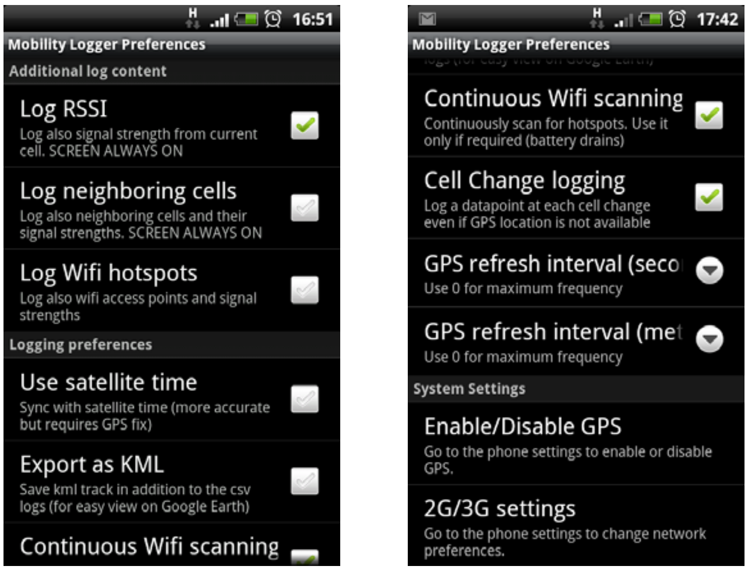
Available settings:

By pressing the menu button it is possible to enter the settings panel. Settings can be changed also while logging, but will be effective only for the next logging session. This is to avoid csv files with different number of elements per line. The Settings view looks as depicted in Fig. 3.

Here a list of the available settings followed by the respective explanation:

- *Log RSSI*: If enabled, each logged data point will also contain the measured RSSI (Received Signal Strength Indicator) from the current cell. PLEASE NOTE that the Android operating system provides RSSI through the API, ONLY AND ONLY IF the screen is ON. Therefore, if this option is enabled the application will try to keep the screen ON (but at minimum brightness) by acquiring a wakelock. If you switch off the screen (for example by pressing the power button) the RSSI value will not be updated anymore (at least as of Gingerbread).

- *Log neighboring cells*: If enabled, each logged data point will also contain the list of neighboring cells including the respective measured RSSI (therefore screen on). In case of a 2G network, neighboring cells will be identified by their cell-ID. In case of a 3G network, neighboring cells will be identified by their PSC (Primary Scrambling Code).


16:51
Mobility Logger Preferences
Additional log content
Log RSSI ✓
Log also signal strength from current cell. SCREEN ALWAYS ON
Log neighboring cells ☐
Log also neighboring cells and their signal strengths. SCREEN ALWAYS ON
Log Wifi hotspots ☐
Log also wifi access points and signal strengths
Logging preferences
Use satellite time ☐
Sync with satellite time (more accurate but requires GPS fix)
Export as KML ☐
Save kml track in addition to the csv logs (for easy view on Google Earth)
Continuous Wifi scanning

17:42
Mobility Logger Preferences
Continuous Wifi scanning ✓
Continuously scan for hotspots. Use it only if required (battery drains)
Cell Change logging ✓
Log a datapoint at each cell change even if GPS location is not available
GPS refresh interval (seco
Use 0 for maximum frequency
GPS refresh interval (met
Use 0 for maximum frequency
System Settings
Enable/Disable GPS
Go to the phone settings to enable or disable GPS.
2G/3G settings
Go to the phone settings to change network preferences.

Fig. 3 – Preference list (screenshots)

- ***Log Wifi hotspots:*** If enabled, each logged data point will include data about the currently visible Access Points, i.e. SSID, BSSID, Frequency, RSSI, Encryption algorithm.

- ***Export as KML:*** If enabled, the application will generate two logs: <date>_<time>.csv and <date>_<time>.kml. You can open the kml in Google Earth. Normal datapoints are marked with a **green** point. Cell changes are marked with a **yellow** point. Changes of Location Area Code are marked with a **red** point. Each placemark contains a baloon as shown in Fig. 4.

- ***Continuous Wifi scanning:*** If enabled, the application will continuously scan for new access points. The application will execute a new scan as soon as the current one finishes. This is a bit power consuming, but required if you want to get everything at every instant. If this option is NOT enabled, the scans will be performed as designed by your phone manufacturer (for me, they are done every 60s).

