Android SDK - Reference Manual
Version 1.2.2

Introduction

This document will guide you through the integration process of the InLocus SDK.

The InLocus SDK includes several groups of services:

1. Mobility Status User State Changes
    a. Activity in which the user is engaged (i.e. walking, running, driving etc.)
    b. Type of location in which the user is situated (i.e. restaurant, home, mall etc.)
2. Locations current and history
3. Geo Fence Functions
4. Proximity Notifications

The rest of the document lists the required integration steps for each of the above groups of services and any additional maintenance information.

While the InLocus SDK document is built to be self-explanatory, we recommend that you consider a session with the InLocus team.

What's InLocus SDK for Android

InLocus SDK for Android is a Java library that is compatible with Android 4.3 or higher and with smartphones that support Bluetooth Low Energy (BLE). The SDK reports events such as the user's current location, as well as various user actions.The framework sends an event that must be managed by the developer side code for handling these user actions.

InLocus SDK includes the following features

1. **Mobility Status and User State Changes**
    a. Activity in which the user is engaged (i.e. walking, running, driving etc.)
    b. Type of location in which the user is situated (i.e. restaurant, home, mall etc.)
2. **Locations**
    a. User location with longitude and latitude
3. **Geo-Fence Functions**
    a. User Entering and exiting specified geofence.
4. **Proximity**
    a.Get notified when people enter inside a designated area and handling the notification(s).

InLocus SDK Dependencies

In Android Studio there are at least 2 different gradle scripts. The Top Level (or Project script) and the Low Level (or App) script. Our Top Level (or project level) build.gradle script looks like this:

// Top level build file where you can add configuration options common to all sub-projects/modules.

```
android {
    compileSdkVersion 26
    defaultConfig {
        minSdkVersion 21
        targetSdkVersion 26
        versionCode 1
        versionName "1.2.2"
testInstrumentationRunner
"android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled true
proguardFiles     getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support.constraint:constraint-
layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation
'com.android.support.test:runner:1.0.2'
    androidTestImplementation
'com.android.support.test.espresso:espresso-core:3.0.2'
    implementation 'com.google.android.gms:play-services-
location:11.8.0'
implementation 'com.squareup.okhttp3:okhttp:3.10.0'
}
```


Note: Associative libraries are added in the SDK Folder.

Android Manifest Permissions (Android API <= 22)

In this manual we are using Android API version > 18 because API Level 18 was the first API release that supported natively Bluetooth Low Energy scan. Our Target API is the API level (API 26) Android 8 Oreo.
In the following part of the manual we will see how to handle natively the permissions under Android 6 Marshmallow since a lot has been changed since Lollipop (API 21 and 22). This part is necessary for being compatible with (18 >= "Android API Level" <= 22) ;)

```
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"></uses-
permission>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission
android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission
android:name="android.permission.INTERNET"></uses-permission>
<uses-permission
android:name="android.permission.BLUETOOTH_ADMIN"></uses-
permission>
<uses-permission
android:name="android.permission.READ_PHONE_STATE" />
<uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED"></uses-
permission>
<uses-permission
android:name="android.permission.BLUETOOTH"></uses-permission>
<uses-permission
android:name="com.google.android.gms.permission.ACTIVITY_RECOGNIT
ION"></uses-permission>
<uses-permission
android:name="android.permission.WAKE_LOCK"></uses-permission>
<uses-permission
android:name="android.permission.CHANGE_WIFI_STATE"></uses-
permission>
```

Android Marshmallow 6.0 Permissions

As explained in the previous step Android 6 Marshmallow has integrated a new permission framework. Apps designed for Android 6.0 will now ask for a permission when they need it. In Beaconstalk, one step before the activation of the Beaconstalk Service we need to request for permissions of GPS, Account (to handle user logs and crashes) and access to external storage.

For doing it, we created two methods the first is called checkPermissions and the seconds overrides the onRequestPermissionsResult method. Here is how they looks like. Feel free to change it anyway anytime.

```
    if
        (ContextCompat.checkSelfPermission(ProxAwa
        reActivity.this,
        android.Manifest.permission.ACCESS_FINE_LO
        CATION) !=
        PackageManager.PERMISSION_GRANTED) {
       //  Request the needed permissions
       ActivityCompat.requestPermissions(ProxAwareActivi
       ty.this, new
       String[]{android.Manifest.permission.ACCESS_FINE_
       LOCATION,
       Manifest.permission.WRITE_EXTERNAL_STORAGE},
       PERMISSION_REQUEST_CODE);
}
else { //permissions granted
      Initialize_and_start_InLocusSDK();
       // this is a function that runs the service according
}


//Activity should override this function. Android will call this
function after user granted/denied permissions
@Override
public void onRequestPermissionsResult(int requestCode, String
permissions[], int[]
grantResults) {
switch (requestCode) {
      case PERMISSION_REQUEST_CODE:
      if (grantResults.length > 0 &&
      grantResults[0] ==
      PackageManager.PERMISSION_GRANTED) {
      Initialize_and_start_InLocusSDK();

            // this is a function that runs the service
            according to the instructions above
      }
      break;
}
```

Features

The user state mechanism will help you identify the user's current activity and location. The API is divided into two categories:

Mobility: contains various user activity types, such as running, walking, driving, etc.

