Software SDK USER GUIDE

Elo Touch Solutions

I-Series Interactive Signage ESY10i1, ESY15i1, ESY22i1
Android ECM (ELO-KIT-ECMG2-AND)

SW602422 Rev A

Disclaimer

The information in this document is subject to change without notice. Elo Touch Solutions, Inc. and its affiliates (collectively "Elo") makes no representations or warranties with respect to the contents herein, and specifically disclaims any implied warranties of merchantability or fitness for a particular purpose. Elo reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Elo to notify any person of such revisions or changes.

Trademark Acknowledgments

AccuTouch, CarrollTouch, Elo (logo), Elo Touch Solutions, Elo TouchSystems, IntelliTouch, iTouch are trademarks of Elo and its Affiliates. Windows is a trademark of Microsoft Corporation.

Table of Contents

Chapter 1: Introduction

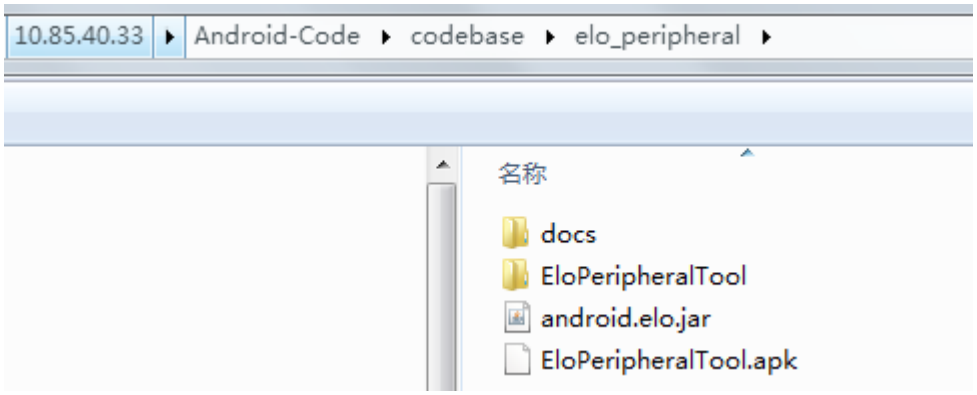
Overview

ELO software SDK allows developers to use the specific API for their own application. This SDK allows developers to control the ELO peripheral Test App, GPIO interface and use iBeacon functions.

Before developing applications with functions leveraging iBeacon technology, please make sure to first visit Apple iBeacon website (https://developer.apple.com/ibeacon/) to review and sign the online agreement before you develop related applications. iBeacon related documents are available from Apple’s website.

Chapter 2: SDK File Structure

File Structure



1. Docs: Javadoc for ibeacon/peripheral api

2. EloPeripheralTool: Source code for EloPeripheralTool

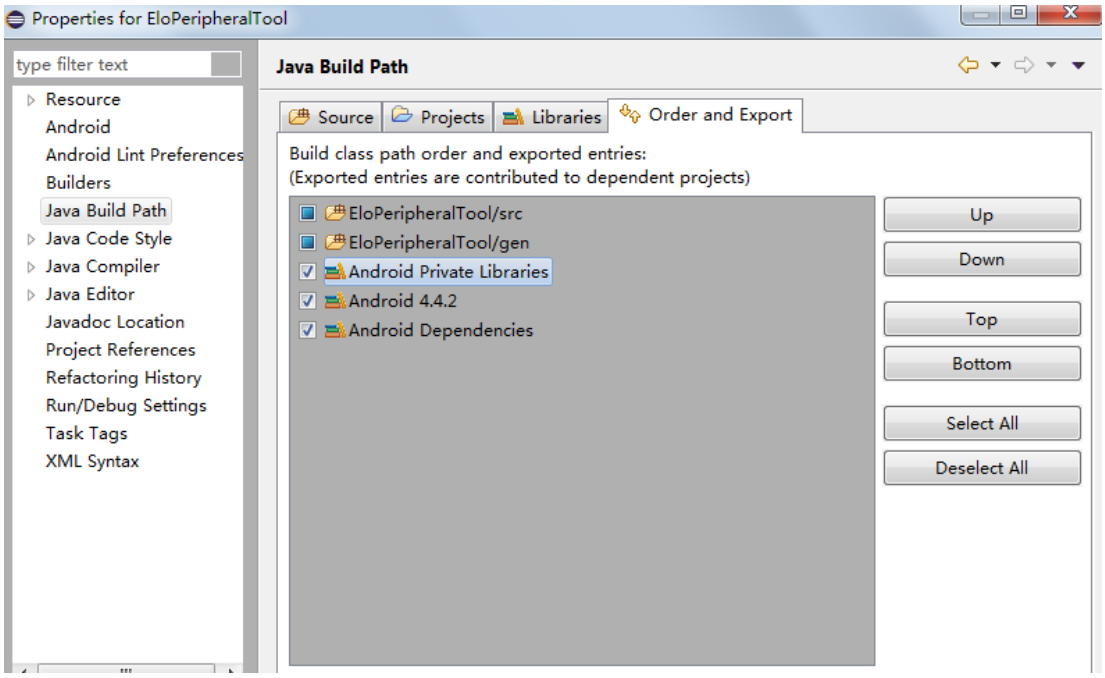
3. android.elo.jar: SDK jar for iBeacon/peripheral API

