RECO SDK for Android ver.English Documentation

Release 0.1.5

2014, Perples Inc.

December 20, 2014

This document describes how to use the RECO SDK of the Perples Inc. to develop android application.

RECO SDK includes the followings:

1. RECO SDK libraries

```
reco-sdk-android_[VERSION_NUMBER].jar
reco-sdk-android_[VERSION_NUMBER]_JavaDoc.jar (optional)
```

2. Sample Code

```
reco-client-android-sample
```

Note: This document is based on the Eclipse Luna Service Release 1.

CHAPTER

ONE

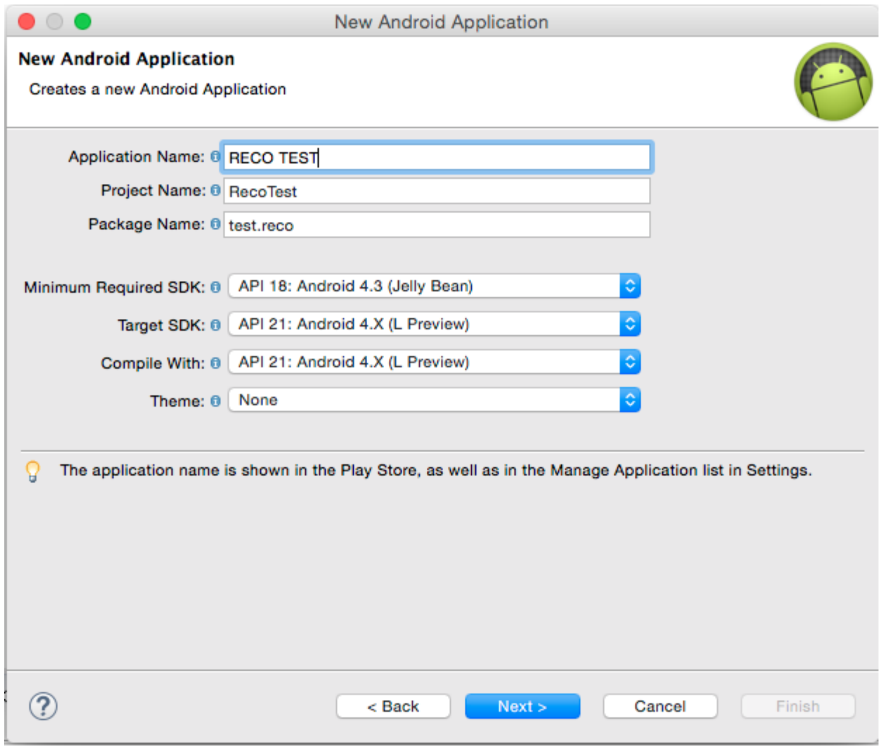
PRE-REQUISITES

The followings are prerequisites to use RECO SDK for Android.

1.1 Android API Version

RECO SDK supports the following API version or higher. Target SDK version or Complie SDK version can be any version higher than 18.

- Minimum Required SDK: API 18: Android 4.3 (Jelly Bean)
- Target SDK: API 18: Android 4.3 (Jelly Bean)
- Complie With: API 18: Android 4.3 (Jelly Bean)



New Android Application

Creates a new Android Application

Application Name: RECO TEST
Project Name: RecoTest
Package Name: test.reco

Minimum Required SDK: API 18: Android 4.3 (Jelly Bean)
Target SDK: API 21: Android 4.X (L Preview)
Compile With: API 21: Android 4.X (L Preview)
Theme: None