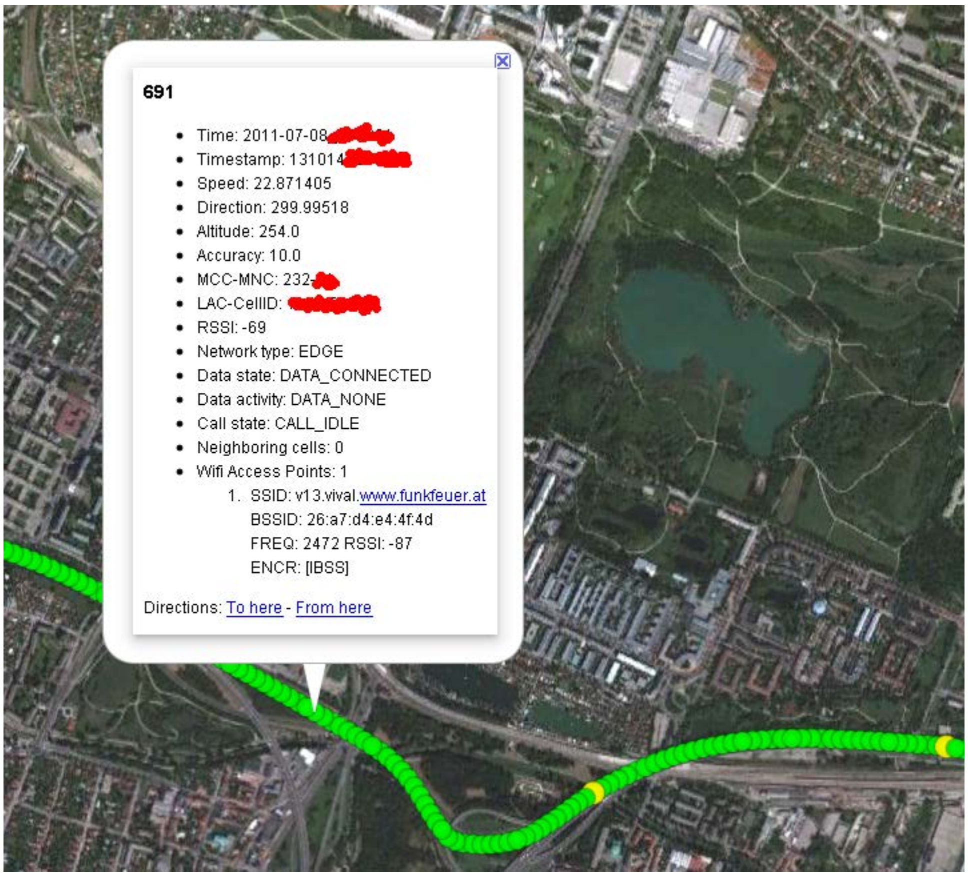
A small hint for rooted users: If you don't want to perform continuous scans, but you find the 60s time interval too large, you can just change the value of the periodic scans by modifying the *build.prop* file in the directory /system/. A search on google will help you to find out the right key to modify. This has effect on a system level, so remember to put it back to the original value when you are finished.

- ***Cell Change logging:*** If enabled, a new data point will be added when there's a cell change AND no GPS coordinates are present. There's no reason to disable this. So I'll probably remove this option.

- ***Logging interval (seconds):*** The interval for saving data-points. Data points will be saved even if no changes are detected by the GPS chipsets. (Set this to zero if you prefer asynchronous logging based on events from GPS and cell changes).

- ***GPS Refresh Interval (Seconds)*** Minimum number of seconds between each datapoint (Suggested value: zero).

- ***GPS Refresh Interval (Meters):*** Minimum number of meters between each data point. Too large values make your log inaccurate. Too small values make your log crowded of points. I always use 10, since in my mobile phones, values below 10 generate datapoints even if I am not moving (probably due to the inaccuracy of the GPS chipset).


691

- Time: 2011-07-08[redacted]
- Timestamp: 131014[redacted]
- Speed: 22.871405
- Direction: 299.99518
- Altitude: 254.0
- Accuracy: 10.0
- MCC-MNC: 232-[redacted]
- LAC-CellID: [redacted]
- RSSI: -69
- Network type: EDGE
- Data state: DATA_CONNECTED
- Data activity: DATA_NONE
- Call state: CALL_IDLE
- Neighboring cells: 0
- Wifi Access Points: 1
    1. SSID: v13.vival.www.funkfeuer.at
       BSSID: 26:a7:d4:e4:4f:4d
       FREQ: 2472 RSSI: -87
       ENCR: [IBSS]

Directions: To here - From here

Fig. 4 – KML Output example on Google Earth

5. Software Modules

The application consists of two *Activities* and one *Service*:

1) MainActivity

This class extends the android *TabActivity* class and overrides few of the lifecycle methods. In particular:

- *onCreate*() is executed when the application is launched. It gets the references to all the objects in the gui, it declares and acquires all required *SystemServices*, draws the GUI with the tabs, and initialize the toggle button waiting for any user touch.
- *onResume*() is executed right after *onCreate()* and/or whenever the application comes in foreground after being in background. Here the application registers all the listeners and starts reacting to changes of locations, network states, wifi, etc.
- *onPause*() is executed whenever the application is not anymore visible, i.e. the user put the GUI in background or another app takes the focus (e.g. a call is received). Here ALL listeners are unregistered and potential wake-locks are released. Note that the listeners registered in the MainActivity serve only the purpose of updating the GUI, therefore it does not make sense to keep them running if the GUI is not in foreground.

After executing *onCreate*() and *onResume*() the application has registered all the required listeners. Whenever a listener is notified of a change, the corresponding method is called reacting accordingly, i.e. updating the value in memory and the GUI. For details, refer to the source code.

2) PreferencesActivity

This class is just used for constructing the Settings view. It actually reads an xml resource file and draws the Preferences list with the defined categories, fields, and types.

3) LoggingService

This class extends the android *Service* class. Similarly to the MainActivity it overrides some lifecycle methods, such as *onCreate*(), *onDestroy*(), *onBind*(), and *onStartCommand*().

onCreate() is called when the user presses the start button in the GUI. This method reads the preferences resource, opens and initializes files and other necessary values, and registers all required listeners and broadcast receivers. Differently from the MainActivity, these listeners remain registered until the user presses the stop button, i.e. the service remains working in background until the user explicitly opens the MainActivity and decides to stop it. Pressing Stop calls the *onDestroy*() method that close the files, unregisters the listeners and the receivers, and shut the service down.

The methods invoked by the listeners and the receivers are similar to the ones also present in the MainActivity. This time however, the changes of state (depending on the user preferences) do not lead to an update of any GUI, but are written to the output log files. Note that registering independent listeners and receivers twice (once in the MainActivity and once in the LoggingService) is a choice led by the necessity to keep the application fast and light, i.e. avoiding inter-process communication and/or messenger class between activities and services.

6. Example of output

The output format of each line in the CSV file follows the following scheme:

```
System_timestamp;GPS_timestamp;latitude;longitude;altitude;speed;direction;gpsaccuracy;mcc;mnc;lac;cellid;signalstrength;network_type;data_state;data_activity;voice_activity;neighboring_cells;visible_accesspoints;
```

The following snippet instead depicts an example of a single line from the output file.

```
1310111533838;1310111533000;48.129801;16.29461;317.0;11.846891;101.988716;5.0;232;01;3104;26710;-81;EDGE;DATA_CONNECTED;DATA_NONE;CALL_IDLE;0;4|SSID: FRITZ!Box Fon WLAN 7140 Annex A, BSSID: 00:1f:3f:18:0e:8f, capabilities: [WPA-PSK-TKIP][WPA2-PSK-CCMP], level: -90, frequency: 2437|SSID: PBS-2226C1, BSSID: 30:39:f2:22:26:c6, capabilities: [WPA-PSK-TKIP][WPS], level: -92, frequency: 2412|SSID: PBS-AB4F51, BSSID: 00:25:53:ab:4f:58, capabilities: [WPA-PSK-TKIP][WPS], level: -94, frequency: 2437|SSID: SpeedTouchBC5BBF, BSSID: 00:90:d0:f1:6b:4b, capabilities: [WPA-PSK-TKIP+CCMP][WPA2-PSK-TKIP+CCMP], level: -97, frequency: 2412;
```

Apart from the timestamps and the location details from the GPS chipsets (first eight fields), this line shows the following information:

- The phone was in an Austrian network (MCC=232);
- The phone was connected to the A1 network (MNC=01);
- The phone was connected to the cell 26710 belonging to the LAC 3104;
- The phone perceived a signal strength of -81dBm;
- The phone was in EDGE mode;
- The phone was attached to the Packet Switched domain of the network and had an active PDP context (DATA_CONNECTED);
- The phone was not transferring any data (DATA_NONE);
- The phone was in idle state for the Circuit Switched domain (CALL_IDLE);
- No neighboring cells were present (0);
- Four Access points (separated by the character “ | ”) were visible in two different channels (2412MHz, 2437MHz), along with SSID, BSSID and signal strengths. All of them had WPA or WPA2 encryption enabled.