4. EloPeripheralTool.apk: Sample app
   Note: Screenshot of EloPeripheralTool application below:

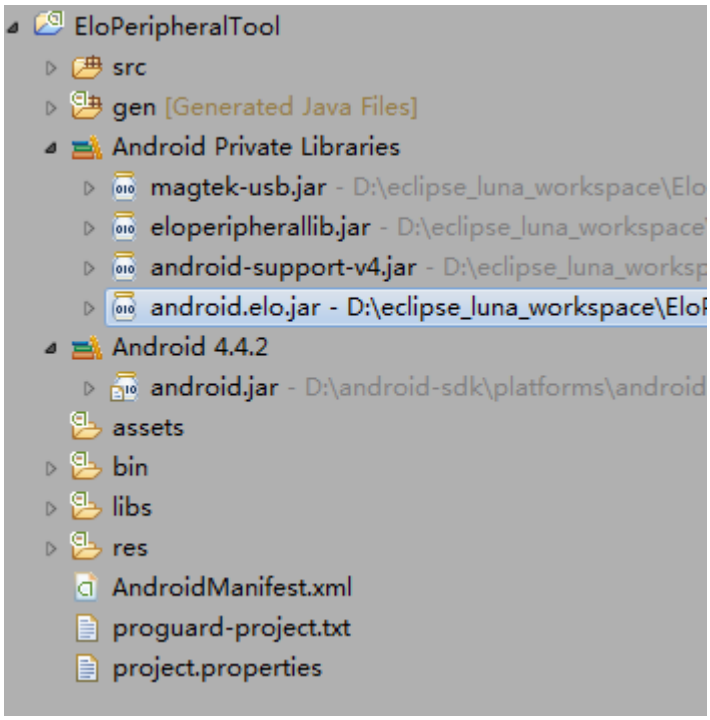
Chapter 3: How to use SDK

Procedure for Eclipse

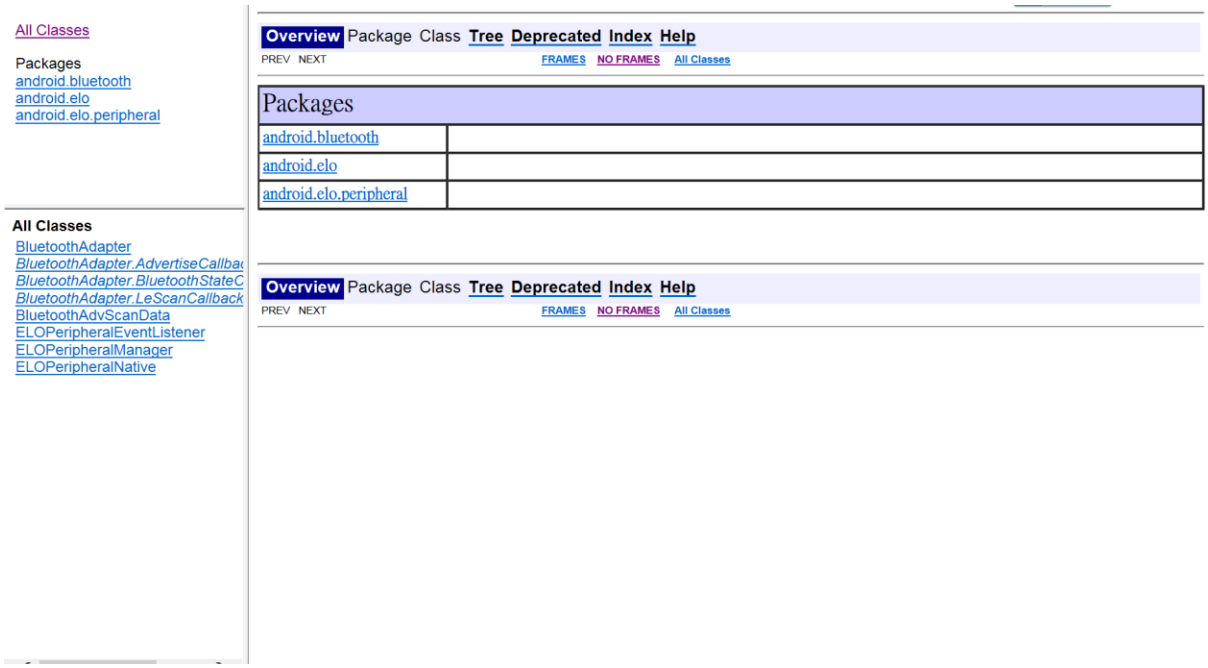
1. Modify the Java Build Path class order to make the android.elo.jar be ahead of the android.jar.