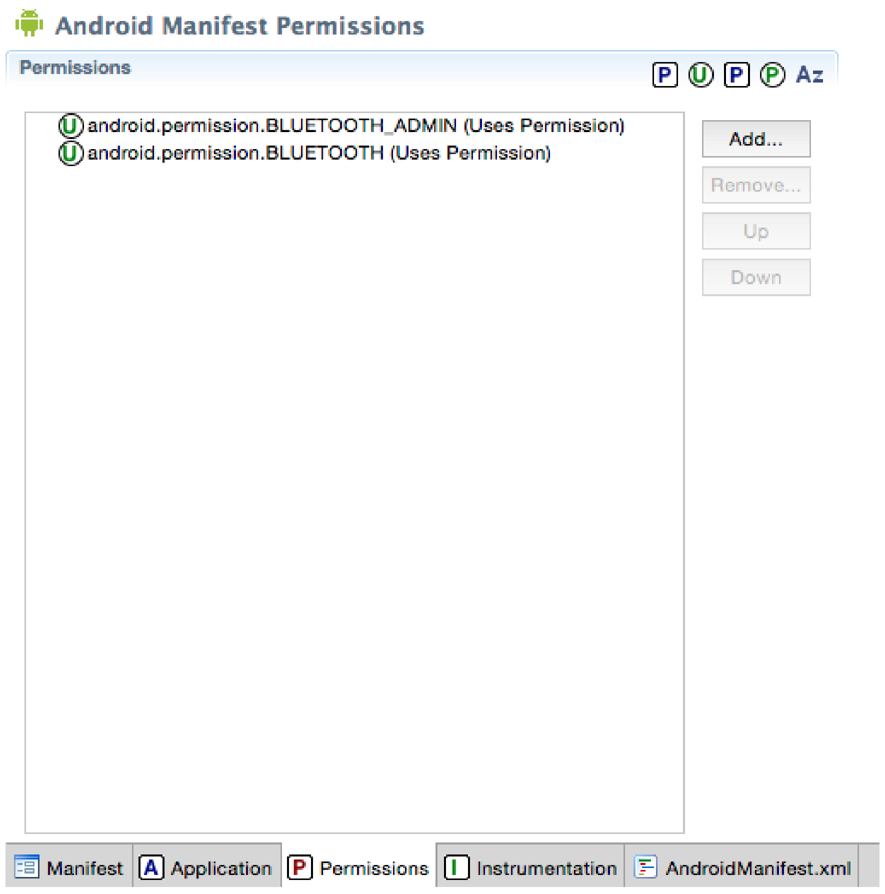
The application name is shown in the Play Store, as well as in the Manage Application list in Settings.

1.2 AndroidManifest.xml permission

RECO SDK requires the permission of **BLUETOOTH_ADMIN** and **BLUETOOTH**.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Note: BLUETOOTH_ADMIN is required to allow the application to discover and pair bluetooth devices. BLUETOOTH is required to use BLUETOOTH_ADMIN.



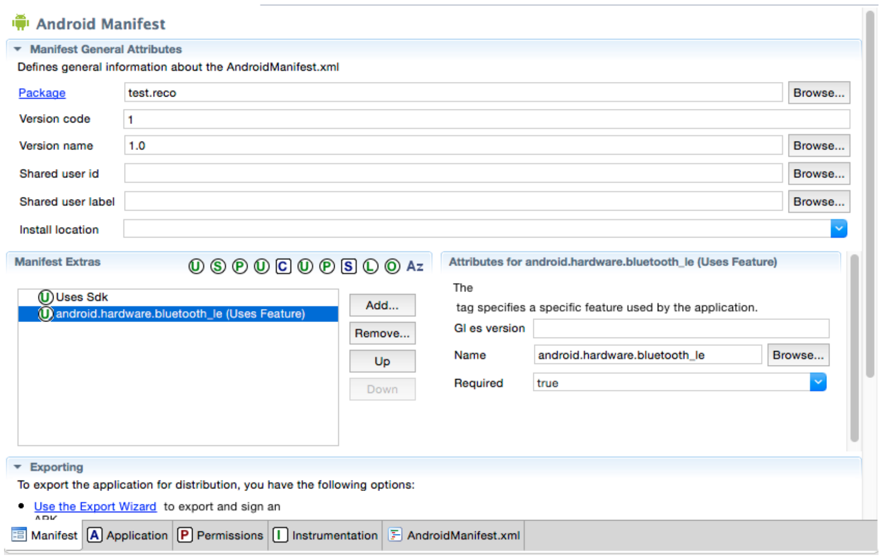
android.permission.BLUETOOTH_ADMIN (Uses Permission)
android.permission.BLUETOOTH (Uses Permission)

In addition, you should **include RECOBeaconService inside <application> </application>** in order to start region monitoring and region ranging.

```
<service android:name="com.perples.recosdk.RECOBeaconService" />
```

In order to declare that your application is available to BLE-capable devices only, include the following.

