ITRC Data Documentation

Title	ITRC Data Documentation v 1.0
Document Number	
Author (Organization)	Rischan Mafrur, Advanced Network Lab, CNU
Project Team	
Creation Date	2015-02-16
Last Modified	2015-02-17
Version	1.0
Status	First Version
Project Link	https://github.com/rischanlab/Rfunf
Path of Data	C:\ITRC_DATA (Rischan PC)

Revision History:

Modified By	Date	Version	Comments
Rischan	2015-02-17	1.0	First Version

Contents

Research Overview

Nowadays, smartphone capability has increased significantly. Smartphone has equipped with high processor, bigger memory, bigger storage and etc. With this equipment, smartphone has capability to running complex application. Many sensor also has embedded to the smartphone. With this sensor and log capability of smartphone, we can develop many useful system or application in different domain such as healthcare (elderly monitoring system, human fall detection), transportation (monitoring road and traffic condition), personal and social behavior, environmental monitoring (pollution, weather), and etc. To develop such system, we have to collect the user personal data and then analyze it. There are two ways to collect personal data from the users based on user involvement, they are:

1. Participatory sensing
2. Opportunistic sensing

Participatory sensing means the application still need user's intervention to complete their task. The examples for such application need user to taking text input for each time period, taking picture and etc. On the other hand, opportunistic sensing means application does not need user's intervention to complete their task, users not involved in making decisions instead smart phone itself make decisions according to the sensed and stored data.

Our research focus on opportunistic sensing which is we develop application which does not need user’s intervention. To analyze the result, because of the data that we collected does not have any label so we prefer to use unsupervised learning.

In this research we develop two systems are:

1. Application data collector
2. Data extraction and visualization

Application data collector is application that we used for collecting user’s personal data. This application is android application. After we have all of data from the user, we have to extract and visualize, and also analyze it. To extract, visualize, and analyze the data, we use R programing language.

Application Data Collector

To develop application data collector, we do not develop from scratch, we use Funf library. The Funf Open Sensing Framework is an Android-based extensible framework, originally developed at the MIT Media Lab, for doing phone-based mobile sensing. Funf provides a reusable set of functionalities enabling the collection and configuration for a broad range of data types. Funf is open sourced under the LGPL license. Figure 1 shows Funf framework can collect many of sensing from smartphone such location, movement, communication and usage, social proximity, and many more. In this document, we do not describe details about Funf architecture but we describe about the data that we have collected and how to extract, visualize and analyze it. More details about Funf architecture can be seen in the main site of Funf[1] and also Funf developer site[2].

Table 1: **List of probes and time period of recording**

No.	Probes	Interval,duration(s)
#1	Location(GPS)	300
#2	Wi-Fi	300
#3	Bluetooth	300
#4	Battery	300
#5	Call Log	86400
#6	SMS Log	86400
#7	Application Installed	86400
#8	Hardware Info	86400
#9	Contact	86400
#10	Browser Search Log	86400
#11	Browser Bookmark	86400
#12	Light Sensor	120,0.07
#13	Proximity	120,0.07
#14	Temperature	120,0.07
#15	Magnetic Field	120,0.07
#16	Pressure	120,0.07
#17	Activity Log	120,0.07
#18	Screen Status	120,0.07
#19	Running Application	120,0.07

Data Description

Our application follows opportunistic sensing because we do not want to bothering user much. To do that we must define the time (interval and duration), when the application will request the data from the smartphone. Interval means how many times in second system will send data request to the smartphone. The example, we set interval 300 seconds means 5 minutes, so application will request and store the data for every 5 minutes. Duration is used in sensor data because without duration is useless to collect the sensors data. The example of duration, when we set interval 120 seconds and duration 0.07 s so the application will send data request to the smartphone for every 2 minutes and the system will record the data during 0.07 seconds.

Table 1.shows the interval and duration from each probes. Those interval and duration already tested and we thought those setting was optimum one but we can change those setting by change the value on the string.xml in android project. Figure 2a shows the

[1]http://www.funf.org/
[2] https://code.google.com/p/funf-open-sensing-framework/wiki/FunfArchitecture

string.xml file in the directory of android project and Figure 2b shows inside the string.xml file, we can change value of interval and duration in that file.

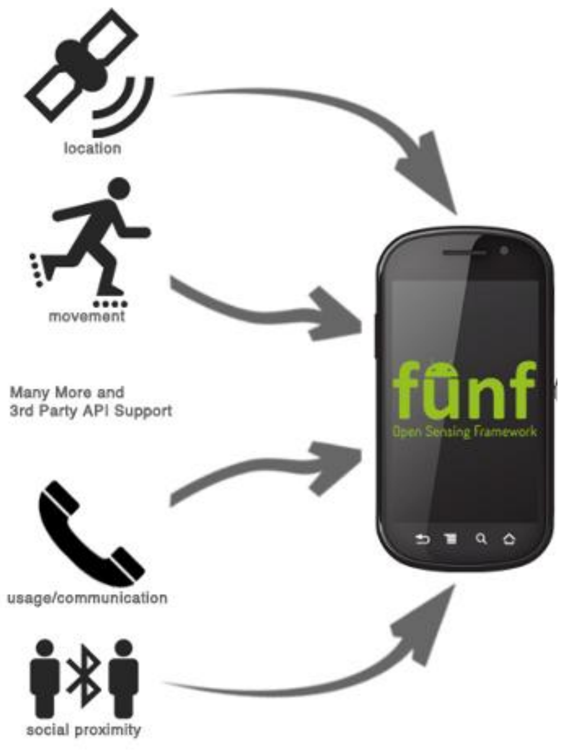
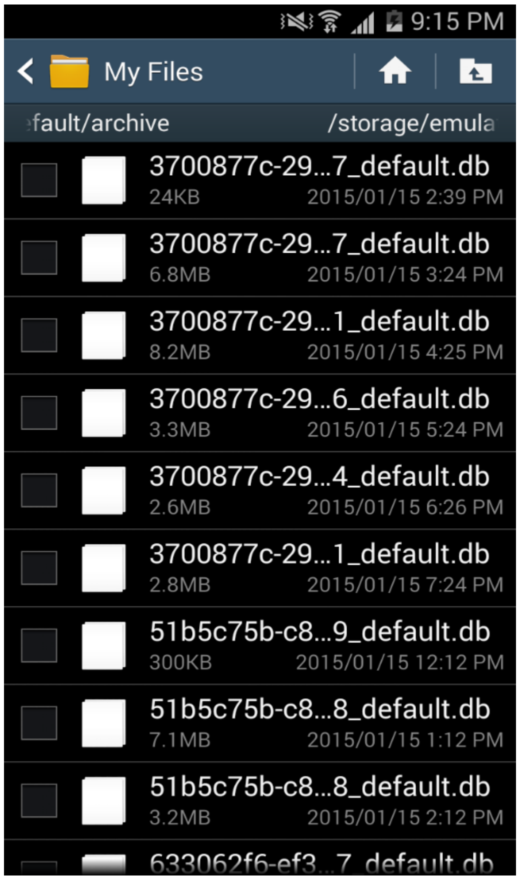