Location: contains various location types, such as office, home, mall, hospital, airport, etc.

Each user state (Running, Mall …) contains different values, depending on the state type and the confidence level (0 - 1) indicating the likelihood of the current user state.

The use state can be in Enter or Exit status, depending on your configuration. For example, you can decide to be notified that the user enters a running state when the confidence level is above 70% and exits when the confidence level is below 30%. (Note that Exit will only occur if the state was first Entered.)

Registering to Events

InLocus Service sends events in broadcast using the BroadcastReceiver. Few events (Mobility and proximity notifications) are sent also using an event listener but sometimes happens that the event listener never wake up after the smartphone was in sleep mode. However this behaviour has been not detected on all smartphones and all versions of Android.

For this reason we suggest you to use BroadcastReceiver to receive events.

Proximity Events(Beacon+WiFi+GPS)

Event occurs when a user Enters or Exits proximity of geofence or Beacon or WiFi signal.

```
public class ProximityReceiver extends BroadcastReceiver{

@Override

public void onReceive(Context context, Intent intent) {

            String results=intent.getStringExtra("notification");
            //result Data fields needs to be defined.

      }

}
```

results *will be a JSON with following fields:-*

```
{
"notification_type": "TEXT",
"notification_title": "Notification Title",
”content”:{
```



```
        "notification_text": "Notification Text body",
        "image_url": "",
        "uri": "https://inlocus.in"
            }
}
```

Here is **the** AndroidManifest.xml code:

```
<receiver android:name=". ProximityReceiver " >

<intentfilter>

<action android:name="com.proximity.inlocus.notification" />
</intentfilter>
</receiver>
```

Sharing information with InLocus

The application can share information with InLocus service by adding the following Code to your main activity before starting the InLocus service. If your application gets mobile number of user use following line of code for sharing.

```
SharedPreferences
preferences=this.getSharedPreferences("SHAREDLOCUSPREFERE
", Context.MODE_PRIVATE);
SharedPreferences.Editor edit=preferences.edit();
edit.putString("mobile","add user mobile number
in string");
```

After sharing the information you can confirm it by using following code.

```
edit.commit();
```


Initialization and Stoppage

Initialization of InLocus SDK

In this part, we will integrate all the necessary code to initialize InLocus SDK.

Here is the code:

```
public class MainActivity extends Activity implements InitializeCallback
{

   private InLocusSDK inLocus;

   @Override
   protected void onCreate(@Nullable Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main_activity);
       inLocus = new InLocusSDK();
       inLocus.initializeInLocus(MainActivity.this, "API_KEY", this);
```

NOTE: Please create an application on the dashboard to obtain the API_KEY.

Setup ScanSettings

```
BleScanning:Beacon Scanning enabled(boolean).
WiFiScanning:WiFi Scanning enabled(boolean).
GPSScanning:Location Scanning enabled(boolean).
ScanInterval:Interval between successive scans in minutes(long).
Notification:Getting proximity notifications from SDK(boolean).

ScanSettings scanSetting=new
ScanSettings(BleScanning,WiFiScanning,Location,ScanInterval(mins)(
long),Notification);
```

Adding important Dependencies

Please add following dependencies to your project.

```
implementation 'com.google.android.gms:play-services-location:11.8.0'
implementation 'com.squareup.okhttp3:okhttp:3.10.0'
```


Starting of InLocus Service

```
@Override
public void oninitSuccess(String result) {

   ScanSettings scanSetting=new ScanSettings(true,true,true,5l,true);
   boolean started=inLocus.startTheServices(scanSetting,context);

}

@Override
public void oninitFailed(String result) {

   Log.e(TAG, "oninitFailed: FAILED" + result);
}
```

Stopping of InLocus Service

In this part, we will integrate all the necessary code to stop the InLocus service.
Here is the code:

```
inLocus.stopTheServices(new
ScanSettings(false,false,false,5l,false),this);
```

Important Information About User States

- Exclusivity – Some of the user states are not mutually exclusive. A user can be in an office that also resides in a mall, and thus be in two locations at the same time.
- Training – To provide reliable results, detection of home and office locations require several days of initiation within the user device. Please take this into account when using these states in your app.
- Confidence level – This parameter describes how confident the service is that the user is actually in a specific state. The values are between 0 (not in this state) and 1 (100% sure the user is in this state). It is possible to request that updates for a specific state be sent from the service, starting from a specific confidence level. For example, you can request notification of an Asleep state only if the confidence level is 0.75 or above. In this case, notification will be sent every time the confidence level reaches 0.75.