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



Android Manifest

Manifest General Attributes

Defines general information about the AndroidManifest.xml

Package: test.reco

Version code: 1

Version name: 1.0

Shared user id:

Shared user label:

Install location:

Manifest Extras

Uses Sdk

android.hardware.bluetooth_le (Uses Feature)

Attributes for android.hardware.bluetooth_le (Uses Feature)

The
tag specifies a specific feature used by the application.

Gl es version:

Name: android.hardware.bluetooth_le

Required: true

Exporting

To export the application for distribution, you have the following options:

- Use the Export Wizard to export and sign an APK

Note: If you want to make your application available to devices that do not support BLE, you should still include this element in AndroidManifest.xml, but set required=”false”. You can determine BLE availability by using the following method in RECO SDK at run-time.

- When monitoring regions:

```
RECOBeaconManager recoManager = RECOBeaconManager.getInstance(context);
if(recoManager.isMonitoringAvailable()) {
    //You can start region monitoring if the device supports BLE.
}
```

- When ranging regions:

```
if(recoManager.isRangingAvailable()) {
    //You can start region ranging if the device supports BLE.
}
```

CHAPTER TWO

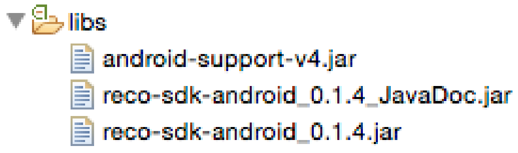
GETTING STARTED

2.1 Configuration for the SDK

Note: This document is based on the Eclipse Luna Service Release 1.

1. Copy the following files in the RECO SDK Package to "libs" directory in your project.

```
reco-sdk-android_[VERSION_NUMBER].jar
reco-sdk-android_[VERSION_NUMBER]_JavaDoc.jar (Optional)
```



```
libs
    android-support-v4.jar
    reco-sdk-android_0.1.4_JavaDoc.jar
    reco-sdk-android_0.1.4.jar
```

2. You neet to include the following permissions to AndroidManifest.xml. For more detailed information, please refer to the previous chapter, "AndroidManifest.xml permission".

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true" />
```

3. You should include RECOBeaconService inside <application> </application>. For more detailed information, please refer to the previous chapter, "AndroidManifest.xml permission".

```
<service android:name="com.perples.recosdk.RECOBeaconService" />
```

2.2 Defining RECOBeacon, RECOBeaconRegion

A beacon region is an area defined by the device's proximity to BLE (Bluetooth Low Energy) beacons. A region can be defined by the bluetooth signal from a BLE beacon.

RECOBeaconRegion is identified by a combination of the following values.

- **Proximity UUID(universally unique identifier)** : 128-bit value that uniquely identifies one or more beacons as a certain type or from a certain organization.
- **Major** : 16-bit unsigned integer that can be used to group related beacons that have the same proximity UUID.
- **Minor** : 16-bit unsigned integer that differentiates beacons with the same proximity UUID and major value.

For example, if a RECOBeaconRegion is defined by proximity UUID only, the region is defined to the extent of any regions receiving signals containing the proximity UUID.

RECOBeaconRegion uses an unique identifier to differentiate regions within your application, when an instance of RECOBeaconRegion is generated. An unique identifier must not be null, **the old region is replaced by the new one if an existing region with the same identifier already exists in your application.**

Furthermore, if you do not put proximity UUID, it is set to the proximity UUID of RECOBeacon provided by Perples Inc.

RECOBeacon is a device that broadcasts the BLE(Bluetooth Low Energy) signal containing data which is complying with the iBeacon standard. It represents beacons detected by the device while ranging regions.

You do not create instances of the RECOBeacon directly. The only way to get the instances of RECOBeacon in our SDK is to range beacons in a certain region. After receiving beacons in the region using callback method, you can get the information in RECOBeacon.

2.3 Connecting to RECOBeaconService for Region Monitoring and Region Ranging

Note: RECO SDK for Android performs monitoring regions and ranging regions using the instance of the RECOBeaconManager. The instance of the RECOBeaconManager must be connected to the RECOBeaconService before starting to monitor and range regions. When the connection between the RECOBeaconManager and RECOBeaconService is established, RECOServiceConnectListener interface is required.

1. Implement RECOServiceConnectListener and its callback method.