Figure 1. Funf Open Sensing Framework



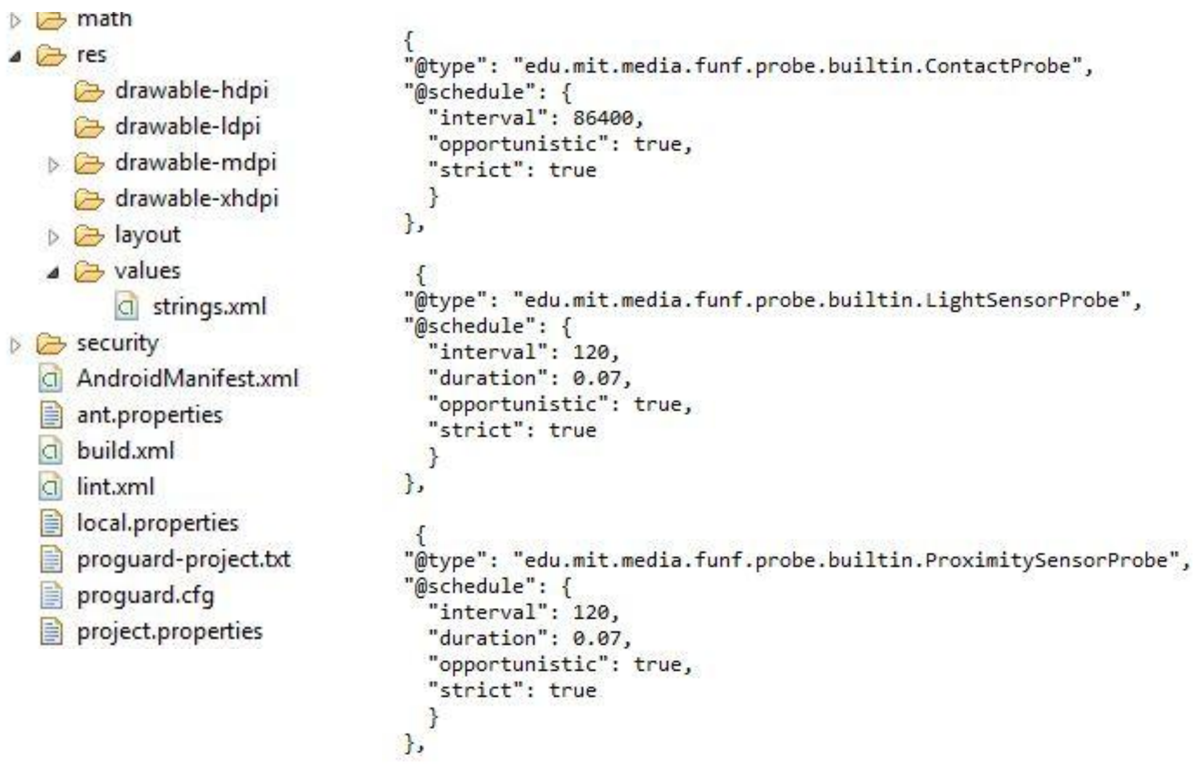
3700877c-29...7_default.db 24KB 2015/01/15 2:39 PM
3700877c-29...7_default.db 6.8MB 2015/01/15 3:24 PM
3700877c-29...1_default.db 8.2MB 2015/01/15 4:25 PM
3700877c-29...6_default.db 3.3MB 2015/01/15 5:24 PM
3700877c-29...4_default.db 2.6MB 2015/01/15 6:26 PM
3700877c-29...1_default.db 2.8MB 2015/01/15 7:24 PM
51b5c75b-c8...9_default.db 300KB 2015/01/15 12:12 PM
51b5c75b-c8...8_default.db 7.1MB 2015/01/15 1:12 PM
51b5c75b-c8...8_default.db 3.2MB 2015/01/15 2:12 PM
633062f6-ef3...7_default.db

Figure 3. Personal data in user's smartphone

To make easy for remembering, we classify the data to three of data categorization, are:

1. On Request Data (Current Data)
2. Historical Data (Saved in Android db)
3. Continuous Data (Sensor data)

On request data means we try to ask current values from android system such as location, battery, nearby Bluetooth and etc. Historical data means the data that already store in android database so we try to access and collect it, the example of historical data are contact, call log, sms log, and etc. Continuous data means we can get those data continuously such as sensor data (accelerometer, gyroscope, magnetic field, and etc). Another important thing is because we are living in time dimension space so every data has timestamp. Funf already has features to collect time, Funf using UNIX UTC (Coordinated Universal Time) which is (Unix time or POSIX time or Unix timestamp) is the number of seconds that have elapsed since January 1, 1970. To convert UNIX time to the human readable time, we can use POSIX function in R or another programming language.



```
▷ math
◢ res
    drawable-hdpi
    drawable-ldpi
  ▷ drawable-mdpi
    drawable-xhdpi
  ▷ layout
  ◢ values
      strings.xml
▷ security
  AndroidManifest.xml
  ant.properties
  build.xml
  lint.xml
  local.properties
  proguard-project.txt
  proguard.cfg
  project.properties
```

```
{
"@type": "edu.mit.media.funf.probe.builtin.ContactProbe",
"@schedule": {
  "interval": 86400,
  "opportunistic": true,
  "strict": true
  }
},

 {
"@type": "edu.mit.media.funf.probe.builtin.LightSensorProbe",
"@schedule": {
  "interval": 120,
  "duration": 0.07,
  "opportunistic": true,
  "strict": true
  }
},

 {
"@type": "edu.mit.media.funf.probe.builtin.ProximitySensorProbe",
"@schedule": {
  "interval": 120,
  "duration": 0.07,
  "opportunistic": true,
  "strict": true
  }
},
```

(a) (b)

Figure 2. (a) strings.xml file in project directory, (b) inside the string.xml file

Data that we collected using our application will be store in SQLite database format with (*.db) extension, the view of data can be seen in Figure 3. To open those database, we can use SQLite browser that can be download in SQLite browser main site[3].

[3] http://sqlitebrowser.org/

On Request Data

Table 2.shows the table of On Request Data. The table contain four columns, _id is automatically generated by database engine, name means the name of probes (sensors), timestamp column is time when system store the data to the phone's storage, and value is the value that returned from the sensors. On request data has four of probes are location, nearby Wi-Fi, nearby Bluetooth, and battery.

Table 2. On Request Data Table

_id	name	timestamp	value (JSON)
	SimpleLocationProbe	Unix UTC	
	WifiProbe		
	BluetoothProbe		
	BatteryProbe		

Simple Location Probe

Location is one of the most important information from the user. In this research, we try to get the location information from the users. The value that returned by system is like this:

```
{"mAccuracy":1625.0,"mAltitude":0.0,"mBearing":0.0,"mElapsedRealtimeNan
os":21989372000000,"mExtras":{"networkLocationSource":"cached","network
LocationType":"cell","noGPSLocation":{"mAccuracy":1625.0,"mAltitude":0.
0,"mBearing":0.0,"mElapsedRealtimeNanos":21989372000000,"mHasAccuracy":
true,"mHasAltitude":false,"mHasBearing":false,"mHasSpeed":false,"mIsFro
mMockProvider":false,"mLatitude":35.1837595,"mLongitude":126.9052379,"m
Provider":"network","mSpeed":0.0,"mTime":1403484137091},"travelState":"
stationary"},"mHasAccuracy":true,"mHasAltitude":false,"mHasBearing":fal
se,"mHasSpeed":false,"mIsFromMockProvider":false,"mLatitude":35.1837595
,"mLongitude":126.9052379,"mProvider":"network","mSpeed":0.0,"mTime":14
03484137091,"timestamp":1403484137.255}
```

That data which from location probes representing a geographic location. A location can consist of a latitude, longitude, timestamp, and other information such as bearing, altitude and velocity. All locations generated by the *LocationManager* are guaranteed to have a valid latitude, longitude, and timestamp (both UTC time and elapsed real-time since boot) and all other parameters are optional. In general, usually we only use latitude and longitude to define the human location, but in this data we have many of data, another data such as accuracy, bearing, altitude, and elapse real time are explained below.