2. Import the android.elo.jar into Android Private Libraries to avoid conflict with Bluetooth API.



3. After those are done, you could develop with ELO API. If you have questions for API definition, please go to docs/elo folder and launch the index.html. You could get the details of API.



Chapter 4: Peripheral API

Glossary

Abbreviations:

- SoC: System on a chip

Terms

- IDLE_MODE: ELO specification for device interaction with POWER_KEY action
- MSR: Magnetic Stripe Reader
- BCR: Barcode Reader
- NFC: Near Field Communication
- ELOPeripheralService: The fundamental class to provide different functions for ELO application development.

System Overview

Fig 4-1 below shows the software stack of Android system. This implementation based on Android design and extends the framework functionality at java services layer to fulfill customer requirement. The extended functionality at java service is named as “ELOPeripheralService” which provides the interfaces for application development.

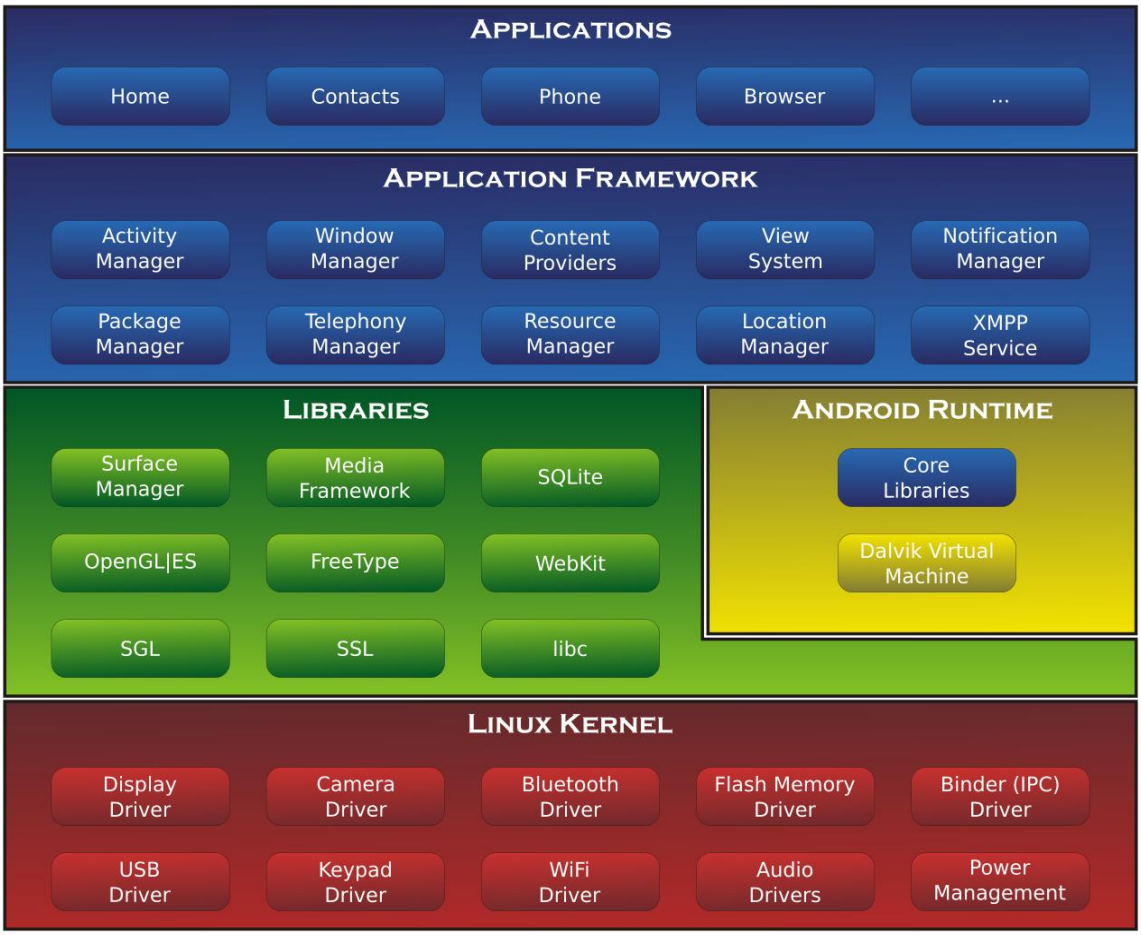


Fig 4-1. Android System Structure

1. Android Framework Java Service

Based on developed programming language, Android services can be divided into native (C base) and java service. Android services are a kind of server-client structure where the server side can receive a request from the client and output the result to client. Also, the server-client structure can deal with synchronization issues from multi-client requests. Therefore, this framework ability, which forms the fundamental class “ELOPeripheralService,” is based on this kind of server-client design feature from Android.

2. ELOPeripheralService

ELOPeripheralService is the major module to act client request from application layer. In order to complete the Android framework system design for service then below classes is added into system:

- ELOPeripheralNative

  Execution of some native-C function flows for ELOPeripheralService

- ELOPeripheralManager

  Provides the interface to the application layer, which is the client component relative to ELOPeripheralService

- EloPeripheralEventListener

  Application extends abstract class to receive the driven event from EloPeripheralManager

  Uses event driven notification method to notify the application program about the status change

- IELOPeripheralService.aidl

  Android programming language for java layer IPC communication

  Used for Client-Server (ELOPeripheralManager-ELOPeripheralService) IPC communication where sending the application's request to server side

- IELOPeripheralServiceListener.aidl

  Android programming language for java layer IPC communication

  Used for server to notify the event driven status change to ELOPeripheralManager then ELOPeripheralManager will notify to application layer