```
import com.perples.recosdk.RECOBeaconManager;
import com.perples.recosdk.RECOServiceConnectListener;

public class YourClassName implements RECOServiceConnectListener {

    ..
    ..
    ..

    public void onServiceConnect() {
        //Write the code when the connection to RECOBeaconService is established
    }
}
```

2. Create an instance of the RECOBeaconManager.

```
//RECOBeaconManager.getInstance(Context, boolean) gets context and
//setting for ranging timeout while monitoring in the background as parameters.
RECOBeaconManager recoManager = RECOBeaconManager.getInstance(this, true);
```

Note: As of release 0.1.5, RECOBeaconManager.getInstance(Context) method is replaced by RECOBeaconManager.getInstance(Context, boolean) method. If you create an instance of the RECOBeaconManager with RECOBeaconManager.getInstance(Context), the default setting for ranging timeout while monitoring in the background is true. For more detailed information, please refer to the "Notes on Development".

3. Set RECOServiceConnectListener interface, and connect the instance of the RECOBeaconManager to the RECOBeaconService.

```
recoManager.bind(this);
```

Note: onServiceConnect() callback method in the RECOServiceConnectListener is called when the connection to the RECOBeaconService is established after calling this method. When the connection is already established, the callback method will not be called. You can check the status of the connection using isBound() method in the RECOBeaconManager before calling bind() method.

```
if(recoManager.isBound()) {
    //Write the code when the connection to the RECOBeaconService is already established
} else {
    recoManager.bind(this);
}
```

4. Call unbind() method when you want to disconnect to the RECOBeaconService.

```
recoManager.unbind();
```

For more detailed information, please refer to the sample project, especially RECOActivity.java, RECOMonitoringActivity.java, and RECORangingActivity.java, in the RECO SDK Package.

2.4 Monitoring RECOBeaconRegion

Note: RECO SDK for Android perform monitoring regions using the instance of the RECOBeaconManager. Before monitoring regions, the instance of the RECOBeaconManager must be connected to the RECOBeaconService, and RECOMonitoringListener interface is required.

1. Implement RECOMonitoringListener and its callback methods.

```
import com.perples.recosdk.RECOBeaconRegion;
import com.perples.recosdk.RECOBeaconRegionState;
import com.perples.recosdk.RECOMonitoringListener;

public class YourClassName implements RECOMonitoringListener {

    ..
    ..
    ..

    public void didDetermineStateForRegion(RECOBeaconRegionState recoRegionState,
                                           RECOBeaconRegion recoRegion) {
        //After starting monitoring a region, this method is called
        //      when the state of the monitored region is changed
        //          such as entry or exit to the region.
        //This method is also called
        //      when didEnterRegion(RECOBeaconRegion) or didExitRegion(RECOBeaconRegion)
        //          callback method is called.
        //Write the code when the state of the monitored region is changed
    }


    public void didEnterRegion(RECOBeaconRegion recoRegion) {
        //When the device is entered the monitored region, this method is called.
        //Write the code when the device is entered the region
    }

    public void didExitRegion(RECOBeaconRegion recoRegion) {
        //When the device is exited the monitored region, this method is called.
        //Write the code when the device is exited the region
    }

    public void didStartMonitoringForRegion(RECOBeaconRegion recoRegion) {
        //This method is called right after starting monitoring a region
        //Write the code when starting monitoring the region
    }

}
```

2. Set RECOMonitoringListener interface, and start monitoring.