Accuracy

Get the estimated accuracy of this location, in meters. We define accuracy as the radius of 68% confidence. In other words, if you draw a circle centered at this location's latitude and longitude, and with a radius equal to the accuracy, then there is a 68% probability that the true location is inside the circle.

Altitude

Get the altitude if available, in meters above the WGS 84 (World Geodetic System) reference ellipsoid. If this location does not have an altitude then 0.0 is returned. The coordinate origin of WGS 84 is meant to be located at the Earth's center of mass; the error is believed to be less than 2 cm.

Bearing

Get the bearing, in degrees. Bearing is the horizontal direction of travel of this device, and is not related to the device orientation. If this location does not have a bearing then 0.0 is returned.

Elapsed Real Time

Note that the UTC time on a device is not monotonic: it can jump forwards or backwards unpredictably. So always use *getElapsedRealtimeNanos()* when calculating time deltas. On the other hand, *getTime()* is useful for presenting a human readable time to the user, or for carefully comparing location fixes across reboot or across devices.

More details about the key and values from the location probes can be seen in Android API documentation through this link.

http://developer.android.com/reference/android/location/Location.html#

Nearby Wi-Fi Probe

One of important information about the user is nearby Wi-Fi. This probes will collect all of Wi-Fi information which is near with user. This data also can be collected by using our application data collector. Returned value from the system is looks like:

```
{"BSSID":"b0:c7:45:7d:0f:7c","SSID":"rischan","capabilities":"[WPA2-
PSK-CCMP+TKIP][ESS]","frequency":5180,"level":-
46,"timestamp":1403476993.05}
```

We have 6 couple of keys and values, BSSID, SSID (Access Point name), capabilities, frequency, level, and timestamp.

Capabilities

Describes the authentication, key management, and encryption schemes supported by the access point.

Frequency

The frequency in MHz of the channel over which the client is communicating with the access point.

Level

The detected signal level in dBm, also known as the RSSI. Use *calculateSignalLevel(int, int)* to convert this number into an absolute signal level which can be displayed to a user.

Nearby Bluetooth Probe

Beside of nearby Wi-Fi, one of important information related with human is nearby Bluetooth. This probes will collect all of Wi-Fi information which is near with user. The value that we get from our application looks like:

```
{android.bluetooth.device.extra.DEVICE":{"mAddress":"74:F0:6D:E8:ED:67"
},"android.bluetooth.device.extra.NAME":"RRI-ITMS
PC","android.bluetooth.device.extra.RSSI":-
79,"timestamp":1404128054.397}
```

We have information about the device, in this case Bluetooth device address, also Bluetooth name, RSSI, and timestamp.

android.bluetooth.device.extra.RSSI

Used as an optional short extra field in ACTION_FOUND intents. Contains the RSSI value of the remote device as reported by the Bluetooth hardware. Constant Value: "android.bluetooth.device.extra.RSSI“. More details about Bluetooth documentation can be seen in Android API documentation through this link

http://developer.android.com/reference/android/bluetooth/BluetoothDevice.html

Battery Probe

Battery Probe will collect battery information from the user’s smartphone such as charging or discharging, health condition, level, and etc. The value that returned by system looks like:

```
{"charge_type":0,"health":2,"icon-small":17303540,
"level":89,"online":1,          "scale":100,"status":3,"technology":"Li-
ion","temperature":305,"timestamp":1403476991.281,"voltage":4138}
```

We have 11 couple of keys and values from those data. The description about the meaning of values of charge_type, health, status, and voltage can be seen below:

Charge Type value meaning

- BATTERY_PLUGED_AC =1
- BATTERY_PLUGGED_USB =2

- BATTERY_PLUGGED_WIRELESS=4

Status value meaning

- BATTERY_STATUS_CHARGING =2
- BATTERY_STATUS_DISCHARGING =3
- BATTERY_STATUS_FULL =5
- BATTERY_STATUS_NOT_CHARGING =4
- BATTERY_STATUS_UNKNOWN =1

Health values meaning

- BATTERY_HEALTH_COLD =7
- BATTERY_HEALTH_DEAD =4
- BATTERY_HEALTH_GOOD =2
- BATTERY_HEALTH_OVERHEAT =3
- BATTERY_HEALTH_OVER_VOLTAGE =5
- BATTERY_HEALTH_UNKNOWN =1
- BATTERY_HEALTH_UNSPECIFIED_FAILURE =5

Voltage

Integer containing the current battery voltage level. Constant Value: "voltage".

More details about Battery documentation can be seen in Android API documentation in this link http://developer.android.com/reference/android/os/BatteryManager.html

Historical Data

Table 3.shows the table of historical data. The table contain four columns, _id is automatically generated by database engine, name means the name of probes (sensors), timestamp column is time when system store the data to the phone’s storage, and value is the value that returned from the sensors. Historical data are the call log data, SMS log, the list of installed application in user’s smartphone, user’s smartphone device (hardware) info, bookmark in smartphone browser, Log search (history) in smartphone browser, and contact in user’s smartphone.

To protect user privacy we use SHA to hash the privacy information such as user name in contact, phone number, name of caller, and etc.

Table 3. Historical Data Table

_id	name	timestamp	value (JSON)
	CallLogProbe	Unix UTC	
	SmsProbe		
	ApplicationsProbe		
	HardwareInfoProbe		
	BrowserBookmarksProbe		
	BrowserSearchesProbe		
	ContactProbe		

Call Log Probe

The data from Call log probes looks like:

```
{"_id":2172,"date":1403874310514,"duration":160,"name":"{\"ONE_WAY_HASH
\":\"d5c7034c3a03ea8ec287f7e8f082d6ec8c07ffb1\"}","number":"{\"ONE_WAY_
HASH\":\"d4f6776ca772a1d8fadb157ef323e906d78d8d9a,"timestamp":140387431
0.514,"type":2}
```

We have 7 couple of JSON keys and values, date is the date when user call (incoming/outgoing) the date in UNIX timestamp format, duration is the duration of user when he/she make call, name and number are hashed, timestamp, and type. The value of type explained below:

Type value meaning

- INCOMING_TYPE = 1
- OUTGOING_TYPE = 2
- MISSED_TYPE = 3

Sms Log Probe

The returned data from user's smartphone of the SMS log probes looks like:

```
{"address":"dad42137da1fcasdsaga54d6c0f0dc8cd1d42e7a3","body":"{\"ONE_W
AY_HASH\":\"c1f3942137da1fca36554d6c0f0dc8cd1d42e7a3\"}","body-byte-
len":90,"body-token-byte-len":"3-52-16-16-","body-token-
count":4,"date":1403316814524,"read":true,"thread_id":215,"timestamp":1
403316814.524,"type":1}
```

Similar with call log, the address, body text are hashed because this information is related to user privacy. Even though we encrypted some of information but we do not lose the pattern of information. If the address (phone number) is same the output of SHA hash also same. In this probe, we also collect the pattern of body token count, based on that data we know the length of message, the number of letters in each words, and etc.