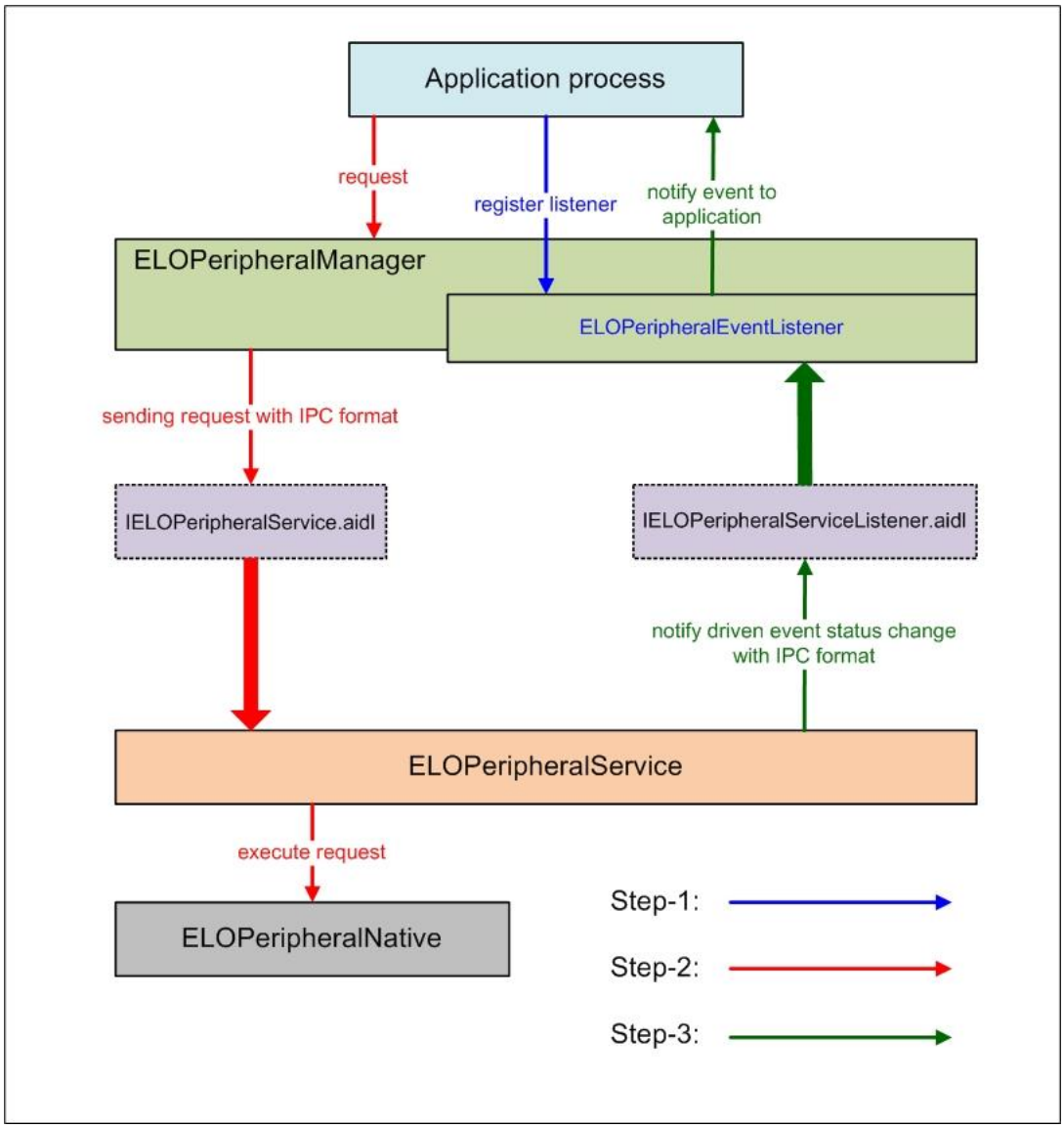


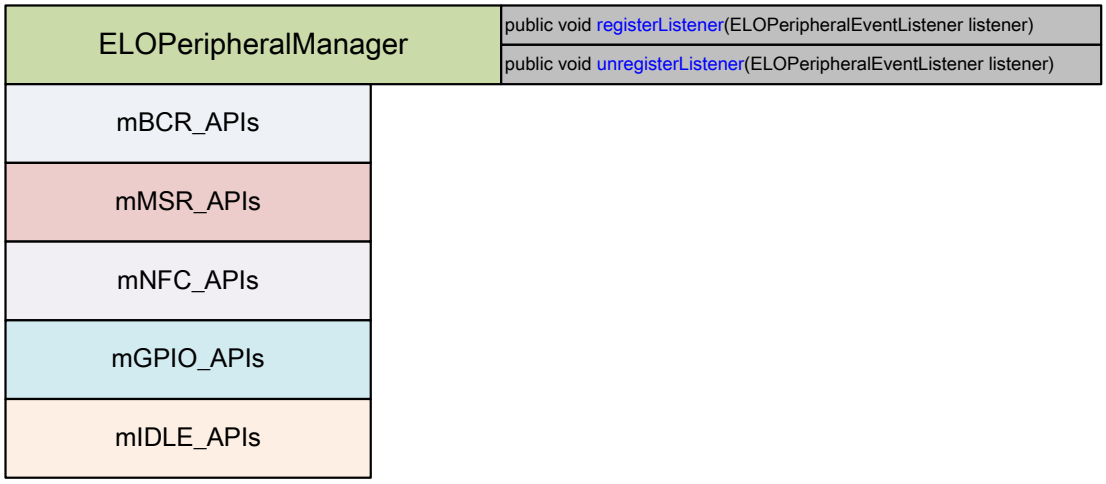
Figure 2-2. ELOPeripheralService

ELOPeripheralManager and ELOPeripheralEventListener

As we mentioned at last section, ELOPeripheralManager and ELOPeripheralEventListener will be the component to provide interface for application development. Therefore, we provide the inner member data of ELOPeripheralManager and ELOPeripheralEventListener as below.