```
recoManager.setMonitoringListener(this);

//You can set scan period. The default is 1 second.
```

```
recoManager.setScanPeriod(TIME IN MILLISECOND);

//You can set sleep period, which represents the time lag between scanning.
//The default is 10 seconds.
recoManager.setSleepPeriod(TIME IN MILLISECOND);

ArrayList<RECOBeaconRegion> monitoringRegions = new ArrayList<RECOBeaconRegion>();
monitoringRegions.add(new RECOBeaconRegion(YOUR REGION UUID, YOUR REGION UNIQUE IDENTIFIER)
monitoringRegions.add(new RECOBeaconRegion(YOUR REGION UUID, YOUR REGION MAJOR,
                                YOUR REGION UNIQUE IDENTIFIER));

for(RECOBeaconRegion region : monitoringRegions) {
    try {
        //You can set expiration time of the region. The default is 5 minutes.
        region.setRegionExpirationTimeMillis(TIME_IN_MILLISECOND);
        recoManager.startMonitoringForRegion(region);
    } catch (RemoteException e) {
        //Write the code when Remote Exception is occurred.
    } catch (NullPointerException e) {
        //Write the code when Null Pointer Exception is occurred.
    }
}
```

3. Stop monitoring.

```
for(RECOBeaconRegion region : monitoringRegions) {
    try {
        recoManager.stopMonitoringForRegion(region);
    } catch (RemoteException e) {
        //Write the code when Remote Exception is occurred.
    } catch (NullPointerException e) {
        //Write the code when Null Pointer Exception is occurred.
    }
}
```

For more detailed information, please refer to the sample project, especially RECOActivity.java, RECOMonitoringActivity.java, and RECOBackgroundService.java, in the RECO SDK Package.

2.5 Ranging RECOBeaconRegion

Note: RECO SDK for Android perform ranging regions using the instance of the RECOBeaconManager. Before raning regions, the instance of the RECOBeaconManager must be connected to the RECOBeaconService, and RECORangingListener interface is required.

1. Implement RECORangingListener and its callback method.

```
import com.perples.recosdk.RECOBeaconRegion;
import com.perples.recosdk.RECOBeaconRegionState;
import com.perples.recosdk.RECORangingListener;

public class YourClassName implements RECORangingListener {

    ..
    ..
    ..

    public void didRangeBeaconsInRegion(Collection<RECOBeacon> recoBeacons,
                                        RECOBeaconRegion recoRegion) {
        //This callback method is called every second
        //                for ranged regions with their ranged beacons
        //Write the code when ranged RECOBeacon list is received for the ranged region
    }
```

2. Set RECORangingListener interface and start ranging.

```
recoManager.setRangingListener(this);

ArrayList<RECOBeaconRegion> rangingRegions = new ArrayList<RECOBeaconRegion>();
rangingRegions.add(new RECOBeaconRegion(YOUR REGION UUID, YOUR REGION UNIQUE IDENTIFIER));
rangingRegions.add(new RECOBeaconRegion(YOUR REGION UUID, YOUR REGION MAJOR,
                                        YOUR REGION UNIQUE IDENTIFIER));

for(RECOBeaconRegion region : rangingRegions) {
    try {
        recoManager.startRangingBeaconsInRegion(region);
    } catch (RemoteException e) {
        //Write the code when Remote Exception is occurred.
    } catch (NullPointerException e) {
        //Write the code when Null Pointer Exception is occurred.
    }
}
```

3. Stop ranging.

