**Bluetooth LE Library for Android provided with a Sample Application to test.**

*\*\*note- After building, apk can be found at the following path:*

*AndroidStudioProjects\Bluetooth-LE-Library---Android-master\sample\_app\build\outputs\apk*

This library offers

\* Easy access to a Bluetooth LE device's Advertisement Records.

\* A simple running average RSSI reading keeping.

\* Calculate distance between BLE shield and the android smartphone receiving the signal.

This will only work on devices with Android 4.3 (API Level 18) and above.

**Including the Library in Project**

This project is available as an artifact for use with Gradle.

**Gradle is a build system.**  
The build system automatically takes all the source files (.java or .xml), then applies the appropriate tool (e.g. takes java class files and converts them to dex files), and groups all of them into one compressed file, i.e., APK.

To use that, add the following blocks to your build.gradle file:

```groovy

repositories {

maven {

url "https://dl.bintray.com/alt236/maven"

}

}

dependencies {

compile 'uk.co.alt236:bluetooth-le-library-android:1.0.0'

}

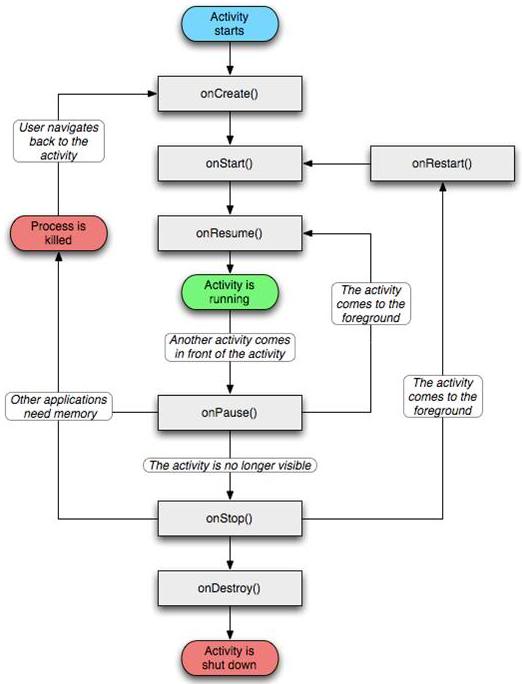
```

If you \*really\* need a Jar file, fork the project and execute `./gradlew clean build generateRelease` at the root of the project.

This will create a zip file under `<PROJECT\_ROOT>/library/build/` the Jar can be found inside.

**FLOW OF CONTROL IN AN ANDROID APPLICATION.**

Activities are one of the fundamental building blocks of apps on the Android platform. They serve as the entry point for a user's interaction with an app, and are also central to how a user navigates within an app (as with the Back button) or between apps (as with the Recent button).



**Using the Library**

In the **`onLeScan()` method** of `BluetoothAdapter.LeScanCallback()` **create a new BluetoothLeDevice** with the given information.

LeScanCallback() is used to deliver LE Scan results.

example:

private BluetoothAdapter.LeScanCallback mLeScanCallback = new BluetoothAdapter.LeScanCallback() {

@Override

public void onLeScan(final BluetoothDevice device, int rssi, byte[] scanRecord) {

final BluetoothLeDevice deviceLe = new BluetoothLeDevice(device, rssi, scanRecord, System.currentTimeMillis());

runOnUiThread(new Runnable() {

@Override

public void run() {

mDeviceStore.addDevice(deviceLe);

mLeDeviceListAdapter.replaceData(mDeviceStore.getDeviceList());

}

});