1. ELOPeripheralManager

Inside this manager contains some inner class member as below and for each inner class provides specific functionality to peripheral equipment or defined action flow.



- public void registerListener(ELOPeripheralEventListener listener)
    - Provides interface for application to register the feedback structure instance
    - Needs to register before request (Activity life cycle: onResume)
- public void unregisterListener(ELOPeripheralEventListener listener)
    - Provides interface for application to unregister the feedback structure instance
    - Needs to unregister before leave the application (Activity life cycle: onPause)

1.1. BCR

```
public class BCR {
    public boolean activeBCR() throws RemoteException {}
    public boolean disactiveBCR() throws RemoteException {}
    public boolean isBCRDeviceConnected() throws RemoteException {}
}
```

- activeBCR
    - Activates the BCR device to reading the barcode dataBCR is active and successful reading the barcode sequence data then system by “ELOPeripheralEventListener” to notify application about the receiving data
- disactiveBCR

- Deactivate the BCR device reading action

- isBCRDeviceConnected
  - Check the BCR device is connection with Android device or not

1.2. GPIOs

```
public class GPIOs {
    public void pullHighGPIO(String iface) throws RemoteException {}
    public void pullLowGPIO(String iface) throws RemoteException {}
    public String[] getGPIOInterafces() throws RemoteException {}
}
```

- pullHighGPIO
  - Pull high the state of GPIO-iface

- PullLowGPIO
  - Pull low the state of GPIO-iface disactiveBCR

- getGPIOInterfaces
  - Get the list string data of interest GPIOs at designed system.
  - Refer to Table 3-1 GPIOs list

Note:

- The pull action only workable with output pin defined GPIO
- The input pin defined GPIO can notify the application about the status change by “ELOPeripheralEventListener” also

GPIOs	PIN define
gpio80	Output
gpio81	Input
gpio82	Input

Table 3-1 GPIOs List

1.3. IDLE

```
public class IDLE {
    public boolean activeIdleMode() {}

    public boolean disactiveIdleMode() {}
}
```

- activeIdleMode
    - Force system to enter IDLE_MODE (IDLE_MODE on)
- disactiveIdleMode
    - Force system to leave IDLE_MODE (IDLE_MODE off)

2. ELOPeripheralEventListener

```
public abstract class ELOPeripheralEventListener {
    /**
     * Active barcode scanner reading function by USB-ID-PIN
     */
    void onBCR_StateChange(int state, String data) {
    }

    void onGPIO_StateChange(int state, String data) {
    }
}
```

- onBCR_StateChange
    - Refer to Table 3-2
    - Notify the application about the BCR status change data
- onGPIO_StateChange
    - Refer to Table 3-2
    - Notify the application about the GPIOs status change data

BCR state	value
ELOPeripheralManager.BCR_STATE_DEVICE_CONNECTION	1 << 0
ELOPeripheralManager.BCR_STATE_DEVICE_DISCONNECTION	1 << 1
ELOPeripheralManager.BCR_STATE_DATA_RECEIVIED	1 << 2
ELOPeripheralManager.BCR_STATE_PIN_AUTO_DISABLE	1 << 3

GPIO state	value
ELOPeripheralManager.GPIO_STATE_HIGH	1 << 4
ELOPeripheralManager.GPIO_STATE_LOW	1 << 5

GPIO data
gpio81
gpio82

Table 3-2

Bluetooth Adapter

Class BluetoothAdapter
java.lang.Object

└ android.bluetooth.BluetoothAdapter

public final class BluetoothAdapter
extends java.lang.Object

Represents the local device Bluetooth adapter.

The BluetoothAdapter lets you perform fundamental Bluetooth tasks, such as initiate device discovery, query a list of bonded (paired) devices, instantiate a BluetoothDevice using a known MAC address, and create a BluetoothServerSocket to listen for connection requests from other devices, and start a scan for Bluetooth LE devices.

To get a BluetoothAdapter representing the local Bluetooth adapter, when running on JELLY_BEAN_MR1 and below, call the static getDefaultAdapter() method; when running on JELLY_BEAN_MR2 and higher, retrieve it through Context.getSystemService(java.lang.String) with Context.BLUETOOTH_SERVICE. Fundamentally, this is your starting point for all Bluetooth actions. Once you have the local adapter, you can get a set of BluetoothDevice objects representing all paired devices withgetBondedDevices(); start device discovery with startDiscovery(); or create a BluetoothServerSocket to listen for incoming connection requests withlistenUsingRfcommWithServiceRecord(String,UUID); or start a scan for Bluetooth LE devices with startLeScan(LeScanCallback callback).