In SMS log probes data we have key “type”, the meaning of type value explained below:

Type value meaning

- MESSAGE_TYPE_ALL =0
- MESSAGE_TYPE_INBOX =1
- MESSAGE_TYPE_SENT =2
- MESSAGE_TYPE_DRAFT =3
- MESSAGE_TYPE_OUTBOX =4
- MESSAGE_TYPE_FAILED =5
- MESSAGE_TYPE_QUEUED =6

If the type message (SMS) is SENT, the data have key “status”, and the meaning of the value in key “status” explained below:

Status value meaning

- STATUS_NONE =-1
- STATUS_COMPLETE =0
- STATUS_PENDING =32
- STATUS_FAILED =64

Installed Application probe

The data from Installed application probes can be seen below, this data is only from one application data:

```
{"dataDir":"/data/data/com.lifevibes.trimapp”,"enabled":true,"enabledSe
tting":0,"icon":2130837526,
"installed":true,"installedTimestamp":null,"isTrusted":0,"nativeLibrary
Dir":"/data/data/com.lifevibes.trimapp/lib","packageName":"com.lifevibe
s.trimapp","processName":"com.lifevibes.trimapp”,"sourceDir":"/system/a
pp/TrimApp_phone_J.apk","targetSdkVersion":17,"taskAffinity":"com.lifev
ibes.trimapp”,"timestamp":1403476969.264,"uid":10142}
```

Installed application probes collect the list of installed applications in user’s smartphone. The data above is an example of one data from one application. Based on that data we can determine the name of application using the name of package and that data also provides the information about the directory that used by application.

Hardware Info Probe

To know user’s smartphone specification, we can use this probes. The data from this probes looks like:

```
{"androidId":"5a0d4221916c50ce","bluetoothMac":"08:D4:2B:2A:05:3D","bra
nd":"samsung","deviceId":"354257050990298","model":"SHV-
E250K","timestamp":1403489373.257,"wifiMac":"08:D4:2B:2A:05:3E"}
```

Based on data from hardware info probes we can know the information about device Bluetooth mac, device brand, phone model, and Wi-Fi mac.

Bookmark and Log Search Probe

This application also provides the bookmark probes and log search probes. Bookmark probes will collect all of bookmark data from user's smartphone browser. Search log probes will collect all of history (search log) from user's smartphone browser. The data from bookmark probes looks like:

{"_id":999,"bookmark":0,"created":0,"date":1403860944022,"timestamp":1403860944.022,"title":"홈 - 내학사행정","url":"http://portal.jnu.ac.kr/Education/Webservice_S/Default.aspx","visits":125}

Based on that data we have information about the date in UNIX timestamp format, the title of bookmark, url, and how many user visit those bookmark.

The data from search log probes looks like:

{"_id":2,"date":1383925223295,"search":"facebook","timestamp":1383925223.295}

We have date in UNIX timestamp, the log search, and also timestamp.

Contact Probe

The data from contact probes can be seen below:

{"contactData":"contact_id":3,"custom_ringtone":"{\"ONE_WAY_HASH\":\"\"}","display_name":"{\"ONE_WAY_HASH\":\"50bf609648d98370521094b6b724d240bd469610\"}","in_visible_group":1,"last_time_contacted":0, photo_id":0,"send_to_voicemail":0,"starred":0,"times_contacted":0,"timestamp":1404296933.626}

Similar with Call and SMS log, the data which related with user privacy were hashed. From the contact data we can know the information about the name of people in contact (hashed), group, last time contacted, and many more. Even though the name of contact was hashed but we still can analyze the pattern. When the user try to contact one of person, if the name is same the output of hash also same, so we still have the pattern data, even we do not know exactly the name of people whom contacted by user.

Continuous Data

Table 4.shows the table of continuous data. The table contain four columns, _id is automatically generated by database engine, name means the name of probes (sensors), timestamp column is time when system store the data to the phone's storage, and value is the value that returned from the sensors.

Table 4. Continuous Data Table

_id	name	timestamp	value (JSON)
	LightSensorProbe	Unix UTC	{"accuracy":1,"lux":**121.0**, "timestamp":**1402725082**.124436}
	ProximitySensorProbe		{"accuracy":0,"distance":**8.0**, "timestamp":**1402725082**.030173}
	TemperatureSensorProbe		
	MagneticFieldSensorProbe		{"accuracy":2,"timestamp":**1402725082**.028829,"x":**-26**.939999,"y":**-8**.5199995,"z":**-24**.06}
	PressureSensorProbe		{"accuracy":0,"pressure":**999**.82, "timestamp":**1402725083**.016377}
	ScreenProbe		{"screenOn":**true**,"timestamp":**1402725416**.351}
	RunningApplicationsProbe		
	ActivityProbe		

Android Sensors Data

The Sensor Probes that we used are light sensor, proximity sensor, temperature sensor, magnetic field sensor, and pressure sensor. Table 4. Shows the table of continuous data and on the last column, we can see the example value from each sensor probes. The data which came from android sensor have accuracy the meaning of accuracy are described below:

Accuracy

Meaning of accuracy values in sensors data, are:

- SENSOR_STATUS_ACCURACY_HIGH means this sensor is reporting data with maximum accuracy, return value = 3.
- SENSOR_STATUS_ACCURACY_LOW means this sensor is reporting data with low accuracy, calibration with the environment is needed, return value = 1
- SENSOR_STATUS_ACCURACY_MEDIUM means this sensor is reporting data with an average level of accuracy, calibration with the environment may improve the readings, return value = 2.
- SENSOR_STATUS_UNRELIABLE means the values returned by this sensor cannot be trusted, calibration is needed or the environment doesn't allow readings, return value = 0.
- SENSOR_STATUS_NO_CONTACT means the values returned by this sensor cannot be trusted because the sensor had no contact with what it was measuring

(for example, the heart rate monitor is not in contact with the user), return value = -1.

The details information about android sensors can be seen in Table 5.

Table 5. Android Sensors Explanation

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_LIGHT	Hardware	Measures the ambient light level (illumination) in lx. values[0]: Ambient light level in SI lux units	Controlling screen brightness.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.
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. values[0]: Proximity sensor distance measured in centimeters	Phone position during a call.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (℃). This sensor implementation varies across devices and this sensor was replaced with theTYPE_AMBIENT_TEMPERATURE sensor in API Level 14. values[0]: ambient (room) temperature in degree Celsius.	Monitoring temperatures.

Running Application Probe

We have installed application probes which can collect the list of installed application in user’s smartphone. To know user interest we also try to collect the current applications which user used or running application.