```
for(RECOBeaconRegion region : rangingRegions) {
    try {
        recoManager.stopRangingBeaconsInRegion(region);
    } catch (RemoteException e) {
        //Write the code when Remote Exception is occurred.
    } catch (NullPointerException e) {
        //Write the code when Null Pointer Exception is occurred.
    }
}
```

For more detailed information, please refer to the sample project, especially RECOActivity.java and RECORangingActivity.java, in the RECO SDK Package.

2.6 Checking a state of RECOBeaconRegion

Note: RECO SDK for Android can check the state of each monitored regions with requestStateForRegion(RECOBeaconRegion) method using the instance of RECOBeaconManager. Before requesting state of regions, the instance of the RECOBeaconManager must be connected to the RECOBeaconService, and RECOMonitoringListener is required. In addition, the requested region must be monitored.

Warning: requestStateForRegion(RECOBeaconRegion) method performs asynchronously, and delivers the results to didDetermineStateForRegion(RECOBeaconRegionState, RECOBeaconRegion) callback method.

1. Implement RECOMonitoringListener and its callback methods.

```
import com.perples.recosdk.RECOBeaconRegion;
import com.perples.recosdk.RECOBeaconRegionState;
import com.perples.recosdk.RECOMonitoringListener;

public class YourClassName implements RECOMonitoringListener {

    ..
    ..
    ..

    public void didDetermineStateForRegion(RECOBeaconRegionState recoRegionState,
                                            RECOBeaconRegion recoRegion) {
        //The result of requestStateForRegion(RECOBeaconRegion) method is received
        //Write the code when the result is received.
    }

    public void didEnterRegion(RECOBeaconRegion recoRegion) {
        //Write the code when the device is entered the region
    }

    public void didExitRegion(RECOBeaconRegion recoRegion) {
        //Write the code when the device is exited the region
    }

    public void didStartMonitoringForRegion(RECOBeaconRegion recoRegion) {
        //This method is called right after starting monitoring a region
        //Write the code when starting monitoring the region
    }

}
```

2. Set RECOMonitoringListener interface and call requestStateForRegion(RECOBeaconRegion).

```
RECOBeaconRegion region = new RECOBeaconRegion(YOUR REGION UUID,
                                                    YOUR REGION UNIQUE IDENTIFIER);
recoManager.setMonitoringListener(this);
recoManager.startMonitoringForRegion(region);
recoManager.requestStateForRegion(region);
```

For more detailed information, please refer to the sample project, especially RECOActivity.java and RECOMonitoringActivity.java, in the RECO SDK Package.

2.7 Notes on Development

1. Running in the Background

- **Region Monitoring** : Region Monitoring can be run in the background. For more detailed information, please refer to the sample project, especially RECOBackgroundMonitoringService.java, in the RECO SDK Package.

Warning: Scan period, sleep period, and the number of beacons may impact the battery usage.

- **Region Ranging** : Region Ranging can be run in the background, but we do not recommend. However, while monitoring in the background, it can help scan beacons around when enter or exit events are occurred.

For more detailed information, please refer to the sample project, especially RECOBackgroundRangingService.java, in the RECO SDK Package.

Warning: Ranging is to scan beacons around continuously, which means DOES NOT sleep to scan. It may impact the battery drain, so running ranging in the background is not recommended. Also, ranging will timeout after 10 seconds ranging while monitoring (in the background). If you do not want to set timeout for ranging while monitoring in the background, please set false for the setting.

```
//If you do not want to set timeout for ranging while monitoring in the background,
//create an instance of RECOBeaconManager as follows
//warning: it affetcs the battery consumption
RECOBeaconManager recoManager = RECOBeaconManager.getInstance(this, false);
```

2. A Known Android Bug

There is a known android bug that some android devices scan BLE devices only once. (link: http://code.google.com/p/android/issues/detail?id=65863) To resolve the bug in our SDK, you can use setDiscontinuousScan() method of the RECOBeaconManager. This method is to set whether the device scans BLE devices continuously or discontinuously. The default is set as FALSE. Please set TRUE only for specific devices.

Warning: If you set TRUE, didRangeBeaconsInRegion() callback method may not be called every second.

3. Exception

- **RemoteException** : RECOBeaconManager is not bound to RECOBeaconService. Call RECOBeaconManager.bind(RECOServiceConnectListner serviceConnectLisnter) first.

- **NullPointerException** : Null RECOBeacon parameter is received, or RECOBeaconManager does not have RECOMonitoringListener or RECORangingListener depending on methods (monitoring or ranging). You should set proper Listener first.