Note: Most methods require the Manifest.permission.BLUETOOTH permission and some also require the Manifest.permission.BLUETOOTH_ADMIN;Manifest.permission.BLUETOOTH_PRIVILEGED;

1. How to use ibeacon advertise

- **Get BluetoothAdapter:**

  You should get BluetoothAdapter instance by BluetoothManager.getAdapter() defined in BluetoothManager.

- **Get BluetoothAdvScanData:**

  You should get BluetoothAdvScanData instance by getAdvScanData() defined in BluetoothAdvScanData.

- **Set manufacturer data**

  Call BluetoothAdvScanData BluetoothAdvScanData.setManufacturerData(int, byte[]) to set manufacturer data.

- **Start advertising:**

  Now you can
  call startAdvertising(android.bluetooth.BluetoothAdapter.AdvertiseCallback) to start advertise.

- **whether advertising:**

  you can call isAdvertising() to check start advertise or not.

- **Stop advertising:**

  Now you can
  call stopAdvertising(android.bluetooth.BluetoothAdapter.AdvertiseCallback) to stop advertise.

Nested Class Summary	
static interface	**BluetoothAdapter.AdvertiseCallback** Interface for BLE advertising callback.
static interface	**BluetoothAdapter.BluetoothStateChangeCallback**
static interface	**BluetoothAdapter.LeScanCallback** Callback interface used to deliver LE scan results.
class	**BluetoothAdapter.StateChangeCallbackWrapper**

2. BluetoothAdapter.AdvertiseCallback

Enclosing class:
BluetoothAdapter

---

public static interface **BluetoothAdapter.AdvertiseCallback**

Interface for BLE advertising callback.

Method Summary	
void	**onAdvertiseStart**(int status) Callback when advertise starts.
void	**onAdvertiseStop**(int status) Callback when advertise stops.

Method Detail

onAdvertiseStart

void **onAdvertiseStart**(int status)

Callback when advertise starts.

Parameters:
status - - BluetoothAdapter.ADVERTISE_CALLBACK_SUCCESS for success, others for failure.

---

onAdvertiseStop

void **onAdvertiseStop**(int status)

Callback when advertise stops.

Parameters:
status - - BluetoothAdapter.ADVERTISE_CALLBACK_SUCCESS for success, others for failure.

3. BluetoothAdapter.BluetoothStateChangeCallback

android.bluetooth

Interface BluetoothAdapter.BluetoothStateChangeCallback

Enclosing class:

BluetoothAdapter

---

public static interface **BluetoothAdapter.BluetoothStateChangeCallback**

Method Summary

void	**onBluetoothStateChange**(boolean on)

Method Detail

onBluetoothStateChange

void **onBluetoothStateChange**(boolean on)

4. BluetoothAdapter.LeScanCallback

android.bluetooth

Interface BluetoothAdapter.LeScanCallback

Enclosing class:

BluetoothAdapter

---

public static interface **BluetoothAdapter.LeScanCallback**

Callback interface used to deliver LE scan results.

See Also:

BluetoothAdapter.startLeScan(LeScanCallback), BluetoothAdapter.startLeScan(UUID[], LeScanCallback)

Method Summary

void	**onLeScan**(android.bluetooth.BluetoothDevice device, int rssi, byte[] scanRecord) Callback reporting an LE device found during a device scan initiated by the BluetoothAdapter.startLeScan(android.bluetooth.BluetoothAdapter.LeScanCallback) function.

Method Detail

onLeScan

void **onLeScan**(android.bluetooth.BluetoothDevice device,
int rssi,
byte[] scanRecord)

Callback reporting an LE device found during a device scan initiated by the BluetoothAdapter.startLeScan(android.bluetooth.BluetoothAdapter.LeScanCallback) function.

Parameters:

device - Identifies the remote device
rssi - The RSSI value for the remote device as reported by the Bluetooth hardware. 0 if no RSSI value is available.
scanRecord - The content of the advertisement record offered by the remote device.