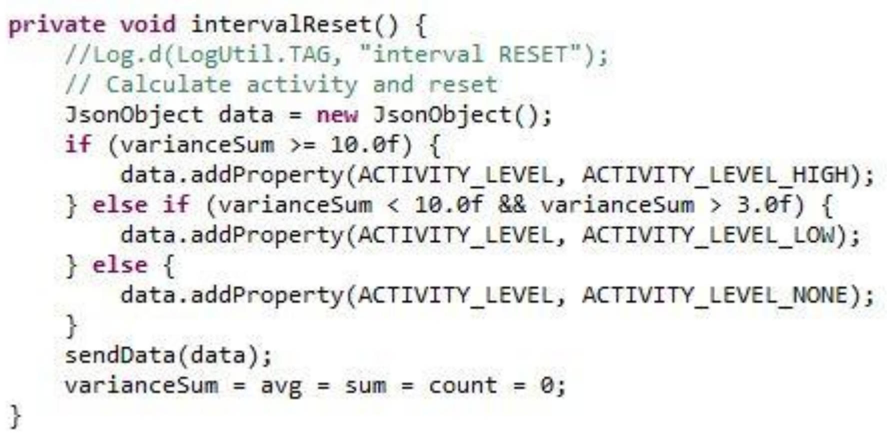
The data from running application probes looks like:

{"duration":6.143,"taskInfo":{"baseIntent":{"mAction":"android.intent.action.MAIN","mCategories":["android.intent.category.LAUNCHER"],"mComponent":{"mClass":"kr.ac.jnu.netsys.MainActivity","mPackage":"edu.mit.media.funf.wifiscanner"},
"mPackage":"edu.mit.media.funf.wifiscanner","mWindowMode":0},"id":30,"persistentId":30},"timestamp":1402725116.144}

We have information about the name of application package which is in current running also the time usage (duration).

Activity Probe

In our application we do not collet the accelerometer value directly but we use algorithm to determine the status of activities which are none, low, or high. We use sum of variance to detect the user activity based on accelerometer value. The details algorithm can be seen in Figure 4.



```
private void intervalReset() {
    //Log.d(LogUtil.TAG, "interval RESET");
    // Calculate activity and reset
    JsonObject data = new JsonObject();
    if (varianceSum >= 10.0f) {
        data.addProperty(ACTIVITY_LEVEL, ACTIVITY_LEVEL_HIGH);
    } else if (varianceSum < 10.0f && varianceSum > 3.0f) {
        data.addProperty(ACTIVITY_LEVEL, ACTIVITY_LEVEL_LOW);
    } else {
        data.addProperty(ACTIVITY_LEVEL, ACTIVITY_LEVEL_NONE);
    }
    sendData(data);
    varianceSum = avg = sum = count = 0;
}
```

Figure 4. Activity log algorithm

The value of activity probes that we get from the application looks like:

{"activityLevel":"none","timestamp":1402725083.715}

Data Summarization

We have finished to collect user personal data. The all of data stored in Rischan PC using path C:\ITRC_DATA. The data has been archived in zip format with name “itrc 2014 userdata finalpoint 20140903.zip” with size 4.25 GB. The extracted data can be accessed in path C:\ITRC_DATA\itrc 2014 userdata finalpoint 20140903\. The size of all of data after extracted is 28.7 GB. Extracted data contain 47 directories in different name for each student data. We have tied to looking information about those data such as the size of data from each student and starting point also ending point of data recording. The result of data summarization which contain with name of directories, size, starting point, and ending point can be seen in Table 6.

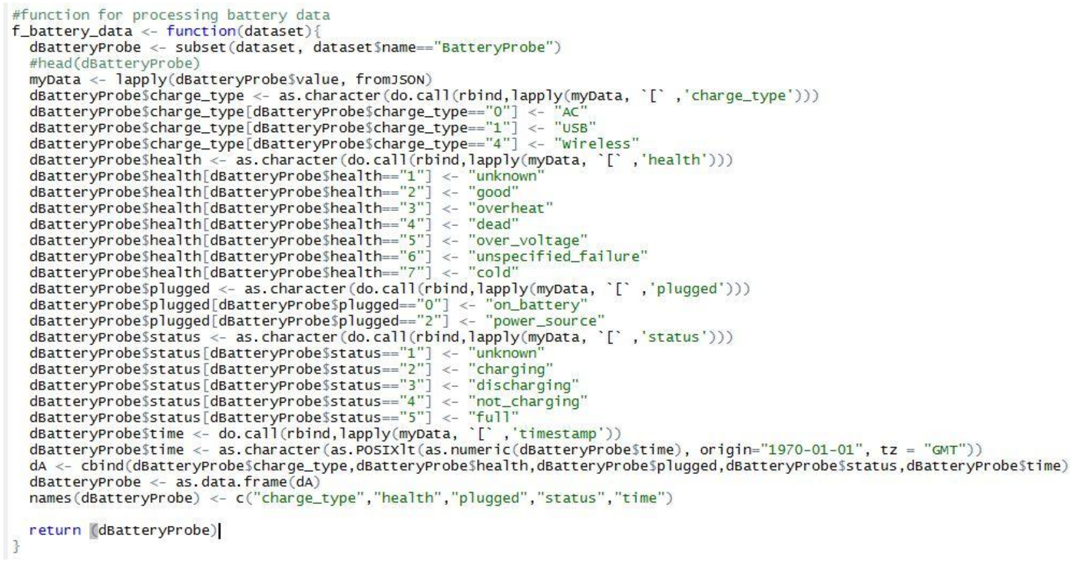
Table 6. Data Summarization from 47 students.

No.	Data ID	Size (MB)	Starting Point	Ending Point
1.	ENFP_0719	628	6/30/2014 8:26	8/20/2014 0:18
2.	ENFP_0773	664	6/26/2014 12:34	8/18/2014 4:58
3.	ENFP_2012	661	6/27/2014 6:11	9/2/2014 3:57
4.	ENTJ_5868	6890	6/27/2014 5:31	8/13/2014 0:00
5.	ENTJ_6454	121	6/26/2014 5:32	8/6/2014 18:53
6.	ENTJ_6966	272	7/2/2014 7:24	8/19/2014 11:22
7.	ENTP_5623	455	6/30/2014 4:49	8/19/2014 20:57
8.	ESFJ_2301	145	6/27/2014 5:31	8/20/2014 2:58
9.	ESFJ_9284	158	6/26/2014 12:34	8/18/2014 4:58
10.	ESFP_0912	278	6/26/2014 5:28	8/18/2014 8:53
11.	ESFP_3295	-		
12.	ESFP_4634	486	6/27/2014 5:25	8/20/2014 4:10
13.	ESFP_7467	607	6/26/2014 5:27	8/19/2014 7:18
14.	ESTJ_0371	2390	7/3/2014 16:21	8/16/2014 21:03
15.	ESTJ_3022	183	6/26/2014 5:28	8/18/2014 23:22
16.	ESTJ_5071	1920	7/2/2014 2:34	9/11/2014 1:49
17.	ESTJ_5190	258	7/30/2014 6:04	8/24/2014 1:43
18.	ESTJ_5824	173	6/26/2014 5:29	8/18/2014 3:51
19.	ESTJ_6510	756	6/27/2014 5:30	8/20/2014 8:09
20.	ESTP_4301	232	6/26/2014 5:29	8/20/2014 4:39
21.	ESTP_5154	990	6/27/2014 5:31	8/13/2014 0:00
22.	INFP_1993	432	6/26/2014 5:31	8/20/2014 0:31
23.	INTJ_5498	342	6/26/2014 5:28	8/20/2014 2:49
24.	INTJ_7906	312	6/14/2014 11:00	8/16/2014 23:01
25.	INTP_3739	1030	6/27/2014 5:28	8/18/2014 5:58
26.	INTP_6399	199	6/26/2014 5:29	8/12/2014 8:32
27.	INTP_9712	180	6/26/2014 5:37	8/16/2014 18:05
28.	ISFJ_2057	183	6/27/2014 5:32	8/14/2014 23:19
29.	ISFJ_2711	767	7/31/2014 0:51	8/20/2014 6:59
30.	ISFJ_7328	133	6/30/2014 7:09	8/19/2014 23:37
31.	ISFP_4030	2380	6/27/2014 6:11	9/2/2014 3:57
32.	ISFP_4282	613	6/27/2014 5:27	8/20/2014 2:46
33.	ISTJ_0178	158	6/26/2014 5:28	8/19/2014 5:05
34.	ISTJ_0386	284	6/26/2014 5:27	8/19/2014 7:18
35.	ISTJ_2068	339	6/26/2014 5:29	8/18/2014 5:30
36.	ISTJ_2837	186	6/27/2014 5:27	8/22/2014 5:41
37.	ISTJ_3052	131	6/27/2014 5:27	8/20/2014 3:41
38.	ISTJ_4659	325	7/2/2014 2:34	9/11/2014 1:49