}

};

```

Once you have created a device, we have to choose one of the following options on Home screen:

* Scan

then Stop

* About
* Share

**public boolean** onOptionsItemSelected(**final** MenuItem item) {  
 **switch** (item.getItemId()) {  
 **case** R.id.***menu\_scan***:  
 startScanPrepare();  
 **break**;  
 **case** R.id.***menu\_stop***:  
 **mScanner**.scanLeDevice(-1, **false**);  
 invalidateOptionsMenu();  
 **break**;  
 **case** R.id.***menu\_about***:  
 DialogFactory.*createAboutDialog*(**this**).show();  
 **break**;  
 **case** R.id.***menu\_share***:  
 **new** Sharer().shareDataAsEmail(**this**, **mDeviceStore**);  
 }  
 **return true**;  
}

After the scan, you can access the following information:

\* `getAddress()` : Gets the MAC Address of the device

\* `getAdRecordStore()`: Gives access to a device's Ad Records

\* `getDevice()`: Gives access to the standard BluetoothDevice object

\* `getFirstRssi()`: Retrieves the RSSI value which was used when the object was created

\* `getFirstTimestamp()` Retrieves the timestamp (in millis) which was used when the object was created

\* `getRssi()` Gets the current RSSI measurement (see note below).

\* `getScanRecord()` Retrieves the RAW scan record array

\* `getTimestamp()` Gets the timestamp of the last RSSI measurement

\* `getRunningAverageRssi()` Retrieves the internally calculated running average RSSI value and calculates distance based on this average value. The function thus returns distance. Name is a misnomer. *Default TX value is taken as -58dbm.*

\* `getcount()` Gives the number of values taken as sample for calculating the average value. Max number of samples is 10.

\* `get value` Gives a list of samples taken during the period of scanning.

\*\*Note:\*\* The Running Average RSSI is not updated automatically (i.e. the library does not monitor on its own in the background). To add another measurement, you need to call `updateRssiReading(long timestamp, int rssiReading)`.

**PAUSE**

**protected void** onPause() {  
 **super**.onPause();  
 **mScanner**.scanLeDevice(-1, **false**);  
}

# sending -1 as parameter, maps it to duration. It is easily observable in the code that it would scan for duration>0. Hence it would pause the activity.

**RESUME**

public void onResume() {

super.onResume();

if (mBluetoothUtils.isBluetoothOn()) {

mTvBluetoothStatus.setText(R.string.on);

} else {

mTvBluetoothStatus.setText(R.string.off);

}

if (mBluetoothUtils.isBluetoothLeSupported()) {

mTvBluetoothLeStatus.setText(R.string.supported);

} else {

mTvBluetoothLeStatus.setText(R.string.not\_supported);

}

invalidateOptionsMenu();

}

An **IBeaconDevice** extends BluetoothLeDevice, so you still have access to the same methods as before**. In addition you can do the following:**

\* `getAccuracy()`: Gets the estimated Accuracy of the reading in meters based on a simple running average calculation

\* `getCalibratedTxPower()`: Gets the calibrated TX power of the iBeacon device as reported

\* `getCompanyIdentifier()`: Gets the iBeacon company identifier (this should always be 0x004C for Apple)

\* `getDistanceDescriptor()`: Gets the estimated Distance descriptor (an enum)

\* `getIBeaconData()`: Gets the raw IBeaconManufacturerData object.

\* `getUUID()`: Gets the device's UUID

\* `getMajor()`: Gets the device's Major value

\* `getMinor()`: Gets the device's Minor value

**Lookup Functions**

You can also lookup values and convert them to human friendly strings:

\* `BluetoothClassResolver.resolveDeviceClass(int btClass)`: Will try to resolve a Blueotooth Device class

\* `CompanyIdentifierResolver.getCompanyName(int companyId, String fallback)`: Will try to resolve a Company identifier to the company name

\* `GattAttributeResolver.getAttributeName(String uuid, String fallback)`: Will try to convert a UUID to its name.

## Library Changelog

\* v0.0.1

\* First public release

\* v0.0.2:

\* Attempting to create an iBeaconDevice from a device which is not an iBeacon will now throw an IllegalArgumentException exception.

\* Fixed a ConcurrentModificationException on getRunningAverageRssi()

\* Added some Estimote UUIDs

\* v1.0.0:

\* Migrated project to Android Studio/ gradle

\* Note that the API has slightly changed in this version.

\* We now use the more generic `BeaconUtils.getBeaconType()` method instead of `IBeaconUtils.isThisAnIBeacon()`

\* Fix for [issue 5](https://github.com/alt236/Bluetooth-LE-Library---Android/issues/5)

\* Fix for [issue 9](https://github.com/alt236/Bluetooth-LE-Library---Android/issues/9)

## Sample Application Changelog

\* v0.0.1

\* First public release

\* v0.0.2:

\* Can now export scanned devices as a CSV file.

\* v0.0.3:

\* UI Refresh.

\* v1.0.0:

\* Migrated project to Android Studio/ gradle

\* Using version v1.0.0 of the library project

\* v1.1.0:

\* App refactor and materialisation.

\* Added runtime permissions.

\* v1.1.1:

\* Fix for [issue 23](https://github.com/alt236/Bluetooth-LE-Library---Android/issues/23)

## Permission Explanation

You will need the following permissions to access the Bluetooth Hardware

\* `android.permission.BLUETOOTH`

\* `android.permission.BLUETOOTH\_ADMIN`

In addition one of the following is needed from API 23 and above to scan for BT LE devices:

\* `android.permission.ACCESS\_COARSE\_LOCATION`

\* `android.permission.ACCESS\_FINE\_LOCATION `

## TODO

\* Tidy up Javadoc. There is quite a lot of it that is template

\* Add parsers for common Ad Records.

## Links

\* Github: [https://github.com/alt236/Bluetooth-LE-Library---Android]()

## Credits

Author: [Alexandros Schillings](https://github.com/alt236).

\* The Accuracy calculation algorithm was taken from: http://stackoverflow.com/questions/20416218/understanding-ibeacon-distancing

\* The AdRecord parser was taken from: https://github.com/devunwired/accessory-samples

\* The sample application has been adapted from Android's Bluetooth LE example

\* Google Play and the Google Play logo are trademarks of Google Inc.

All logos are the property of their respective owners.

The code in this project is licensed under the Apache Software License 2.0.

Copyright (c) 2014-2017 Alexandros Schillings.