5. BluetoothAdvScanData

android.bluetooth

Class BluetoothAdvScanData

java.lang.Object
└ **android.bluetooth.BluetoothAdvScanData**

---

public final class **BluetoothAdvScanData**
extends java.lang.Object

This class provides the public APIs to set advertising and scan response data when BLE device operates in peripheral mode.
The exact format is defined in Bluetooth 4.0 specification, Volume 3, Part C, Section 11

How to use API

- **AndroidManifest.xml**

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

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

- **Import classes**

  import android.bluetooth.BluetoothAdapter;

  import android.bluetooth.BluetoothAdapter.AdvertiseCallback;

  import android.bluetooth.BluetoothAdvScanData;

  import android.bluetooth.BluetoothManager;

- **Use details**

  private BluetoothAdapter mBluetoothAdapter;

  private BluetoothAdvScanData mBluetoothAdvScanData;

  private BluetoothAdapter.AdvertiseCallback mAdvertiseCallback;

```
//get instance
final BluetoothManager bluetoothManager =
        (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
mBluetoothAdapter = bluetoothManager.getAdapter();
mBluetoothAdvScanData = mBluetoothAdapter.getAdvScanData();


final byte[] manufacturerData = new byte[] {
        (byte) 0x4c, (byte) 0x00, (byte) 0x02, (byte) 0x15, // fix
        // proximity uuid 01020304-0506-0708-1112-131415161718
        (byte) 0x01, (byte) 0x02, (byte) 0x03, (byte) 0x04, // uuid
        (byte) 0x05, (byte) 0x06, (byte) 0x07, (byte) 0x08, // uuid
        (byte) 0x11, (byte) 0x12, (byte) 0x13, (byte) 0x14, // uuid
        (byte) 0x15, (byte) 0x16, (byte) 0x17, (byte) 0x18, // uuid
        (byte) 0x01, (byte) 0x01, // major 257
        (byte) 0x02, (byte) 0x02, // minor 514
        (byte) 0xc5 // Tx Power -59
};
mBluetoothAdvScanData.setManufacturerData(1, manufacturerData);
mBluetoothAdapter.startAdvertising(getAdvertiseCallback());
```

```
//whether BLE is currently advertising, if true stop advertising

if(mAdapter.isAdvertising()){

        mBluetoothAdapter.stopAdvertising(getAdvertiseCallback());

}
```

Field Summary	
static int	**AD** Available data types of BluetoothAdvScanData.
static int	**EIR**
static int	**SCAN_RESPONSE**

Constructor Summary
BluetoothAdvScanData(android.bluetooth.IBluetoothGatt mBluetoothGatt, int dataType)

Method Summary	
int	**getDataType**()
byte[]	**getManufacturerData**() Get manufacturer data.
byte[]	**getServiceData**() Get service data
java.util.List<android.os.ParcelUuid>	**getServiceUuids**() Get an immutable list of service uuids that will be advertised.
void	**removeManufacturerCodeAndData**(int manufacturerCode) Remove manufacturer data based on given manufacturer code.
boolean	**setManufacturerData**(int manufacturerCode, byte[] manufacturerData) Set manufactureCode and manufactureData.
boolean	**setServiceData**(byte[] serviceData) Set service data.

Method Detail

getDataType

public int **getDataType**()

Returns:
advertising data type.

---

setManufacturerData

public boolean **setManufacturerData**(int manufacturerCode,
byte[] manufacturerData)

Set manufactureCode and manufactureData.

Parameters:
manufacturerCode - - unique identifier for the manufacturer
manufacturerData - - data associated with the specific manufacturer.
Returns:
true if manufacturer data is set, false if there is no enough room to set manufacturer data or the data is already set.

---

getManufacturerData

public byte[] **getManufacturerData**()

Get manufacturer data.

Returns:
manufacturer data

---

setServiceData

public boolean **setServiceData**(byte[] serviceData)

Set service data. Note the service data can only be set when advertising;

Parameters:
serviceData -

---

getServiceData

public byte[] **getServiceData**()

Get service data

Returns:
service data.

---

getServiceUuids

public java.util.List<android.os.ParcelUuid> **getServiceUuids**()

Get an immutable list of service uuids that will be advertised.

Returns:
a list containing uuids

---

removeManufacturerCodeAndData

public void **removeManufacturerCodeAndData**(int manufacturerCode)

Remove manufacturer data based on given manufacturer code.

Parameters:
manufacturerCode -