39.	ISTJ_4667	156	6/26/2014 5:29	8/15/2014 10:44
40.	ISTJ_4700	170	7/3/2014 6:50	8/25/2014 13:08
41.	ISTJ_4753	363	6/26/2014 5:29	8/18/2014 23:48
42.	ISTJ_4968	95	7/3/2014 16:21	8/16/2014 21:03
43.	ISTJ_9139	473	7/3/2014 16:21	8/20/2014 5:57
44.	ISTJ_9576	198	7/4/2014 1:00	8/18/2014 7:12
45.	ISTP_3948	500	6/26/2014 5:29	8/20/2014 1:28
46.	ISTP_7676	365	6/27/2014 5:31	8/19/2014 22:11
47.	XXXX_XXXX	434	6/27/2014 5:31	8/21/2014 6:02

Data Extraction

We have described the condition of our data that we have collected also the meaning of the value of data itself. We can use all of those data, or only one, two or more probes data according to our purpose. The value from those data is still in raw (JSON) format, to extract the most important values or values that we need we have to develop code to do that. We already develop the code for data extraction. The code can be accessed it in https://github.com/rischanlab/Rfunf in Data Extraction directory.



```r
#function for processing battery data
f_battery_data <- function(dataset){
  dBatteryProbe <- subset(dataset, dataset$name=="BatteryProbe")
  #head(dBatteryProbe)
  myData <- lapply(dBatteryProbe$value, fromJSON)
  dBatteryProbe$charge_type <- as.character(do.call(rbind,lapply(myData, `[` ,'charge_type')))
  dBatteryProbe$charge_type[dBatteryProbe$charge_type=="0"] <- "AC"
  dBatteryProbe$charge_type[dBatteryProbe$charge_type=="1"] <- "USB"
  dBatteryProbe$charge_type[dBatteryProbe$charge_type=="4"] <- "Wireless"
  dBatteryProbe$health <- as.character(do.call(rbind,lapply(myData, `[` ,'health')))
  dBatteryProbe$health[dBatteryProbe$health=="1"] <- "unknown"
  dBatteryProbe$health[dBatteryProbe$health=="2"] <- "good"
  dBatteryProbe$health[dBatteryProbe$health=="3"] <- "overheat"
  dBatteryProbe$health[dBatteryProbe$health=="4"] <- "dead"
  dBatteryProbe$health[dBatteryProbe$health=="5"] <- "over_voltage"
  dBatteryProbe$health[dBatteryProbe$health=="6"] <- "unspecified_failure"
  dBatteryProbe$health[dBatteryProbe$health=="7"] <- "cold"
  dBatteryProbe$plugged <- as.character(do.call(rbind,lapply(myData, `[` ,'plugged')))
  dBatteryProbe$plugged[dBatteryProbe$plugged=="0"] <- "on_battery"
  dBatteryProbe$plugged[dBatteryProbe$plugged=="2"] <- "power_source"
  dBatteryProbe$status <- as.character(do.call(rbind,lapply(myData, `[` ,'status')))
  dBatteryProbe$status[dBatteryProbe$status=="1"] <- "unknown"
  dBatteryProbe$status[dBatteryProbe$status=="2"] <- "charging"
  dBatteryProbe$status[dBatteryProbe$status=="3"] <- "discharging"
  dBatteryProbe$status[dBatteryProbe$status=="4"] <- "not_charging"
  dBatteryProbe$status[dBatteryProbe$status=="5"] <- "full"
  dBatteryProbe$time <- do.call(rbind,lapply(myData, `[` ,'timestamp'))
  dBatteryProbe$time <- as.character(as.POSIXlt(as.numeric(dBatteryProbe$time), origin="1970-01-01", tz = "GMT"))
  dA <- cbind(dBatteryProbe$charge_type,dBatteryProbe$health,dBatteryProbe$plugged,dBatteryProbe$status,dBatteryProbe$time)
  dBatteryProbe <- as.data.frame(dA)
  names(dBatteryProbe) <- c("charge_type","health","plugged","status","time")

  return (dBatteryProbe)
}
```

Figure 5. Example of data extraction (Battery Probes)

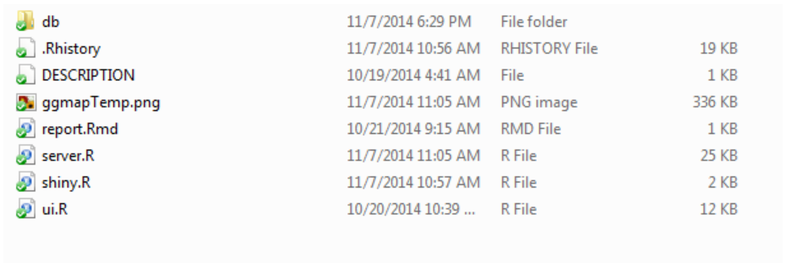
The example of data extraction using R code can be seen in Figure 5. We can see the full code in my github project that I already gave before.

Data Visualization

Visualize the Data in Web Application

After we collected personal user data from user's smartphone, we need to analyze and visualize it. We also develop web application to visualize the data. We use R language to preprocessing, processing, and analysis personal user data.

Traditional tools for data analysis such as matlab, R, SPSS and etc, only support for plotting the result such as in figure with (jpeg, jpg, png) format and pdf. When we need generate document report or maybe want to expose the result in web, we have to copy the result to the web. The problem is when the data changed, we need to plot again and copy again to our document report or to the web. This application solve that problem. We use shiny library from R studio which can support to generate reproducible result for research with beautiful, interactive and responsive web layout. Figure 6. Shows the list of file in shiny project directory.



db	11/7/2014 6:29 PM	File folder	
.Rhistory	11/7/2014 10:56 AM	RHISTORY File	19 KB
DESCRIPTION	10/19/2014 4:41 AM	File	1 KB
ggmapTemp.png	11/7/2014 11:05 AM	PNG image	336 KB
report.Rmd	10/21/2014 9:15 AM	RMD File	1 KB
server.R	11/7/2014 11:05 AM	R File	25 KB
shiny.R	11/7/2014 10:57 AM	R File	2 KB
ui.R	10/20/2014 10:39 ...	R File	12 KB

Figure 6. List of files of this framework

This application contain three main files: ui.R, server.R, and shiny.R.

1. ui.R is file that contain script to manage ui and layout of shiny application. The number of code is 202 lines with size 12 KB.
2. server.R is file that contain main script for loading, preprocessing, processing, and analysis the data, script using R language. Actually, This file contain many of functions, I just put all of functions in one file. The number of code in this file is 547 lines with size 25 KB.
3. shiny.R is config file, such as the user key, and about application configuration. This file only contain 37 lines with size 1 KB.
4. Report.Rmd this is just an example of Rmarkdown report. By using this file we can download file from plotting result (ex: regression plot) in many format such as PDF, HTML, and DOCX. The important thing is the file contain R source code and the result, so users who download that result plot they can see the source code and when they try that code in R environment will get same result. So, usually researcher call it "reproducible sample".

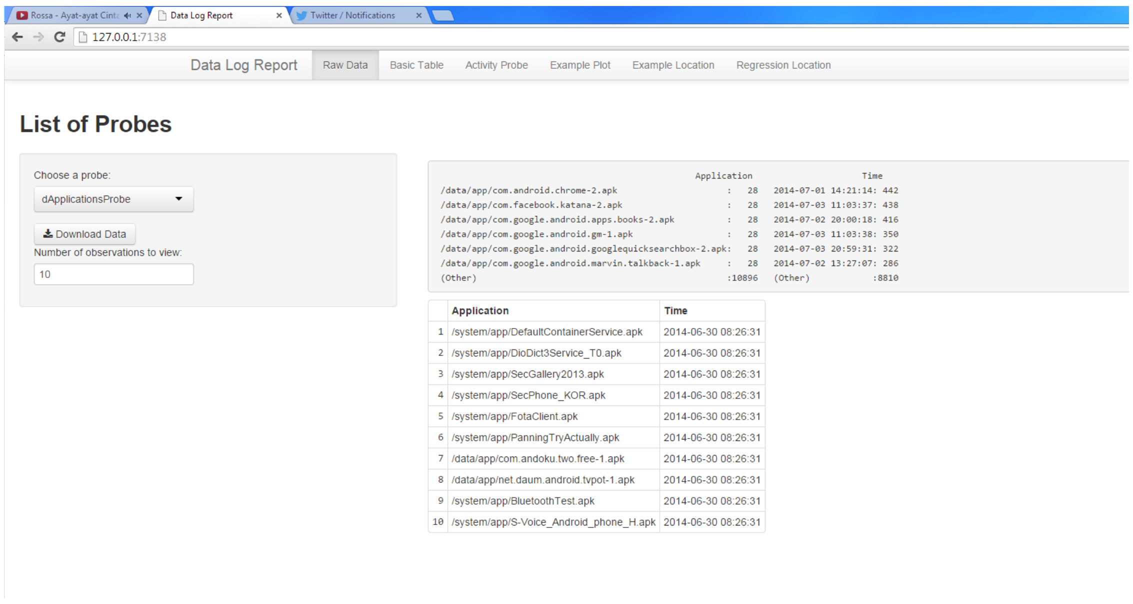
The user data that we collected can be copy to folder '*db*'. To start this application, we need to go to parent directory before '*datalog'* directory. Directory '*datalog'* means the name of this application. This web application can be downloaded in https://github.com/rischanlab/ITRC.

To start this application using this command:

```
runApp("datalog",display.mode = "showcase")
```

Actually the purpose for creating this application is to running in server (Shiny server), but for the example, we also can run this application in desktop environment. See the video demo_example.mov in this directory. To run this application we need some software requirements, on the next page we explain about how to setup environment in Ubuntu 12.04 or later. Example layout with this application can be seen in Figure 7. Data summary, Figure 8. Data summary using responsive table, Figure 9. Example plot of nearby Access Point, and Figure 10. Example plot of regression function that can be download as the reproducible research document.

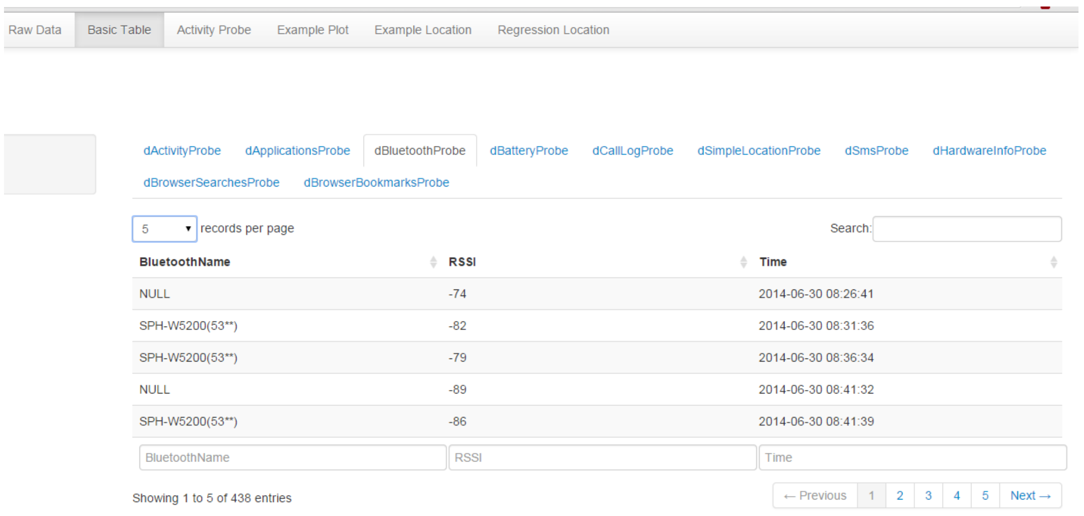
Example Layout of this application.



```
                                          Application                Time
/data/app/com.android.chrome-2.apk                    :   28   2014-07-01 14:21:14: 442
/data/app/com.facebook.katana-2.apk                   :   28   2014-07-03 11:03:37: 438
/data/app/com.google.android.apps.books-2.apk         :   28   2014-07-02 20:00:18: 416
/data/app/com.google.android.gm-1.apk                 :   28   2014-07-03 11:03:38: 350
/data/app/com.google.android.googlequicksearchbox-2.apk:   28   2014-07-03 20:59:31: 322
/data/app/com.google.android.marvin.talkback-1.apk    :   28   2014-07-02 13:27:07: 286
(Other)                                               :10896   (Other)             :8810
```

	Application	Time
1	/system/app/DefaultContainerService.apk	2014-06-30 08:26:31
2	/system/app/DioDict3Service_T0.apk	2014-06-30 08:26:31
3	/system/app/SecGallery2013.apk	2014-06-30 08:26:31
4	/system/app/SecPhone_KOR.apk	2014-06-30 08:26:31
5	/system/app/FotaClient.apk	2014-06-30 08:26:31
6	/system/app/PanningTryActually.apk	2014-06-30 08:26:31
7	/data/app/com.andoku.two.free-1.apk	2014-06-30 08:26:31
8	/data/app/net.daum.android.tvpot-1.apk	2014-06-30 08:26:31
9	/system/app/BluetoothTest.apk	2014-06-30 08:26:31
10	/system/app/S-Voice_Android_phone_H.apk	2014-06-30 08:26:31

Figure 7. Example plot of user data log (Installed Application in User's smartphone)



BluetoothName	RSSI	Time
NULL	-74	2014-06-30 08:26:41
SPH-W5200(53**)	-82	2014-06-30 08:31:36
SPH-W5200(53**)	-79	2014-06-30 08:36:34
NULL	-89	2014-06-30 08:41:32
SPH-W5200(53**)	-86	2014-06-30 08:41:39
BluetoothName	RSSI	Time

Showing 1 to 5 of 438 entries

Figure 8. Different layout to show the data (Nearby Bluetooth data)

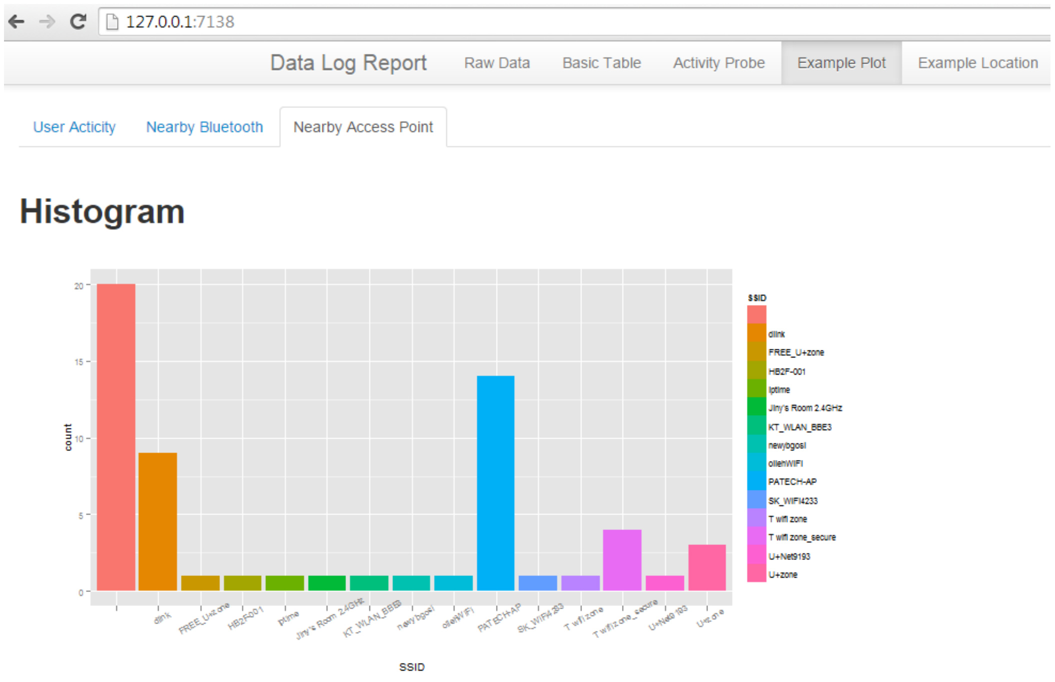


Figure 9. Example plot of Nearby Access Point

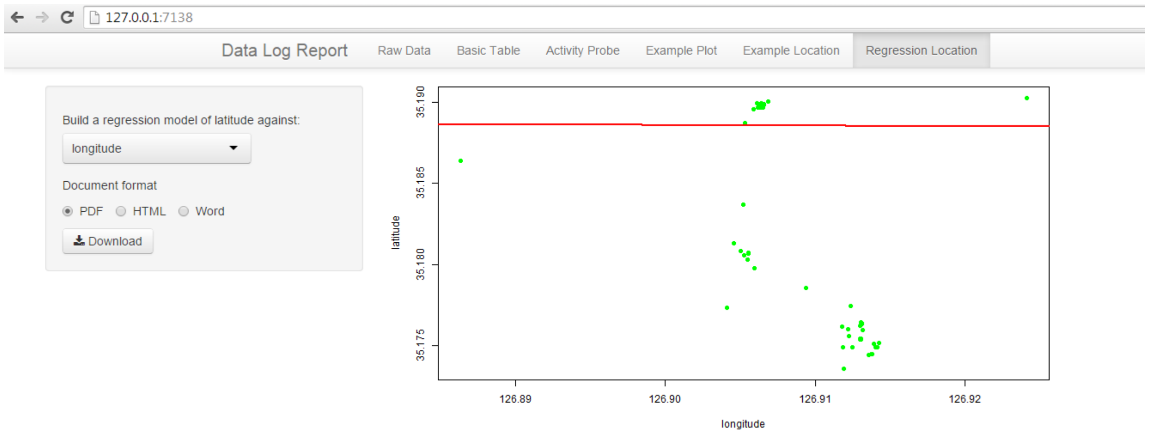


Figure 10. An Example of regression plot that can be download for reproducible research

Setup and Install Environment

Because of this application is for server side purpose, and to run this application, we have to setup and configure the environment. In this page, we explain about how to setup environment that can be used to run our framework.

OS requirement: **Ubuntu 12.04 or later**

Shiny Server, Currently only available for a pre-built binary for the 64-bit architecture. Running on other architectures will require building from source. Before installing Shiny Server, you'll need to install R and the Shiny package. To install the latest version of R you should first add the CRAN repository to your system.

You can then install R using the following command:

```
$ sudo apt-get install r-base
```

NOTE: if you do not add the CRAN Debian in your Ubuntu repository this command will install the version of R corresponding to your current system version. Since this version of R may be a year or two old it is strongly recommended that you add the CRAN repositories so you can run the most up to date version of R. You'll also need to install the Shiny R package before installing Shiny Server:

```
$ sudo su --c "R -e \"install.packages('shiny', repos='http://cran.rstudio.com/')\""
```

Once you've installed R and the Shiny package, execute the following commands in a terminal window to install *gdebi* (which is used to install Shiny Server and all of its dependencies) and Shiny Server.

```
$ sudo apt-get install gdebi-core
```

```
$        wget        http://download3.rstudio.org/ubuntu-12.04/x86_64/shiny-server-1.2.3.368-amd64.deb
```

```
$ sudo gdebi shiny-server-1.2.3.368-amd64.deb
```

Next Steps

Once installed, view the Administrator's Guide to learn how to manage and configure Shiny Server.

R library needed:

1. library(shinyapps)
2. library(shiny)
3. library(shiny)
4. library("RSQLite")
5. library("rjson")
6. library("ggplot2")
7. library("scales")
8. library("ggmap")
9. library("rmarkdown")
10. library("rmarkdown")

After the environment is ready, enter to R console and then run this command:

```
runApp("datalog",display.mode = "showcase")
```

Limitations

Our application still need many of improvements, we have some of limitations in this application as follows:

1. This application does not have function that can send automatically the data from user to the server. Current version still need manually way which is using USB to copy the data from user's phone to the PC server.
2. This application will produces many of duplication data for the historical data. Historical data means the data already store in Android db. This application will collect historical data every one day, so the data will have many of duplication.
3. For the sensor data, using this application, we cannot set the sampling rate of the sensors.